Space Physics Data Facility
NASA/Goddard Space Flight Center
Greenbelt, Maryland 20771 (U.S.A.)

This software may be copied or redistributed as long as it is not sold for profit, but it can be incorporated into any other substantive product with or without modifications for profit or non-profit. If the software is modified, it must include the following notices:

- The software is not the original (for protection of the original author’s reputations from any problems introduced by others)
- Change history (e.g. date, functionality, etc.)

This Copyright notice must be reproduced on each copy made. This software is provided as is without any express or implied warranties whatsoever.

Internet – gsfc-edf-support@lists.nasa.gov
2.2.1 CDFeClose ................................................. 29
2.2.2 CDFeCloseCDF ............................................ 30
2.2.3 CDFcOpen ............................................... 31
2.2.4 CDFcOpenCDF ............................................. 32
2.2.5 CDFcDelete ............................................... 33
2.2.6 CDFcDeleteCDF .......................................... 34
2.2.7 CDFeDoc ................................................ 35
2.2.8 CDFeError .............................................. 36
2.2.9 CDFeGetCacheSize ...................................... 37
2.2.10 CDFeGetChecksum ...................................... 37
2.2.11 CDFeGetCompression .................................. 38
2.2.12 CDFeGetCompressionCacheSize ....................... 39
2.2.13 CDFeGetCompressionInfo .............................. 40
2.2.14 CDFeGetCopyright ..................................... 41
2.2.15 CDFeGetDecoding ...................................... 42
2.2.16 CDFeGetEncoding ...................................... 42
2.2.17 CDFeGetFileBackward ................................ 43
2.2.18 CDFeGetFormat ........................................ 44
2.2.19 CDFeGetLeapSecondLastUpdated ...................... 45
2.2.20 CDFeGetMajority ...................................... 46
2.2.21 CDFeGetName .......................................... 46
2.2.22 CDFeGetNegtoPosfp0Mode ............................ 47
2.2.23 CDFeGetReadOnlyMode ................................ 48
2.2.24 CDFeGetStageCacheSize .............................. 49
2.2.25 CDFeGetValidate ...................................... 50
2.2.26 CDFeGetVersion ...................................... 50
2.2.27 CDFeGetzMode ........................................ 51
2.2.28 CDFeInquire .......................................... 52
2.2.29 CDFeInquireCDF ...................................... 53
2.2.30 CDFeOpen ............................................. 55
2.2.31 CDFeOpenCDF ......................................... 56
2.2.32 CDFeSelect ............................................ 57
2.2.33 CDFeSelectCDF ........................................ 58
2.2.34 CDFeSetCacheSize ..................................... 59
2.2.35 CDFeSetChecksum ...................................... 60
2.2.36 CDFeSetCompression .................................. 60
2.2.37 CDFeSetCompressionCacheSize ....................... 61
2.2.38 CDFeSetDecoding ...................................... 62
2.2.39 CDFeSetEncoding ...................................... 63
2.2.40 CDFeSetFileBackward ................................ 64
2.2.41 CDFeSetFormat ........................................ 64
2.2.42 CDFeSetLeapSecondLastUpdated ...................... 65
2.2.43 CDFeSetMajority ...................................... 66
2.2.44 CDFeSetNegtoPosfp0Mode ............................ 67
2.2.45 CDFeSetReadOnlyMode ................................ 68
2.2.46 CDFeSetStageCacheSize .............................. 68
2.2.47 CDFeValidate .......................................... 69
2.2.48 CDFeSetzMode ........................................ 70

4.3 Variables ..................................................... 71
4.3.1 CDFeCloserVar .......................................... 71
4.3.2 CDFeClosezVar ......................................... 72
4.3.3 CDFeConfirmrVarExistence ............................ 73
4.3.4 CDFeConfirmrVarPadValueExistence .................. 73
4.3.5 CDFeConfirmzVarExistence ............................ 74
4.3.6 CDFeConfirmzVarPadValueExistence .................. 75
4.3.7 CDFeCreateVar ......................................... 76
CDFcreatezVar ................................................................. 78
CDFdeleterVar ................................................................. 79
CDFdeletezVarRecords ....................................................... 80
CDFdeletezVarRecordsRenumber ........................................... 81
CDFdeletezVar ................................................................. 82
CDFdeletezVarRecords ....................................................... 83
CDFdeletezVarRecordsRenumber ........................................... 84
CDFgetMaxWrittenRecNums .................................................. 85
CDFgetMaxVars ................................................................. 86
CDFgetMaxzVars ............................................................... 87
CDFgetVarAllocRecords ..................................................... 88
CDFgetVarBlockingFactor ................................................... 88
CDFgetVarCacheSize .......................................................... 89
CDFgetVar Compression ....................................................... 90
CDFgetVarData ................................................................. 91
CDFgetVarDataType ............................................................ 92
CDFgetVarDimVariances ..................................................... 93
CDFgetVarInfo ................................................................. 94
CDFgetVarMaxAllocRecNum .................................................. 95
CDFgetVarMaxWrittenRecNum .............................................. 96
CDFgetVarName ............................................................... 97
CDFgetVarNumElements ..................................................... 98
CDFgetVarNumRecsWritten .................................................. 99
CDFgetVarPadValue ........................................................... 99
CDFgetVarRecordData .......................................................100
CDFgetVarRecVariance ......................................................101
CDFgetVarReservePercent .................................................102
CDFgetVarsDimSizes ........................................................103
CDFgetVarSeqData ...........................................................104
CDFgetVarSeqPos .............................................................105
CDFgetVarsMaxWrittenRecNum ............................................106
CDFgetVarsNumDims ........................................................107
CDFgetSparseRecords .......................................................107
CDFgetVar .......................................................... 78
CDFgetVarAllocRecords .....................................................109
CDFgetVarBlockingFactor ...................................................110
CDFgetVarCacheSize ........................................................111
CDFgetVarCompression ......................................................112
CDFgetVarData .................................................................113
CDFgetVarDataType ............................................................114
CDFgetVarDimSizes ..........................................................115
CDFgetVarDimVariances .....................................................116
CDFgetVarInfo .................................................................117
CDFgetVarMaxAllocRecNum ..................................................118
CDFgetVarMaxWrittenRecNum ..............................................119
CDFgetVarName ...............................................................120
CDFgetVarNumElements .....................................................121
CDFgetVarNumRecsWritten ..................................................122
CDFgetVarPadValue ...........................................................123
CDFgetVarRecordData .......................................................124
CDFgetVarRecVariance ......................................................125
CDFgetVarReservePercent .................................................126
CDFgetVarSeqData ...........................................................127
CDFgetVarSeqPos .............................................................128
CDFgetVarsMaxWrittenRecNum ............................................128
| 4.3.64 | CDFgetzVarSparseRecords ................................................................. | 129 |
| 4.3.65 | CDFHyperGetVarData ................................................................................ | 130 |
| 4.3.66 | CDFHyperGetzVarData .............................................................................. | 132 |
| 4.3.67 | CDFHyperPutrVarData ............................................................................... | 134 |
| 4.3.68 | CDFHyperPutzVarData ............................................................................... | 135 |
| 4.3.69 | CDFInquirerVar ....................................................................................... | 137 |
| 4.3.70 | CDFinquirezVar ....................................................................................... | 139 |
| 4.3.71 | CDFputrVarData ....................................................................................... | 140 |
| 4.3.72 | CDFputrVarPadValue ................................................................................ | 141 |
| 4.3.73 | CDFputrVarRecordData ............................................................................. | 142 |
| 4.3.74 | CDFputrVarSeqData .................................................................................. | 143 |
| 4.3.75 | CDFputzVarData ....................................................................................... | 144 |
| 4.3.76 | CDFputzVarPadValue ................................................................................ | 146 |
| 4.3.77 | CDFputzVarRecordData ............................................................................. | 147 |
| 4.3.78 | CDFputzVarSeqData .................................................................................. | 148 |
| 4.3.79 | CDFrenamerVar ....................................................................................... | 149 |
| 4.3.80 | CDFrenamezVar ....................................................................................... | 150 |
| 4.3.81 | CDFsetrVarAllocBlockRecords .................................................................. | 150 |
| 4.3.82 | CDFsetrVarAllocRecords .......................................................................... | 151 |
| 4.3.83 | CDFsetrVarBlockingFactor ....................................................................... | 152 |
| 4.3.84 | CDFsetrVarCacheSize ............................................................................... | 153 |
| 4.3.85 | CDFsetrVarCompression .......................................................................... | 154 |
| 4.3.86 | CDFsetrVarDataSpec ................................................................................ | 155 |
| 4.3.87 | CDFsetrVarDimVariances ......................................................................... | 156 |
| 4.3.88 | CDFsetrVarInitialRecs ............................................................................ | 157 |
| 4.3.89 | CDFsetrVarRecVariance .......................................................................... | 158 |
| 4.3.90 | CDFsetrVarReservePercent ....................................................................... | 158 |
| 4.3.91 | CDFsetrVarsCacheSize ............................................................................. | 159 |
| 4.3.92 | CDFsetrVarSeqPos ................................................................................... | 160 |
| 4.3.93 | CDFsetrVarSparseRecords ....................................................................... | 161 |
| 4.3.94 | CDFsetzVarAllocBlockRecords .................................................................. | 162 |
| 4.3.95 | CDFsetzVarAllocRecords .......................................................................... | 163 |
| 4.3.96 | CDFsetzVarBlockingFactor ....................................................................... | 164 |
| 4.3.97 | CDFsetzVarCacheSize ............................................................................... | 165 |
| 4.3.98 | CDFsetzVarCompression .......................................................................... | 165 |
| 4.3.99 | CDFsetzVarDataSpec ................................................................................ | 166 |
| 4.3.100 | CDFsetzVarDimVariances ....................................................................... | 167 |
| 4.3.101 | CDFsetzVarInitialRecs ........................................................................... | 168 |
| 4.3.102 | CDFsetzVarRecVariance .......................................................................... | 169 |
| 4.3.103 | CDFsetzVarReservePercent ..................................................................... | 170 |
| 4.3.104 | CDFsetzVarsCacheSize ........................................................................... | 171 |
| 4.3.105 | CDFsetzVarSeqPos .................................................................................. | 172 |
| 4.3.106 | CDFsetzVarSparseRecords ...................................................................... | 173 |
| 4.3.107 | CDFvarClose ............................................................................................ | 174 |
| 4.3.108 | CDFvarCreate ......................................................................................... | 174 |
| 4.3.109 | CDFvarGet ................................................................................................ | 176 |
| 4.3.110 | CDFvarHyperGet ...................................................................................... | 177 |
| 4.3.111 | CDFvarHyperPut ...................................................................................... | 178 |
| 4.3.112 | CDFvarInquire ......................................................................................... | 180 |
| 4.3.113 | CDFvarNum ................................................................................................ | 181 |
| 4.3.114 | CDFvarPut ................................................................................................ | 182 |
| 4.3.115 | CDFvarRename ......................................................................................... | 183 |
| 4.4 | Attributes/Entries .................................................................................... | 184 |
| 4.4.1 | CDFattrCreate .......................................................................................... | 184 |
| 4.4.2 | CDFattrEntryInquire .................................................................................. | 185 |
| 4.4.3 | CDFattrGet ............................................................................................... | 187 |
5 Interpreting CDF Status Codes .........................................................235

6 EPOCH Utility Routines .................................................................236

6.1 computeEPOCH ............................................................................236
6.2 EPOCHbreakdown ......................................................................237
6.3 toEncodeEPOCH ..........................................................................237
6.4 encodeEPOCH .............................................................................237
6.5 encodeEPOCH1 ............................................................................238
7 TT2000 Utility Routines .................................................. 249

8 CDF Utility Methods ....................................................... 255
8.14 CDFgetStringEncoding ........................................................................................................259
8.15 CDFgetStringFormat ........................................................................................................259
8.16 CDFgetStringMajority .......................................................................................................259
8.17 CDFgetStringSparseRecord ...............................................................................................260

9 CDF Exception Methods ........................................................................................................261

9.1 CDFgetCurrentStatus ...........................................................................................................261
9.2 CDFgetStatusMsg ..............................................................................................................261
Chapter 1

1 Compiling

VB-CDF distribution is packaged in a self-extracting installer. Once the installer is downloaded and run, all distributed files, i.e., APIs, test programs, batch files, help information and the document, will be placed into a directory of choice, and environment variables, PATH and CsharpCDFDir, are automatically set. If an older version already exists in the host machine, the installer will try to remove it before the new one is installed.

To VB, CDF library is unmanaged code distributed in the native DLL. The distributed .DLLs were built from a 32-bit (x86) Windows and they can be run on a 32-bit Windows via the x86-compatible Common Language Runtime (CLR), as well as a 64-bit Windows under WOW64.

1.1 Namespaces

Several classes are created for VB applications that facilitate the calls to the native CDF DLL. The CDF namespace has been set up to include these CDF related classes: CDFConstants, CDFException, CDFAPIs, and CDFUtils. CDFConstants provides commonly used constants that mimic to those defined in the .DLL CDFException provides the exception handling when a failed CDF operation is detected. CDFAPIs provide all (static) public (and private) methods that VB applications can call to interact with the similar, underlining functions provided by the CDF Standard Interface in the .DLL. CDFUtils provides several small utility tools. These classes are distributed in the form of signed assemblies, as .DLLs. To facilitate the access to functions in DLL, each VB application must use the “cdf” namespace in order to call the VB-CDF APIs. The following namespaces should be included by VB applications that call CDF APIs:

```csharp
imports System
imports System.Runtime.InteropServices
imports CDF
```

1.2 Base Classes

CDFAPIs is the main class that provides the VB-CDF APIs. Class CDFAPIs inherits from CDFConstants class, which defines all constants referenced by the CDF. A VB application, if inheriting from the CDFAPIs class, can call all
CDFAPIs methods and refer CDFConstants’ constants directly, without specifying their class names. CDFException class inherits from VB’s Exception class and CDFUtils class inherits from CDFConstants class as well.

### 1.3 Compiling with Compiler Options

If a test application, e.g., TestCDF.vb, resides in the same directory as all distributed .dll files, the following command can be used to create an executable

```
vbc /platform:x86 /r:CDFAPIs.dll,CDFException.dll,
CDFConstants.dll,CDFUtils.dll TestCDF.vb
```

`vbc.exe`, the VB compiler, can be called automatically from an IDE such as Visual Studio .NET, or run from the command line if the PATH environment variable is set properly. `vbc.exe` can be found in the `Windows’s .NET Framework` directory, `<windows>\Microsoft.NET\Framework\v#.#` (v#.# as v3.5 or in the latest release version).

 `/platform:x86` option is required for the Windows running 64-bit OS as VB-CDF is built on an x86 (32-bit) platform.

When the VB-CDF package is installed, the PATH environment variable is automatically modified to include the installation directory so the native CDF .DLL, `dllcdfcsharp.dll`, becomes available when a VB application calls CDF functions. Once the executable, TestCDF.exe, is created, it can be run from any directory.

If the VB applications that call CDF APIs reside in the directories other than the VB-CDF installation directory, the following compilation command can be used to create an executable (.exe):

```
vbc /platform:x86
/lib:%CsharpCDFDir%
/r:cdffapis.dll,cdfconstants.dll,cdfexception.dll,cdfutils.dll
TestCDF.vb
```

where environment variable `CsharpCDFDir`, the installation directory for VB-CDF package, is set when the installer is run.

When the executable is run, an exception of “FileNotFoundException” will be encountered as CDFAPIs could not be loaded. It’s because the distributed CDF assemblies are considered `private` in the .NET environment. The .NET Framework’s runtime, Common Language Runtime (CLR), will not be able to locate the files if the application resides in a different directory from the called assemblies. To make these assemblies `global` so CLR can locate, they need to be placed in the `Global Assembly Cache (GAC)` repository. Use the following steps to do so:

```
gacutil /i CDFConstants.dll
gacutil /i CDFException.dll
gacutil /i CDFAPIs.dll
gacutil /i CDFUtils.dll
```

`gacutil.exe` (Global Assembly Cache utility) is a *Microsoft Software Development Kits (SDKs)* utility that can insert, list and remove the assemblies to and from GAC. Gacutil.exe usually can be found at `<Program Files>\Microsoft SDKs\Windows\v#.##\bin` (v#.## as v6.0A or in the latest release version). Use “gacutil /u” to remove assemblies of older versions form GAC.

`ildasm.exe` is another SDKs utility that can be used to browse the assemblies for information as versions, keys, etc.
1.4 Sample programs

A couple of sample programs are included for distribution. Qst2vb.vb and Qst2vb2.vb, the quick test programs for VB. Qst2vb.vb uses the VB value type for data read and write to a CDF file. Qst2vb2.vb passes in the base class objects for arguments while reading the data from a CDF. Qts2cEpoch.vb, Qst2cEpoch16.vb and Qst2cTT2000.vb are three sample programs that show how EPOCH-related functions are used. A batch file, tocompileVB.bat, is distributed along with the sample programs. Execute it from a Command Prompt window to compile the programs into executables (.exe). Run totestvb.bat to test the executables to make sure they all work fine.
Chapter 2

2 Programming Interface

2.1 Item Referencing

The following sections describe various aspects of the programming interface for VB applications.

For VB applications, all item numbers are referenced starting at zero (0). These include variable, attribute, and attribute entry numbers, record numbers, dimensions, and dimension indices. Note that both rVariables and zVariables are numbered starting at zero (0).

2.2 Compatible Types

As VB and CDF .DLL may have different sizes of the same data types, e.g. long, the size compatibility must be enforced when passing the data between the two. On 32-bit Windows, 4-byte long has been used all over in the CDF .DLL. However, long in VB is defined as 8-byte. So, to make the size compatible, 4-byte integer is used, instead, in VB for each long type variable in the .DLL. For CDF data of type CDF_CHAR, or CDF_UCHAR, it is represented by a string in VB. They are not size compatible, so conversion, performed in the APIs, is needed between a character array in .DLL and string in VB.

The VB-CDF operations normally involve two variables: the operation status, status, and the CDF identifier, id:

<table>
<thead>
<tr>
<th>status</th>
<th>All VB-CDF functions, except CDFvarNum, CDFgetVarNum, CDFattrNum and CDFgetAttrNum, return an operation status. This status is defined as an integer in .DLL and VB. The CDFerror method can be used to inquire the meaning of any status code. Appendix A lists the possible status codes along with their explanations. Chapter 5 describes how to interpret status codes.</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>An identifier (or handle) for a CDF that must be used when referring to a CDF. This identifier has a type of long in VB. A new identifier is established whenever a CDF is created or opened, establishing a connection to that CDF on disk. This long value is used in all subsequent operations on a particular CDF. The value must not be altered by an application.</td>
</tr>
</tbody>
</table>
2.3 CDFConstants

CDF defines a set of constants that are used all over the .DLL. These constants are mimicked in CDFConstants class with compatible data types.

2.4 CDF status

These constants are of same type as the operation status, mentioned in 2.2.

CDF_OK                     A status code indicating the normal completion of a CDF function.
CDF_WARN                   Threshold constant for testing severity of non-normal CDF status codes.

Status less than CDF_OK normally indicate an error. For most cases, an exception will be thrown.

2.5 CDF Formats

SINGLE_FILE   The CDF consists of only one file. This is the default file format.
MULTI_FILE    The CDF consists of one header file for control and attribute data and one additional file for each variable in the CDF.

2.6 CDF Data Types

One of the following constants must be used when specifying a CDF data type for an attribute entry or variable.

CDF_BYTE      1-byte, signed integer.
CDF_CHAR      1-byte, signed character.
CDF_INT1      1-byte, signed integer.
CDF_UCHAR     1-byte, unsigned character.
CDF_UINT1     1-byte, unsigned integer.
CDF_INT2      2-byte, signed integer.
CDF_UINT2     2-byte, unsigned integer.
CDF_INT4      4-byte, signed integer.
CDF_UINT4 4-byte, unsigned integer.
CDF_INT8 8-byte, signed integer.
CDF_REAL4 4-byte, floating point.
CDF_FLOAT 4-byte, floating point.
CDF_REAL8 8-byte, floating point.
CDF_DOUBLE 8-byte, floating point.
CDF_EPOCH 8-byte, floating point.
CDF_EPOCH16 two 8-byte, floating point.
CDF_TIME_TT2000 8-byte, signed integer.

The following table depicts the equivalent data type between the CDF and VB:

<table>
<thead>
<tr>
<th>CDF Data Type</th>
<th>VB Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDF_BYTE</td>
<td>sbyte</td>
</tr>
<tr>
<td>CDF_INT1</td>
<td>sbyte</td>
</tr>
<tr>
<td>CDF_UINT1</td>
<td>byte</td>
</tr>
<tr>
<td>CDF_INT2</td>
<td>short</td>
</tr>
<tr>
<td>CDF_UINT2</td>
<td>ushort</td>
</tr>
<tr>
<td>CDF_INT4</td>
<td>integer</td>
</tr>
<tr>
<td>CDF_UINT4</td>
<td>uinteger</td>
</tr>
<tr>
<td>CDF_INT8</td>
<td>long</td>
</tr>
<tr>
<td>CDF_REAL4</td>
<td>single</td>
</tr>
<tr>
<td>CDF_FLOAT</td>
<td>single</td>
</tr>
<tr>
<td>CDF_REAL8</td>
<td>double</td>
</tr>
<tr>
<td>CDF_DOUBLE</td>
<td>double</td>
</tr>
<tr>
<td>CDF_EPOCH</td>
<td>double</td>
</tr>
<tr>
<td>CDF_EPOCH16</td>
<td>double(2)</td>
</tr>
<tr>
<td>CDF_TIME_TT2000l</td>
<td>long</td>
</tr>
<tr>
<td>CDF_CHAR</td>
<td>string</td>
</tr>
<tr>
<td>CDF_UCHAR</td>
<td>string</td>
</tr>
</tbody>
</table>

CDF_CHAR and CDF_UCHAR are considered character data types. These are significant because only variables of these data types may have more than one element per value (representing the length of the string, where each element is a character).

**NOTE:** Keep in mind that an long is 8 bytes and that an integer is 4 bytes. Use integer for CDF data types CDF_INT4 and CDF_UINT4, rather than long. Use long for CDF_INT8 and CDF_TIME_TT2000 data types.

---

1 CDF_EPOCH16 has two doubles, which corresponds to an array as double() in VB.
2.7 Data Encodings

A CDF's data encoding affects how its attribute entry and variable data values are stored (on disk). Attribute entry and variable values passed into the CDF library (to be written to a CDF) should always be in the host machine's native encoding. Attribute entry and variable values read from a CDF by the CDF library and passed out to an application will be in the currently selected decoding for that CDF (see the Concepts chapter in the CDF User's Guide).

<table>
<thead>
<tr>
<th>Encoding Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOST_ENCODING</td>
<td>Indicates host machine data representation (native). This is the default encoding, and it will provide the greatest performance when reading/writing on a machine of the same type.</td>
</tr>
<tr>
<td>NETWORK_ENCODING</td>
<td>Indicates network transportable data representation (XDR).</td>
</tr>
<tr>
<td>VAX_ENCODING</td>
<td>Indicates VAX data representation. Double-precision floating-point values are encoded in Digital's D_FLOAT representation.</td>
</tr>
<tr>
<td>ALPHAVMSd_ENCODING</td>
<td>Indicates DEC Alpha running OpenVMS data representation. Double-precision floating-point values are encoded in Digital's D_FLOAT representation.</td>
</tr>
<tr>
<td>ALPHAVMSg_ENCODING</td>
<td>Indicates DEC Alpha running OpenVMS data representation. Double-precision floating-point values are encoded in Digital's G_FLOAT representation.</td>
</tr>
<tr>
<td>ALPHAVMSi_ENCODING</td>
<td>Indicates DEC Alpha running OpenVMS data representation. Double-precision floating-point values are encoded in IEEE representation.</td>
</tr>
<tr>
<td>ALPHAOSF1_ENCODING</td>
<td>Indicates DEC Alpha running OSF/1 data representation.</td>
</tr>
<tr>
<td>SUN_ENCODING</td>
<td>Indicates SUN data representation.</td>
</tr>
<tr>
<td>SGI_ENCODING</td>
<td>Indicates Silicon Graphics Iris and Power Series data representation.</td>
</tr>
<tr>
<td>DECSTATION_ENCODING</td>
<td>Indicates DECstation data representation.</td>
</tr>
<tr>
<td>IBMRS_ENCODING</td>
<td>Indicates IBMRS data representation (IBM RS6000 series).</td>
</tr>
<tr>
<td>HP_ENCODING</td>
<td>Indicates HP data representation (HP 9000 series).</td>
</tr>
<tr>
<td>IBMPC_ENCODING</td>
<td>Indicates PC data representation.</td>
</tr>
<tr>
<td>NeXT_ENCODING</td>
<td>Indicates NeXT data representation.</td>
</tr>
<tr>
<td>MAC_ENCODING</td>
<td>Indicates Macintosh data representation.</td>
</tr>
<tr>
<td>ARM_LITTLE_ENCODING</td>
<td>Indicates ARM architecture running little-endian data representation.</td>
</tr>
<tr>
<td>ARM_BIG_ENCODING</td>
<td>Indicates ARM architecture running big-endian data representation.</td>
</tr>
<tr>
<td>IA64VMSi_ENCODING</td>
<td>Indicates Itanium 64 running OpenVMS data representation. Double-precision floating-point values are encoded in IEEE representation.</td>
</tr>
</tbody>
</table>
IA64VMSd_ENCODING Indicates Itanium 64 running OpenVMS data representation. Double-precision floating-point values are encoded in Digital’s D_FLOAT representation.

IA64VMSg_ENCODING Indicates Itanium 64 running OpenVMS data representation. Double-precision floating-point values are encoded in Digital’s G_FLOAT representation.

When creating a CDF (via CDFcreate) or respecifying a CDF's encoding (via CDFsetEncoding), you may specify any of the encodings listed above. Specifying the host machine's encoding explicitly has the same effect as specifying HOST_ENCODING.

When inquiring the encoding of a CDF, either NETWORK_ENCODING or a specific machine encoding will be returned. (HOST_ENCODING is never returned.)

2.8 Data Decodings

A CDF’s decoding affects how its attribute entry and variable data values are passed out to a calling application. The decoding for a CDF may be selected and reselected any number of times while the CDF is open. Selecting a decoding does not affect how the values are stored in the CDF file(s) - only how the values are decoded by the CDF library. Any decoding may be used with any of the supported encodings. The Concepts chapter in the CDF User's Guide describes a CDF’s decoding in more detail.

HOST_DECODING Indicates host machine data representation (native). This is the default decoding.

NETWORK_DECODING Indicates network transportable data representation (XDR).

VAX_DECODING Indicates VAX data representation. Double-precision floating-point values will be in Digital’s D_FLOAT representation.

ALPHAVMSd_DECODING Indicates DEC Alpha running OpenVMS data representation. Double-precision floating-point values will be in Digital’s D_FLOAT representation.

ALPHAVMSg_DECODING Indicates DEC Alpha running OpenVMS data representation. Double-precision floating-point values will be in Digital’s G_FLOAT representation.

ALPHAVMSi_DECODING Indicates DEC Alpha running OpenVMS data representation. Double-precision floating-point values will be in IEEE representation.

ALPHAOSF1_DECODING Indicates DEC Alpha running OSF/1 data representation.

SUN_DECODING Indicates SUN data representation.

SGi_DECODING Indicates Silicon Graphics Iris and Power Series data representation.

DECSTATION_DECODING Indicates DECstation data representation.

IBMRS_DECODING Indicates IBMRS data representation (IBM RS6000 series).
HP_DECODING Indicates HP data representation (HP 9000 series).

IBMPC_DECODING Indicates PC data representation.

NeXT_DECODING Indicates NeXT data representation.

MAC_DECODING Indicates Macintosh data representation.

ARM_LITTLE_DECODING Indicates ARM architecture running little-endian data representation.

ARM_BIG_DECODING Indicates ARM architecture running big-endian data representation.

IA64VMSi_DECODING Indicates Itanium 64 running OpenVMS data representation. Double-precision floating-point values are encoded in IEEE representation.

IA64VMSd_DECODING Indicates Itanium 64 running OpenVMS data representation. Double-precision floating-point values are encoded in Digital’s D_FLOAT representation.

IA64VMSg_DECODING Indicates Itanium 64 running OpenVMS data representation. Double-precision floating-point values are encoded in Digital’s G_FLOAT representation.

The default decoding is HOST_DECODING. The other decodings may be selected via the CDFsetDecoding method. The Concepts chapter in the CDF User's Guide describes those situations in which a decoding other than HOST_DECODING may be desired.

2.9 Variable Majorities

A CDF's variable majority determines the order in which variable values (within the variable arrays) are stored in the CDF file(s). The majority is the same for rVariables and zVariables.

ROW_MAJOR C-like array ordering for variable storage. The first dimension in each variable array varies the slowest. This is the default.

COLUMN_MAJOR Fortran-like array ordering for variable storage. The first dimension in each variable array varies the fastest.

Knowing the majority of a CDF's variables is necessary when performing hyper reads and writes. During a hyper read the CDF library will place the variable data values into the memory buffer in the same majority as that of the variables. The buffer must then be processed according to that majority. Likewise, during a hyper write, the CDF library will expect to find the variable data values in the memory buffer in the same majority as that of the variables.

The majority must also be considered when performing sequential reads and writes. When sequentially reading a variable, the values passed out by the CDF library will be ordered according to the majority. When sequentially writing a variable, the values passed into the CDF library are assumed (by the CDF library) to be ordered according to the majority.

As with hyper reads and writes, the majority of a CDF's variables affect multiple variable reads and writes. When performing a multiple variable write, the full-physical records in the buffer passed to the CDF library must have the
CDF's variable majority. Likewise, the full-physical records placed in the buffer by the CDF library during a multiple variable read will be in the CDF's variable majority.

For C applications the compiler-defined majority for arrays is row major. The first dimension of multi-dimensional arrays varies the slowest in memory.

2.10 Record/Dimension Variances

Record and dimension variances affect how variable data values are physically stored.

VARY True record or dimension variance.
NOVARY False record or dimension variance.

If a variable has a record variance of VARY, then each record for that variable is physically stored. If the record variance is NOVARY, then only one record is physically stored. (All of the other records are virtual and contain the same values.)

If a variable has a dimension variance of VARY, then each value/subarray along that dimension is physically stored. If the dimension variance is NOVARY, then only one value/subarray along that dimension is physically stored. (All other values/subarrays along that dimension are virtual and contain the same values.)

2.11 Compressions

The following types of compression for CDFs and variables are supported. For each, the required parameters are also listed. The Concepts chapter in the CDF User's Guide describes how to select the best compression type/parameters for a particular data set. Among the available types, GZIP provides the best result.

NO_COMPRESSION No compression.
RL_COMPRESSION Run-length encoding compression. There is one parameter.
1. The style of run-length encoding. Currently, only the run-length encoding of zeros is supported. This parameter must be set to RLE_OF_ZEROS.
HUFF_COMPRESSION Huffman compression. There is one parameter.
1. The style of Huffman encoding. Currently, only optimal encoding trees are supported. An optimal encoding tree is determined for each block of bytes being compressed. This parameter must be set to OPTIMAL_ENCODING_TREES.
AHUFF_COMPRESSION Adaptive Huffman compression. There is one parameter.
1. The style of adaptive Huffman encoding. Currently, only optimal encoding trees are supported. An optimal encoding tree is determined...
for each block of bytes being compressed. This parameter must be set to OPTIMAL_ENCODING_TREES.

GZIP_COMPRESSION  Gnu's "zip" compression. There is one parameter.

1. The level of compression. This may range from 1 to 9. 1 provides the least compression and requires less execution time. 9 provide the most compression but require the most execution time. Values in-between provide varying compromises of these two extremes. 6 normally provides a better balance between compression and execution.

2.12  Sparseness

2.12.1  Sparse Records

The following types of sparse records for variables are supported.

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO_SPARSERECORDS</td>
<td>No sparse records.</td>
</tr>
<tr>
<td>PAD_SPARSERECORDS</td>
<td>Sparse records - the variable's pad value is used when reading values from a missing record.</td>
</tr>
<tr>
<td>PREV_SPARSERECORDS</td>
<td>Sparse records - values from the previous existing record are used when reading values from a missing record. If there is no previous existing record the variable's pad value is used.</td>
</tr>
</tbody>
</table>

2.12.2  Sparse Arrays

The following types of sparse arrays for variables are supported.\(^3\)

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO_SPARSEARRAYS</td>
<td>No sparse arrays.</td>
</tr>
</tbody>
</table>

Note: sparse array is not supported and will not be implemented.

2.13  Attribute Scopes

Attribute scopes are simply a way to explicitly declare the intended use of an attribute by user applications (and the CDF toolkit).

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLOBAL_SCOPE</td>
<td>Indicates that an attribute's scope is global (applies to the CDF as a whole).</td>
</tr>
</tbody>
</table>

\(^2\) Disabled for PC running 16-bit DOS/Windows 3.x.

\(^3\) Obviously, sparse arrays are not yet supported.
VARIABLE_SCOPE Indicates that an attribute's scope is by variable. (Each rEntry or zEntry corresponds to an rVariable or zVariable, respectively.)

### 2.14 Read-Only Modes

Once a CDF has been opened, it may be placed into a read-only mode to prevent accidental modification (such as when the CDF is simply being browsed). Read-only mode is selected via CDFsetReadOnlyMode method. When read-only mode is set, all metadata is read into memory for future reference. This improves overall metadata access performance but is extra overhead if metadata is not needed. Note that if the CDF is modified while not in read-only mode, subsequently setting read-only mode in the same session will not prevent future modifications to the CDF.

- **READONLYon** Turns on read-only mode.
- **READONLYoff** Turns off read-only mode.

### 2.15 zModes

Once a CDF has been opened, it may be placed into one of two variations of zMode. zMode is fully explained in the Concepts chapter in the CDF User's Guide. A zMode is selected via CDFsetzMode method.

- **zMODEoff** Turns off zMode.
- **zMODEon1** Turns on zMode/1.
- **zMODEon2** Turns on zMode/2.

### 2.16 -0.0 to 0.0 Modes

Once a CDF has been opened, the CDF library may be told to convert -0.0 to 0.0 when read from or written to that CDF. This mode is selected via CDFsetNegtoPosfp0Mode method.

- **NEGtoPOSfp0on** Convert -0.0 to 0.0 when read from or written to a CDF.
- **NEGtoPOSfp0off** Do not convert -0.0 to 0.0 when read from or written to a CDF.

### 2.17 Operational Limits

These are limits within the CDF library. If you reach one of these limits, please contact CDF User Support.

- **CDF_MAX_DIMS** Maximum number of dimensions for the rVariables or a zVariable.
CDF_MAX_PARMS 
Maximum number of compression or sparseness parameters.

The CDF library imposes no limit on the number of variables, attributes, or attribute entries that a CDF may have. on the PC, however, the number of rVariables and zVariables will be limited to 100 of each in a multi-file CDF because of the 8.3 naming convention imposed by MS-DOS.

2.18 Limits of Names and Other Character Strings

CDF_PATHNAME_LEN 
Maximum length of a CDF file name. A CDF file name may contain disk and directory specifications that conform to the conventions of the operating systems being used (including logical names on OpenVMS systems and environment variables on UNIX systems).

CDF_VAR_NAME_LEN256 
Maximum length of a variable name.

CDF_ATTR_NAME_LEN256 
Maximum length of an attribute name.

CDF_COPYRIGHT_LEN 
Maximum length of the CDF Copyright text.

CDF_STATUSTEXT_LEN 
Maximum length of the explanation text for a status code.

2.19 Backward File Compatibility with CDF 2.7

By default, a CDF file created by CDF V3.0 or a later release is not readable by any of the CDF releases before CDF V3.0 (e.g. CDF 2.7.x, 2.6.x, 2.5.x, etc.). The file incompatibility is due to the 64-bit file offset used in CDF 3.0 and later releases (to allow for files greater than 2G bytes). Note that before CDF 3.0, 32-bit file offset was used.

There are two ways to create a file that’s backward compatible with CDF 2.7 and 2.6, but not 2.5. A method, CDFsetFileBackward, can be called to control the backward compatibility from an application before a CDF file is created (i.e. CDFcreateCDF). This method takes an argument to control the backward file compatibility. Passing a flag value of BACKWARDFILEOn, defined in CDFConstants, to the method will cause new files being created to be backward compatible. The created files are of version V2.7.2, not V3.*. This option is useful for those who wish to create and share files with colleagues who still use a CDF V2.7/V2.6 library. If this option is specified, the maximum file size is limited to 2G bytes. Passing a flag value of BACKWARDFILEOff will use the default file creation mode and newly created files will not be backward compatible with older libraries. The created files are of version 3.* and thus their file sizes can be greater than 2G bytes. Not calling this method has the same effect of calling the method with an argument value of BACKWARDFILEOff.

The following example creates two CDF files: “MY_TEST1.cdf” is a V3.* file while “MY_TEST2.cdf” a V2.7 file.

```plaintext
try
    status = CDFcreateCDF("MY_TEST1", id1)
```

```plaintext
    Dim status as integer
    Dim id1 as long, id2 as long
    ‘ CDF identifier.
    ‘ Returned status code.
```
CDFsetFileBackward(BACKWARDFILEon)
status = CDFCreateCDF("MY_TEST2", id2)
catch ex as Exception
    end try
Another method is through an environment variable and no method call is needed (and thus no code change involved in any existing applications). The environment variable, CDF_FILEBACKWARD on Windows, is used to control the CDF file backward compatibility. If its value is set to “TRUE”, all new CDF files are backward compatible with CDF V2.7 and 2.6. This applies to any applications or CDF tools dealing with creation of new CDFs. If this environment variable is not set, or its value is set to anything other than “TRUE”, any files created will be of the CDF 3.* version and these files are not backward compatible with the CDF 2.7.2 or earlier versions.

Normally, only one method should be used to control the backward file compatibility. If both methods are used, the method call through CDFsetFileBackward will take the precedence over the environment variable.

You can use the CDFgetFileBackward method to check the current value of the backward-file-compatibility flag. It returns 1 if the flag is set (i.e. create files compatible with V2.7 and 2.6) or 0 otherwise.

dim flag as integer ' Returned status code.
flag = CDFgetFileBackward()

2.20 Checksum

To ensure the data integrity while transferring CDF files from/to different platforms at different locations, the checksum feature was added in CDF V3.2 as an option for the single-file format CDF files (not for the multi-file format). By default, the checksum feature is not turned on for new files. Once the checksum bit is turned on for a particular file, the data integrity check of the file is performed every time it is open and a new checksum is computed and stored when it is closed. This overhead (performance hit) may be noticeable for large files. Therefore, it is strongly encouraged to turn off the checksum bit once the file integrity is confirmed or verified.

If the checksum bit is turned on, a 16-byte signature message (a.k.a. message digest) is computed from the entire file and appended to the end of the file when the file is closed (after any create/write/update activities). Every time such file is open, other than the normal steps for opening a CDF file, this signature, serving as the authentic checksum, is used for file integrity check by comparing it to the re-computed checksum from the current file. If the checksums match, the file’s data integrity is verified. Otherwise, an error message is issued. Currently, the valid checksum modes are: NO_CHECKSUM and MD5_CHECKSUM, both defined in CDFConstants class. With MD5_CHECKSUM, the MD5 algorithm is used for the checksum computation. The checksum operation can be applied to CDF files that were created with V2.7 or later.

There are several ways to add or remove the checksum bit. One way is to use the method call with a proper checksum mode. Another way is through the environment variable. Finally, CDFedit and CDFconvert (CDF tools included as part of the standard CDF distribution package) can be used for adding or removing the checksum bit. Through the Interface call, you can set the checksum mode for both new or existing CDF files while the environment variable method only allows to set the checksum mode for new files.
The environment variable **CDF_CHECKSUM** on Windows is used to control the checksum option. If its value is set to “**MD5**”, all new CDF files will have their checksum bit set with a signature message produced by the MD5 algorithm. If the environment variable is not set or its value is set to anything else, no checksum is set for the new files.

The following example set a new CDF file with the MD5 checksum and set another existing file’s checksum to none.

```vbnet
Dim id1 as long, id2 as long
Dim status as integer
Dim checksum as integer

status = CDFCreateCDF("MY_TEST1”, id1)
status = CDFsetChecksum (id1, MD5_CHECKSUM)
status = CDFclose(id1)

status = CDFopen("MY_TEST2", id2)
status = CDFsetChecksum (id2, NO_CHECKSUM)
status = CDFclose(id2)
```

### 2.21 Data Validation

To ensure the data integrity of CDF files and secure operation of CDF-based applications, a data validation feature has been added to the CDF opening logic. This process, as the default, performs sanity checks on the data fields in the CDF’s internal data structures to make sure that the values are within valid ranges and consistent with the defined values/types/entries. It also ensures that the variable and attribute associations within the file are valid. Any compromised CDF files, if not validated properly, could cause applications to function unexpectedly, e.g., segmentation fault due to a buffer overflow. The main purpose of this feature is to safeguard the CDF operations, catch any bad data in the file and end the application gracefully if any bad data is identified. Using this feature, in most cases, will slow down the file opening process especially for large or very fragmented files. Therefore, it is recommended that this feature be turned off once a file’s integrity is confirmed or verified. Or, the file in question may need a file conversion, which will consolidate the internal data structures and eliminate the fragmentations. Check the **cdfconvert** tool program in the CDF User’s Guide for further information.  

This validation feature is controlled by setting/unsetting the environment variable **CDF_VALIDATE** on Windows is not set or set to “**yes**”, all CDF files are subjected to the data validation process. If the environment variable is set to “**no**”, then no validation is performed. The environment variable can be set at logon or through the command line, which goes into effect during a terminal session, or within an application, which is good only while the application is running. Setting the environment variable, using C method **CDFsetValidate**, at application level will overwrite the setup from the command line. The validation is set to be on when **VALIDATEFILEon** is passed in as an argument. **VALIDATEFILEoff** will turn off the validation. The function, **CDFgetValidate**, will return the validation mode, 1 (one) means data being validated, 0 (zero) otherwise. If the environment variable is not set, the default is to validate the data.

---

4 The data validation during the open process will not check the variable data. It is still possible that data could be corrupted, especially compression is involved. To fully validate a CDF file, use cfdump tool with “-detect” switch.
CDF file upon opening.

The following example sets the data validation on when the CDF file, “TEST”, is open.

```vbscript
dim id as long
Dim status as integer

CDFsetValidate (VALIDATEFILEon)
status = CDFopen(“TEST”, id)
```

The following example turns off the data validation when the CDF file, “TEST” is open.

```vbscript
dim id as long
Dim status as integer

CDFsetValidate (VALIDATEFILEoff)
status = CDFopen(“TEST”, id)
```

### 2.22 8-Byte Integer

Both data types of CDF_INT8 and CDF_TIME_TT2000 use 8-bytes signed integer. VB’s “long” type is the one that matches to these two types.

### 2.23 Leap Seconds

CDF’s CDF_TIME_TT2000 is the epoch value in nanoseconds since J2000 (2000-01-01T12:00:00.000000000) with leap seconds included. The CDF uses an external or internal table for computing the leap seconds. The external table, if present and properly pointed to by a predefined environment variable, will be used over the internal one. When the VB package is installed, the external table and environment variables are set so it can be used. If the external table is deleted or no longer pointed by the environment variable, the internal, hard-coded table in the library is used. When a new leap second is added, if the external table is updated accordingly, then the software does not need to be upgraded. Refer to CDF User’s Guide for leap seconds.

A tool program, CDFleapsecondsInfo distributed with the CDFpackage, will show how the table is referred and when the last leap second was added. Optionally, it can dump the table contents.
Chapter 3

3 Understanding the Application Interface

This chapter provides some basic information about the VB’s Application Interfaces (APIs) to CDF, and the native CDF .DLL. The following chapter will describe each API in detail.

3.1 Arguments Passing

Each CDF API has a sequence of parameters, which define the set of arguments that must be provided for that method in VB applications. Being a strongly typed language, VB’s APIs to CDF follow the same rules for the parameters. Arguments for APIs that perform CDF data get, put or inquire operations are required to have the signatures of the defined VB value/string type or basic Object classes.

The input parameters in APIs for the CDF identifier, variable number, attribute number, entry number, record number, record counts and record indices, etc, are always of fixed types. They must be a scalar of type long for CDF identifier, integer for variable/attribute/entry number and record number/count, or an array of integers, integer(), for variable dimensional sizes/variances and record data indices, counts and intervals. The output parameters must be in either of the defined type or the VB base Object class. For example, for a returned data of type integer, the passing argument in the calling application can be either a defined integer variable, or a variable of object class. Compilation error will occur if any one of the such arguments from the applications does not match to that defined in the API.

A CDF identifier, when a CDF is open or created, is presented as a long variable, even in the underlying C# and CDF native library it is a pointer.

For example, CDFsetEncoding and CDFgetEncoding are used to set and get the data encoding of a CDF. Both APIs take two parameters, the CDF identifier, always a long, and the encoding, an integer. CDFsetEncoding take both parameters from applications for input, while CDFgetEncoding has the CDF identifier as input and the encoding for output. The following code shows how these methods can be used.

To set a CDF’s encoding,

dim status as integer
dim id as long
dim encoding as integer

...
encoding = IBMPC_ENCODING  
status = CDFsetEncoding(id, encoding)

The CDF identifier, id, is set when a CDF is open or created. The encoding is set to PC encoding, defined in CDFConstants class.

Similarly, to get the CDF’s encoding:

status = CDFgetEncoding(id, encoding)

 APIs that read or write CDF data, either variable’s data (and their pad value) or metadata, are flexible when dealing with data of different pre-defined CDF types, e.g., CDF_INT1, CDF_UINT1, CDF_FLOAT, CDF_CHAR, CDF_EPOCH, etc. To pass the data value(s) to the APIs, one of the following forms can be used, depending on the data type: **VB numeric type or string in a scalar or array or simply the VB base object class.** String or an array of strings involves data of CDF_CHAR or CDF_UCHAR type. As VB’s character/string has a different characteristic from the ASCII-based code in the CDF native DLL library, some manipulations are performed by the APIs when dealing with such data. VB objects can be used, as a general form for all data value(s), when reading/writing data from CDF. The called APIs will handle the passed object and map it to its corresponding CDF data type. **Type casting** the objects returned by the APIs may be needed.

For example, methods: CDFputzVarData and CDFgetzVarData are used to write and read a single data value for a zVariable in a CDF. Both take five parameters. The first four, the CDF identifier, variable number, record number and indices, are for input and of fixed types of: long, integer, integer and an array of integers (integer()), respectively. The last parameter is for data value, as an input for CDFputzVarData or an output for CDFgetzVarData. To call CDFputzVarData, the data value has to be defined to match to variable’s underlying data type and given a value. It is passed in as is. To retrieve the data by CDFgetzVarData, just specifies the variable with a proper data type and pass in to the API.

The following samples show how these arguments are set up to write a data value to record 1, indices (1,1) for zVariable, “zVar1”, a 2-dimentional of CDF_INT2.

```vba
dim status as integer  
dim id as long  
dim varNum as integer  
dim recNum as integer = 1  
dim indices() as integer = {1,1}  
dim value as short = 100  
...
varNum = CDFvarNum (id, “zVar1”)  
status = CDFputzVarData(id, varNum, recNum, indices, value)
```

To read the data value the same variable at the same record and indices:

```vba
dim value as short  
...
status = CDFgetzVarData(id, varNum, recNum, indices, value)
```

Similarly, value can be defined as a VB base object:

```vba
Dim valueo as object  
status = CDFgetzVarData(id, varNum, recNum, indices, valueo)
```

Either use such statement:
Dim value as short = valueo

Or, use a proper type casting method, such as CType or DirectCast for a scalar, to make it a value type after the object is returned. For object of an array, just assign it to a properly type-defined, dimensional variable.

dim value as short = CType(valueo, short)

APIs that handle multiple data values reads and writes, e.g., CDFputzVarRecordData and CDFgetzVarRecordData for writing and reading a full data record an zVariable, are similar. They both take four parameters: the first three, as input, are the CDF identifier, variable number, record number of the fixed types of long, integer and integer, respectively, and the last one is the data values, input for CDFputzVarRecordData or output for CDFgetzVarRecordData. The data values have to be defined (and assigned for input), according to the variable’s underlying data type, and passed in as is.

The following samples show how the arguments are set in CDFputzVarRecordData to write the full record 1 for zVariable, “zVar1”, a 2-dim (2,3) of type short. The first one passes the data value object as is, while the second one uses a pointer to the data values.

dim status as integer
dim id as long
dim varNum as integer
dim recNum as integer = 1
Dim values(,) as short = { {1,2,3},{11,12,13}}
…
varNum = CDFvarNum (id, “zVar1”)
status = CDFputzVarRecordData(id, varNum, recNum, values)

For CDFgetzVarRecordData to read back the same variable’s record data, one can use the same arguments as CDFputzVarRecordData.

dim id as long
dim varNum as integer
dim recNum as integer = 1
Dim values (,) as short
…
varNum = CDFvarNum (id, “zVar1”)
status = CDFgetzVarRecordData(id, varNum, recNum, values)

Console.WriteLine("{0},{1},{2}" + Environment.NewLine + "{3},{4},{5}",values(0.0),values(0.1),values(0.2),_ values(1.0),values(1.1), values(1.2))

Alternatively, use a base object for the output:

Dim valueso as object
…
status = CDFgetzVarRecordData(id, varNum, recNum, valueso)

Dim values(,) as short = valueo

Console.WriteLine("{0},{1},{2}" + Environment.NewLine + "{3},{4},{5}",values(0.0),values(0.1),values(0.2),_ values(1.0),values(1.1), values(1.2))
3.2 Multi-Dimensional Arrays

For data involved multidimensional arrays, CDF’s native .DLL data structure is equivalent to the **rectangular array** in VB. Multidimensional arrays of jagged type are not supported by APIs. An extra dimension is added to the retrieved data if the operations involve multiple records. For example, to read two full records from a variable of two-dimensions, 3-by-4 by the hyper get method, the returned will be a three-dimensional, 2-by-3-by-4, object. Conversely, if the hyper read skips certain dimension(s) from an operation, the returned object’s dimensionality will be reduced accordingly. For example, to read a row or column from a variable’s two-dimensional record, the returned will be a single array of either column or row count.

3.3 Data Type Equivalent

The following list shows the data types used by CDF and their corresponding types in VB:

- CDF_INT1  sbyte
- CDF_INT2  short
- CDF_INT4  int
- CDF_INT8  long
- CDF_UINT1 byte
- CDF_UINT2 ushort
- CDF_UINT4 uint
- CDF_BYTE  sbyte
- CDF_REAL  single
- CDF_FLOAT  single
- CDF_DOUBLE double
- CDF_REAL8 double
- CDF_EPOCH double
- CDF_EPOCH16 double(2)
- CDF_TIME_TT2000 long
- CDF_CHAR  string (with manipulation)
- CDF_UCHAR string (with manipulation)

3.4 Fixed Statement

Fixed statement is required to pin VB managed data objects, mainly arrays of numeric data, so that pointers of the objects can be safely used and passed to the CDF APIs. By doing so, the objects’ addresses in the heap won’t be moved around by the garbage collector during the operation.

For example, CDFHyperGetVarData method can be called to retrieve a number of data values for a zVariable. For instance, the following application code can be used to read the first four (4) records from a zVariable of 2-dim (2,3) of type CDF_INT4. The declared data buffer, a 3-dimensional of int, is blocked in the fixed statement when the call is made.
dim id as long
dim status as integer
dim varNum as integer
dim recNum as integer = 0, recCount as integer = 4, recInterval as integer = 1
dim indices() as integer = {0, 0}
dim counts() as integer = {2, 3}
dim intervals() as integer = {1, 1}
dim data(4,2,3) as integer
   ' Dimension: record number, row, column
...
...
status = CDFhyperGetzVarData (id, varNum, recNum, recCount, recInterval, indices, counts, intervals, data)
...

3.5 Exception Handling

Except a few APIs, each call to a CDF method will return an operation status. If the status is abnormal, less than CDF_OK, an exception might be thrown. It is recommended that the code for the CDF-based application be surrounded by a try-catch block so an exception can be caught and handled. The methods to check the existence of a CDF entity, e.g., entry, attribute, variable, will not throw exception if that entity is not in the CDF. The returned, informational status will reflect so. Once an exception is thrown, the thrown object, if initiated from the CDF APIs, is a CDFException class object. There are a couple of class methods, GetCurrentStatus and GetStatusMsg, which can be used to acquire the status when an exception is thrown and the descriptive information about that exception.

dim id as long
dim status as integer
dim encoding as integer
try
    status = CDFopen("TEST", id)
    ...
    status = CDFgetEncoding(id, encoding)
    ....
    status = CDFclose(id)
catch ex as Exception
    Console.WriteLine("Exception: "+ex.ToString())
    Or,
    dim status1 as integer = ex.GetCurrentStatus()
    Console.WriteLine("Exception: "+ex.GetStatusMsg(status1))
}

3.6 Dimensional Limitations

The VB to CDF APIs follow the same dimensional restriction as in the CDF native DLL: a limit of ten (10) dimensions a CDF variable’s numeric typed data record can have. For string typed data, represented in a CDF file with CDF_CHAR or CDF_UCHAR type, a limit of four (4) dimensions is applied.
Chapter 4

4 Application Interface

This chapter covers all Application Interfaces (APIs) that VB applications can call to interact with CDF. Since C# APIs to CDF had already been developed, they are the base for all .Net Framework applications for CDF. Pointers are used extensively for passing the data, e.g., CDF identifier as void *, between C# applications, C# APIs and CDF native DLL. Such pointer-based functions are hard to handle in VB application. For that, a new set of APIs is added to C# APIs suite to specifically allow VB applications to use C# functions without the use of pointers.

There are two types of variables (rVariable and zVariable) in CDF, and they can happily coexist in a CDF: Every rVariable in a CDF must have the same number of dimensions and dimension sizes while each zVariable can have its own dimensionality. Since all the rVariables in a CDF must have the same dimensions and dimension sizes, there'll be a lot of disk space wasted if a few variables need big arrays and many variables need small arrays. Since zVariable is more efficient in terms of storage and offers more functionality than rVariable, use of zVariable is strongly recommended. As a matter of fact, there’s no reason to use rVariables at all if you are creating a CDF file from scratch. One may wonder why there are rVariables and zVariables, not just zVariables. When CDF was first introduced, only rVariables were available. The inefficiencies with rVariables were quickly realized and addressed with the introduction of zVariables in later CDF releases.

The description for each API will detail its parameters: their types, for input or output and what the method returns. APIs that handle read/write of variable data and attribute entry may use a special indicator: TYPE, to specify the parameters that can have different signatures. The acceptable data types for such method are specified. For example, CDFgetzVarData method, returning a single zVariable value, is described as:

integer CDFgetEncoding ( id as long, varNum as integer, recNum as integer, indices as integer(), value as TYPE) {
  out -- Completion status code.
  in -- CDF identifier.
  in -- Variable number.
  in -- Record number.
  in -- Dimension indices.
  out -- Data value.
  TYPE -- VB value/string type or object
}

TYPE, as specified, can be defined a VB value or string (matching to the variable’s underlying data type) or simply a VB base Object. The following sample shows how the API is used to retrieve a data value from the zVariable “my_var”, a 2-dimensional, CDF_INT4 type at indices of {1,1} for record 1:

dim status as integer
dim indices() as integer = {1, 1}
dim id as long
dim value as integer
....
status = CDFgetEncoding(id, CDFvarNum(id, “my_var”), 1, indices, value)
Alternatively, value can be defined as object:

```
  dim value as object
  ....
  status = CDFgetEncoding(id, CDFvarNum(id, ”my_var”), 1, indices, value)
```

APIs are grouped, based on the CDF entities they operate on. These groups consist of general library information, CDF as a whole, variable and attribute/entry.

### 4.1 Library Information

The functions in this section are related to the current CDF library being used for the CDF operations, and they provide useful information such as the current library version number and Copyright notice.

#### 4.1.1 CDFgetDataTypeSize

```
integer CDFgetDataTypeSize (            
  '  out -- Completion status code.
  in -- CDF data type.
  '  out -- # of bytes for the given type.
dataType as integer,         
numBytes as integer)         
```

CDFgetDataTypeSize returns the size (in bytes) of the specified CDF data type.

The arguments to CDFgetDataTypeSize are defined as follows:

- `dataType` The CDF supported data type.
- `numBytes` The size of `dataType`.

#### 4.1.1.1. Example(s)

The following example returns the size of the data type CDF_INT4 that is 4 bytes.

```
  ....
  dim status as integer
  Dim numBytes as integer      
    ' Returned status code.
  ' Number of bytes.
  try                             
    ....
    status = CDFgetDataTypeSize(CDF_INT4, &numBytes)
    ....
  ... catch ex as Exception
```
4.1.2  CDFgetLibraryCopyright

integer CDFgetLibraryCopyright (copyright as string)

CDFgetLibraryCopyright returns the Copyright notice of the CDF library being used.

The arguments to CDFgetLibraryCopyright are defined as follows:

  copyright       The Copyright notice.

4.1.2.1. Example(s)

The following example returns the Copyright of the CDF library being used.

  ..
  ..
  ..
  dim status as integer
  Dim copyright as string
  ..
  try
  ....
  status = CDFgetLibraryCopyright(copyright)
  ...
  ...
  catch ex as Exception
  ...
  end try

4.1.3  CDFgetLibraryVersion

integer CDFgetLibraryVersion (version as integer,
   release as integer,
   increment as integer,
   subIncrement as string)

CDFgetLibraryVersion returns the version and release information of the CDF library being used.

The arguments to CDFgetLibraryVersion are defined as follows:

  version      Library version.
  release      Library release.
  increment    Library increment.
  subIncrement Library sub-increment.
version    The library version number.
release    The library release number.
increment  The library incremental number.
subIncrement The library sub-incremental string, a single character.

4.1.3.1. Example(s)

The following example returns the version and release information of the CDF library that is being used.

```
dim status as integer ' Returned status code.
Dim version as integer ' CDF library version number.
Dim release as integer ' CDF library release number.
Dim increment as integer ' CDF library incremental number.
Dim subIncrement as string ' CDF library sub-incremental character.
	ry
....
status = CDFgetLibraryVersion( version, release, increment, subIncrement)
....
catch ex as Exception
....
end try
```

4.1.4 CDFgetStatusText

```
dim varNum as integer CDFgetStatusText(
status as integer, ' out -- Completion status code.
message as string) ' in -- The status code.

CDFgetStatusText is identical to CDFerror, a legacy CDF function, (see section 4.2.8), and the use of this method is strongly encouraged over CDFerror as it might not be supported in the future. This method is used to inquire the text explanation of a given status code. Chapter 5 explains how to interpret status codes and Appendix A lists all of the possible status codes.

The arguments to CDFgetStatusText are defined as follows:

status       The status code to check.
message      The explanation of the status code.
4.1.4.1. Example(s)

The following example displays the explanation text for the error code that is returned from a call to CDFopenCDF.

```
...,

  dim id as long
  dim status as integer
  Dim text as string
  ...
  try
      ....
      status = CDFopenCDF ("giss_wetl", id)
      ...
      status = CDFclose(id)
  ...,
  catch ex as Exception
      text = CDFgetStatusMsg(ex.CDFgetCurrentStatus()) ...
  end try
```

4.2 CDF

The functions in this section provide CDF file-specific operations. Any operations involving variables or attributes are described in the following sections. This CDF has to be a newly created or opened from an existing one.

4.2.1 CDFclose

```
Integer CDFclose( id as long)
```

CDFclose closes the specified CDF. The CDF’s cache buffers are flushed the CDF’s open file is closed (or files in the case of a multi-file CDF) and the CDF identifier is made available for reuse.

**NOTE:** You must close a CDF with CDFclose to guarantee that all modifications you have made will actually be written to the CDF’s file(s). If your program exits, normally or otherwise, without a successful call to CDFclose, the CDF’s cache buffers are left unflushed.

The arguments to CDFclose are defined as follows:

- **id** - The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopen.

4.2.1.1. Example(s)
The following example will close an open CDF.

```
    dim id as long
    dim status as integer

    try
    ....
    status = CDFopen("...", id)
    status = CDFclose (id)
    catch ex as Exception
    ...
    end try
```

### 4.2.2 CDFcloseCDF

The function `CDFcloseCDF` is defined as follows:

```
Integer CDFcloseCDF (    
    out -- Completion status code.
    id as long)    
    in -- CDF identifier.
```

CDFcloseCDF closes the specified CDF. This method is identical to CDFclose, a legacy CDF function. The use of this method is strongly encouraged over CDFclose as it might not be supported in the future. The CDF's cache buffers are flushed the CDF's open file is closed (or files in the case of a multi-file CDF) and the CDF identifier is made available for reuse.

**NOTE:** You must close a CDF with CDFcloseCDF to guarantee that all modifications you have made will actually be written to the CDF's file(s). If your program exits, normally or otherwise, without a successful call to CDFcloseCDF, the CDF's cache buffers are left unflushed.

The arguments to CDFcloseCDF are defined as follows:

- `id` The identifier of the CDF. This identifier must have been initialized by a call to CDFcreateCDF or CDFopenCDF.

### 4.2.2.1. Example(s)

The following example will close an open CDF.

```
    dim id as long
    dim status as integer

    try
    ....
```
status = CDFopenCDF ("giss_wetl", id)
...
status = CDFcloseCDF (id)
catch ex as Exception
...
end try

4.2.3  CDFcreate

Integer CDFcreate(
  CDFname as string,
  numDims as integer,
  dimSizes as integer(),
  encoding as integer,
  majority as integer,
  id as long)
  ' out -- Completion status
  ' in -- CDF file name.
  ' in -- Number of dimensions, rVariables.
  ' in -- Dimension sizes, rVariables.
  ' in -- Data encoding.
  ' in -- Variable majority.
  ' out -- CDF identifier.

CDFcreate, a legacy CDF function, creates a CDF as defined by the arguments. A CDF cannot be created if it already exists. (The existing CDF will not be overwritten.) If you want to overwrite an existing CDF, you must first open it with CDFopenCDF, delete it with CDFdeleteCDF, and then recreate it with CDFcreate. If the existing CDF is corrupted, the call to CDFopen will fail. (An error code will be returned.) In this case you must delete the CDF at the command line. Delete the dotCDF file (having an extension of .cdf), and if the CDF has the multi-file format, delete all of the variable files (having extensions of .v0,.v1,.. and .z0,.z1,.. ).

The arguments to CDFcreate are defined as follows:

   CDFname  The file name of the CDF to create. (Do not specify an extension.) This may be at most CDF_PATHNAME_LEN characters. A CDF file name may contain disk and directory specifications that conform to the conventions of the operating system being used (including logical names on OpenVMS systems and environment variables on UNIX systems).

   UNIX:  File names are case-sensitive.

   numDims  Number of dimensions the rVariables in the CDF are to have. This may be as few as zero (0) and at most CDF_MAX_DIMS.

   dimSizes  The size of each dimension. Each element of dimSizes specifies the corresponding dimension size. Each size must be greater then zero (0). For 0-dimensional rVariables this argument is ignored (but must be present).

   encoding  The encoding for variable data and attribute entry data. Specify one of the encodings described in Section 2.7.

   majority  The majority for variable data. Specify one of the majorities described in Section 2.9.

   id  The identifier for the created CDF. This identifier must be used in all subsequent operations on the CDF.

When a CDF is created, both read and write access are allowed. The default format for a CDF created with CDFcreate is specified in the configuration file of your CDF distribution. Consult your system manager for this default.
NOTE: CDFclose must be used to close the CDF before your application exits to ensure that the CDF will be correctly written to disk.

4.2.3.1. Example(s)

The following example creates a CDF named “test1.cdf” with network encoding and row majority.

```vba
dim id as long
Dim status as integer
dim numDims as integer = 3
Dim dimSizes() as integer = {180,360,10}
dim majority as integer  =  ROW_MAJOR

try
    status =  CDFcreate  ("test1",  numDims,  dimSizes,  NETWORK_ENCODING, majority,  id)
    catch ex as Exception
        ...
end try
```

4.2.4 CDFcreateCDF

Integer CDFcreateCDF(
    cdfName as string,
    id as long)

CDFcreateCDF creates a CDF file. This method is a simple form of CDFcreate without the number of dimensions, dimensional sizes, encoding and majority arguments. It is the better method if only zVariables are to be created in the CDF. The created CDF will use the default encoding (HOST_ENCODING) and majority (ROW_MAJOR). A CDF cannot be created if it already exists. (The existing CDF will not be overwritten.) If you want to overwrite an existing CDF, you can either manually delete the file or open it with CDFopenCDF, delete it with CDFdeleteCDF, and then recreate it with CDFcreateCDF. If the existing CDF is corrupted, the call to CDFopenCDF will fail. (An error code will be returned.) In this case you must delete the CDF at the command line. Delete the dotCDF file (having an extension of .cdf), and if the CDF has the multi-file format, delete all of the variable files (having extensions of .v0,.v1,. . . and .z0,.z1,. . .).

Note that a CDF file created with CDFcreateCDF can only accept zVariables, not rVariables. But this is fine since zVariables are more flexible than rVariables. See the third paragraph of Chapter 3 for the differences between rVariables and zVariables.

The arguments to CDFcreateCDF are defined as follows:

CDFname The file name of the CDF to create. (Do not specify an extension.) This may be at most CDF_PATHNAME_LEN characters. A CDF file name may contain disk and directory specifications that conform to the conventions of the operating system being used (including logical names on OpenVMS systems and environment variables on UNIX systems).
UNIX: File names are case-sensitive.

id

The identifier for the created CDF. This identifier must be used in all subsequent operations on the CDF.

When a CDF is created, both read and write access are allowed. The default format for a CDF created with CDFcreateCDF is specified in the configuration file of your CDF distribution. Consult your system manager for this default.

NOTE: CDFcloseCDF must be used to close the CDF before your application exits to ensure that the CDF will be correctly written to disk.

4.2.4.1. Example(s)

The following example creates a CDF named “test1.cdf” with the default encoding and majority.

```plaintext
dim id as long  ' CDF identifier.
dim status as integer  ' Returned status code.

try
   ...
   status = CDFcreateCDF ("test1", id)
   ...
   status = CDFclose (id)
   catch ex as Exception 
      ...
end try
```

4.2.5  CDFdelete

integer CDFdelete(
   id as long)  ' out -- Completion status code.

CDFdelete, a legacy CDF function, deletes the specified CDF. The CDF files deleted include the dotCDF file (having an extension of .cdf), and if a multi-file CDF, the variable files (having extensions of .v0,.v1,. . . and .z0,.z1,. . . ).

You must open a CDF before you are allowed to delete it. If you have no privilege to delete the CDF files, they will not be deleted. If the CDF is corrupted and cannot be opened, the CDF file(s) must be deleted at the command line.

The arguments to CDFdelete are defined as follows:

id

The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopen.
4.2.5.1. Example(s)

The following example will open and then delete an existing CDF.

```vba
try
..
  dim id as long                        ' CDF identifier.
  Dim status as integer                ' Returned status code.
.. try
    ..
      status = CDFopen ("test2", id)
      status = CDFdelete (id)
    ..
  catch ex as Exception
    ..
  end try
```

4.2.6 CDFdeleteCDF

integer CDFdeleteCDF(                         ' out -- Completion status code.
  id as long)                                 ' in -- CDF identifier.

CDFdeleteCDF deletes the specified CDF. This method is identical to CDFdelete, and the use of this method is strongly encouraged over CDFdelete as it might not be supported in the future. The CDF files deleted include the dotCDF file (having an extension of .cdf), and if a multi-file CDF, the variable files (having extensions of .v0,.v1,.., and .z0,.z1,...).

You must open a CDF before you are allowed to delete it. If you have no privilege to delete the CDF files, they will not be deleted. If the CDF is corrupted and cannot be opened, the CDF file(s) must be deleted at the command line.

The arguments to CDFdeleteCDF are defined as follows:

id The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopen.

4.2.6.1. Example(s)

The following example will open and then delete an existing CDF.

```vba
try
..
  dim id as long                        ' CDF identifier.
  Dim status as integer                ' Returned status code.
.. try
```
dim status as integer  ' Returned status code.
.
try
....
  status = CDFopenCDF ("test2", id)
  ....
  status = CDFdeleteCDF(id)
  ...
catch ex as Exception
  ...
end try

4.2.7  CDFdoc

integer CDFdoc(
  id as long,
  version as integer,
  release as integer,
  copyright as string)
  ' out -- Completion status code.
  ' in -- CDF identifier.
  ' out -- Version number.
  ' out -- Release number.
  ' out -- copyright.

CDFdoc is used to inquire general information about a CDF. The version/release of the CDF library that created the
CDF is provided (e.g., CDF V3.1 is version 3, release 1) along with the CDF copyright notice. The copyright notice is
formatted for printing without modification.

The arguments to CDFdoc are defined as follows:

  id  The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate
       (or CDFcreateCDF) or CDFopen.

  version  The version number of the CDF library that created the CDF.

  release  The release number of the CDF library that created the CDF.

  copyright  The Copyright notice of the CDF library that created the CDF. This string will contain a
              newline character after each line of the Copyright notice.

4.2.7.1.  Example(s)

The following example returns and displays the version/release and copyright notice.

  .
  .
  dim id as long  ' CDF identifier.
  dim status as integer  ' Returned status code.
  Dim version as integer  ' CDF version number.
  Dim release as integer  ' CDF release number.
  Dim copyright as string  ' Copyright notice.
try 
....
status =  CDFdoc  (id, version, release, copyright)
....
catch ex as Exception 
....
end try

4.2.8   CDFerror

integer CDFerror(               ‘ out -- Completion status code.
status as integer,             ‘ in -- Status code.
message as string)             ‘ out -- Explanation text.

CDFerror, a legacy CDF function, is used to inquire the explanation of a given status code (not just error codes). Chapter 5 explains how to interpret status codes and Appendix A lists all of the possible status codes.

The arguments to CDFerror are defined as follows:

status          The status code to check.
message         The explanation of the status code.

4.2.8.1.  Example(s)

The following example displays the explanation text if an error code is returned from a call to CDFopen.

try 
....
status =  CDFopen  ("giss_wetl",  id)
....
catch ex as Exception 
   dim status as integer1 =  CDFerror(ex.GetCurrentStatus(), out text)  
....
end try

5 A legacy CDF function. While it is still available in V3.1, CDFgetStatusText is the preferred function for it.
4.2.9 CDFgetCacheSize

integer CDFgetCacheSize (  
id as long,  
numBuffers as integer)  

CDFgetCacheSize returns the number of cache buffers being used for the dotCDF file when a CDF is open. Refer to the CDF User’s Guide for description of caching scheme used by the CDF library.

The arguments to CDFgetCacheSize are defined as follows:

id The identifier of the CDF. This identifier must have been initialized by a call to CDFcreateCDF (or CDFcreate) or CDFopen.

numBuffers The number of cache buffers.

4.2.9.1. Example(s)

The following example returns the cache buffers for the open CDF file.

```pascal
try
    status = CDFgetCacheSize (id, numBuffers)
    ...
end try
```

4.2.10 CDFgetChecksum

integer CDFgetChecksum (  
id as long,  
checksum as integer)  

CDFgetChecksum returns the checksum mode of a CDF. The CDF checksum mode is described in Section 2.20.

The arguments to CDFgetChecksum are defined as follows:
id
The identifier of the CDF. This identifier must have been initialized by a call to
CDFcreateCDF (or CDFcreate) or CDFopen.

checksum
The checksum mode (NO_CHECKSUM or MD5_CHECKSUM).

### 4.2.10.1. Example(s)

The following example returns the checksum code for the open CDF file.

```vbscript
...  
...  
...  dim id as long    ' CDF identifier.
dim status as integer ' Returned status code.
dim checksum as integer ' CDF’s checksum.
...
try
    ...
    status = CDFgetChecksum (id, checksum)
    ...
...  catch ex as Exception
    ...
end try
```

### 4.2.11 CDFgetCompression

```vbscript
integer CDFgetCompression (    ' out -- Completion status code.
id as long,    ' in -- CDF identifier.
compressionType as integer,    ' out -- CDF’s compression type.
compressionParms as integer(),  ' out -- Compression parameters.
compressionPercentage as integer)  ' out -- Compressed percentage.
```

CDFgetCompression gets the compression information of the CDF. It returns the compression type (method) and, if compressed, the compression parameters and compression rate. CDF compression types/parameters are described in Section 2.11. The compression percentage is the result of the compressed file size divided by its original, uncompressed file size.6

The arguments to CDFgetCompression are defined as follows:

- **id** The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

---

6 The compression ratio is (100 – compression percentage): the lower the compression percentage, the better the compression ratio.
compressionType  The type of the compression.
compressionParms  The parameters of the compression.
compressionPercentage  The compression rate.

4.2.11.1. Example(s)

The following example returns the compression information of the open CDF file.

dim id as long
  ' CDF identifier.
dim status as integer
  ' Returned status code.
dim compressType as integer
  ' CDF’s compression type.
Dim compressionParms() as integer
  ' Compression parameters.
dim compressionPercentage as integer
  ' Compression rate.

try
  ....
  status = CDFgetCompression (id, compression, compressionParms, compressionPercentage)
  ...
catch ex as Exception
  ...
end try

4.2.12  CDFgetCompressionCacheSize

integer CDFgetCompressionCacheSize (id as long,
numBuffers as integer)
  ' in  --  CDF identifier.
  ' out  --  CDF’s compressed cache buffers.

CDFgetCompressionCacheSize gets the number of cache buffers used for the compression scratch CDF file. Refer to the CDF User’s Guide for description of caching scheme used by the CDF library.

The arguments to CDFgetCompressionCacheSize are defined as follows:

Id  The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
numBuffers  The number of cache buffers.
4.2.12.1. Example(s)

The following example returns the number of cache buffers used for the scratch file from the compressed CDF file.

```vba
... 
... 
dim id as long
dim status as integer
dim numBuffers as integer
... 
try
... 
status = CDFgetCompressionCacheSize (id, numBuffers)
... 
catch ex as Exception
... 
end try
```

4.2.13  CDFgetCompressionInfo

```vba
integer CDFgetCompressionInfo ( 
CDFname as string, 
compType as integer, 
cParms.as integer() 
cSize as long. 
uSize as long).
```

CDFgetCompressionInfo returns the compression type/parameters of a CDF without having to open the CDF. This refers to the compression of the CDF - not of any compressed variables.

The arguments to CDFgetCompressionInfo are defined as follows:

- **CDFname** The pathname of a CDF file without the .cdf file extension.
- **compType** The CDF compression type.
- **cParms** The CDF compression parameters.
- **cSize** The compressed CDF file size.
- **uSize** The size of CDF when decompress the originally compressed CDF.

4.2.13.1. Example(s)

The following example returns the compression information from a “unopen” CDF named “MY_TEST.cdf”.

...
dim status as integer
dim compType as integer
dim cParms as integer()
Dim cSize as long
Dim uSize as long

try
    status = CDFgetCompressionInfo("MY_TEST", compType, cParms, cSize, uSize)
    ...
catch ex as Exception
    ...
end try

4.2.14 CDFgetCopyright

integer CDFgetCopyright ( id as long, copyright as string)

CDFgetCopyright gets the Copyright notice in a CDF.

The arguments to CDFgetCopyright are defined as follows:

id The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate
    (or CDFcreateCDF) or CDFopenCDF.

copyright CDF Copyright.

4.2.14.1. Example(s)

The following example returns the Copyright in a CDF.

dim id as long
    ' CDF identifier.
dim status as integer
    ' Returned status code.
Dim copyright as string
    ' CDF’s copyright.

try
    ....
status = CDFgetCopyright (id, copyright)
...
...
catch ex as Exception
...
end try

4.2.15  CDFgetDecoding

integer CDFgetDecoding ( id as long, 
decoding as integer)  

CDFgetDecoding returns the decoding code for the data in a CDF. The decodings are described in Section 2.8.

The arguments to CDFgetDecoding are defined as follows:

id   The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

decoding The decoding of the CDF.

4.2.15.1. Example(s)

The following example returns the decoding for the CDF.

...  
...  

dim id as long  
dim status as integer  
dim decoding as integer  

try  
...  
status = CDFgetDecoding(id, decoding)  
...  
catch ex as Exception  
...  
end try

4.2.16  CDFgetEncoding
integer CDFgetEncoding (id as long, encoding as integer)  
          ‘ out -- Completion status code. 
          ‘ in -- CDF identifier. 
          ‘ out -- CDF encoding.

CDFgetEncoding returns the data encoding used in a CDF. The encodings are described in Section 2.7.

The arguments to CDFgetEncoding are defined as follows:

    id                The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
    encoding         The encoding of the CDF.

4.2.16.1. Example(s)

The following example returns the data encoding used for the given CDF.

    try 
    ....
    status = CDFgetEncoding(id, encoding)
    ....
    catch ex as Exception
    ....
end try

4.2.17  CDFgetFileBackward

integer CDFgetFileBackward()  
          ‘ out -- File Backward Mode.

CDFgetFileBackward returns the backward mode information dealing with the creation of a new CDF file. A mode of value 1 indicates when a new CDF file is created, it will be a backward version of V2.7, not the current library version.

The arguments to CDFgetFileBackward are defined as follows:

    N/A
4.2.17.1. **Example(s)**

In the following example, the CDF’s file backward mode is acquired.

```plaintext
... 
... 
... 

dim id as long  ' CDF identifier.  
dim status as integer  ' Returned status code.  
dim mode as integer  ' Backward mode.  
... 
try  
... 
mode = CDFgetFileBackward ()  
if mode = 1 then  
... 
end if  
catch ex as Exception  
... 
end try  
```

4.2.18 **CDFgetFormat**

```plaintext
integer CDFgetFormat (  
    id as long,  
    format as integer)  
    ' out -- Completion status code.  
    ' in -- CDF identifier.  
    ' out -- CDF format.  
```

CDFgetFormat returns the file format, single or multi-file, of the CDF. The formats are described in Section 2.5.

The arguments to CDFgetFormat are defined as follows:

- **id**
  - The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

- **format**
  - The format of the CDF.

4.2.18.1. **Example(s)**

The following example returns the file format of the CDF.

```plaintext
... 
... 
... 

dim id as long  ' CDF identifier.  
dim status as integer  ' Returned status code.  
```
4.2.19 CDFgetLeapSecondLastUpdated

integer CDFgetLeapSecondLastUpdated ( id as long, lastUpdated as integer) ' out -- Completion status code. ' in -- CDF identifier. ' out -- CDF format.

CDFgetLeapSecondLastUpdated returns the leap second last updated date from the CDF. This value indicates what/if the leap second table this CDF is based on. It is of YYYYMMDD form. The value can also be negative 1 (-1), the field not set (for older CDFs), or zero (0) if the leap second table is not being accessed. This field is only relevant to TT2000 data in the CDF.

The arguments to CDFgetLeapSecondLastUpdated are defined as follows:

id The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

lastUpdated The date that the latest leap second was added to the leap second table.

4.2.19.1. Example(s)

The following example returns the date that the last leap second was added to the leap second table from the CDF.

dim id as long ' CDF identifier.
dim status as integer ' Returned status code.
dim lastUpdated as integer ' Format.

try
  status = CDFgetLeapSecondLastUpdated(id, lastUpdated)
  ...
  ...
  catch ex as Exception
  ...
end try
### 4.2.20 CDFgetMajority

```plaintext
integer CDFgetMajority (  
id as long,  
majority as integer)  
```

CDFgetMajority returns the variable majority, row or column-major, of the CDF. The majorities are described in Section 2.9.

The arguments to CDFgetMajority are defined as follows:

- **id** The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **majority** The variable majority of the CDF.

#### 4.2.20.1. Example(s)

The following example returns the majority of the CDF.

```plaintext
...  
try  
  status = CDFgetMajority (id, majority)  
...  
catch ex as Exception  
...  
end try
```

### 4.2.21 CDFgetName

```plaintext
integer CDFgetName (  
id as long,  
name as string)  
```

CDFgetName returns the file name of the specified CDF.
The arguments to CDFgetName are defined as follows:

- **id**  The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **name**  The file name of the CDF.

### 4.2.21.1. Example(s)

The following example returns the name of the CDF.

```vbnet
dim id as long  ' CDF identifier.
dim status as integer  ' Returned status code.
Dim name as string  ' Name of the CDF.

try
  ....
  status = CDFgetName (id, name)
  ....
...  catch ex as Exception
    ...
end try
```

### 4.2.22  CDFgetNegtoPosfp0Mode

```vbnet
integer CDFgetNegtoPosfp0Mode (  ' out -- Completion status code.
  id as long,  ' in -- CDF identifier.
  negtoPosfp0 as integer)  ' out -- -0.0 to 0.0 mode.
```

CDFgetNegtoPosfp0Mode returns the –0.0 to 0.0 mode of the CDF. You can use CDFsetNegtoPosfp0 method to set the mode. The –0.0 to 0.0 modes are described in Section 2.16.

The arguments to CDFgetNegtoPosfp0Mode are defined as follows:

- **id**  The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **negtoPosfp0**  The –0.0 to 0.0 mode of the CDF.

### 4.2.22.1. Example(s)
The following example returns the \(-0.0\) to \(0.0\) mode of the CDF.

```vbnet
dim id as long
   ' CDF identifier.
dim status as integer
   ' Returned status code.
Dim negtoPosfp0 as integer
   ' -0.0 to 0.0 mode.

try
   ....
   status = CDFgetNegtoPosfp0Mode (id, negtoPosfp0)

   ....
catch ex as Exception
   ....
end try
```

### 4.2.23 **CDFgetReadOnlyMode**

```vbnet
integer CDFgetReadOnlyMode(
   id as long,
   readOnlyMode as integer)
   ' out -- Completion status code.
   ' in -- CDF identifier.
   ' out -- CDF read-only mode.
```

CDFgetReadOnlyMode returns the read-only mode for a CDF. You can use CDFsetReadOnlyMode to set the mode of readOnlyMode. The read-only modes are described in Section 2.14.

The arguments to CDFgetReadOnlyMode are defined as follows:

- **id**
  The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

- **readOnlyMode**
  The read-only mode (READONLYon or READONLYoff).

### 4.2.23.1. Example(s)

The following example returns the read-only mode for the given CDF.

```vbnet
dim id as long
   ' CDF identifier.
Dim status as integer
   ' CDF read-only mode.
dim readMode as integer

try
   ....
```

48
status = CDFgetReadOnlyMode (id, readMode)
...
catch ex as Exception
...
end try

4.2.24  CDFgetStageCacheSize

integer CDFgetStageCacheSize(  
  id as long,  
  numBuffers as integer)  

CDFgetStageCacheSize returns the number of cache buffers being used for the staging scratch file a CDF. Refer to the CDF User’s Guide for the description of the caching scheme used by the CDF library.

The arguments to CDFgetStageCacheSize are defined as follows:

id          The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

numBuffers  The number of cache buffers.

4.2.24.1. Example(s)

The following example returns the number of cache buffers used in a CDF.

status = CDFgetStageCacheSize (id, numBuffers)
4.2.25 CDFgetValidate

integer CDFgetValidate() ‘ out – CDF validation mode.

CDFgetValidate returns the data validation mode. This information reflects whether when a CDF is open, its certain data fields are subjected to a validation process. 1 is returned if the data validation is to be performed, 0 otherwise.

The arguments to CDFgetVersion are defined as follows:

N/A

4.2.25.1. Example(s)

In the following example, it gets the data validation mode.

```
try
    ... validate = CDFgetValidate ()
    ...
catch ex as Exception
    ...
end try
```

4.2.26 CDFgetVersion

integer CDFgetVersion( id as long, version as integer, release as integer, increment as integer) ‘ out -- Completion status code.
’ in -- CDF identifier.
’ out -- CDF version.
’ out -- CDF release.
’ out -- CDF increment.

CDFgetVersion returns the version/release information for a CDF file. This information reflects the CDF library that was used to create the CDF file.

The arguments to CDFgetVersion are defined as follows:
id The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

version The CDF version number.

release The CDF release number.

increment The CDF increment number.

4.2.26.1. Example(s)

In the following example, a CDF’s version/release is acquired.

```vbnet
dim id as long ' CDF identifier.
dim status as integer ' Returned status code.
dim version as integer ' CDF version.
dim release as integer ' CDF release.
dim increment as integer ' CDF increment.

try
    status = CDFgetVersion (id, version, release, increment)

end try
```

4.2.27 CDFgetzMode

integer CDFgetzMode(
    id as long, ' in -- CDF identifier.
    zMode as integer) ' out -- CDF zMode.

CDFgetzMode returns the zMode for a CDF file. The zModes are described in Section 2.15.

The arguments to CDFgetzMode are defined as follows:

id The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

zMode The CDF zMode.
4.2.27.1. Example(s)

In the following example, a CDF’s zMode is acquired.

```
        .
        .
        dim id as long
        dim status as integer
        dim zMode as integer
        .
        try
        ....
        status = CDFgetzMode (id, zMode)
        ...
        ... catch ex as Exception ...
        end try
```

4.2.28 CDFinquire

integer CDFinquire( 
    id as long, 
    numDims as integer, 
    dimSizes as integer(), 
    encoding as integer, 
    majority as integer, 
    maxRec as integer, 
    numVars as integer, 
    numAttrs as integer)

CDFinquire returns the basic characteristics of a CDF. An application needs to know the number of rVariable dimensions and their sizes before it can access rVariable data (since all rVariables’ dimension and dimension size are the same). Knowing the variable majority can be used to optimize performance and is necessary to properly use the variable hyper functions (for both rVariables and zVariables).

The arguments to CDFinquire are defined as follows:

id The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopen.
numDims The number of dimensions for the rVariables in the CDF.
dimSizes The dimension sizes of the rVariables in the CDF. dimSizes is a 1-dimensional array containing one element per dimension. Each element of dimSizes receives the corresponding dimension size. For 0-dimensional rVariables this argument is ignored (but must be present).
encoding  The encoding of the variable data and attribute entry data. The encodings are defined in Section 2.7.

majority The majority of the variable data. The majorities are defined in Section 2.9.

maxRec The maximum record number written to an rVariable in the CDF. Note that the maximum record number written is also kept separately for each rVariable in the CDF. The value of maxRec is the largest of these. Some rVariables may have fewer records actually written. Use CDFrVarMaxWrittenRecNum to inquire the maximum record written for an individual rVariable.

numVars The number of rVariables in the CDF.

numAttrs The number of attributes in the CDF.

4.2.28.1. Example(s)

The following example returns the basic information about a CDF.

```

dim id as long
dim status as integer
dim numDims as integer
Dim dimSizes() as integer
dim encoding as integer
dim majority as integer
dim maxRec as integer

dim numVars as integer
dim numAttrs as integer

try
...
status = CDFinquire (id, numDims, dimSizes, encoding, majority, _
 maxRec, numVars, numAttrs)
...
catch ex as Exception
...
end try
```

4.2.29  CDFinquireCDF

```
integer CDFinquireCDF(
  id as long,
  numDims as integer,
  dimSizes as integer(),

  
  ' out -- Completion status code.
  ' in  -- CDF identifier
  ' out -- Number of dimensions for rVariables.
  ' out -- Dimension sizes for rVariables.
```

53
encoding as integer,        '  out -- Data encoding.
majority as integer,       '  out -- Variable majority.
maxrRec as integer,        '  out -- Maximum record number among rVariables.
numrVars as integer,       '  out -- Number of rVariables in the CDF.
maxzRec as integer,        '  out -- Maximum record number among zVariables.
numzVars as integer,       '  out -- Number of zVariables in the CDF.
numAttrs as integer)       '  out -- Number of attributes in the CDF.

CDFinquireCDF returns the basic characteristics of a CDF. This method expands the method CDFinquire by acquiring extra information regarding the zVariables. Knowing the variable majority can be used to optimize performance and is necessary to properly use the variable hyper-get/put functions.

The arguments to CDFinquireCDF are defined as follows:

id The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
numDims The number of dimensions for the rVariables in the CDF. Note that all the rVariables’ dimensionality in the same CDF file must be the same.
dimSizes The dimension sizes of the rVariables in the CDF (note that all the rVariables’ dimension sizes in the same CDF file must be the same). dimSizes is a 1-dimensional array containing one element per dimension. Each element of dimSizes receives the corresponding dimension size. For 0-dimensional rVariables this argument is ignored (but must be present).
encoding The encoding of the variable data and attribute entry data. The encodings are defined in Section 2.7.
majority The majority of the variable data. The majorities are defined in Section 2.9.
maxrRec The maximum record number written to an rVariable in the CDF. Note that the maximum record number written is also kept separately for each rVariable in the CDF. The value of maxRec is the largest of these.
umrVars The number of rVariables in the CDF.
maxzRec The maximum record number written to a zVariable in the CDF. Note that the maximum record number written is also kept separately for each zVariable in the CDF. The value of maxRec is the largest of these. Some zVariables may have fewer records than actually written. Use CDFgetzVarMaxWrittenRecNum to inquire the actual number of records written for an individual zVariable.
umzVars The number of zVariables in the CDF.
numAttrs The number of attributes in the CDF.

4.2.29.1. Example(s)

The following example returns the basic information about a CDF.

.
dim id as long
  '  CDF identifier.
dim status as integer
  '  Returned status code.
dim numDims as integer
  '  Number of dimensions, rVariables.
Dim dimSizes() as integer
  '  Dimension sizes, rVariables.
dim encoding as integer
  '  Data encoding.
dim majority as integer
  '  Variable majority.
dim maxRec as integer
  '  Maximum record number, rVariables.
dim numrVars as integer
  '  Number of rVariables in CDF.
dim maxzRec as integer
  '  Maximum record number, zVariables.
dim numzVars as integer
  '  Number of zVariables in CDF.
dim numAttrs as integer
  '  Number of attributes in CDF.
.
try
  status = CDFinquireCDF (id, numDims, dimSizes, encoding, majority, 
                          maxRec, numrVars, maxzRec, numzVars, numAttrs)
  ...
  ...
catch ex as Exception 
  ...
end try

4.2.30  CDFopen

integer CDFopen(
  out status as integer, 
  out id as long, 
  in CDFname as string,
)
  '  Completion status code.
  '  CDF file name.
  '  CDF identifier.

CDFopen, a legacy CDF function, opens an existing CDF. The CDF is initially opened with only read access. This allows multiple applications to read the same CDF simultaneously. When an attempt to modify the CDF is made, it is automatically closed and reopened with read/write access. (The method will fail if the application does not have or cannot get write access to the CDF.)

The arguments to CDFopen are defined as follows:

CDFname The file name of the CDF to open. (Do not specify an extension.) This may be at most CDF_PATHNAME_LEN characters. A CDF file name may contain disk and directory specifications that conform to the conventions of the operating system being used (including logical names on OpenVMS systems and environment variables on UNIX systems).

UNIX: File names are case-sensitive.

id The identifier for the opened CDF. This identifier must be used in all subsequent operations on the CDF.

NOTE: CDFclose must be used to close the CDF before your application exits to ensure that the CDF will be correctly written to disk.
4.2.30.1. Example(s)

The following example will open a CDF named “NOAA1.cdf”.

```vbnet
' CDF identifier.
' Returned status code.
' file name of CDF.
dim id as long
dim status as integer
Dim CDFname as string = "NOAA1"
try
try
  status = CDFopen (CDFname, id)
catch ex as Exception
  ...
end try
```

4.2.31  CDFopenCDF

Integer CDFopenCDF(  
  out -- Completion status code.
  CDFname as string,  
in -- CDF file name.
  id as long)  
  out -- CDF identifier.

CDFopenCDF opens an existing CDF. This method is identical to CDFopen, and the use of this method is strongly encouraged over CDFopen as it might not be supported in the future. The CDF is initially opened with only read access. This allows multiple applications to read the same CDF simultaneously. When an attempt to modify the CDF is made, it is automatically closed and reopened with read/write access. The method will fail if the application does not have or cannot get write access to the CDF.

The arguments to CDFopenCDF are defined as follows:

```
CDFname The file name of the CDF to open. (Do not specify an extension.) This may be at most CDF_PATHNAME_LEN characters. A CDF file name may contain disk and directory specifications that conform to the conventions of the operating system being used (including logical names on OpenVMS systems and environment variables on UNIX systems).

UNIX: File names are case-sensitive.

id The identifier for the opened CDF. This identifier must be used in all subsequent operations on the CDF.
```

NOTE: CDFcloseCDF must be used to close the CDF before your application exits to ensure that the CDF will be correctly written to disk.
4.2.31.1. Example(s)

The following example will open a CDF named “NOAA1.cdf”.

```
    .
    .
    .
    dim id as long          '  CDF identifier.
    dim status as integer   '  Returned status code.
    Dim CDFname as string = "NOAA1"  ' file name of CDF.
    .
    try
    ....
    status = CDFopenCDF (CDFname, id)
    ...
    catch ex as Exception
    ...
    end try
```

4.2.32  CDFselect

integer CDFselect(          '  out -- Completion status code.
    id as long)              '  in -- CDF identifier.

CDFselect selects an opened CDF as the current CDF. Only one CDF is allowed to be current. To access data from a CDF, that CDF must be selected as the current. This method is no longer needed as the methods involved CDF operations always need the CDF identifier, as the first argument, so it can be set as current before other operations can be applied.

The arguments to CDFselect are defined as follows:

- **id**: The identifier for the opened CDF. This identifier must be used in all subsequent operations on the CDF.

**NOTE**: When a CDF is opened, it becomes the current. No CDF is current after CDFcloseCDF is called to close the file.

4.2.32.1. Example(s)

The following example will select a CDF named “NOAA1.cdf” as the current CDF while another file “NOAA2.cdf” is also opened.

```
    .
    .
    .
    dim id1 as long, id2 as long          '  CDF identifier.
```
4.2.33  CDFselectCDF

integer CDFselectCDF(
    id as long)
    ' out -- Completion status code.
    ' in -- CDF identifier.

CDFselectCDF selects an opened CDF as the current CDF. Only one CDF is allowed to be current. To access data from
a CDF, that CDF must be selected as the current. This method is no longer needed as the methods involved CDF
operations always need the CDF identifier, as the first argument, so it can be set as current before other operations can
be applied. This method is identical to CDFselect.

The arguments to CDFselectCDF are defined as follows:

    id
    The identifier for the opened CDF. This identifier must be used in all subsequent operations
    on the CDF.

NOTE: When a CDF is opened, it becomes the current. No CDF is current after CDFcloseCDF is called to close the
file.

4.2.33.1. Example(s)

The following example will select a CDF named “NOAA1.cdf” as the current CDF while another file “NOAA2.cdf” is
also opened.

    try
        ....
        status = CDFopenCDF (CDFname1, id1)  
        status = CDFopenCDF (CDFname2, id2)  
        status = CDFselect(id1)  
        ....
        status = CDFclose(id1)  
        status = CDFclose(id2)  
        catch ex as Exception  
            ...  
    end try

    dim id1 as long, i2 as long
    dim status as integer
    Dim CDFname1 as string = "NOAA1"
    Dim CDFname2 as string = "NOAA2"
try
....
status = CDFopenCDF (CDFname1, id1)
status = CDFopenCDF (CDFname2, id2)
status = CDFselectCDF(id1)
....
status = CDFclose(id1)
status = CDFclose(id2)
catch ex as Exception
...
end try

4.2.34 CDFsetCacheSize

integer CDFsetCacheSize ( 
    id as long, 
    numBuffer as integer)

CDFsetCacheSize specifies the number of cache buffers being used for the dotCDF file when a CDF is open. Refer to the CDF User’s Guide for the description of the cache scheme used by the CDF library.

The arguments to CDFsetCacheSize are defined as follows:

id The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

numBuffers The number of cache buffers.

4.2.34.1. Example(s)

The following example extends the number of cache buffers to 500 for the open CDF file. The default number is 300 for a single-file format CDF on Unix systems.

dim id as long
    ' CDF identifier.
dim status as integer
    ' Returned status code.
dim cacheBuffers as integer
    ' CDF’s cache buffers.
cacheBuffers = 500
try
    ....
    status = CDFsetCacheSize (id, cacheBuffers)
    ...
...
catch ex as Exception
...
end try

4.2.35  CDFsetChecksum

integer CDFsetChecksum ( id as long, checksum as integer)

CDFsetChecksum specifies the checksum mode for the CDF. The CDF checksum mode is described in Section 2.20.

The arguments to CDFsetChecksum are defined as follows:

id  The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

checksum  The checksum mode (NO_CHECKSUM or MD5_CHECKSUM).

4.2.35.1. Example(s)

The following example turns off the checksum flag for the open CDF file.

. .
. .
dim id as long  ' CDF identifier.
dim status as integer  ' Returned status code.
dim checksum as integer  ' CDF’s checksum.
. .
checksum= NO_CHECKSUM
try
....
status = CDFsetChecksum (id, checksum)
...
...
catch ex as Exception
...
end try

4.2.36  CDFsetCompression

integer CDFsetCompression ( id as long,)

. .
. .
compressionType as integer, CompressionParms as integer())

CDFsetCompression specifies the compression type and parameters for a CDF. This compression refers to the CDF, not of any variables. The compressions are described in Section 2.11.

The arguments to CDFsetCompression are defined as follows:

id The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

compressionType The compression type.

compressionParms The compression parameters.

4.2.36.1. Example(s)

The following example uses GZIP.6 to compress the CDF file.

```vba

compressionType = GZIP_COMPRESSION
compressionParms(0) = 6
try
    status = CDFsetCompression (id, compressionType, compressionParms) …
...catch ex as Exception
    ...
eend try
```

4.2.37 CDFsetCompressionCacheSize

integer CDFsetCompressionCacheSize ( id as long, numBuffers as integer)

CDFsetCompressionCacheSize specifies the number of cache buffers used for the compression scratch CDF file. Refer to the CDF User’s Guide for the description of the cache scheme used by the CDF library.

The arguments to CDFsetCompressionCacheSize are defined as follows:
id The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

compressionNumBuffers The number of cache buffers.

### 4.2.37.1. Example(s)

The following example extends the number of cache buffers used for the scratch file from the compressed CDF file to 100. The default cache buffers is 80 for Unix systems.

```dim id as long
   dim status as integer
   dim numBuffers as integer = 100
   try
      status = CDFsetCompressionCacheSize (id, numBuffers)
   catch ex as Exception
      ...
   end try```

### 4.2.38  CDFsetDecoding

integer CDFsetDecoding ( id as long, decoding as integer)  

CDFsetDecoding sets the decoding of a CDF. The decodings are described in Section 2.8.

The arguments to CDFsetDecoding are defined as follows:

- **id** The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **decoding** The decoding of a CDF.

### 4.2.38.1. Example(s)

The following example sets NETWORK_DECODING to be the decoding scheme in the CDF.
4.2.39 CDFsetEncoding

integer CDFsetEncoding (  
id as long,  
encoding as integer) 

CDFsetEncoding specifies the data encoding of the CDF. A CDF’s encoding may not be changed after any variable values have been written. The encodings are described in Section 2.7.

The arguments to CDFsetEncoding are defined as follows:

id The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

encoding The encoding of the CDF.

4.2.39.1. Example(s)

The following example sets the encoding to HOST_ENCODING for the CDF.
status = CDFsetEncoding(id, encoding)
...
...
catch ex as Exception
...
end try

4.2.40 CDFsetFileBackward

void CDFsetFileBackward(
    mode as integer)  ' in -- File backward Mode.

CDFsetFileBackward sets the backward mode. When the mode is set as FILEBACKWARDOn, any new CDF files
created are of version 2.7, instead of the underlining library version. If mode FILEBACKWARDOff is used, the default
for creating new CDF files, the library version is the version of the file.

The arguments to CDFsetFileBackward are defined as follows:

 mode The backward mode.

4.2.40.1. Example(s)

In the following example, it sets the file backward mode to FILEBACKWARDOff, which means that any files to be
created will be of version V3.*, the same as the library version.

try
...
...
    CDFsetFileBackward (FILEBACKWARDOff)
...
...
catch ex as Exception
...
end try

4.2.41 CDFsetFormat

integer CDFsetFormat (  ' out -- Completion status code.
    id as long,  ' in -- CDF identifier.
    format as integer)  ' in -- CDF format.
CDFsetFormat specifies the file format, either single or multi-file format, of the CDF. A CDF’s format may not be changed after any variable values have been written. The formats are described in Section 2.5.

The arguments to CDFsetFormat are defined as follows:

id The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

format The file format of the CDF.

4.2.41.1. Example(s)

The following example sets the file format to MULTI_FILE for the CDF. The default is SINGLE_FILE format.

```plaintext
dim id as long
    ' CDF identifier.
Dim status as integer
    ' Returned status code.
Dim format as integer
    ' Format.

format = MULTI_FILE
try
    status = CDFsetFormat(id, format)
    ...
    ...
catch ex as Exception
    ...
end try
```

4.2.42  CDFsetLeapSecondLastUpdated

integer CDFsetLeapSecondLastUpdated ( out -- Completion status code.
id as long,     ' in -- CDF identifier.
lastUpdated as integer)     ' in -- Leap second last updated date

CDFsetLeapSecondLastUpdated respecifies the leap second last updated date in the CDF. The value, in YYYYMMDD form, indicates what/if the leap second table this CDF is based upon. The value is either a valid entry in the currently used leap second table, or zero (0). Value zero means the CDF is not using any leap second table. This field is only relevant to TT2000 data. Normally, this function is used for older CDFs that have not had the field set.

The arguments to CDFsetLeapSecondLastUpdated are defined as follows:

id The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

lastUpdated The date the latest leap second was added to the leap second table.
4.2.42.1. Example(s)

The following example resets the leap second last updated date in the CDF. Likely, the file’s field was not set originally (an older CDF).

```
dim id as long
  ' CDF identifier.
dim status as integer
  ' Returned status code.
dim lastUpdated as integer
  ' Leap second last updated.

lastUpdated = 20150701
try
  ....
  status = CDFsetLeapSecondLastUpdated (id, lastUpdated)
  ...
  ...
catch ex as Exception
  ...
end try
```

4.2.43  CDFsetMajority

```
integer CDFsetMajority (  
    id as long,  
    majority as integer)  
  ' out -- Completion status code.  
  ' in -- CDF identifier.  
  ' in -- CDF variable majority.

CDFsetMajority specifies the variable majority, either row or column-major, of the CDF. A CDF’s majority may not be changed after any variable values have been written. The majorities are described in Section 2.9.

The arguments to CDFsetMajority are defined as follows:

id
  The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

majority
  The variable majority of the CDF.

4.2.43.1. Example(s)

The following example sets the majority to COLUMN_MAJOR for the CDF. The default is ROW_MAJOR.

```
dim id as long
  ' CDF identifier.
Dim status as integer
Dim majority as integer
.
.
majority = COLUMN_MAJOR
try
    .
    status = CDFsetMajority (id, majority)
    ...
    ...
catch ex as Exception
    ...
end try

4.2.44 CDFsetNegtoPosfp0Mode

integer CDFsetNegtoPosfp0Mode ( 
    id as long, 
    negtoPosfp0 as integer)
.
.
CDFsetNegtoPosfp0Mode specifies the –0.0 to 0.0 mode of the CDF. The –0.0 to 0.0 modes are described in Section 2.16.

The arguments to CDFsetNegtoPosfp0Mode are defined as follows:

 id The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

 negtoPosfp0 The –0.0 to 0.0 mode of the CDF.

4.2.44.1. Example(s)

The following example sets the –0.0 to 0.0 mode to ON for the CDF.

.
.
.
dim id as long
Dim status as integer
Dim negtoPosfp0 as integer
.
.
negtoPosfp0 = NEGtoPOSfp0on
try
    .
    status = CDFsetNegtoPosfp0Mode (id, negtoPosfp0)
    ...
    ...
catch ex as Exception
    ...

67
end try

4.2.45  **CDFsetReadOnlyMode**

integer CDFsetReadOnlyMode( id as long, readOnlyMode as integer) ' out -- Completion status code. ' in -- CDF identifier. ' in -- CDF read-only mode.

CDFsetReadOnlyMode specifies the read-only mode for a CDF. The read-only modes are described in Section 2.14.

The arguments to CDFsetReadOnlyMode are defined as follows:

- **id**  
The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

- **readOnlyMode**  
The read-only mode.

4.2.45.1. Example(s)

The following example sets the read-only mode to OFF for the CDF.

```vba
... dim id as long ' CDF identifier. Dim readMode as integer ' CDF read-only mode. Dim status as integer ...

readMode = READONLYoff try .... status = CDFsetReadOnlyMode (id, readMode) ...
... catch ex as Exception ...
end try
```

4.2.46  **CDFsetStageCacheSize**

integer CDFsetStageCacheSize( id as long, numBuffers as integer) ' out -- Completion status code. ' in -- CDF identifier. ' in -- The stage cache size.
CDFsetStageCacheSize specifies the number of cache buffers being used for the staging scratch file a CDF. Refer to the CDF User’s Guide for the description of the caching scheme used by the CDF library.

The arguments to CDFsetStageCacheSize are defined as follows:

```
id
numBuffers
```

- `id` The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- `numBuffers` The number of cache buffers.

### 4.2.46.1. Example(s)

The following example sets the number of stage cache buffers to 10 for a CDF.

```
dim id as long
Dim numBuffers as integer
Dim status as integer

numBuffers = 10
try
    status = CDFsetStageCacheSize (id, numBuffers)
... catch ex as Exception
... end try
```

### 4.2.47 CDFsetValidate

```
void CDFsetValidate(
mode as integer)
```

CDFsetValidate sets the data validation mode. The validation mode dedicates whether certain data in an open CDF file will be validated. This mode should be set before the any files are opened. Refer to Data Validation Section 2.21.

The arguments to CDFsetVersion are defined as follows:

```
mode
```

- `mode` The validation mode.
4.2.47.1. Example(s)

In the following example, it sets the validation mode to be on, so any following CDF files are subjected to the data validation process when they are open.

```plaintext
try
    ...
    CDFsetValidate (VALIDATEFILEon)
    ...
catch ex as Exception
    ...
end try
```

4.2.48 CDFsetzMode

```plaintext
integer CDFsetzMode(id as long, zMode as integer)
```

CDFsetzMode specifies the zMode for a CDF file. The zModes are described in Section 2.15 and see the Concepts chapter in the CDF User’s Guide for a more detailed information on zModes. zMode is used when dealing with a CDF file that contains 1) rVariables, or 2) rVariables and zVariables. If you want to treat rVariables as zVariables, it’s highly recommended to set the value of zMode to zMODEon2.

The arguments to CDFsetzMode are defined as follows:

- **id** (The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.)
- **zMode** (The CDF zMode.)

4.2.48.1. Example(s)

In the following example, a CDF’s zMode is specified to zMODEon2: all rVariables are treated as zVariables with NOVARY dimensions being eliminated.

```plaintext
dim id as long
Dim status as integer
Dim zMode as integer

zMode = zMODEon2
try
```
status = CDFsetzMode (id, zMode)
...
catch ex as Exception

4.3 Variables

The methods in this section are all CDF variable-specific. A variable, either a rVariable or zVariable, is identified by its unique name in a CDF or a variable number. Before you can perform any operation on a variable, the CDF in which it resides in must be opened.

4.3.1 CDFcloserVar

integer CDFcloserVar(
    id as long,
    varNum as integer)

CDFcloserVar closes the specified rVariable file from a multi-file format CDF. Note that rVariables in a single-file CDF don’t need to be closed. The variable's cache buffers are flushed before the variable's open file is closed. However, the CDF file is still open.

NOTE: For the multi-file CDF, you must close all open variable files to guarantee that all modifications you have made will actually be written to the CDF's file(s). If your program exits, normally or otherwise, without a successful call to CDFcloseCDF, the CDF's cache buffers are left unflushed.

The arguments to CDFcloserVar are defined as follows:

id The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

varNum The variable number for the open rVariable’s file. This identifier must have been initialized by a call to CDFcreateVar or CDFgetVarNum.

4.3.1.1 Example(s)

The following example will close an open rVariable file from a multi-file CDF.

dim id as long
Dim status as integer
Dim varNum as integer
try
    ....
    varNum = CDFgetVarNum (id, “VAR_NAME1”)
    .
    status = CDFcloserVar (id, varNum)
...
catch ex as Exception
    ...
end try

4.3.2 CDFcloserVar

integer CDFcloserVar(  
    id as long,  
    varNum as integer)  

CDFcloserVar closes the specified zVariable file from a multi-file format CDF. Note that zVariables in a single-file CDF don’t need to be closed. The variable's cache buffers are flushed before the variable's open file is closed. However, the CDF file is still open.

NOTE: For the multi-file CDF, you must close all open variable files to guarantee that all modifications you have made will actually be written to the CDF's file(s). If your program exits, normally or otherwise, without a successful call to CDFcloseCDF, the CDF's cache buffers are left unflushed.

The arguments to CDFcloserVar are defined as follows:

- **id**: The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **varNum**: The variable number for the open zVariable's file. This identifier must have been initialized by a call to CDFcreatezVar or CDFgetVarNum.

4.3.2.1. Example(s)

The following example will close an open zVariable file from a multi-file CDF.

. 
. 
. 
.
    dim id as long  
    Dim status as integer  
    Dim varNum as integer  
    . 
    try
        ....
        varNum = CDFgetVarNum (id, “VAR_NAME1”)
        .
    .
4.3.3  

CDFconfirmrVarExistence

integer CDFconfirmrVarExistence(        · out -- Completion status code.
    id as long,                           · in -- CDF identifier.
    varName as string)                    · in -- rVariable name.

CDFconfirmrVarExistence confirms the existence of a rVariable with a given name in a CDF. If the rVariable does not exist, an error code will be returned. No exception is thrown if the variable is not found.

The arguments to CDFconfirmrEntryExistence are defined as follows:

    id    The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

    varName The rVariable name to check.

4.3.3.1.  Example(s)

The following example checks the existence of rVariable “MY_VAR” in a CDF.

    .
    .
    .
    dim id as long                          · CDF identifier.
    Dim status as integer                   · Returned status code.
    .
    try
    ....
    status = CDFconfirmrVarExistence (id, “MY_VAR”)  · in -- rVariable name.
    if status <> CDF_OK then UserStatusHandler (status)
    ...
    ...
    catch ex as Exception
    ...
    end try

4.3.4  

CDFconfirmrVarPadValueExistence
integer CDFconfirmrVarPadValueExistence(
    id as long,
    varNum as integer)
    ' out -- Completion status code.
    ' in -- CDF identifier.
    ' in -- rVariable number.

CDFconfirmrVarPadValueExistence confirms the existence of an explicitly specified pad value for the specified rVariable in a CDF. If an explicit pad value has not been specified, the informational status code NO_PADVALUE_SPECIFIED will be returned. No exception is thrown if the variable’s pad value is not defined.

The arguments to CDFconfirmrVarPadValueExistence are defined as follows:

id The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
varNum The rVariable number.

4.3.4.1. Example(s)

The following example checks the existence of the pad value of rVariable “MY_VAR” in a CDF.

```
try
    varNum = CDFgetVarNum(id, "MY_VAR")
    status = CDFconfirmrVarPadValueExistence (id, varNum)
    if status <> NO_PADVALUE_SPECIFIED then
        ...
    end if
... catch ex as Exception
... end try
```

4.3.5 CDFconfirmzVarExistence

integer CDFconfirmzVarExistence(
    id as long,
    varName as string)
    ' out -- Completion status code.
    ' in -- CDF identifier.
    ' in -- zVariable name.
CDFconfirmzVarExistence confirms the existence of a zVariable with a given name in a CDF. If the zVariable does not exist, an error code will be returned. No exception is thrown if the variable is not found.

The arguments to CDFconfirmrEntryExistence are defined as follows:

- **id**: The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **varName**: The zVariable name to check.

### 4.3.5.1. Example(s)

The following example checks the existence of zVariable “MY_VAR” in a CDF.

```vbscript
dim id as long
Dim status as integer

try
    status = CDFconfirmzVarExistence (id, “MY_VAR”)  
    if status <> CDF_OK then UserStatusHandler (status) 
    ...
    ...catch ex as Exception
    ...
end try
```

### 4.3.6. CDFconfirmzVarPadValueExistence

```vbscript
integer CDFconfirmzVarPadValueExistence(
    id as long,
    varNum as integer)
```

CDFconfirmzVarPadValueExistence confirms the existence of an explicitly specified pad value for the specified zVariable in a CDF. If an explicit pad value has not been specified, the informational status code NO_PADVALUE_SPECIFIED will be returned. No exception is thrown if the variable’s pad value is not defined.

The arguments to CDFconfirmzVarPadValueExistence are defined as follows:

- **id**: The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **varNum**: The zVariable number.
4.3.6.1.  Example(s)

The following example checks the existence of the pad value of variable “MY_VAR” in a CDF.

```vba
dim id as long
Dim status as integer
Dim varNum as integer

try
    varNum = CDFgetVarNum(id, "MY_VAR")
    status = CDFconfirmzVarPadValueExistence (id, varNum)
    if status <> NO_PADVALUE_SPECIFIED then
        end if
    ...
    ...
catch ex as Exception
    ...
end try
```

4.3.7  CDFcreaterVar

```vba
integer CDFcreaterVar(
    id as long,       ' out -- Completion status code.
    varName as string, ' in -- CDF identifier.
    dataType as integer, ' in -- rVariable name.
    numElements as integer, ' in -- Data type.
    recVariance as integer, ' in -- Number of elements (of the data type).
    dimVariances as integer(), ' in -- Dimension variances.
    varNum as integer)    ' out -- rVariable number.
```

CDFcreaterVar is used to create a new rVariable in a CDF. A variable (rVariable or rVariable) with the same name must not already exist in the CDF.

The arguments to CDFcreaterVar are defined as follows:

- **id**
  The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

- **varName**
  The name of the rVariable to create. This may be at most CDF_VAR_NAME_LEN256 characters. Variable names are case-sensitive.

- **dataType**
  The data type of the new rVariable. Specify one of the data types defined in Section 2.6.
The number of elements of the data type at each value. For character data types (CDF_CHAR and CDF_UCHAR), this is the number of characters in the string (each value consists of the entire string). For all other data types this must always be one (1) - multiple elements at each value are not allowed for non-character data types.

recVariance
The rVariable's record variance. Specify one of the variances defined in Section 2.10.

dimVariances
The rVariable's dimension variances. Each element of dimVariances specifies the corresponding dimension variance. For each dimension specify one of the variances defined in Section 2.10. For 0-dimensional rVariables this argument is ignored (but must be present).

varNum
The number assigned to the new rVariable. This number must be used in subsequent CDF function calls when referring to this rVariable. An existing rVariable's number may be determined with the CDFgetVarNum function.

4.3.7.1. Example(s)

The following example will create several rVariables in a 2-dimensional CDF.

dim id as long
Dim status as integer
Dim EPOCHrecVary as integer = VARY
Dim LATrecVary as integer = NOVARY
Dim LONrecVary as integer = NOVARY
Dim TMPrecVary as integer = VARY
Dim EPOCHdimVarys() as integer = {NOVARY,NOVARY}
Dim LATdimVarys() as integer = {VARY,VARY}
Dim LONdimVarys() as integer = {VARY,VARY}
Dim TMPdimVarys() as integer = {VARY,VARY}
Dim EPOCHvarNum as integer
Dim LATvarNum as integer
Dim LONvarNum as integer
Dim TMPvarNum as integer

try
status = CDFcreaterVar (id, "EPOCH", CDF_EPOCH, 1, EPOCHrecVary, _
                      EPOCHdimVarys, EPOCH varNum)
status = CDFcreaterVar (id, "LATITUDE", CDF_INT2, 1, LATrecVary, LATdimVarys, LATvarNum)
status = CDFcreaterVar (id, "LONITUDE", CDF_INT2, 1, LONrecVary, LONdimVarys, LONvarNum)
status = CDFcreaterVar (id, "TEMPERATURE", CDF_REAL4, 1, TMPrecVary, _
                      TMPdimVarys, TMPvarNum)

catch ex as Exception
...
end try

77
4.3.8 CDFcreatezVar

integer CDFcreatezVar(
  id as long,
  varName as string,
  dataType as integer,
  numElements as integer,
  numDims as integer,
  dimSizes as integer(),
  recVariance as integer,
  dimVariances as integer(),
  varNum as integer
)  

CDFcreatezVar is used to create a new zVariable in a CDF. A variable (rVariable or zVariable) with the same name must not already exist in the CDF.

The arguments to CDFcreatezVar are defined as follows:

id
The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

varName
The name of the zVariable to create. This may be at most CDF_VAR_NAME_LEN256 characters. Variable names are case-sensitive.

dataType
The data type of the new zVariable. Specify one of the data types defined in Section 2.6.

numElements
The number of elements of the data type at each value. For character data types (CDF_CHAR and CDF_UCHAR), this is the number of characters in the string (each value consists of the entire string). For all other data types this must always be one (1) - multiple elements at each value are not allowed for non-character data types.

numDims
Number of dimensions the zVariable. This may be as few as zero (0) and at most CDF_MAX_DIMS.

dimSizes
The size of each dimension. Each element of dimSizes specifies the corresponding dimension size. Each size must be greater than zero (0). For 0-dimensional zVariables this argument is ignored (but must be present).

recVariance
The zVariable's record variance. Specify one of the variances defined in Section 2.10.

dimVariances
The zVariable's dimension variances. Each element of dimVariances specifies the corresponding dimension variance. For each dimension specify one of the variances defined in Section 2.10. For 0-dimensional zVariables this argument is ignored (but must be present).

varNum
The number assigned to the new zVariable. This number must be used in subsequent CDF function calls when referring to this zVariable. An existing zVariable's number may be determined with the CDFgetVarNum function.
4.3.8.1. Example(s)

The following example will create several zVariables in a CDF. In this case EPOCH is a 0-dimensional, LAT and LON are 2-dimensional, and TMP is a 1-dimensional.

```plaintext
try
    status = CDFcreatezVar (id, "EPOCH", CDF_EPOCH, 1, 0, EPOCHdimSizes, EPOCHrecVary, EPOCHdimVarys, EPOCHvarNum)
    status = CDFcreatezVar (id, "LATITUDE", CDF_INT2, 1, 2, LATLONdimSizes, LATrecVary, LATdimVarys, LATvarNum)
    status = CDFcreatezVar (id, "LONGITUDE", CDF_INT2, 1, 2, LATLONdimSizes, LONrecVary, LONdimVarys, LONvarNum)
    status = CDFcreatezVar (id, "TEMPERATURE", CDF_REAL4, 1, 1, TMPdimSizes, TMPrecVary, TMPdimVarys, TMPvarNum)
end try
```

4.3.9 CDFdeleterVar

```plaintext
integer CDFdeleterVar(
    id as long,                ' out -- Completion status code.
    varNum as integer)         ' in -- CDF identifier.
                              ' in -- rVariable identifier.
```

CDFdeleterVar deletes the specified rVariable from a CDF.

The arguments to CDFdeleterVar are defined as follows:
id The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

varNum The rVariable number to be deleted.

4.3.9.1. Example(s)

The following example deletes the rVariable named MY_VAR in a CDF.

```
try
    varNum = CDFgetVarNum (id, "MY_VAR")
    status = CDFdeleterVar (id, varNum)
    ...
catch ex as Exception
    ...
end try
```

4.3.10 CDFdeleterVarRecords

integer CDFdeleterVarRecords(
  id as long,  ' in -- CDF identifier.
  varNum as integer,  ' in -- rVariable identifier.
  startRec as integer,  ' in -- Starting record number.
  endRec as integer)  ' in -- Ending record number.

CDFdeleterVarRecords deletes a range of data records from the specified rVariable in a CDF. If this is a variable with sparse records, the remaining records after deletion will not be renumbered.7

The arguments to CDFdeleterVarRecords are defined as follows:

id The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

varNum The identifier of the rVariable.

7 Normal variables without sparse records have contiguous physical records. Once a section of the records get deleted, the remaining ones automatically fill the gap.
4.3.10.1. Example(s)

The following example deletes 11 records (from record numbered 11 to 21) from the rVariable “MY_VAR” in a CDF. Note: The first record is numbered as 0.

```
\begin{verbatim}
  \begin{verbatim}
  Dim id as long
  Dim status as integer
  Dim varNum as integer
  Dim startRec as integer
  Dim endRec as integer
  try
    varNum = CDFgetVarNum (id, "MY_VAR")
    startRec = 10
    endRec = 20
    status = CDFdeleterVarRecords (id, varNum, startRec, endRec)
  …
  catch ex as Exception
    …
  end try
\end{verbatim}
\end{verbatim}
```

4.3.11  CDFdeleterVarRecordsRenumber

integer CDFdeleterVarRecordsRenumber(
  id as long,
  varNum as integer,
  startRec as integer,
  endRec as integer
)  \hfill{\texttt{\texttt{' out -- Completion status code.}} \texttt{\texttt{' in -- CDF identifier.}} \texttt{\texttt{' in -- rVariable identifier.}} \texttt{\texttt{' in -- Starting record number.}} \texttt{\texttt{' in -- Ending record number.}}}

CDFdeleterVarRecordsRenumber deletes a range of data records from the specified rVariable in a CDF. If this is a variable with sparse records, the remaining records after deletion will be renumbered, just like non-sparse variable’s records.

The arguments to CDFdeleterVarRecordsRenumber are defined as follows:

\begin{itemize}
  \item \texttt{id} \quad The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
\end{itemize}
varNum  The identifier of the rVariable.

startRec  The starting record number to delete.

endRec  The ending record number to delete.

4.3.11.1. Example(s)

The following example deletes 11 records (from record numbered 11 to 21) from the rVariable “MY_VAR” in a CDF. Note: The first record is numbered as 0. If the last record number is 100, then after the deletion, the record will be 89.

```
... try
  varNum = CDFgetVarNum (id, “MY_VAR”)
  startRec = 10
  endRec = 20
  status = CDFdeleterVarRecordsRenumber (id, varNum, startRec, endRec)
... catch ex as Exception
  ... end try
```

4.3.12  CDFdeletezVar

integer CDFdeletezVar(  
id as long,  
varNum as integer)  

CDFdeletezVar deletes the specified zVariable from a CDF.

The arguments to CDFdeletezVar are defined as follows:

```
  id  The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
  varNum  The zVariable number to be deleted.
```
4.3.12.1. Example(s)

The following example deletes the zVariable named MY_VAR in a CDF.

```
try
  varNum = CDFgetVarNum (id, “MY_VAR”)  
  status = CDFdeletezVar (id, varNum)  
  ...
catch ex as Exception
  ...
end try
```

4.3.13  CDFdeletezVarRecords

integer CDFdeletezVarRecords(
  id as long,  
  varNum as integer,  
  startRec as integer,  
  endRec as integer)  

CDFdeletezVarRecords deletes a range of data records from the specified zVariable in a CDF. If this is a variable with sparse records, the remaining records after deletion will not be renumbered.

The arguments to CDFdeletezVarRecords are defined as follows:

- **id**: The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **varNum**: The identifier of the zVariable.
- **startRec**: The starting record number to delete.
- **endRec**: The ending record number to delete.
### 4.3.13.1. Example(s)

The following example deletes 11 records (from record numbered 11 to 21) from the zVariable “MY_VAR” in a CDF.

*Note: The first record is numbered as 0.*

```vba
' The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
Dim id as long
Dim status as integer
Dim varNum as integer
Dim startRec as integer
Dim endRec as integer
```

```vba
try
... varNum = CDFgetVarNum(id, "MY_VAR")
... startRec = 10
... endRec = 20
... status = CDFdeletezVarRecords(id, varNum, startRec, endRec)
... catch ex as Exception
... end try
```

### 4.3.14 CDFdeletezVarRecordsRenumber

```vba
integer CDFdeletezVarRecordsRenumber(id as long, varNum as integer, startRec as integer, endRec as integer)
```

CDFdeletezVarRecordsRenumber deletes a range of data records from the specified zVariable in a CDF. If this is a variable with sparse records, the remaining records after deletion will be renumbered, just like non-sparse variable’s records.

The arguments to CDFdeletezVarRecordsRenumber are defined as follows:

- **id**
  - The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

- **varNum**
  - The identifier of the zVariable.

- **startRec**
  - The starting record number to delete.

- **endRec**
  - The ending record number to delete.
4.3.14.1. Example(s)

The following example deletes 11 records (from record numbered 11 to 21) from the zVariable “MY_VAR” in a CDF. Note: The first record is numbered as 0. If the last record number is 100, then after the deletion, the record will be 89.

```vba
try
    varNum = CDFgetVarNum (id, "MY_VAR")
    startRec = 10
    endRec = 20
    status = CDFdeletezVarRecordsRenumber (id, varNum, startRec, endRec)
    ... 
    catch ex as Exception
        ... 
    end try
```

4.3.15  CDFgetMaxWrittenRecNums

```vba
integer CDFgetMaxWrittenRecNums ( 
    id as long,  
    rVarsMaxNum as integer,  
    zVarsMaxNum as integer)  
```

CDFgetMaxWrittenRecNums returns the maximum written record number for the rVariables and zVariables in a CDF. The maximum record number for rVariables or zVariables is one less than the maximum number of records among all respective variables.

The arguments to CDFgetMaxWrittenRecNums are defined as follows:

- **id**: The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **rVarsMaxNum**: The maximum record number among all rVariables.
- **zVarsMaxNum**: The maximum record number among all zVariables.
4.3.15.1. Example(s)

The following example returns the maximum written record numbers among all rVariables and zVariables of the CDF.

```
    dim id as long ' CDF identifier.
    Dim status as integer ' Returned status code.
    Dim rVarsMaxNum as integer ' Maximum record number among all rVariables.
    Dim zVarsMaxNum as integer ' Maximum record number among all zVariables.

    try
        ....
        status = CDFgetMaxWrittenRecNums (id, rVarsMaxNum, zVarsMaxNum)
        ...
        catch ex as Exception
            ...
    end try
```

4.3.16  CDFgetNumrVars

```
    integer CDFgetNumrVars ( id as long,
        numVars as integer) ' out -- Completion status code.
        ' in -- CDF identifier.
        ' out -- Total number of rVariables.
```

CDFgetNumrVars returns the total number of rVariables in a CDF.

The arguments to CDFgetNumrVars are defined as follows:

- \textit{id} The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- numVars The number of rVariables.

4.3.16.1. Example(s)

The following example returns the total number of rVariables in a CDF.

```
    dim status as integer ' Returned status code.
    dim id as long ' CDF identifier.
    Dim numVars as integer ' Number of zVariables.
```
try
....
status = CDFgetNumrVars (id, numVars)
....
catch ex as Exception
....
end try

4.3.17 CDFgetNumzVars

integer CDFgetNumzVars (  
id as long,  
numVars as integer)  

CDFgetNumzVars returns the total number of zVariables in a CDF.

The arguments to CDFgetNumzVars are defined as follows:

id The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

numVars The number of zVariables.

4.3.17.1. Example(s)

The following example returns the total number of zVariables in a CDF.

try
....
status = CDFgetNumzVars (id, numVars)
....
catch ex as Exception
....
end try
4.3.18 CDFgetrVarAllocRecords

integer CDFgetrVarAllocRecords(
    id as long,
    varNum as integer,
    numRecs as integer)

`CDFgetrVarAllocRecords` returns the number of records allocated for the specified rVariable in a CDF. Refer to the CDF User’s Guide for a description of allocating variable records in a single-file CDF.

The arguments to `CDFgetrVarAllocRecords` are defined as follows:

- *id*: The identifier of the current CDF. This identifier must have been initialized by a call to `CDFcreate` (or `CDFcreateCDF`) or `CDFopenCDF`.
- *varNum*: The rVariable number.
- *numRecs*: The number of allocated records.

4.3.18.1. Example(s)

The following example returns the number of allocated records for rVariable “MY_VAR” in a CDF.

```plaintext
...  dim id as long
      Dim varNum as integer
      Dim numRecs as integer
      Dim status as integer
  try
    varNum = CDFgetVarNum (id, “MY_VAR”)  
    status = CDFgetrVarAllocRecords (id, varNum, numRecs)
  ...  
catch ex as Exception
  ...  
end try
```

4.3.19 CDFgetrVarBlockingFactor

integer CDFgetrVarBlockingFactor(

`CDFgetrVarBlockingFactor`
id as long, varNum as integer, bf as integer)

CDFgetVarBlockingFactor returns the blocking factor for the specified rVariable in a CDF. Refer to the CDF User’s Guide for a description of the blocking factor.

The arguments to CDFgetVarBlockingFactor are defined as follows:

- **id**: The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **varNum**: The rVariable number.
- **bf**: The blocking factor. A value of zero (0) indicates that the default blocking factor will be used.

### 4.3.19.1. Example(s)

The following example returns the blocking factor for the rVariable “MY_VAR” in a CDF.

```plaintext
... dim id as long ' CDF identifier.
Dim varNum as integer ' rVariable number.
Dim bf as integer ' The blocking factor.
Dim status as integer.
... try ....
   varNum = CDFgetVarNum (id, “MY_VAR”) status = CDFgetVarBlockingFactor (id, varNum, bf) .
   catch ex as Exception ...
end try
```

### 4.3.20 CDFgetVarCacheSize

integer CDFgetVarCacheSize(
  id as long, varNum as integer,
  numBuffers as integer)

CDFgetVarCacheSize returns the number of cache buffers being for the specified rVariable in a CDF. This operation is not applicable to a single-file CDF. Refer to the CDF User’s Guide for a description of caching scheme used by the CDF library.
The arguments to CDFgetrVarCacheSize are defined as follows:

- **id**: The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **varNum**: The rVariable number.
- **numBuffers**: The number of cache buffers.

### 4.3.20.1. Example(s)

The following example returns the number of cache buffers for rVariable “MY_VAR” in a CDF.

```vbs
''
''
dim id as long
Dim varNum as integer
Dim numBuffers as integer
dim status as integer
''
try
....
    varNum = CDFgetVarNum (id, "MY_VAR")
....
    status = CDFgetrVarCacheSize (id, varNum, numBuffers)
...
catch ex as Exception
...
end try
```

### 4.3.21 CDFgetrVarCompression

```vbs
integer CDFgetrVarCompression(  
id as long,  
varNum as integer,  
compType as integer,  
cParms as integer(),  
cPct as integer)  
```

CDFgetrVarCompression returns the compression type/parameters and compression percentage of the specified rVariable in a CDF. Refer to Section 2.11 for a description of the CDF supported compression types/parameters. The compression percentage is the result of the compressed size from all variable records divided by its original, uncompressed variable size.

The arguments to CDFgetrVarCompression are defined as follows:
The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

varNum
The rVariable number.

compType
The compression type.

cParms
The compression parameters.

cPct
The percentage of the uncompressed size of rVariable’s data values needed to store the compressed values.

4.3.21.1. Example(s)

The following example returns the compression information for rVariable “MY_VAR” in a CDF.

```vbnet
dim id as long
Dim varNum as integer
Dim compType as integer
Dim cParms(1) as integer
Dim cPct as integer

try
    varNum = CDFgetVarNum (id, “MY_VAR”) 
    status = CDFgetrVarCompression (id, varNum, compType, cParms, cPct)
    ...
    catch ex as Exception
        ...
end try
```

4.3.22 CDFgetrVarData

integer CDFgetrVarData(
    id as long,
    varNum as integer,
    recNum as integer,
    indices as integer(),
    value as TYPE)

CDFgetrVarData returns a data value from the specified indices, the location of the element, in the given record of the specified rVariable in a CDF.

The arguments to CDFgetrVarData are defined as follows:
id

The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

varNum

The rVariable number.

recNum

The record number.

indices

The dimension indices within the record.

value

The data value.

4.3.22.1. Example(s)

The following example returns two data values, the first and the fifth element, in Record 0 from rVariable “MY_VAR”, a 2-dimensional (2 by 3) CDF_DOUBLE type variable, in a row-major CDF.

```vbnet
dim id as long
Dim varNum as integer
Dim recNum as integer
Dim indices(2) as integer
Dim value1 as double, value2 as double

try
    varNum = CDFgetVarNum (id, “MY_VAR”)  
    recNum = 0
    indices(0) = 0
    indices(1) = 0
    status = CDFgetrVarData (id, varNum, recNum, indices, value1)
    indices(0) = 1
    indices(1) = 1
    object value2o
    status = CDFgetrVarData (id, varNum, recNum, indices, value2o)
    value2 = value2o
    ...
    ...
catch ex as Exception
    ...
end try
```

4.3.23 CDFgetrVarDataType

```vbnet
integer CDFgetrVarDataType(
    id as long,
    varNum as integer,
    ‘ out -- Completion status code.
    ‘ in -- CDF identifier.
    ‘ in -- Variable number.
```
CDFgetrVarDataType returns the data type of the specified rVariable in a CDF. Refer to Section 2.6 for a description of the CDF data types.

The arguments to CDFgetrVarDataType are defined as follows:

- **id**: The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **varNum**: The rVariable number.
- **dataType**: The data type.

### 4.3.23.1. Example(s)

The following example returns the data type of rVariable “MY_VAR” in a CDF.

```vba
... dim id as long ' CDF identifier.
Dim varNum as integer ' rVariable number.
Dim dataType as integer ' The data type.
try
    varNum = CDFgetVarNum (id, “MY_VAR”)
    status = CDFgetrVarDataType (id, varNum, dataType)
... catch ex as Exception ...
end try
```

### 4.3.24 CDFgetrVarDimVariances

integer CDFgetrVarDimVariances(
    id as long,
    varNum as integer,
    dimVarys as integer())

CDFgetrVarDimVariances returns the dimension variances of the specified rVariable in a CDF. For 0-dimensional rVariable, this operation is not applicable. The dimension variances are described in section 2.10.

The arguments to CDFgetrVarDimVariances are defined as follows:
id The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

varNum The rVariable number.

dimVarys The dimension variances.

### 4.3.24.1. Example(s)

The following example returns the dimension variances of the 2-dimensional rVariable “MY_VAR” in a CDF.

```plaintext
    try
    ....
    status = CDFgetrVarDimVariances (id, CDFgetVarNum (id, “MY_VAR”), dimVarys)

    catch ex as Exception
    ...
    end try
```

### 4.3.25 CDFgetrVarInfo

```plaintext
integer CDFgetrVarInfo(
    id as long,
    varNum as integer,
    dataType as integer,
    numElems as integer,
    numDims as integer,
    dimSizes as integer())
```

CDFgetrVarInfo returns the basic information about the specified rVariable in a CDF.

The arguments to CDFgetrVarInfo are defined as follows:

- **id** The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **varNum** The rVariable number.
**4.3.25.1. Example(s)**

The following example returns the basic information of rVariable “MY_VAR” in a CDF.

```plaintext
... try
  ...
  dim id as long
  Dim dataType as integer
  Dim numElems as integer
  Dim numDims as integer
  Dim dimSizes() as integer
  dim status as integer
  ...
  status = CDFgetrVarInfo (id, CDFgetVarNum (id, “MY_VAR”), dataType, numElems, numDims, dimVarys)
...
  catch ex as Exception
  ...
  end try
```

**4.3.26  CDFgetrVarMaxAllocRecNum**

```plaintext
integer CDFgetrVarMaxAllocRecNum(
  id as long,
  varNum as integer,
  maxRec as integer)

CDFgetrVarMaxAllocRecNum returns the number of records allocated for the specified rVariable in a CDF.
```

The arguments to CDFgetrVarMaxAllocRecNum are defined as follows:

- **id**  
  The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
4.3.26.1. Example(s)

The following example returns the maximum allocated record number for the rVariable “MY_VAR” in a CDF.

```lisp
... 

dim id as long
Dim maxRec as integer
Dim status as integer.

try
    ... 
    status = CDFgetrVarMaxAllocRecNum (id, CDFgetVarNum (id, “MY_VAR”), maxRec)
... 
catch ex as Exception
    ...
end try
```

4.3.27 CDFgetrVarMaxWrittenRecNum

```lisp
integer CDFgetrVarMaxWrittenRecNum ( 
    id as long, int amazingly_cool_variable, 
    varNum as integer, int variable_number, 
    maxRec as integer) 

    ' out -- Completion status code. 
    ' in -- CDF identifier. 
    ' in -- Variable number. 
    ' out -- Maximum written record number. 
```

CDFgetrVarMaxWrittenRecNum returns the maximum record number written for the specified rVariable in a CDF.

The arguments to CDFgetrVarMaxWrittenRecNum are defined as follows:

id
The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

varNum
The rVariable number.

maxRec
The maximum written record number.

4.3.27.1. Example(s)

The following example returns the maximum record number written for the rVariable “MY_VAR” in a CDF.

```lisp
... 
```
Dim id as long  ' CDF identifier.
Dim maxRec as integer  ' The maximum record number.
Dim status as integer.

try
    ...
    status = CDFgetrVarMaxWrittenRecNum(id, CDFgetVarNum(id, "MY_VAR"), maxRec)
    ...
catch ex as Exception
    ...
end try

### 4.3.28 CDFgetrVarName

integer CDFgetrVarName(id as long,
                          varNum as integer,
                          varName as string)  

CDFgetrVarName returns the name of the specified rVariable, by its number, in a CDF.

The arguments to CDFgetrVarName are defined as follows:

- **id**: The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **varNum**: The rVariable number.
- **varName**: The name of the variable.

#### 4.3.28.1. Example(s)

The following example returns the name of the rVariable whose variable number is 1.

dim id as long  ' CDF identifier.
Dim varNum as integer  ' rVariable number.
Dim varName as string  ' The name of the variable.

varNum = 1
try
    ...
    varName = CDFgetrVarName(id, varNum, varName)
    ...
end try
status = CDFgetrVarName (id, varNum, varName)
...
catch ex as Exception
...
end try

### 4.3.29 CDFgetrVarNumElements

integer CDFgetrVarNumElements(
  id as long,
  varNum as integer,
  numElems as integer)

CDFgetrVarNumElements returns the number of elements for each data value of the specified rVariable in a CDF. For character data type (CDF_CHAR and CDF_UCHAR), the number of elements is the number of characters in the string. For other data types, the number of elements will always be one (1).

The arguments to CDFgetrVarNumElements are defined as follows:

- **id**
  The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

- **varNum**
  The rVariable number.

- **numElems**
  The number of elements.

### 4.3.29.1. Example(s)

The following example returns the number of elements for the data type from rVariable “MY_VAR” in a CDF.

```
    dim id as long
    Dim numElems as integer
    Dim status as integer.
    try
        ....
        status = CDFgetrVarNumElements (id, CDFgetVarNum (id, “MY_VAR”), numElems) ...
    ...
    catch ex as Exception
        ...
    end try
```
4.3.30  CDFgetrVarNumRecsWritten

integer CDFgetrVarNumRecsWritten(
    id as long,
    varNum as integer,
    numRecs as integer)

    ' out -- Completion status code.
    ' in -- CDF identifier.
    ' in -- Variable number.
    ' out -- Number of written records.

CDFgetrVarNumRecsWritten returns the number of records written for the specified rVariable in a CDF. This number may not correspond to the maximum record written if the rVariable has sparse records.

The arguments to CDFgetrVarNumRecsWritten are defined as follows:

    id                The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
    varNum            The rVariable number.
    numRecs           The number of written records.

4.3.30.1. Example(s)

The following example returns the number of written records from rVariable “MY_VAR” in a CDF.

    ...
    ...
    dim id as long
    Dim numRecs as integer
    Dim status as integer.
    try
        ....
        status = CDFgetrVarNumRecsWritten (id, CDFgetVarNum (id, “MY_VAR”), numRecs)
    ...
    catch ex as Exception
        ...
    end try

4.3.31  CDFgetrVarPadValue

integer CDFgetrVarPadValue(
    id as long,
    varNum as integer,
    value as TYPE)

    ' out -- Completion status code.
    ' in -- CDF identifier.
    ' in -- Variable number.
    ' out -- Pad value.
    ' TYPE -- VB value/string type or object.
CDFgetrVarPadValue returns the pad value of the specified rVariable in a CDF. If a pad value has not been explicitly specified for the rVariable through CDFsetrVarPadValue, the informational status code NO_PADVALUE_SPECIFIED will be returned. Since a variable’s pad value is an optional, no exception is thrown while trying to get its value if its value is not set. It’s recommended to check the returned status after the method is called.

The arguments to CDFgetrVarPadValue are defined as follows:

- **id**: The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **varNum**: The rVariable number.
- **value**: The pad value.

### 4.3.31.1. Example(s)

The following example returns the pad value from rVariable “MY_VAR”, a CDF_INT4 type variable, in a CDF.

```vba
dim id as long
Dim padValue as integer
Dim status as integer.

try
    object padValueo
    status = CDFgetrVarPadValue (id, CDFgetVarNum (id, “MY_VAR”), padValueo)
    if status <> NO_PADVALUE_SPECIFIED then
        padValue = Ctype(padValueo, integer)
    end if
end try
```

### 4.3.32 CDFgetrVarRecordData

integer CDFgetrVarRecordData(
    id as long,
    varNum as integer,
    dim recNum as integer,
    buffer as TYPE )

- **out** -- Completion status code.
- **in** -- CDF identifier.
- **in** -- Variable number.
- **in** -- Record number.
- **out** -- Record data.
- **TYPE** -- VB value/string type (likely an array) or object.
CDFgetVarRecordData returns an entire record at a given record number for the specified rVariable in a CDF. The buffer should be large enough to hold the entire data values form the variable.

The arguments to CDFgetVarRecordData are defined as follows:

- **id**: The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **varNum**: The rVariable number.
- **recNum**: The record number.
- **buffer**: The buffer holding the entire record data.

### 4.3.32.1. Example(s)

The following example will read two full records (record numbers 2 and 5) from rVariable “MY_VAR”, a 2-dimension (2 by 3), CDF_INT4 type variable, in a CDF. The variable’s dimension variances are all VARY.

```vbnet
    try
        varNum = CDFgetVarNum (id, "MY_VAR")
        status = CDFgetrVarRecordData (id, varNum, 2, buffer1)
        dim buffer2o as object
        status = CDFgetrVarRecordData (id, varNum, 5, buffer2o)
        buffer2 = buffer2o
    ...```

### 4.3.33 CDFgetrVarRecVariance

integer CDFgetrVarRecVariance(id as long, varNum as integer, recVary as integer)

- **id**: CDF identifier.
- **varNum**: rVariable number.
- **recVary**: The data holding buffer – pre-allocation.
- **buffer2o**: The data holding buffer – API allocation.
- **status**: Completion status code.
- **varNum**: Variable number.
- **recVary**: Record variance.
CDFgetrVarRecVariance returns the record variance of the specified rVariable in a CDF. The record variances are described in Section 2.10.

The arguments to CDFgetrVarRecVariance are defined as follows:

- id: The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- varNum: The rVariable number.
- recVary: The record variance.

4.3.33.1. Example(s)

The following example returns the record variance for the rVariable “MY_VAR” in a CDF.

```
...,

    dim id as long
    Dim recVary as integer
    Dim status as integer

    try
        status = CDFgetrVarRecVariance (id, CDFgetVarNum (id, “MY_VAR”), recVary) ...
    ...
    catch ex as Exception
        ...
    end try
```

4.3.34  CDFgetrVarReservePercent

integer CDFgetrVarReservePercent(id as long, varNum as integer, percent as integer)  ' out -- Completion status code.
  ' in -- CDF identifier.
  ' in -- Variable number.
  ' out -- Reserve percentage.

CDFgetrVarReservePercent returns the compression reserve percentage being used for the specified rVariable in a CDF. This operation only applies to compressed rVariables. Refer to the CDF User’s Guide for a description of the reserve scheme used by the CDF library.

The arguments to CDFgetrVarReservePercent are defined as follows:

- id: The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- varNum: The rVariable number.
percent  The reserve percentage.

4.3.34.1. Example(s)

The following example returns the compression reserve percentage from the compressed rVariable “MY_VAR” in a CDF.

```
...  
  
  dim id as long  ` CDF identifier.
  Dim percent as integer  ` The compression reserve percentage.
  dim status as integer
  try
  ....
  status = CDFgetrVarReservePercent (id, CDFgetVarNum (id, “MY_VAR”), percent)
  ...
  ...
  catch ex as Exception
  ...
  end try
```

4.3.35  CDFgetrVarsDimSizes

```
integer CDFgetrVarsDimSizes(  ` out -- Completion status code.
  id as long,  ` in -- CDF identifier.
  dimSizes as integer())  ` out -- Dimension sizes.
```

CDFgetrVarsDimSizes returns the size of each dimension for the rVariables in a CDF. (all rVariables have the same dimensional sizes.) For 0-dimensional rVariables, this operation is not applicable.

The arguments to CDFgetrVarsDimSizes are defined as follows:

```
id  The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

  dimSizes  The dimension sizes. Each element of dimSizes receives the corresponding dimension size.
```

4.3.35.1. Example(s)

The following example returns the dimension sizes for rVariables in a CDF.

...  
  ```
4.3.36  CDFgetrVarSeqData

integer CDFgetrVarSeqData(
    id as long,
    varNum as integer,
    value as TYPE)

CDFgetrVarSeqData reads one value from the specified rVariable in a CDF at the current sequential value (position). After the read, the current sequential value is automatically incremented to the next value. An error is returned if the current sequential value is past the last record of the rVariable. Use CDFsetrVarSeqPos method to set the current sequential value (position).

The arguments to CDFgetrVarSeqData are defined as follows:

- **id**: The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **varNum**: The rVariable number from which to read data.
- **value**: The buffer to store the value.

4.3.36.1. Example(s)

The following example will read the first two data values from the beginning of record number 2 (from a 2-dimensional rVariable whose data type is CDF_INT4) in a CDF.
recNum = 2
indices(0) = 0
indices(1) = 0
try
....
status = CDFsetrVarSeqPos(id, varNum, recNum, indices)
status = CDFgetrVarSeqData(id, varNum, value1)
object value2o
status = CDFgetrVarSeqData(id, varNum, value2o)
value2 = value2o
...
... catch ex as Exception
...
end try

4.3.37  CDFgetrVarSeqPos

integer CDFgetrVarSeqPos (id, varNum, recNum, indices)  

CDFgetrVarSeqPos returns the current sequential value (position) for sequential access for the specified rVariable in a CDF. Note that a current sequential value is maintained for each rVariable individually. Use CDFsetrVarSeqPos method to set the current sequential value.

The arguments to CDFgetrVarSeqPos are defined as follows:

id  The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
varNum  The rVariable number.
recNum  The rVariable record number.
indices  The dimension indices. Each element of indices receives the corresponding dimension index. For 0-dimensional rVariable, this argument is ignored, but must be presented.

4.3.37.1. Example(s)

The following example returns the location for the current sequential value (position), the record number and indices within it, from a 2-dimensional rVariable named MY_VAR in a CDF.

. .
. . 
dim id as long  
Dim recNum as integer  
  ' CDF identifier.
  ' The record number.
Dim indices() as integer  ' The indices.
dim status as integer

try
....
  status = CDFgetrVarSeqPos (id, CDFgetVarNum (id, "MY_VAR"), recNum, indices)
...
catch ex as Exception
...
end try

4.3.38  CDFgetrVarsMaxWrittenRecNum

integer CDFgetrVarsMaxWrittenRecNum(
  id as long,  ' out -- Completion status code.
  recNum as integer)  ' in -- CDF identifier.
  ' out -- Maximum record number.

CDFgetrVarsMaxWrittenRecNum returns the maximum record number among all of the rVariables in a CDF. Note that this is not the number of written records but rather the maximum written record number (that is one less than the number of records). A value of negative one (-1) indicates that rVariables contain no records. The maximum record number for an individual rVariable may be acquired using the CDFgetrVarMaxWrittenRecNum method call.

Suppose there are three rVariables in a CDF: Var1, Var2, and Var3. If Var1 contains 15 records, Var2 contains 10 records, and Var3 contains 95 records, then the value returned from CDFgetrVarsMaxWrittenRecNum would be 95.

The arguments to CDFgetrVarsMaxWrittenRecNum are defined as follows:

id  The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

recNum  The maximum written record number.

4.3.38.1. Example(s)

The following example returns the maximum record number for all of the rVariables in a CDF.

  .
  .
  Dim id as long  ' CDF identifier.
  Dim recNum as integer  ' The maximum record number.
  Dim status as integer.
  .
  try
  ....
  status = CDFgetrVarsMaxWrittenRecNum (id, recNum)
  ...
  ...
catch ex as Exception
  ...
end try

4.3.39  CDFgetrVarsNumDims

integer CDFgetrVarsNumDims(
  id as long,
  numDims as integer)

CDFgetrVarsNumDims returns the number of dimensions (dimensionality) for the rVariables in a CDF.

The arguments to CDFgetrVarsNumDims are defined as follows:

id  The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
numDims  The number of dimensions.

4.3.39.1. Example(s)

The following example returns the number of dimensions for rVariables in a CDF.

try
  status = CDFgetrVarsNumDims (id, numDims)
... catch ex as Exception ...
end try

4.3.40  CDFgetrVarSparseRecords

integer CDFgetrVarSparseRecords(
  id as long,
  varNum as integer,
  sRecordsType as integer)

CDFgetrVarSparseRecords returns the sparse records type for the rVariable in a CDF.

The arguments to CDFgetrVarSparseRecords are defined as follows:

id  CDF identifier.
varNum  The variable number.
sRecordsType  The sparse records type.
CDFgetrVarSparseRecords returns the sparse records type of the rVariable in a CDF. Refer to Section 2.12.1 for the description of sparse records.

The arguments to CDFgetrVarSparseRecords are defined as follows:

- **id**: The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **varNum**: The variable number.
- **sRecordsType**: The sparse records type.

### 4.3.40.1. Example(s)

The following example returns the sparse records type of the rVariable “MY_VAR” in a CDF.

```vbnet
    Dim id as long ' CDF identifier.
    Dim sRecordsType as integer ' The sparse records type.
    Dim status as integer.

    try
        status = CDFgetrVarSparseRecords (id, CDFgetVarNum (id, “MY_VAR”), sRecordsType) ...
    catch ex as Exception ...
    end try
```

### 4.3.41 CDFgetVarNum

```vbnet
    integer CDFgetVarNum( ' out -- Variable number.
        id as long, ' in -- CDF identifier.
        varName as string) ' in -- Variable name.
```

CDFgetVarNum returns the variable number for the given variable name (rVariable or zVariable). If the variable is found, CDFgetVarNum returns its variable number - which will be equal to or greater than zero (0). If an error occurs (e.g., the variable does not exist in the CDF), an error code (of type int) is returned, and an exception is thrown. Error codes are less than zero (0). The returned variable number should be used in the functions of the same variable type, rVariable or zVariable. If it is an rVariable, functions dealing with rVariables should be used. Similarly, functions for zVariables should be used for zVariables.

The arguments to CDFgetVarNum are defined as follows:

---

8 Since no two variables, either rVariable or zVariable, can have the same name, this function now returns the variable number for the given rVariable or zVariable name (if the variable name exists in a CDF).
id        The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

varName   The name of the variable to search. This may be at most CDF_VAR_NAME_LEN256 characters. Variable names are case-sensitive.

CDFgetVarNum may be used as an embedded function call where an rVariable or zVariable number is needed.

4.3.41.1. Example(s)

In the following example CDFgetVarNum is used as an embedded function call when inquiring about a zVariable.

```
try
  ...
  dim id as long
  Dim status as integer
  Dim varName as string
  Dim dataType as integer
  Dim numElements as integer
  Dim numDims as integer
  Dim dimSizes() as integer
  Dim dimVariances() as integer
  ...
  try
    ....
    status = CDFinquirezVar (id, CDFgetVarNum (id,"LATITUDE"), varName, dataType, _
                              numElements, numDims, dimSizes , recVariance, dimVariances)
    ...
    ...
  catch ex as Exception
    ...
  end try
```

In this example the zVariable named LATITUDE was inquired. Note that if LATITUDE did not exist in the CDF, the call to CDFgetVarNum would have returned an error code. Passing that error code to CDFinquirezVar as a zVariable number would have resulted in CDFinquirezVar also returning an error code. Also note that the name written into varName is already known (LATITUDE). In some cases the zVariable names will be unknown - CDFinquirezVar would be used to determine them. CDFinquirezVar is described in Section 4.3.66.

4.3.42 CDFgetzVarAllocRecords

```
integer CDFgetzVarAllocRecords( id as long,
                          varNum as integer,
                          numRecs as integer)
```

' out -- Completion status code.
' in -- CDF identifier.
' in -- Variable number.
' out -- Allocated number of records.
CDFgetzVarAllocRecords returns the number of records allocated for the specified zVariable in a CDF. Refer to the CDF User’s Guide for a description of allocating variable records in a single-file CDF.

The arguments to CDFgetzVarAllocRecords are defined as follows:

- **id**: The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **varNum**: The zVariable number.
- **numRecs**: The number of allocated records.

### Example(s)

The following example returns the number of allocated records for zVariable “MY_VAR” in a CDF.

```vba
... try .... varNum = CDFgetVarNum (id, “MY_VAR”) status = CDFgetzVarAllocRecords (id, varNum, numRecs) ...
... catch ex as Exception ...
end try
```

### CDFgetzVarBlockingFactor

integer CDFgetzVarBlockingFactor(id as long, varNum as integer, bf as integer)

CDFgetzVarBlockingFactor returns the blocking factor for the specified zVariable in a CDF. Refer to the CDF User’s Guide for a description of the blocking factor.

The arguments to CDFgetzVarBlockingFactor are defined as follows:

- **id**: The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
varNum The zVariable number.

bf The blocking factor. A value of zero (0) indicates that the default blocking factor will be used.

### 4.3.43.1. Example(s)

The following example returns the blocking factor for the zVariable “MY_VAR” in a CDF.

```vba
dim id as long
Dim varNum as integer
Dim bf as integer
dim status as integer
try
....
  varNum = CDFgetVarNum (id, “MY_VAR”) status = CDFgetzVarBlockingFactor (id, varNum, bf) .
catch ex as Exception
...
catch try
```

### 4.3.44 CDFgetzVarCacheSize

```vba
type CDFgetzVarCacheSize( id as long, varNum as integer, numBuffers as integer)
```

CDFgetzVarCacheSize returns the number of cache buffers being for the specified zVariable in a CDF. This operation is not applicable to a single-file CDF. Refer to the CDF User’s Guide for a description of caching scheme used by the CDF library.

The arguments to CDFgetzVarCacheSize are defined as follows:

- **id** The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **varNum** The zVariable number.
- **numBuffers** The number of cache buffers.
4.3.44.1. Example(s)

The following example returns the number of cache buffers for zVariable “MY_VAR” in a CDF.

```
try
    varNum = CDFgetVarNum (id, “MY_VAR”)
    status = CDFgetzVarCacheSize (id, varNum, numBuffers)
... catch ex as Exception
... end try
```

4.3.45  CDFgetzVarCompression

integer CDFgetzVarCompression(
    id as long,                        ' out -- Completion status code.
    varNum as integer,                 ' in -- CDF identifier.
    compType as integer,               ' in -- Variable number.
    cParms as integer(),              ' out -- Compression type.
    cPct as integer)                   ' out -- Compression parameters.
        ' out -- Compression percentage.

CDFgetzVarCompression returns the compression type/parameters and compression percentage of the specified zVariable in a CDF. Refer to Section 2.11 for a description of the CDF supported compression types/parameters. The compression percentage is the result of the compressed size from all variable records divided by its original, uncompressed variable size.

The arguments to CDFgetzVarCompression are defined as follows:

- `id`: The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- `varNum`: The zVariable number.
- `compType`: The compression type.
- `cParms`: The compression parameters.
- `cPct`: The percentage of the uncompressed size of zVariable’s data values needed to store the compressed values.
### 4.3.45.1. Example(s)

The following example returns the compression information for zVariable “MY_VAR” in a CDF.

```vba
... 
... 
dim id as long
Dim varNum as integer
Dim compType as integer
Dim cParms() as integer
Dim cPct as integer
Dim status as integer.

try
... 
varNum = CDFgetVarNum (id, “MY_VAR”)
status = CDFgetzVarCompression (id, varNum, compType, cParms, cPct)
... 
catch ex as Exception
... 
end try
```

### 4.3.46  CDFgetzVarData

```vba
integer CDFgetzVarData(
  id as long,
  varNum as integer,
  dim recNum as integer,
  indices as integer(),
  value as TYPE)
```

CDFgetzVarData returns a data value from the specified indices, the location of the element, in the given record of the specified zVariable in a CDF.

The arguments to CDFgetzVarData are defined as follows:

- **id**  
  The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

- **varNum**  
  The zVariable number.

- **recNum**  
  The record number.

- **indices**  
  The dimension indices within the record.

- **value**  
  The data value.
4.3.46.1. Example(s)

The following example returns two data values, the first and the fifth element, in Record 0 from zVariable “MY_VAR”, a 2-dimensional (2 by 3) CDF_DOUBLE type variable, in a row-major CDF.

```plaintext
...

    dim id as long
    Dim varNum as integer
    Dim recNum as integer
    Dim indices(2) as integer
    Dim value1 as double, value2 as double

    try
      varNum = CDFgetVarNum (id, “MY_VAR”)  ' CDF identifier.
      recNum = 0  ' zVariable number.
      indices(0) = 0  ' The record number.
      indices(1) = 0  ' The dimension indices.
      status = CDFgetzVarData (id, varNum, recNum, indices, value1)  ' The data values.
      indices(0) = 1
      indices(1) = 1
      object value2o
      status = CDFgetzVarData (id, varNum, recNum, indices, value2o)
      value2 = value2o
    ...
    catch ex as Exception
      ...
    end try

```

4.3.47  CDFgetzVarDataType

```plaintext
integer CDFgetzVarDataType(
    id as long,
    varNum as integer,
    dataType as integer)  '
      out -- Completion status code.
      in -- CDF identifier.
      in -- Variable number.
      out -- Data type.
```

CDFgetzVarDataType returns the data type of the specified zVariable in a CDF. Refer to Section 2.6 for a description of the CDF data types.

The arguments to CDFgetzVarDataType are defined as follows:

```plaintext
    id
      The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
```
varNum | The zVariable number.
dataType | The data type.

4.3.47.1. Example(s)

The following example returns the data type of zVariable “MY_VAR” in a CDF.

```vbnet
dim id as long
Dim varNum as integer
Dim dataType as integer
Dim status as integer.

try
    varNum = CDFgetVarNum (id, “MY_VAR”)
    status = CDFgetzVarDataType (id, varNum, dataType)
...catch ex as Exception
    ...
end try
```

4.3.48 CDFgetzVarDimSizes

```vbnet
integer CDFgetzVarDimSizes( id as long,
varNum as integer,
dimSizes as integer)
```

CDFgetzVarDimSizes returns the size of each dimension for the specified zVariable in a CDF. For 0-dimensional zVariables, this operation is not applicable.

The arguments to CDFgetzVarDimSizes are defined as follows:

- `id` | The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- `varNum` | The zVariable number
- `dimSizes` | The dimension sizes. Each element of `dimSizes` receives the corresponding dimension size.
4.3.48.1. Example(s)

The following example returns the dimension sizes for zVariable “MY_VAR” in a CDF.

```plaintext
dim id as long  ' CDF identifier.
dim dimSizes() as integer  ' Dimensional sizes.
Dim status as integer

.try
    status = CDFgetzVarDimSizes (id, CDFgetVarNum (id, “MY_VAR”), dimSizes)
... catch ex as Exception
... end try
```

4.3.49  CDFgetzVarDimVariances

```plaintext
integer CDFgetzVarDimVariances(
    id as long,  ' out -- Completion status code.
    varNum as integer,  ' in -- CDF identifier.
    dimVarys as integer())  ' in -- Variable number.
    ' out -- Dimension variances.
```

CDFgetzVarDimVariances returns the dimension variances of the specified zVariable in a CDF. For 0-dimensional zVariable, this operation is not applicable. The dimension variances are described in section 2.10.

The arguments to CDFgetzVarDimVariances are defined as follows:

- **id**
  - The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

- **varNum**
  - The zVariable number.

- **dimVarys**
  - The dimension variances.

4.3.49.1. Example(s)

The following example returns the dimension variances of the 2-dimensional zVariable “MY_VAR” in a CDF.

```plaintext
dim id as long  ' CDF identifier.
Dim dimVarys() as integer  ' The dimension variances.
```
Dim status as integer.
.
try
 ....

    status = CDFgetzVarDimVariances (id, CDFgetVarNum (id, “MY_VAR”), dimVarys)

...
catch ex as Exception
...
end try

4.3.50  **CDFgetzVarInfo**

integer CDFgetzVarInfo(
   id as long,
   varNum as integer,
   dataType as integer,
   numElems as integer,
   numDims as integer,
   dimSizes as integer())

CDFgetzVarInfo returns the basic information about the specified zVariable in a CDF.

The arguments to CDFgetzVarInfo are defined as follows:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
</table>
| id         | The identifier of the current CDF. This identifier must have been initialized by a call to
            | CDFcreate (or CDFcreateCDF) or CDFopenCDF. |
| varNum     | The zVariable number. |
| dataType   | The data type of the variable. |
| numElems   | The number of elements for the data type of the variable. |
| numDims    | The number of dimensions. |
| dimSizes   | The dimension sizes. |

4.3.50.1. **Example(s)**

The following example returns the basic information of zVariable “MY_VAR” in a CDF.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>CDF identifier.</td>
</tr>
<tr>
<td>varNum</td>
<td>The data type.</td>
</tr>
</tbody>
</table>
Dim numElems as integer
Dim numDims as integer
Dim dimSizes() as integer
Dim status as integer.
.
try
    ....

        status = CDFgetzVarInfo (id, CDFgetVarNum (id, “MY_VAR”), dataType, numElems, _
        numDims, dimVarys)

    ...
catch ex as Exception
    ...
end try try

4.3.51  CDFgetzVarMaxAllocRecNum

type CDFgetzVarMaxAllocRecNum(  
id as long,  
varNum as integer,  
maxRec as integer)

CDFgetzVarMaxAllocRecNum returns the number of records allocated for the specified zVariable in a CDF.

The arguments to CDFgetzVarMaxAllocRecNum are defined as follows:

id  The identifier of the current CDF.  This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

varNum  The zVariable number.

maxRec  The number of records allocated.

4.3.51.1. Example(s)

The following example returns the maximum allocated record number for the zVariable “MY_VAR” in a CDF.

    ...
    ...
    dim id as long
    Dim maxRec as integer
    dim status as integer
    .
    try
        ...
        status = CDFgetzVarMaxAllocRecNum (id, CDFgetVarNum (id, “MY_VAR”), maxRec)
... catch ex as Exception ...
end try

4.3.52  CDFgetzVarMaxWrittenRecNum

integer CDFgetzVarMaxWrittenRecNum (id as long, varNum as integer, maxRec as integer) " out -- Completion status code.
             " in -- CDF identifier.
             " in -- Variable number.
             " out -- Maximum written record number.

CDFgetzVarMaxWrittenRecNum returns the maximum record number written for the specified zVariable in a CDF.

The arguments to CDFgetzVarMaxWrittenRecNum are defined as follows:

id         The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

varNum     The zVariable number.

maxRec     The maximum written record number.

4.3.52.1. Example(s)

The following example returns the maximum record number written for the zVariable “MY_VAR” in a CDF.

... dim id as long Dim maxRec as integer Dim status as integer ... try ..... status = CDFgetzVarMaxWrittenRecNum (id, CDFgetVarNum (id, “MY_VAR”), maxRec) ...
catch ex as Exception ...
end try
4.3.53  CDFgetzVarName

integer CDFgetzVarName(
  id as long,
  varNum as integer,
  varName as string)

‘ out -- Completion status code.
‘ in -- CDF identifier.
‘ in -- Variable number.
‘ out -- Variable name.

CDFgetzVarName returns the name of the specified zVariable, by its number, in a CDF.

The arguments to CDFgetzVarName are defined as follows:

   id         The identifier of the current CDF.  This identifier must have been initialized by a call to
              CDFcreate (or CDFcreateCDF) or CDFopenCDF.

   varNum     The zVariable number.

   varName    The name of the variable.

4.3.53.1. Example(s)

The following example returns the name of the zVariable whose variable number is 1.

   dim id as long
   Dim varNum as integer
   Dim varName as string
   Dim status as integer.

   varNum = 1
   try
      status = CDFgetzVarName (id, varNum, varName)
   catch ex as Exception
      ...
   end try

4.3.54  CDFgetzVarNumDims

integer CDFgetzVarNumDims(
  id as long,
  varNum as integer,
  numDims as integer)

‘ out -- Completion status code.
‘ in -- CDF identifier.
‘ in -- Variable number.
‘ out -- Number of dimensions.
CDFgetzVarNumDims returns the number of dimensions (dimensionality) for the specified zVariable in a CDF.

The arguments to CDFgetzVarNumDims are defined as follows:

- **id**: The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **varNum**: The zVariable number
- **numDims**: The number of dimensions.

### 4.3.54.1. Example(s)

The following example returns the number of dimensions for zVariable “MY_VAR” in a CDF.

```plaintext
try
    status = CDFgetzVarNumDims (id, CDFgetVarNum (id, "MY_VAR"), numDims)
...catch ex as Exception
...
end try
```

### 4.3.55   CDFgetzVarNumElements

integer CDFgetzVarNumElements(
    id as long,
    varNum as integer,
    numElems as integer)  

CDFgetzVarNumElements returns the number of elements for each data value of the specified zVariable in a CDF. For character data type (CDF_CHAR and CDF_UCHAR), the number of elements is the number of characters in the string. For other data types, the number of elements will always be one (1).

The arguments to CDFgetzVarNumElements are defined as follows:

- **id**: The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **varNum**: The zVariable number.
numElems  The number of elements.

4.3.55.1. Example(s)

The following example returns the number of elements for the data type from zVariable “MY_VAR” in a CDF.

```
        .
        .
        .
        .
        dim id as long
        dim numElems as integer
        Dim status as integer.
        .
        try
        .
        ....
        status = CDFgetzVarNumElements (id, CDFgetVarNum (id, “MY_VAR”), numElems) …
        ...
        catch ex as Exception
        ...
        end try
```

4.3.56  CDFgetzVarNumRecsWritten

integer CDFgetzVarNumRecsWritten(  
id as long,  
varNum as integer,  
numRecs as integer)  

‘ out -- Completion status code.  
‘ in -- CDF identifier.  
‘ in -- Variable number.  
‘ out -- Number of written records.

CDFgetzVarNumRecsWritten returns the number of records written for the specified zVariable in a CDF. This number may not correspond to the maximum record written if the zVariable has sparse records.

The arguments to CDFgetzVarNumRecsWritten are defined as follows:

- `id`: The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- `varNum`: The zVariable number.
- `numRecs`: The number of written records.

4.3.56.1. Example(s)

The following example returns the number of written records from zVariable “MY_VAR” in a CDF.
Mandatory CDFgetzVarNumRecsWritten (id, CDFgetVarNum (id, “MY_VAR”), numRecs)

catch ex as Exception
... end try

4.3.57  CDFgetzVarPadValue

integer CDFgetzVarPadValue(
  id as long,
  varNum as integer,
  value as TYPE)

CDFgetzVarPadValue returns the pad value of the specified zVariable in a CDF. If a pad value has not been explicitly specified for the zVariable through CDFsetzVarPadValue, the informational status code NO_PADVALUE_SPECIFIED will be returned. Since a variable’s pad value is an optional, no exception is thrown while trying to get its value if its value is not set. It’s recommended to check the returned status after the method is called.

The arguments to CDFgetzVarPadValue are defined as follows:

  id  The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

  varNum  The zVariable number.

  value  The pad value.

4.3.57.1. Example(s)

The following example returns the pad value from zVariable “MY_VAR”, a CDF_INT4 type variable, in a CDF.

dim id as long  ' CDF identifier.
Dim padValue as integer  ' CDF identifier.
Dim status as integer  ' The pad value.
try
....
dim padValueo as object
status = CDFgetzVarPadValue (id, CDFgetVarNum (id, “MY_VAR”), padValueo)
if status <> NO_PADVALUE_SPECIFIED then
    padValue = CType(padValueo, integer)
end if
....
catch ex as Exception
...
end try

4.3.58  CDFgetzVarRecordData

integer CDFgetzVarRecordData(
    id as long,
    varNum as integer,
    recNum as integer,
    buffer as TYPE)

CDFgetzVarRecordData returns an entire record at a given record number for the specified zVariable in a CDF. The buffer should be large enough to hold the entire data values form the variable.

The arguments to CDFgetzVarRecordData are defined as follows:

id  The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
varNum  The zVariable number.
recNum  The record number.
buffer  The buffer holding the entire record data.

4.3.58.1.  Example(s)

The following example will read two full records (record numbers 2 and 5) from zVariable ‘MY_VAR’, a 2-dimension (2 by 3), CDF_INT4 type variable, in a CDF. The variable’s dimension variances are all VARY.

   ...
   ...
dim id as long  ` CDF identifier.
Dim varNum as integer  ` zVariable number.
Dim buffer1(2,3) as integer ' The data holding buffer – pre-allocation.
Dim buffer2 as object ' The data holding buffer – API allocation.
Dim status as integer.

try
....
varNum = CDFgetVarNum (id, “MY_VAR”)
status = CDFgetzVarRecordData (id, varNum, 2, buffer1)
status = CDFgetzVarRecordData (id, varNum, 5, buffer2)
...
catch ex as Exception
...
end try

### 4.3.59 CDFgetzVarRecVariance

integer CDFgetzVarRecVariance(
   id as long,
   varNum as integer,
   recVary as integer)

CDFgetzVarRecVariance returns the record variance of the specified zVariable in a CDF. The record variances are described in Section 2.10.

The arguments to CDFgetzVarRecVariance are defined as follows:

- **id**: The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **varNum**: The zVariable number.
- **recVary**: The record variance.

#### 4.3.59.1. Example(s)

The following example returns the record variance for the zVariable “MY_VAR” in a CDF.

```vba
dim id as long ' CDF identifier.
Dim recVary as integer ' The record variance.
dim status as integer

try
....
status = CDFgetzVarRecVariance (id, CDFgetVarNum (id, “MY_VAR”), recVary) ...
```
... catch ex as Exception
...
end try

4.3.60  CDFgetzVarReservePercent

integer CDFgetzVarReservePercent(
  id as long,
  varNum as integer,
  percent as integer)
  '  out -- Completion status code.
  '  in -- CDF identifier.
  '  in -- Variable number.
  '  out -- Reserve percentage.

CDFgetzVarReservePercent returns the compression reserve percentage being used for the specified zVariable in a CDF. This operation only applies to compressed zVariables. Refer to the CDF User’s Guide for a description of the reserve scheme used by the CDF library.

The arguments to CDFgetzVarReservePercent are defined as follows:

- **id**
  The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

- **varNum**
  The zVariable number.

- **percent**
  The reserve percentage.

4.3.60.1. Example(s)

The following example returns the compression reserve percentage from the compressed zVariable “MY_VAR” in a CDF.

```vba
... try
   status = CDFgetzVarReservePercent (id, CDFgetVarNum (id, “MY_VAR”), percent)
... catch ex as Exception
... end try
```
4.3.61 CDFgetzVarSeqData

`integer CDFgetzVarSeqData( id as long, varNum as integer, value as TYPE)`

CDFgetzVarSeqData reads one value from the specified zVariable in a CDF at the current sequential value (position). After the read, the current sequential value is automatically incremented to the next value. An error is returned if the current sequential value is past the last record of the zVariable. Use CDFsetzVarSeqPos method to set the current sequential value (position).

The arguments to CDFgetzVarSeqData are defined as follows:

- `id` The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- `varNum` The zVariable number from which to read data.
- `value` The buffer to store the value.

4.3.61.1. Example(s)

The following example will read the first two data values from the beginning of record number 2 (from a 2-dimensional zVariable whose data type is CDF_INT4) in a CDF.

```
dim id as long  ' CDF identifier.
dim varNum as integer  ' The variable number from which to read data
dim value1 as integer, value2 as integer  ' The data value.
dim indices(2) as integer  ' The indices in a record.
dim recNum as integer  ' The record number.

recNum = 2
indices(0) = 0
indices(1) = 0
try
    status = CDFsetzVarSeqPos (id, varNum, recNum, indices)
    status = CDFgetzVarSeqData (id, varNum, value1)
    dim value2o as object
    status = CDFgetzVarSeqData (id, varNum, value2o)
    value2 = value2o
    ...
    catch ex as Exception
        ...
    end try
```
4.3.62 CDFgetzVarSeqPos

```c
integer CDFgetzVarSeqPos( 
    id as long, 
    varNum as integer, 
    recNum as integer, 
    indices as integer())
```

CDFgetzVarSeqPos returns the current sequential value (position) for sequential access for the specified zVariable in a CDF. Note that a current sequential value is maintained for each zVariable individually. Use CDFsetzVarSeqPos method to set the current sequential value.

The arguments to CDFgetzVarSeqPos are defined as follows:

- **id** The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **varNum** The zVariable number.
- **recNum** The zVariable record number.
- **indices** The dimension indices. Each element of indices receives the corresponding dimension index. For 0-dimensional zVariable, this argument is ignored, but must be presented.

4.3.62.1. Example(s)

The following example returns the location for the current sequential value (position), the record number and indices within it, from a 2-dimensional zVariable named MY_VAR in a CDF.

```c
Dim id as long
Dim recNum as integer
Dim indices() as integer
Dim status as integer.

try
    status = CDFgetzVarSeqPos (id, CDFgetVarNum (id, "MY_VAR"), recNum, indices)
    ...catch ex as Exception
    ...end try
```

4.3.63 CDFgetzVarsMaxWrittenRecNum
integer CDFgetzVarsMaxWrittenRecNum(  
id as long,  
recNum as integer)  
  ' out -- Completion status code.  
  ' in -- CDF identifier.  
  ' out -- Maximum record number.

CDFgetzVarsMaxWrittenRecNum returns the maximum record number among all of the zVariables in a CDF. Note that this is not the number of written records but rather the maximum written record number (that is one less than the number of records). A value of negative one (-1) indicates that zVariables contain no records. The maximum record number for an individual zVariable may be acquired using the CDFgetzVarMaxWrittenRecNum method call.

Suppose there are three zVariables in a CDF: Var1, Var2, and Var3. If Var1 contains 15 records, Var2 contains 10 records, and Var3 contains 95 records, then the value returned from CDFgetzVarsMaxWrittenRecNum would be 95.

The arguments to CDFgetzVarsMaxWrittenRecNum are defined as follows:

id                  The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
recNum              The maximum written record number.

4.3.63.1. Example(s)

The following example returns the maximum record number for all of the zVariables in a CDF.

```
  ...
  dim id as long          ' CDF identifier.
  Dim recNum as integer   ' The maximum record number.
  dim status as integer
  try
    ...
    status = CDFgetzVarsMaxWrittenRecNum (id, recNum)
    ...
  catch ex as Exception
    ...
  end try
```

4.3.64   CDFgetzVarSparseRecords

integer CDFgetzVarSparseRecords(  
id as long,  
varNum as integer,  
sRecordsType as integer)  
  ' out -- Completion status code.  
  ' in -- CDF identifier.  
  ' in -- The variable number.  
  ' out -- The sparse records type.

CDFgetzVarSparseRecords returns the sparse records type of the zVariable in a CDF. Refer to Section 2.12.1 for the description of sparse records.
The arguments to CDFgetzVarSparseRecords are defined as follows:

- **id**: The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **varNum**: The variable number.
- **sRecordsType**: The sparse records type.

### 4.3.64.1. Example(s)

The following example returns the sparse records type of the zVariable “MY_VAR” in a CDF.

```vba
Dim id as long
Dim sRecordsType as integer

try
    status = CDFgetzVarSparseRecords (id, CDFgetVarNum (id, "MY_VAR"), sRecordsType) 
...catch ex as Exception
    ...
end try
```

### 4.3.65  CDFhyperGetrVarData

The **CDFhyperGetrVarData** function is used to read one or more values for the specified rVariable. It is important to know the variable majority of the CDF before using this method because the values placed into the data buffer will be in that majority. CDFinquireCDF can be used to determine the default variable majority of a CDF distribution. The Concepts chapter in the CDF User's Guide describes the variable majorities.
The record number starts at 0, not 1. For example, if you want to read the first 5 records, the starting record number (recStart), the number of records to read (recCount), and the record interval (recInterval) should be 0, 5, and 1, respectively. **Note:** you need to provide dummy arrays, with at least one (1) element, for indices, counts and intervals for scalar variables.

The arguments to CDFhyperGetrVarData are defined as follows:

- **id** The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **varNum** The rVariable number from which to read data. This number may be determined with a call to CDFgetVarNum.
- **recStart** The record number at which to start reading.
- **recCount** The number of records to read.
- **recInterval** The reading interval between records (e.g., an interval of 2 means read every other record).
- **indices** The dimension indices (within each record) at which to start reading. Each element of indices specifies the corresponding dimension index. For 0-dimensional rVariable, this argument is ignored (but must be present).
- **counts** The number of values along each dimension to read. Each element of counts specifies the corresponding dimension count. For 0-dimensional rVariable, this argument is ignored (but must be present).
- **intervals** For each dimension, the dimension interval between reading (e.g., an interval of 2 means read every other value). Each element of intervals specifies the corresponding dimension interval. For 0-dimensional rVariable, this argument is ignored (but must be present).
- **buffer** The data holding buffer for the read values. The majority of the values in this buffer will be the same as that of the CDF. This buffer must be large to hold the values. CDFinquirerVar can be used to determine the rVariable's data type and number of elements (of that data type) at each value. If a dimensional array of strings is expected, then use object type.

### 4.3.65.1. Example(s)

The following example will read 3 records of data, starting at record number 13 (14th record), from a rVariable named Temperature. The variable is a 3-dimensional array with sizes (180,91,10) and the CDF’s variable majority is ROW_MAJOR. The record variance is VARY, the dimension variances are (VARY,VARY,VARY), and the data type is CDF_REAL4. This example is similar to the CDFgetrVarData example except that it uses a single call to CDFhyperGetrVarData (rather than numerous calls to CDFgetrVarData).

```plaintext
  .
  .
dim id as long
Dim status as integer
Dim tmp(,,) as single
Dim varN as integer
Dim recStart as integer = 13
Dim recCount as integer = 3
Dim recInterval as integer = 1

  ' CDF identifier.
  ' Returned status code.
  ' Temperature values.
  ' rVariable number.
  ' Start record number.
  ' Number of records to read
  ' Record interval – read every record
```
Dim indices() as integer = {0,0,0}    ' Dimension indices.
Dim counts() as integer = {180,91,10} ' Dimension counts.
Dim intervals() as integer = {1,1,1}   ' Dimension intervals – read all

try
    status = CDFhyperGetrVarData (id, varN, recStart, recCount, recInterval, indices, counts, intervals, tmp)
    ...
    ...
catch ex as Exception
    ... end try

Note that if the CDF's variable majority had been COLUMN_MAJOR, the tmp array would have been declared float tmp(10,91,180,3) for proper indexing.

4.3.66  CDFhyperGetzVarData

integer CDFhyperGetzVarData(id as long, varNum as integer, recStart as integer, recCount as integer, recInterval as integer, indices as integer(), counts as integer(), intervals as integer(), buffer as TYPE)    ' out -- Completion status code.
    ' in -- CDF identifier.
    ' in -- zVariable number.
    ' in -- Starting record number.
    ' in -- Number of records.
    ' in -- Reading interval between records.
    ' in -- Dimension indices of starting value.
    ' in -- Number of values along each dimension.
    ' in -- Reading intervals along each dimension.
    ' out -- Buffer of values.
    ' TYPE -- VB value/string type (likely an array)
    ' or object.

CDFhyperGetzVarData is used to read one or more values for the specified zVariable. It is important to know the variable majority of the CDF before using this method because the values placed into the data buffer will be in that majority. CDFinquireCDF can be used to determine the default variable majority of a CDF distribution. The Concepts chapter in the CDF User's Guide describes the variable majorities.

The record number starts at 0, not 1. For example, if you want to read the first 5 records, the starting record number (recStart), the number of records to read (recCount), and the record interval (recInterval) should be 0, 5, and 1, respectively. Note: you need to provide dummy arrays, with at least one (1) element, for indices, counts and intervals for scalar variables.

The arguments to CDFhyperGetzVarData are defined as follows:

id        The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
varNum    The zVariable number from which to read data. This number may be determined with a call to CDFgetVarNum.
recStart  The record number at which to start reading.
recCount  The number of records to read.

recInterval  The reading interval between records (e.g., an interval of 2 means read every other record).

indices  The dimension indices (within each record) at which to start reading. Each element of indices specifies the corresponding dimension index. For 0-dimensional zVariable, this argument is ignored (but must be present).

counts  The number of values along each dimension to read. Each element of counts specifies the corresponding dimension count. For 0-dimensional zVariable, this argument is ignored (but must be present).

intervals  For each dimension, the dimension interval between reading (e.g., an interval of 2 means read every other value). Each element of intervals specifies the corresponding dimension interval. For 0-dimensional zVariable, this argument is ignored (but must be present).

buffer  The data holding buffer for the read values. The majority of the values in this buffer will be the same as that of the CDF. This buffer must be large to hold the values. CDFinquirezVar can be used to determine the zVariable's data type and number of elements (of that data type) at each value. If a dimensional array of strings is expected, then use object type.

4.3.66.1. Example(s)

The following example will read 3 records of data, starting at record number 13 (14th record), from a zVariable named Temperature. The variable is a 3-dimensional array with sizes (180,91,10) and the CDF's variable majority is ROW_MAJOR. The record variance is VARY, the dimension variances are \{VARY,VARY,VARY\}, and the data type is CDF_REAL4. This example is similar to the CDFgetzVarData example except that it uses a single call to CDFhyperGetzVarData (rather than numerous calls to CDFgetzVarData).

```
dim id as long
Dim status as integer
Dim tmp(,,,) as single
Dim varN as integer
Dim recStart as integer = 13
Dim recCount as integer = 3
Dim recInterval as integer = 1
Dim indices() as integer = {0,0,0}
Dim counts() as integer = {180,91,10}
Dim intervals() as integer = {1,1,1}

try
  varN  = CDFgetVarNum  (id,  "Temperature")
  status = CDFhyperGetzVarData (id, varN, recStart, recCount, recInterval, indices, counts, intervals, _
    tmp)
  ...
catch ex as Exception
  ...
```
end try

Note that if the CDF's variable majority had been COLUMN_MAJOR, the tmp array would have been declared float
tmp(10,91,180,3) for proper indexing.

### 4.3.67 CDFhyperPutrVarData

integer CDFhyperPutrVarData(
| id as long, | ' out -- Completion status code. |
| varNum as integer, | ' in -- CDF identifier. |
| recStart as integer, | ' in -- rVariable number. |
| recCount as integer, | ' in -- Starting record number. |
| recInterval as integer, | ' in -- Number of records. |
| indices as integer(), | ' in -- Writing interval between records. |
| counts as integer(), | ' in -- Dimension indices of starting value. |
| intervals as integer(), | ' in -- Number of values along each dimension. |
| buffer as TYPE) | ' in -- Writing intervals along each dimension. |
| ' in -- Buffer of values. |
| ' TYPE -- VB value/string type (likely an array) |

CDFhyperPutrVarData is used to write one or more values from the data holding buffer to the specified rVariable. It is important to know the variable majority of the CDF before using this method because the values in the data buffer will be written using that majority. CDFinquireCDF can be used to determine the default variable majority of a CDF distribution. The Concepts chapter in the CDF User's Guide describes the variable majorities.

The record number starts at 0, not 1. For example, if you want to write 2 records (10th and 11th record), the starting record number (recStart), the number of records to write (recCount), and the record interval (recInterval) should be 9, 2, and 1, respectively. **Note:** you need to provide dummy arrays, with at least one (1) element, for indices, counts and intervals for scalar variables.

The arguments to CDFhyperPutrVarData are defined as follows:

- **id** The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **varNum** The rVariable number to which write data. This number may be determined with a call to CDFgetVarNum.
- **recStart** The record number at which to start writing.
- **recCount** The number of records to write.
- **recInterval** The interval between records for writing (e.g., an interval of 2 means write every other record).
- **indices** The indices (within each record) at which to start writing. Each element of indices specifies the corresponding dimension index. For 0-dimensional rVariable this argument is ignored (but must be present).
- **counts** The number of values along each dimension to write. Each element of counts specifies the corresponding dimension count. For 0-dimensional rVariable this argument is ignored (but must be present).
intervals  For each dimension, the interval between values for writing (e.g., an interval of 2 means write every other value). Each element of intervals specifies the corresponding dimension interval. For 0-dimensional rVariable this argument is ignored (but must be present).

buffer  The data holding buffer of values to write. The majority of the values in this buffer must be the same as that of the CDF. The values starting at memory address buffer are written to the CDF.

4.3.67.1. Example(s)

The following example writes 2 records to a rVariable named LATITUDE that is a 1-dimensional array with dimension sizes (181). The dimension variances are \{VARY\}, and the data type is CDF_INT2. This example is similar to the CDFputrVarData example except that it uses a single call to CDFhyperPutrVarData rather than numerous calls to CDFputrVarData.

```vbnet
dim id as long
Dim status as integer
Dim i as integer, j as integer
Dim lats(2,181) as short
Dim varN as integer
Dim recStart as integer = 0
Dim recCount as integer = 2
Dim recInterval as integer = 1
Dim indices() as integer = {0}
Dim counts() as integer = {181}
Dim intervals() as integer = {1}

try
... varN  =  CDFgetVarNum  (id,  "LATITUDE")
for i = 0   to 1
    for j = -90   to 90
        lats(i,90+lat)  =  Ctype(j, short)
    next j
next i
...status =  CDFhypperPutrVarData  (id,  varN,  recStart,  recCount,  recInterval,  indices,  counts,  intervals,  lats)
...
catch ex as Exception
...
end try
```

4.3.68  CDFhyperPutzVarData

integer CDFhyperPutzVarData(
    out -- Completion status code.
)
id as long, varNum as integer,
recStart as integer,
recCount as integer,
recInterval as integer,
indices as integer(),
counts as integer(),
intervals as integer(),
buffer as TYPE)

'C in -- CDF identifier.
' C in -- zVariable number.
' C in -- Starting record number.
' C in -- Number of records.
' C in -- Writing interval between records.
' C in -- Dimension indices of starting value.
' C in -- Number of values along each dimension.
' C in -- Writing intervals along each dimension.
' C in -- Buffer of values.
' TYPE -- VB value/string type (likely an array).

CDFhyperPutzVarData is used to write one or more values from the data holding buffer to the specified zVariable. It is important to know the variable majority of the CDF before using this method because the values in the data buffer will be written using that majority. CDFinquireCDF can be used to determine the default variable majority of a CDF distribution. The Concepts chapter in the CDF User's Guide describes the variable majorities.

The record number starts at 0, not 1. For example, if you want to write 2 records (10th and 11th record), the starting record number (recStart), the number of records to write (recCount), and the record interval (recInterval) should be 9, 2, and 1, respectively. Note: you need to provide dummy arrays, with at least one (1) element, for indices, counts and intervals for scalar variables.

The arguments to CDFhyperPutzVarData are defined as follows:

id The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

varNum The zVariable number to which write data. This number may be determined with a call to CDFgetVarNum.

recStart The record number at which to start writing.

recCount The number of records to write.

recInterval The interval between records for writing (e.g., an interval of 2 means write every other record).

indices The indices (within each record) at which to start writing. Each element of indices specifies the corresponding dimension index. For 0-dimensional zVariable this argument is ignored (but must be present).

counts The number of values along each dimension to write. Each element of counts specifies the corresponding dimension count. For 0-dimensional zVariable this argument is ignored (but must be present).

intervals For each dimension, the interval between values for writing (e.g., an interval of 2 means write every other value). Each element of intervals specifies the corresponding dimension interval. For 0-dimensional zVariable this argument is ignored (but must be present).

buffer The data holding buffer of values to write. The majority of the values in this buffer must be the same as that of the CDF. The values starting at memory address buffer are written to the CDF.

4.3.68.1. Example(s)
The following example writes 2 records to a zVariable named LATITUDE that is a 1-dimensional array with dimension sizes (181). The dimension variances are \{VARY\}, and the data type is CDF\_INT2. This example is similar to the CDFputzVarData example except that it uses a single call to CDFhyperPutzVarData rather than numerous calls to CDFputzVarData.

```
try
  ...
  varN  =  CDFgetVarNum  (id,  "LATITUDE")
  for i= 0  to 1
    for j = -90   to  90
      lats(i,90+lat)  =  Ctype(j, short)
    next j
  next i
  status =  CDFhyperPutzVarData  (id,  varN,  recStart,  recCount,  recInterval, indices, counts, intervals,  lats)
...
catch ex as Exception
...
end try
```

4.3.69  CDFinquirerVar

```
integer CDFinquirezVar(
    id as long,
    varNum as integer,
    varName as string,
    dataType as integer,
    numElements as integer,
    numDims as integer,
    dimSizes as integer(),
    recVariance as integer,
    dimVariances as integer())
```

• out -- Completion status code.
• in -- CDF identifier.
• in -- rVariable number.
• out -- rVariable name.
• out -- Data type.
• out -- Number of elements (of the data type).
• out -- Number of dimensions.
• out -- Dimension sizes
• out -- Record variance.
• out -- Dimension variances.
CDFinquirerVar is used to inquire about the specified rVariable. This method would normally be used before reading rVariable values (with CDFgetrVarData or CDFhyperGetrVarData) to determine the data type and number of elements of that data type.

The arguments to CDFinquirezVar are defined as follows:

- **id**: The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **varNum**: The number of the rVariable to inquire. This number may be determined with a call to CDFgetVarNum (see Section 4.3.41).
- **varName**: The rVariable's name.
- **dataType**: The data type of the rVariable. The data types are defined in Section 2.6.
- **numElements**: The number of elements of the data type at each rVariable value. For character data types (CDF_CHAR and CDF_UCHAR), this is the number of characters in the string. Each value consists of the entire string. For all other data types, this will always be one (1) — multiple elements at each value are not allowed for non-character data types.
- **numDims**: The number of dimensions.
- **dimSizes**: The dimension sizes. It is a 1-dimensional array, containing one element per dimension. Each element of dimSizes receives the corresponding dimension size. For 0-dimensional zVariables this argument is ignored (but must be present).
- **recVary**: The record variance. The record variances are defined in Section 2.10.
- **dimVarys**: The dimension variances. Each element of dimVarys receives the corresponding dimension variance. The dimension variances are described in Section 2.10. For 0-dimensional zVariables this argument is ignored (but a placeholder is necessary).

### 4.3.69.1. Example(s)

The following example returns information about a rVariable named HEAT_FLUX in a CDF.

```plaintext
try
  
  dim id as long
  Dim status as integer
  Dim varName as string
  Dim dataType as integer
  Dim numElems as integer
  Dim recVary as integer
  Dim numDims as integer
  Dim dimSizes() as integer
  Dim dimVarys() as integer

  ....
```
status = CDFinquirerVar(id, CDFgetVarNum(id,"HEAT_FLUX"), varName, dataType, _
numElems, numDims, dimSizes, recVary, dimVarys)
...
...catch ex as Exception
...
end try

4.3.70 CDFinquirezVar

integer CDFinquirezVar(
  id as long,
  varNum as integer,
  varName as string,
  dataType as integer,
  numElements as integer,
  numDims as integer,
  dimSizes as integer(),
  recVariance as integer,
  dimVariances as integer())

CDFinquirezVar is used to inquire about the specified zVariable. This method would normally be used before reading zVariable values (with CDFgetzVarData or CDFhyperGetzVarData) to determine the data type and number of elements of that data type.

The arguments to CDFinquirezVar are defined as follows:

id The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

varNum The number of the zVariable to inquire. This number may be determined with a call to CDFgetVarNum (see Section 4.3.41).

varName The zVariable's name.

dataType The data type of the zVariable. The data types are defined in Section 2.6.

numElements The number of elements of the data type at each zVariable value. For character data types (CDF_CHAR and CDF_UCHAR), this is the number of characters in the string. (Each value consists of the entire string.) For all other data types, this will always be one (1) - multiple elements at each value are not allowed for non-character data types.

numDims The number of dimensions.

dimSizes The dimension sizes. It is a 1-dimensional array, containing one element per dimension. Each element of dimSizes receives the corresponding dimension size. For 0-dimensional zVariables this argument is ignored (but must be present).

recVariance The record variance. The record variances are defined in Section 2.10.
dimVariances The dimension variances. Each element of dimVariances receives the corresponding dimension variance. The dimension variances are described in Section 2.10. For 0-dimensional zVariables this argument is ignored (but a placeholder is necessary).

4.3.70.1. Example(s)

The following example returns information about a zVariable named HEAT_FLUX in a CDF.

```
try
  status = CDFinquirezVar(id, CDFgetVarNum(id,"HEAT_FLUX"), varName, dataType, numElems, numDims, dimSizes, recVary, dimVarys)

catch ex as Exception
```

4.3.71 CDFputrVarData

```
integer CDFputrVarData(id as long, varNum as integer, recNum as integer, indices as integer(), value as TYPE)
```

CDFputrVarData writes a single data value to the specified index, the location of the element, in the given record of the specified rVariable in a CDF.

The arguments to CDFputrVarData are defined as follows:

id The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
varNum The rVariable number.

recNum The record number.

indices The dimension indices within the record.

value The data value.

4.3.71.1. Example(s)

The following example will write two data values, the first and the fifth element, in Record 0 from rVariable “MY_VAR”, a 2-dimensional (2 by 3), CDF_DOUBLE type variable, in a row-major CDF. The first put operation passes the pointer of the data value, while the second operation passes the data value as an object.

```vbnet
dim id as long
Dim varNum as integer
Dim recNum as integer
Dim indices(2) as integer
Dim value1 as double, value2 as double
Dim status as integer.

try
    varNum = CDFgetVarNum (id, “MY_VAR”)  
    recNum = 0
    indices(0) = 0
    indices(1) = 0
    value1 = 10.1
    status = CDFputrVarData (id, varNum, recNum, indices, value1)
    indices(0) = 1
    indices(1) = 1
    value2 = 20.2
    status = CDFputrVarData (id, varNum, recNum, indices, value2)
    ... 
    catch ex as Exception
        ...
end try
```

4.3.72  CDFputrVarPadValue

integer CDFputrVarPadValue(
    id as long,
    varNum as integer,
    value as TYPE)
    ` out -- Completion status code.
    ` in -- CDF identifier.
    ` in -- Variable number.
    ` in -- Pad value.
    ` TYPE – VB value/string type
CDFputrVarPadValue specifies the pad value for the specified rVariable in a CDF. A rVariable's pad value may be specified (or respecified) at any time without affecting already written values (including where pad values were used). The Concepts chapter in the CDF User's Guide describes variable pad values.

The arguments to CDFputrVarPadValue are defined as follows:

- **id**: The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **varNum**: The rVariable number.
- **value**: The pad value.

### 4.3.72.1. Example(s)

The following example sets the pad value to –9999 for rVariable “MY_VAR”, a CDF_INT4 type variable, and “*****” for another rVariable “MY_VAR2”, a CDF_CHAR type with a number of elements of five (5), in a CDF.

```vbnet
' CDF identifier.
Dim id as long
Dim padValue1 as integer = -9999 ' An integer pad value.
Dim padValue2 as string = “*****” ' A string pad value.

try
  ...
  status = CDFputrVarPadValue (id, CDFgetVarNum (id, “MY_VAR”), padValue1)
  status = CDFputrVarPadValue (id, CDFgetVarNum (id, “MY_VAR2”), padValue2)
  ...
  ...
  catch ex as Exception
    ...
end try
```

### 4.3.73 CDFputrVarRecordData

integer CDFputrVarRecordData(
  id as long,
  varNum as integer,
  recNum as integer,
  buffer as TYPE)

\* out -- Completion status code.
\* in -- CDF identifier.
\* in -- Variable number.
\* in -- Record number.
\* in -- Record data.
\* TYPE -- VB value/string type (likely an array)
CDFputrVarRecordData writes an entire record at a given record number for the specified rVariable in a CDF. The buffer should hold the entire data values for the variable. The data values in the buffer should be in the order that corresponds to the variable majority defined for the CDF.

The arguments to CDFputrVarRecordData are defined as follows:

- **id**: The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **varNum**: The rVariable number.
- **recNum**: The record number.
- **buffer**: The buffer holding the entire record values.

### 4.3.73.1. Example(s)

The following example will write one full record (numbered 2) from rVariable “MY_VAR”, a 2-dimension (2 by 3), CDF_INT4 type variable, in a CDF. The variable’s dimension variances are all VARY.

```vbnet
dim id as long
Dim varNum as integer
Dim buffer(2,3) as integer = {{1,2,3},{4,5,6}}

try
...
varNum = CDFvarNum (id,"MY_VAR")
status = CDFputrVarRecordData (id, varNum, 2, buffer)
...
catch ex as Exception
...
end try
```

### 4.3.74 CDFputrVarSeqData

```vbnet
integer CDFputrVarSeqData(
    id as long,
    varNum as integer,
    value as TYPE)
\* out -- Completion status code.
\* in -- CDF identifier.
\* in -- Variable number.
\* in -- Data value.
\* TYPE -- VB value/string type
```
CDFputrVarSeqData writes one value to the specified rVariable in a CDF at the current sequential value (position) for that variable. After the write, the current sequential value is automatically incremented to the next value. Use CDFsetrVarSeqPos method to set the current sequential value (position).

The arguments to CDFputrVarSeqData are defined as follows:

- **id**: The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **varNum**: The rVariable number.
- **value**: The buffer holding the data value.

### 4.3.74.1. Example(s)

The following example will write two data values starting at record number 2 from a 2-dimensional rVariable whose data type is CDF_INT4. The first write will pass in a pointer from the data value, while the second write will pass in the data value object directly.

```plaintext
dim id as long
Dim varNum as integer
Dim value1 as integer, value2 as integer
Dim indices(2) as integer
Dim recNum as integer

dim status as integer

recNum = 2
indices(0) = 1
indices(1) = 2
try
    ....
    value1 = 10
    value2 = -20.
    status = CDFsetrVarSeqPos (id, varNum, recNum, indices)
    status = CDFputrVarSeqData (id, varNum, value1)
    status = CDFputrVarSeqData (id, varNum, value2)
    ...
    ...
    catch ex as Exception
    ...
    end try
```

### 4.3.75 CDFputzVarData

integer CDFputzVarData(
    id as long,
    varNum as integer,
    recNum as integer,
    ....
)
index(0) = 0
index(1) = 0
value1 = 10.1
status = CDFputzVarData (id, varNum, recNum, indices, value1)
index(0) = 1
index(1) = 1
value2 = 20.2
status = CDFputzVarData (id, varNum, recNum, indices, value2)
...  
...  
catch ex as Exception
...

4.3.75.1. Example(s)

The following example will write two data values, the first and the fifth element, in Record 0 from zVariable “MY_VAR”, a 2-dimensional (2 by 3), CDF_DOUBLE type variable, in a row-major CDF. The first put operation passes the pointer of the data value, while the second operation passes the data value as an object.
4.3.76  

CDFputzVarPadValue

integer CDFputzVarPadValue(
    id as long,
    varNum as integer,
    value as TYPE)  

CDFputzVarPadValue specifies the pad value for the specified zVariable in a CDF. A zVariable's pad value may be specified (or respecified) at any time without affecting already written values (including where pad values were used). The Concepts chapter in the CDF User's Guide describes variable pad values.

The arguments to CDFputzVarPadValue are defined as follows:

id  The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

varNum  The zVariable number.

value  The pad value.

4.3.76.1. Example(s)

The following example sets the pad value to –9999 for zVariable “MY_VAR”, a CDF_INT4 type variable, and “*****” for another zVariable “MY_VAR2”, a CDF_CHAR type with a number of elements of five (5), in a CDF.

```plaintext
dim id as long
    ' CDF identifier.
dim padValue1 as integer = -9999
    ' An integer pad value.
Dim padValue2 as string = “*****”
    ' A string pad value.
try
    status = CDFputzVarPadValue (id, CDFgetVarNum (id, “MY_VAR”), padValue1)
    status = CDFputzVarPadValue (id, CDFgetVarNum (id, “MY_VAR2”), padValue2)
end try
```

end try
4.3.77  **CDFputzVarRecordData**

`integer CDFputzVarRecordData(
  id as long,
  varNum as integer,
  recNum as integer,
  buffer as TYPE)
`  

* out -- Completion status code.
* in -- CDF identifier.
* in -- Variable number.
* in -- Record number.
* in -- Record data.
* **TYPE** -- VB value/string type (likely an array)

CDFputzVarRecordData writes an entire record at a given record number for the specified zVariable in a CDF. The buffer should hold the entire data values for the variable. The data values in the buffer should be in the order that corresponds to the variable majority defined for the CDF.

The arguments to CDFputzVarRecordData are defined as follows:

- **id**  
  The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

- **varNum**  
  The zVariable number.

- **recNum**  
  The record number.

- **buffer**  
  The buffer holding the entire record values.

### 4.3.77.1. Example(s)

The following example will write one full record (numbered 2) from zVariable “MY_VAR”, a 2-dimension (2 by 3), CDF_INT4 type variable, in a CDF. The variable’s dimension variances are all VARY.

```vbnet
...  
  dim id as long  
  dim varNum as integer  
  Dim buffer(,)as integer = {{1,2,3},{4,5,6}}  
  Dim status as integer  
  
  try  
  ....  
  varNum = CDFvarNum (id,"MY_VAR")  
  status = CDFputzVarRecordData (id, varNum, 2, buffer)  
  ...  
  catch ex as Exception  
  ...  
  end try  
```
4.3.78  CDFputzVarSeqData

integer CDFputzVarSeqData(
    id as long,
    varNum as integer,
    value as TYPE)                            · out -- Completion status code.
                                                · in -- CDF identifier.
                                                · in -- Variable number.
                                                · in -- Data value.
                                                · TYPE -- VB value/string type

CDFputzVarSeqData writes one value to the specified zVariable in a CDF at the current sequential value (position) for that variable. After the write, the current sequential value is automatically incremented to the next value. Use CDFsetzVarSeqPos method to set the current sequential value (position).

The arguments to CDFputzVarSeqData are defined as follows:

id                        The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
varNum                     The zVariable number.
value                      The buffer holding the data value.

4.3.78.1. Example(s)

The following example will write two data values starting at record number 2 from a 2-dimensional zVariable whose data type is CDF_INT4. The first write will pass in a pointer from the data value, while the second write will pass in the data value object directly.

```
...        · CDF identifier.
...        · The variable number.
...        · The data value.
...        · The indices in a record.
...        · The record number.

dim id as long
    dim varNum as integer
    dim value1 as integer, value2 as integer
    Dim indices(2) as integer
    dim recNum as integer
    Dim status as integer
    recNum = 2
    indices(0) = 1
    indices(1) = 2
    try
        ...        · The identifier.
        value1 = 10
        value2 = -20.
        status = CDFsetzVarSeqPos (id, varNum, recNum, indices)
        status = CDFputzVarSeqData (id, varNum, value1)
        status = CDFputzVarSeqData (id, varNum, value2)
    ...    ...        · The variable number.
    catch ex as Exception
        ...        · The data value.
    end try
```
CDFrenamerVar is used to rename an existing rVariable. A variable (rVariable or zVariable) with the same name must not already exist in the CDF.

The arguments to CDFrenamerVar are defined as follows:

- **id**
  - The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

- **varNum**
  - The number of the rVariable to rename. This number may be determined with a call to CDFgetVarNum.

- **varName**
  - The new rVariable name. This may be at most CDF_VAR_NAME_LEN256 characters. Variable names are case-sensitive.

### 4.3.79.1. Example(s)

In the following example the rVariable named TEMPERATURE is renamed to TMP (if it exists). Note that if CDFgetVarNum returns a value less than zero (0) then that value is not an rVariable number but rather an error code.

```plaintext
try
  varNum = CDFgetVarNum (id, "TEMPERATURE")
  status = CDFrenamerVar (id, varNum, "TMP")
... catch ex as Exception
  ...
end try
```
4.3.80 CDFrenamezVar

integer CDFrenamezVar(
  id as long,
  varNum as integer,
  varName as string)

CDFrenamezVar is used to rename an existing zVariable. A variable (rVariable or zVariable) with the same name must not already exist in the CDF.

The arguments to CDFrenamezVar are defined as follows:

- `id` The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- `varNum` The number of the zVariable to rename. This number may be determined with a call to CDFgetVarNum.
- `varName` The new zVariable name. This may be at most CDF_VAR_NAME_LEN256 characters. Variable names are case-sensitive.

4.3.80.1. Example(s)

In the following example the zVariable named TEMPERATURE is renamed to TMP (if it exists). Note that if CDFgetVarNum returns a value less than zero (0) then that value is not an zVariable number but rather an error code.

```plaintext
dim id as long       ' CDF identifier.
dim status as integer ' Returned status code.
dim varNum as integer ' zVariable number.

try
  ....
  varNum = CDFgetVarNum (id, "TEMPERATURE")
  status = CDFrenamezVar (id, varNum, "TMP")
  ...
  ...
  catch ex as Exception
  ...
end try
```

4.3.81 CDFsetrVarAllocBlockRecords

integer CDFsetrVarAllocBlockRecords(
  id as long,
)

' out -- Completion status code.
' in -- CDF identifier.
CDFsetrVarAllocBlockRecords specifies a range of records to be allocated (not written) for the specified rVariable in a CDF. This operation is only applicable to uncompressed rVariable in single-file CDFs. Refer to the CDF User’s Guide for the descriptions of allocating variable records.

The arguments to CDFsetrVarAllocBlockRecords are defined as follows:

- **id**: The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **varNum**: The rVariable number.
- **firstRec**: The first record number to allocate.
- **lastRec**: The last record number to allocate.

### 4.3.81.1. Example(s)

The following example allocates 10 records, from record numbered 10 to 19, for rVariable “MY_VAR” in a CDF.

```plaintext

dim id as long    ' CDF identifier.
dim firstRec as integer, lastRec as integer    ' The first/last record numbers.
Dim status as integer.

firstRec = 10
lastRec = 19
try
    status = CDFsetrVarAllocBlockRecords (id, CDFgetVarNum (id, "MY_VAR"), firstRec, lastRec)
    ...
catch ex as Exception
    ...
end try
```

### 4.3.82  CDFsetrVarAllocRecords

CDFsetrVarAllocRecords specifies a range of records to be allocated (not written) for the specified rVariable in a CDF. This operation is only applicable to uncompressed rVariable in single-file CDFs. Refer to the CDF User’s Guide for the descriptions of allocating variable records.

The arguments to CDFsetrVarAllocRecords are defined as follows:

- **id**: CDF identifier.
- **varNum**: rVariable number.
- **numRecs**: Number of records.

```plaintext
integer CDFsetrVarAllocRecords(
    id as long,
    varNum as integer,
    numRecs as integer)    ' out -- Completion status code.
    ' in -- CDF identifier.
    ' in -- Variable number.
    ' in -- Number of records.
```
CDFsetrVarAllocRecords specifies a number of records to be allocated (not written) for the specified rVariable in a CDF. The records are allocated beginning at record number zero (0). This operation is only applicable to uncompressed rVariable in single-file CDFs. Refer to the CDF User’s Guide for the descriptions of allocating variable records.

The arguments to CDFsetrVarAllocRecords are defined as follows:

- **id**: The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **varNum**: The rVariable number.
- **numRecs**: The number of records to allocate.

### 4.3.82.1. Example(s)

The following example allocates 100 records, from record numbered 0 to 99, for rVariable “MY_VAR” in a CDF.

```java
...  
...  
  dim id as long              ' CDF identifier.
  dim numRecs as integer     ' The number of records.
  dim status as integer      ' CDF identifier.
  numRecs = 100
  try
    status = CDFsetrVarAllocRecords (id, CDFgetVarNum (id, "MY_VAR"), numRecs)
    ...
  catch ex as Exception
    ...
  end try
```

### 4.3.83 CDFsetrVarBlockingFactor

```
integer CDFsetrVarBlockingFactor( 
  id as long,                      ' in -- CDF identifier.
  varNum as integer,              ' in -- Variable number.
  bf as integer)                   ' in -- Blocking factor.
```

CDFsetrVarBlockingFactor specifies the blocking factor (number of records allocated) for the specified rVariable in a CDF. Refer to the CDF User’s Guide for a description of the blocking factor.

The arguments to CDFsetrVarBlockingFactor are defined as follows:

- **id**: The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
varNum  The rVariable number.

bf      The blocking factor. A value of zero (0) indicates that the default blocking factor is being used.

4.3.83.1. Example(s)

The following example sets the blocking factor to 100 records for rVariable “MY_VAR” in a CDF.

```vbscript
...;

dim id as long
Dim bf as integer
dim status as integer

bf = 100
try
    status = CDFsetrVarBlockingFactor (id, CDFgetVarNum (id, “MY_VAR”), bf)
    ...
catch ex as Exception
    ...
end try
```

4.3.84  CDFsetrVarCacheSize

```vbscript
integer CDFsetrVarCacheSize(
    id as long,           ' in -- CDF identifier.
    varNum as integer,    ' in -- Variable number.
    numBuffers as integer) ' in -- Number of cache buffers.
```

CDFsetrVarCacheSize specifies the number of cache buffers being for the rVariable in a CDF. This operation is not applicable to a single-file CDF. Refer to the CDF User’s Guide for description about caching scheme used by the CDF library.

The arguments to CDFsetrVarCacheSize are defined as follows:

id      The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

varNum  The rVariable number.

numBuffers  The number of cache buffers.
4.3.84.1. Example(s)

The following example sets the number of cache buffers to 10 for rVariable “MY_VAR” in a CDF.

```vba
dim id as long
Dim numBuffers as integer
Dim status as integer
numBuffers = 10
try
    status = CDFsetrVarCacheSize (id, CDFgetVarNum (id, “MY_VAR”), numBuffers)
end try
```

4.3.85 CDFsetrVarCompression

```vba
integer CDFsetrVarCompression(id as long, varNum as integer, compType as integer, cParms as integer())
```

CDFsetrVarCompression specifies the compression type/parameters for the specified rVariable in a CDF. Refer to Section 2.11 for a description of the CDF supported compression types/parameters.

The arguments to CDFsetrVarCompression are defined as follows:

- id: The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- varNum: The rVariable number.
- compType: The compression type.
- cParms: The compression parameters.

4.3.85.1. Example(s)

The following example sets the compression to GZIP.6 for rVariable “MY_VAR” in a CDF.

```vba
...```
dim id as long
Dim compType as integer
Dim cParms(1) as integer
dim status as integer

compType = GZIP_COMPRESSION
cParms(0) = 6
try
....
status = CDFsetrVarCompression (id, CDFgetVarNum (id, “MY_VAR”), compType, cParms)
...
...catch ex as Exception
...
end try

4.3.86  CDFsetrVarDataSpec

integer CDFsetrVarDataSpec(
    id as long,  
    varNum as integer,  
    dataType as integer)  

CDFsetrVarDataSpec respecifies the data type of the specified rVariable in a CDF. The variable’s data type cannot be changed if the new data type is not equivalent (type having a different data size) to the old data type and any values (including the pad value) have been written. Data specifications are considered equivalent if the data types are equivalent. Refer to the CDF User’s Guide for equivalent data types.

The arguments to CDFsetrVarDataSpec are defined as follows:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.</td>
</tr>
<tr>
<td>varNum</td>
<td>The rVariable number.</td>
</tr>
<tr>
<td>dataType</td>
<td>The new data type.</td>
</tr>
</tbody>
</table>

4.3.86.1. Example(s)

The following example respecifies the data type to CDF_INT2 (from its original CDF_UINT2) for rVariable “MY_VAR” in a CDF.
**4.3.87 CDFsetrVarDimVariances**

integer CDFsetrVarDimVariances(id as long, varNum as integer, dimVarys as integer())

CDFsetrVarDimVariances respecifies the dimension variances of the specified rVariable in a CDF. For 0-dimensional rVariable, this operation is not applicable. The dimension variances are described in Section 2.10.

The arguments to CDFsetrVarDimVariances are defined as follows:

- **id**
  - The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFOpenCDF.

- **varNum**
  - The rVariable number.

- **dimVarys**
  - The dimension variances.

### 4.3.87.1. Example(s)

The following example resets the dimension variances to true (VARY) and true (VARY) for rVariable “MY_VAR”, a 2-dimensional variable, in a CDF.

```cpp
dim id as long  ' CDF identifier.
Dim dataType as integer  ' The data type.
Dim status as integer.

dataType = CDF_INT2
try
    ....
    status = CDFsetrVarDataSpec (id, CDFgetVarNum (id, “MY_VAR”), dataType)
    ...
    catch ex as Exception
    ...
end try

DIM DIM_VAR() AS INTEGER = {VARY, VARY}  ' The dimension variances.
nDim varNum as integer  ' rVariable number.
Dim dimVarys() as integer = {VARY, VARY}

try
    ....
    varNum = CDFgetVarNum (id, “MY_VAR”)
```

156
status = CDFsetrVarDimVariances (id, varNum, dimVarys)
...
catch ex as Exception
...
end try

---

4.3.88  CDFsetrVarInitialRecs

integer CDFsetrVarInitialRecs(
  id as long,
  varNum as integer,
  numRecs as integer)

CDFsetrVarInitialRecs specifies a number of records to initially write to the specified rVariable in a CDF. The records are written beginning at record number 0 (zero). This may be specified only once per rVariable and before any other records have been written to that rVariable. If a pad value has not yet been specified, the default is used (see the Concepts chapter in the CDF User’s Guide). If a pad value has been explicitly specified, that value is written to the records. The Concepts chapter in the CDF User's Guide describes initial records.

The arguments to CDFsetrVarInitialRecs are defined as follows:

- id: The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- varNum: The rVariable number.
- numRecs: The initially written records.

---

4.3.88.1. Example(s)

The following example writes the initial 100 records to rVariable “MY_VAR” in a CDF.

```
try
  varNum = CDFgetVarNum (id, “MY_VAR”)
  numRecs = 100
  status = CDFsetrVarInitialRecs (id, varNum, numRecs)
  ...
catch ex as Exception
  ...
```
end try

4.3.89  **CDFsetrVarRecVariance**

integer CDFsetrVarRecVariance( id as long, varNum as integer, recVary as integer)  

CDFsetrVarRecVariance specifies the record variance of the specified rVariable in a CDF. The record variances are described in Section 2.10.

The arguments to CDFsetrVarRecVariance are defined as follows:

- **id**
  - The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

- **varNum**
  - The rVariable number.

- **recVary**
  - The record variance.

4.3.89.1. Example(s)

The following example sets the record variance to VARY (from NOVARY) for rVariable “MY_VAR” in a CDF.

```
dim id as long  
  ' CDF identifier.
dim recVary as integer 
  ' The record variance.
Dim status as integer.

recVary = VARY
try
    status = CDFsetrVarRecVariance (id, CDFgetVarNum (id, “MY_VAR”), recVary)
    ...
catch ex as Exception
    ...
end try
```

4.3.90  **CDFsetrVar ReservePercent**

integer CDFsetrVarReservePercent(  

CDFsetrVarReservePercent
CDFsetrVarReservePercent specifies the compression reserve percentage being used for the specified rVariable in a CDF. This operation only applies to compressed rVariables. Refer to the CDF User's Guide for a description of the reserve scheme used by the CDF library.

The arguments to CDFsetrVarReservePercent are defined as follows:

- **id**: The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **varNum**: The rVariable number.
- **percent**: The reserve percentage.

### 4.3.90.1. Example(s)

The following example sets the reserve percentage to 10 for rVariable “MY_VAR” in a CDF.

```plaintext
... 

dim id as long
dim percent as integer
Dim status as integer.

percent = 10
try
    ....
    status = CDFsetrVarReservePercent (id, CDFgetVarNum (id, “MY_VAR”), percent)
... 
catch ex as Exception
    ...
end try
...
```

### 4.3.91 CDFsetrVarsCacheSize

integer CDFsetrVarsCacheSize(
    id as long,
    numBuffers as integer)

CDFsetrVarsCacheSize specifies the number of cache buffers to be used for all of the rVariable files in a CDF. This operation is not applicable to a single-file CDF. The Concepts chapter in the CDF User's Guide describes the caching scheme used by the CDF library.
The arguments to CDFsetrVarsCacheSize are defined as follows:

- **id**: The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **numBuffers**: The number of buffers.

### 4.3.91.1. Example(s)

The following example sets the number of cache buffers to 10 for all rVariables in a CDF.

```plaintext
dim id as long
    ' CDF identifier.
dim numBuffers as integer
    ' The number of cache buffers.
Dim status as integer.
numBuffers = 10
try
    status = CDFsetrVarsCacheSize (id, numBuffers)
    ...
catch ex as Exception
    ...
end try
```

### 4.3.92  CDFsetrVarSeqPos

integer CDFsetrVarSeqPos(
    id as long,
    varNum as integer,
    dim recNum as integer,
    indices as integer())

    ' out -- Completion status code.
    ' in -- CDF identifier.
    ' in -- Variable number.
    ' in -- Record number.
    ' in -- Indices in a record.

CDFsetrVarSeqPos specifies the current sequential value (position) for sequential access for the specified rVariable in a CDF. Note that a current sequential value is maintained for each rVariable individually. Use CDFgetrVarSeqPos method to get the current sequential value.

The arguments to CDFsetrVarSeqPos are defined as follows:

- **id**: The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **varNum**: The rVariable number.
- **recNum**: The rVariable record number.
indices  The dimension indices. Each element of indices receives the corresponding dimension index. For 0-dimensional rVariable, this argument is ignored, but must be presented.

4.3.92.1. Example(s)

The following example sets the current sequential value to the first value element in record number 2 for a rVariable, a 2-dimensional variable, in a CDF.

```plaintext
.
.
.
dim id as long
Dim varNum as integer
dim recNum as integer
Dim indices(2) as integer
.
.
recNum = 2
indices(0) = 0
indices(1) = 0
try
    status = CDFsetrVarSeqPos (id, varNum, recNum, indices)
    ...
    catch ex as Exception
    ...
end try
```

4.3.93  CDFsetrVarSparseRecords

```plaintext
type CDFsetrVarSparseRecords(
    id as long,  
    varNum as integer,  
    sRecordsType as integer)
```

CDFsetrVarSparseRecords specifies the sparse records type of the specified rVariable in a CDF. Refer to Section 2.12.1 for the description of sparse records.

The arguments to CDFsetrVarSparseRecords are defined as follows:

- **id**  The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **varNum**  The rVariable number.
- **sRecordsType**  The sparse records type.
4.3.93.1. Example(s)

The following example sets the sparse records type to PAD_SPARSERECORDS from its original type for rVariable “MY_VAR” in a CDF.

```vbs

dim id as long ' CDF identifier.
dim sRecordsType as integer ' The sparse records type.
Dim status as integer.

sRecordsType = PAD_SPARSERECORDS
try
    status = CDFsetrVarSparseRecords (id, CDFgetVarNum (id, “MY_VAR”), sRecordsType)
    ...
    ...
catch ex as Exception
    ...
end try
```

4.3.94 CDFsetzVarAllocBlockRecords

```vbs
integer CDFsetzVarAllocBlockRecords( ' out -- Completion status code.
    id as long, ' in -- CDF identifier.
    varNum as integer, ' in -- Variable number.
    firstRec as integer, ' in -- First record number.
    lastRec as integer) ' in -- Last record number.
```

CDFsetzVarAllocBlockRecords specifies a range of records to be allocated (not written) for the specified zVariable in a CDF. This operation is only applicable to uncompressed zVariable in single-file CDFs. Refer to the CDF User’s Guide for the descriptions of allocating variable records.

The arguments to CDFsetzVarAllocBlockRecords are defined as follows:

- **id**: The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **varNum**: The zVariable number.
- **firstRec**: The first record number to allocate.
- **lastRec**: The last record number to allocate.

4.3.94.1. Example(s)

The following example allocates 10 records, from record numbered 10 to 19, for zVariable “MY_VAR” in a CDF.
dim id as long  ’ CDF identifier.
dim firstRec as integer, lastRec as integer  ’ The first/last record numbers.
dim status as integer

firstRec = 10
lastRec = 19
try
    status = CDFsetzVarAllocBlockRecords (id, CDFgetVarNum (id, “MY_VAR”), firstRec, lastRec)
    ...
catch ex as Exception
    ...
end try

4.3.95  CDFsetzVarAllocRecords

integer CDFsetzVarAllocRecords(  
    id as long,  ’ in -- CDF identifier.
    varNum as integer,  ’ in -- Variable number.
    numRecs as integer)  ’ in -- Number of records.

CDFsetzVarAllocRecords specifies a number of records to be allocated (not written) for the specified zVariable in a CDF. The records are allocated beginning at record number zero (0). This operation is only applicable to uncompressed zVariable in single-file CDFs. Refer to the CDF User’s Guide for the descriptions of allocating variable records.

The arguments to CDFsetzVarAllocRecords are defined as follows:

    id          The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
    varNum      The zVariable number.
    numRecs     The number of records to allocate.

4.3.95.1. Example(s)

The following example allocates 100 records, from record numbered 0 to 99, for zVariable “MY_VAR” in a CDF.

    dim id as long
    Dim numRecs as integer
    Dim status as integer.  ’ The number of records.
numRecs = 100
try
....
status = CDFsetzVarAllocRecords (id, CDFgetVarNum (id, “MY_VAR”), numRecs)
....
catch ex as Exception
....
end try

4.3.96 CDFsetzVarBlockingFactor

integer CDFsetzVarBlockingFactor(id as long, varNum as integer, bf as integer)

CDFsetzVarBlockingFactor specifies the blocking factor (number of records allocated) for the specified zVariable in a CDF. Refer to the CDF User’s Guide for a description of the blocking factor.

The arguments to CDFsetzVarBlockingFactor are defined as follows:

id The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

varNum The zVariable number.

bf The blocking factor. A value of zero (0) indicates that the default blocking factor is being used.

4.3.96.1. Example(s)

The following example sets the blocking factor to 100 records for zVariable “MY_VAR” in a CDF.

```plaintext
dim id as long
Dim bf as integer
Dim status as integer.

bf = 100
try
....
status = CDFsetzVarBlockingFactor (id, CDFgetVarNum (id, “MY_VAR”), bf)
....
catch ex as Exception
....
```
4.3.97 CDFsetzVarCacheSize

integer CDFsetzVarCacheSize(
    id as long,
    varNum as integer,
    numBuffers as integer)

CDFsetzVarCacheSize specifies the number of cache buffers being for the zVariable in a CDF. This operation is not applicable to a single-file CDF. Refer to the CDF User's Guide for description about caching scheme used by the CDF library.

The arguments to CDFsetzVarCacheSize are defined as follows:

- **id**: The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **varNum**: The zVariable number.
- **numBuffers**: The number of cache buffers.

4.3.97.1. Example(s)

The following example sets the number of cache buffers to 10 for zVariable “MY_VAR” in a CDF.

```plaintext
Dim id as long
    ' CDF identifier.
Dim numBuffers as integer
    ' The number of cache buffers.
Dim status as integer.

numBuffers = 10
try
    ...
    status = CDFsetzVarCacheSize (id, CDFgetVarNum (id, “MY_VAR”), numBuffers)
    ...
    catch ex as Exception
        ...
end try
```

4.3.98 CDFsetzVarCompression
integer CDFsetzVarCompression(
    id as long,
    varNum as integer,
    compType as integer,
    cParms as integer())

CDFsetzVarCompression specifies the compression type/parameters for the specified zVariable in a CDF. Refer to Section 2.11 for a description of the CDF supported compression types/parameters.

The arguments to CDFsetzVarCompression are defined as follows:

- **id**
  - The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

- **varNum**
  - The zVariable number.

- **compType**
  - The compression type.

- **cParms**
  - The compression parameters.

### 4.3.98.1. Example(s)

The following example sets the compression to GZIP.6 for zVariable “MY_VAR” in a CDF.

```
    dim id as long
    Dim compType as integer
    Dim cParms(1) as integer

    compType = GZIP_COMPRESSION
    cParms(0) = 6
    try
        status = CDFsetzVarCompression (id, CDFgetVarNum (id, "MY_VAR"), compType, cParms)
    catch ex as Exception
        ...
    end try
```

### 4.3.99 CDFsetzVarDataSpec

integer CDFsetzVarDataSpec(
    id as long,
    varNum as integer,
    dataType as integer)

CDFsetzVarDataSpec specifies the data type for the specified zVariable in a CDF. Refer to Section 2.13 for a description of the CDF supported data types.
CDFsetzVarDataSpec respecifies the data type of the specified zVariable in a CDF. The variable’s data type cannot be changed if the new data type is not equivalent (type having a different data size) to the old data type and any values (including the pad value) have been written. Data specifications are considered equivalent if the data types are equivalent. Refer to the CDF User’s Guide for equivalent data types.

The arguments to CDFsetzVarDataSpec are defined as follows:

- **id**: The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **varNum**: The zVariable number.
- **dataType**: The new data type.

### 4.3.99.1. Example(s)

The following example respecifies the data type to CDF_INT2 (from its original CDF_UINT2) for zVariable “MY_VAR” in a CDF.

```plaintext
    dim id as long       ' CDF identifier.
    Dim dataType as integer  ' The data type.
    Dim status as integer
    dataType = CDF_INT2
    try
    ....
    status = CDFsetzVarDataSpec (id, CDFgetVarNum (id, “MY_VAR”), dataType)
    ....
    catch ex as Exception
    ...
    end try
```

### 4.3.100 CDFsetzVarDimVariances

integer CDFsetzVarDimVariances(
    id as long,
    varNum as integer,
    dimVarys as integer())

CDFsetzVarDimVariances respecifies the dimension variances of the specified zVariable in a CDF. For 0-dimensional zVariable, this operation is not applicable. The dimension variances are described in Section 2.10.
The arguments to CDFsetzVarDimVariances are defined as follows:

- **id**: The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **varNum**: The zVariable number.
- **dimVarys**: The dimension variances.

### 4.3.100.1. Example(s)

The following example resets the dimension variances to true (VARY) and true (VARY) for zVariable “MY_VAR”, a 2-dimensional variable, in a CDF.

```plaintext
... 

dim id as long
dim varNum as integer
Dim dimVarys()as integer = {VARY, VARY}
Dim status as integer

try

  varNum = CDFgetVarNum (id, "MY_VAR")
  status = CDFsetzVarDimVariances (id, varNum, dimVarys)

... 

catch ex as Exception

... 

end try
```

### 4.3.101 CDFsetzVarInitialRecs

```plaintext
integer CDFsetzVarInitialRecs(
  id as long,
  varNum as integer,
  numRecs as integer)
```

CDFsetzVarInitialRecs specifies a number of records to initially write to the specified zVariable in a CDF. The records are written beginning at record number 0 (zero). This may be specified only once per zVariable and before any other records have been written to that zVariable. If a pad value has not yet been specified, the default is used (see the Concepts chapter in the CDF User's Guide). If a pad value has been explicitly specified, that value is written to the records. The Concepts chapter in the CDF User's Guide describes initial records.

The arguments to CDFsetzVarInitialRecs are defined as follows:
id The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

varNum The zVariable number.

numRecs The initially written records.

### 4.3.101.1. Example(s)

The following example writes the initial 100 records to zVariable “MY_VAR” in a CDF.

```vba
try
    varNum = CDFgetVarNum (id, “MY_VAR”)
    numRecs = 100
    status = CDFsetzVarInitialRecs (id, varNum, numRecs)
    ...
catch ex as Exception
    ...
end try
```

### 4.3.102 CDFsetzVarRecVariance

The arguments to CDFsetzVarRecVariance are defined as follows:

### CDFsetzVarRecVariance

```vba
integer CDFsetzVarRecVariance(
    id as long,
    varNum as integer,
    recVary as integer)
```

CDFsetzVarRecVariance specifies the record variance of the specified zVariable in a CDF. The record variances are described in Section 2.10.
4.3.102.1. Example(s)

The following example sets the record variance to VARY (from NOVARY) for zVariable “MY_VAR” in a CDF.

```vba
... 
... 
dim id as long ' CDF identifier.
Dim recVary as integer ' The record variance.
Dim status as integer ..
recVary = VARY try ....
status = CDFsetzVarRecVariance (id, CDFgetVarNum (id, “MY_VAR”), recVary) ...
... catch ex as Exception ...
end try
```

4.3.103 CDFsetzVarReservePercent

```vba
integer CDFsetzVarReservePercent( id as long, ' out -- Completion status code.
varNum as integer, ' in -- CDF identifier.
percent as integer) ' in -- Variable number.
' in -- Reserve percentage.
```

CDFsetzVarReservePercent specifies the compression reserve percentage being used for the specified zVariable in a CDF. This operation only applies to compressed zVariables. Refer to the CDF User’s Guide for a description of the reserve scheme used by the CDF library.

The arguments to CDFsetzVarReservePercent are defined as follows:

- **id**: The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **varNum**: The zVariable number.
- **percent**: The reserve percentage.

4.3.103.1. Example(s)

The following example sets the reserve percentage to 10 for zVariable “MY_VAR” in a CDF.

```vba
... 
```


dim id as long  ' CDF identifier.
Dim percent as integer  ' The reserve percentage.
Dim status as integer

percent = 10
try
....
status = CDFsetzVarReservePercent (id, CDFgetVarNum (id, “MY_VAR”), percent)
...
catch ex as Exception
...
end try
.

4.3.104 CDFsetzVarsCacheSize

integer CDFsetzVarsCacheSize(
    id as long,
    numBuffers as integer)

CDFsetzVarsCacheSize specifies the number of cache buffers to be used for all of the zVariable files in a CDF. This operation is not applicable to a single-file CDF. The Concepts chapter in the CDF User's Guide describes the caching scheme used by the CDF library.

The arguments to CDFsetzVarsCacheSize are defined as follows:

id  The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

numBuffers  The number of buffers.

4.3.104.1. Example(s)

The following example sets the number of cache buffers to 10 for all zVariables in a CDF.

dim id as long  ' CDF identifier.
Dim numBuffers as integer  ' The number of cache buffers.
.
numBuffers = 10
try
....
status = CDFsetzVarsCacheSize (id, numBuffers)
... 
... 
catch ex as Exception
...
end try

**4.3.105 CDFsetzVarSeqPos**

integer CDFsetzVarSeqPos(
  id as long,
  varNum as integer,
  dim recNum as integer,
  indices as integer as integer())

CDFsetzVarSeqPos specifies the current sequential value (position) for sequential access for the specified zVariable in a CDF. Note that a current sequential value is maintained for each zVariable individually. Use CDFgetzVarSeqPos method to get the current sequential value.

The arguments to CDFsetzVarSeqPos are defined as follows:

- **id**
  The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

- **varNum**
  The zVariable number.

- **recNum**
  The zVariable record number.

- **indices**
  The dimension indices. Each element of indices receives the corresponding dimension index. For 0-dimensional zVariable, this argument is ignored, but must be presented.

**4.3.105.1. Example(s)**

The following example sets the current sequential value to the first value element in record number 2 for a zVariable, a 2-dimensional variable, in a CDF.

```
.
.
dim id as long
  ' CDF identifier.
dim varNum as integer
  ' The variable number.
dim recNum as integer
  ' The record number.
dim indices(2) as integer
  ' The indices.
.
recNum = 2
indices(0) = 0
indices(1) = 0
try
  status = CDFsetzVarSeqPos (id, varNum, recNum, indices)
...```
... catch ex as Exception...
end try

4.3.106 CDFsetzVarSparseRecords

integer CDFsetzVarSparseRecords(
    id as long,
    varNum as integer,
    sRecordsType as integer)

CDFsetzVarSparseRecords specifies the sparse records type of the specified zVariable in a CDF. Refer to Section 2.12.1 for the description of sparse records.

The arguments to CDFsetzVarSparseRecords are defined as follows:

- **id**
  - The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

- **varNum**
  - The zVariable number.

- **sRecordsType**
  - The sparse records type.

4.3.106.1. Example(s)

The following example sets the sparse records type to PAD_SPARSERECORDS from its original type for zVariable “MY_VAR” in a CDF.

```plaintext
... dim id as long
... dim sRecordsType as integer
... Dim status as integer.
...
... sRecordsType = PAD_SPARSERECORDS
try
    status = CDFsetzVarSparseRecords (id, CDFgetVarNum (id, “MY_VAR”), sRecordsType)
    ...
... catch ex as Exception
    ...
end try
```
4.3.107 CDFvarClose

integer CDFvarClose(
    id as long,
    varNum as integer)

CDFvarClose closes the specified rVariable file from a multi-file format CDF. The variable's cache buffers are flushed before the variable's open file is closed. However, the CDF file is still open.

NOTE: You must close all open variable files to guarantee that all modifications you have made will actually be written to the CDF's file(s). If your program exits, normally or otherwise, without a successful call to CDFclose, the CDF's cache buffers are left unflushed.

The arguments to CDFclose are defined as follows:

id       The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopen.

varNum   The variable number for the open rVariable's file. This identifier must have been initialized by a call to CDFgetVarNum.

4.3.107.1. Example(s)

The following example will close an open rVariable in a multi-file CDF.

```
dim id as long  
   `CDF identifier.

dim status as integer  
   `Returned status code.

try
    status = CDFvarClose (id, CDFvarNum (id, "Flux"))

catch ex as Exception
    ...  
end try
```

4.3.108 CDFvarCreate

integer CDFvarCreate(

9 A legacy CDF function, handling rVariables only. While it is still available in V3.1, CDFcloserVar is the preferred function for it.

10 A legacy CDF function, handling rVariables only. While it is still available in V3.1, CDFcreaterVar is the preferred function for it.
CDFVarCreate is used to create a new rVariable in a CDF. A variable (rVariable or zVariable) with the same name must not already exist in the CDF.

The arguments to CDFVarCreate are defined as follows:

- **id**
  - The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopen.

- **varName**
  - The name of the rVariable to create. This may be at most CDF_VAR_NAME_LEN256 characters. Variable names are case-sensitive.

- **dataType**
  - The data type of the new rVariable. Specify one of the data types defined in Section 2.6.

- **numElements**
  - The number of elements of the data type at each value. For character data types (CDF_CHAR and CDF_UCHAR), this is the number of characters in the string (each value consists of the entire string). For all other data types this must always be one (1) - multiple elements at each value are not allowed for non-character data types.

- **recVariance**
  - The rVariable's record variance. Specify one of the variances defined in Section 2.10.

- **dimVariances**
  - The rVariable's dimension variances. Each element of dimVariances specifies the corresponding dimension variance. For each dimension specify one of the variances defined in Section 2.10. For 0-dimensional rVariables this argument is ignored (but must be present).

- **varNum**
  - The number assigned to the new rVariable. This number must be used in subsequent CDF function calls when referring to this rVariable. An existing rVariable's number may be determined with the CDFvarNum or CDFgetVarNum function.

### 4.3.108.1. Example(s)

The following example will create several rVariables in a 2-dimensional CDF.

```plaintext

dim id as long
    ' CDF identifier.
dim stats as integer
    ' Returned status code.
dim EPOCHrecVary as integer = VARY
    ' EPOCH record variance.
Dim LATrecVary as integer = NOVARY
    ' LAT record variance.
Dim LONrecVary as integer = NOVARY
    ' LON record variance.
Dim TMPrecVary as integer = VARY
    ' TMP record variance.
Dim EPOCHdimVarys() as integer = {NOVARY,NOVARY}
    ' EPOCH dimension variances.
Dim LATdimVarys() as integer = {VARY,VARY}
    ' LAT dimension variances.
```
Dim LONDimVarys() as integer = {VARY, VARY} ' LON dimension variances.
Dim TMPdimVarys() as integer = {VARY, VARY} ' TMP dimension variances.
Dim EPOCHvarNum as integer ' EPOCH zVariable number.
Dim LATvarNum as integer ' LAT zVariable number.
Dim LONvarNum as integer ' LON zVariable number.
Dim TMPvarNum as integer ' TMP zVariable number.

try
    status = CDFvarCreate (id, "EPOCH", CDF_EPOCH, 1, _
        EPOCHrecVary, EPOCHdimVarys, EPOCHvarNum)
    status = CDFvarCreate (id, "LATITUDE", CDF_INT2, 1, _
        LATrecVary, LATdimVarys, LATvarNum)
    status = CDFvarCreate (id, "INTITUDE", CDF_INT2, 1, _
        LONrecVary, LONdimVarys, LONvarNum)
    status = CDFvarCreate (id, "TEMPERATURE", CDF_REAL4, 1, _
        TMPrecVary, TMPdimVarys, TMPvarNum)

catch ex as Exception
    ...
end try

4.3.109 CDFvarGet

integer CDFvarGet(
    id as long,
    varNum as integer,
    dim recNum as integer,
    indices as integer(),
    value as TYPE)

' out -- Completion status code.
' in -- CDF identifier.
' in -- rVariable number.
' in -- Record number.
' in -- Dimension indices.
' out -- Value.
' TYPE -- VB value/string type or object

CDFvarGet is used to read a single value from an rVariable.

The arguments to CDFvarGet are defined as follows:

id The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopen.
varNum The rVariable number from which to read data.
recNum The record number at which to read.
indices The dimension indices within the record.

---

11 A legacy CDF function, handling rVariables only. While it is still available in V3.1, CDFgetrVarData is the preferred function for it.
value  The data value read. This buffer must be large enough to hold the value.

4.3.109.1.  Example(s)

The following example returns two data values, the first and the fifth element, in Record 0 from an rVariable named MY_VAR, a 2-dimensional (2 by 3) CDF_DOUBLE type variable, in a row-major CDF. The first get operation passes the value pointer, while the second operation uses “out” argument modifier.

```plaintext
dim id as long
    ' CDF identifier.
dim recNum as integer
    ' The record number.
dim varNum as integer
    ' The variable number.
Dim indices(2) as integer
    ' The dimension indices.
Dim value1 as double, value2 as double
    ' The data values.
dim status as integer.

try
    varNum = CDFvarNum (id, "MY_VAR")
    recNum = 0
    indices(0) = 0
    indices(1) = 0
    status = CDFvarGet (id, varNum, recNum, indices, value1)
    indices(0) = 1
    indices(1) = 1
    object value2o
    status = CDFvarGet (id, varNum, recNum, indices, value2o)
    value2 = value2o
catch ex as Exception
    ...
end try
```

4.3.110  CDFvarHyperGet12

integer CDFvarHyperGet(
    id as long,
    varNum as integer,
    recStart as integer,
    recCount as integer,
    recInterval as integer,
    indices as integer(),
    counts as integer(),
    intervals as integer(),
    values as TYPE)
    ' out -- Completion status code.
    ' in -- CDF identifier.
    ' in -- rVariable number.
    ' in -- Starting record number.
    ' in -- Number of records.
    ' in -- Subsampling interval between records.
    ' in -- Dimension indices of starting value.
    ' in -- Number of values along each dimension.
    ' in -- Subsampling intervals along each dimension.
    ' out -- Values.

---

12 A legacy CDF function, handling rVariables only. While it is still available in V3.1, CDFhyperGetrVarData is the preferred function for it.
CDFvarHyperGet is used to fill a buffer of one or more values from the specified rVariable. It is important to know the variable majority of the CDF before using CDFvarHyperGet because the values placed into the buffer will be in that majority. CDFinquire can be used to determine the default variable majority of a CDF distribution. The Concepts chapter in the CDF User's Guide describes the variable majorities. **Note:** you need to provide dummy arrays, with at least one (1) element, for indices, counts and intervals for scalar variables.

### 4.3.110.1. Example(s)

The following example will read an entire record of data from an rVariable. The CDF’s rVariables are 3-dimensional with sizes (180,91,10) and CDF’s variable majority is ROW_MAJOR. For the rVariable the record variance is VARY, the dimension variances are \{VARY,VARY,VARY\}, and the data type is CDF_REAL4. This example is similar to the example provided for CDFvarGet except that it uses a single call to CDFvarHyperGet rather than numerous calls to CDFvarGet.

```vba
try
    varN  =  CDFgetVarNum  (id,  "Temperature")
    ...
    status = CDFvarHyperGet (id,  varN,  recStart,  recCount,  recInterval,  indices,  counts,  intervals,  tmp)
    ...
catch ex as Exception
    ...
end try
```

Note that if the CDF's variable majority had been COLUMN_MAJOR, the tmp array would have been declared simple type of `tmp(10,91,180)` for proper indexing.

### 4.3.111 CDFvarHyperPut13

```vba
integer CDFvarHyperPut(                                ' out -- Completion status code.
    ')
```

13 A legacy CDF function, handling rVariables only. While it is still available in V3.1, CDFhyperPutrVarData is the preferred function for it.
CDFvarHyperPut is used to write one or more values from the data holding buffer to the specified rVariable. It is important to know the variable majority of the CDF before using this routine because the values in the buffer to be written must be in the same majority. CDFinquire can be used to determine the default variable majority of a CDF distribution. The Concepts chapter in the CDF User's Guide describes the variable majorities. **Note:** you need to provide dummy arrays, with at least one (1) element, for indices, counts and intervals for scalar variables.

### 4.3.111.1. Example(s)

The following example writes values to the rVariable LATITUDE of a CDF that is an 2-dimensional array with dimension sizes (360,181). For LATITUDE the record variance is NOVARY, the dimension variances are \{NOVARY,VARY\}, and the data type is CDF_INT2. This example is similar to the CDFvarPut example except that it uses a single call to CDFvarHyperPut rather than numerous calls to CDFvarPut.

```
try
    varN  =  CDFvarNum (id,  "LATITUDE")
    for i =  -90  to 90
        lats(90+i)  =  CType(i, short)
    next lat
    status = CDFvarHyperPut (id,  varN,  recStart,  recCount,  recInterval,  indices,  counts,  intervals,  lats)
    ....
catch ex as Exception
    ...
end try
```
4.3.112 CDFvarInquire

integer CDFvarInquire(  
id as long,  
varNum as integer,  
varName as string,  
dataType as integer,  
numElements as integer,  
recVariance as integer,  
dimVariances as integer())  
    
\* out -- Completion status code.  
\* in -- CDF identifier.  
\* in -- rVariable number.  
\* out -- rVariable name.  
\* out -- Data type.  
\* out -- Number of elements (of the data type).  
\* out -- Record variance.  
\* out -- Dimension variances.

CDFvarInquire is used to inquire about the specified rVariable. This method would normally be used before reading rVariable values (with CDFvarGet or CDFvarHyperGet) to determine the data type and number of elements (of that data type).

The arguments to CDFvarInquire are defined as follows:

- **id**
  The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopen.

- **varNum**
  The number of the rVariable to inquire. This number may be determined with a call to CDFvarNum (see Section 4.3.113).

- **varName**
  The rVariable's name.

- **dataType**
  The data type of the rVariable. The data types are defined in Section 2.6.

- **numElements**
  The number of elements of the data type at each rVariable value. For character data types (CDF_CHAR and CDF_UCHAR), this is the number of characters in the string. (Each value consists of the entire string.) For all other data types, this will always be one (1) - multiple elements at each value are not allowed for non-character data types.

- **recVariance**
  The record variance. The record variances are defined in Section 2.10.

- **dimVariances**
  The dimension variances. Each element of dimVariances receives the corresponding dimension variance. The dimension variances are defined in Section 2.10. For 0-dimensional rVariables this argument is ignored (but a placeholder is necessary).

4.3.112.1. **Example(s)**

The following example returns about an rVariable named HEAT_FLUX in a CDF. Note that the rVariable name returned by CDFvarInquire will be the same as that passed in to CDFgetVarNum.

```
\dim id as long     \* CDF identifier.
Dim status as integer \* Returned status code.
```
Dim varName as string  ' rVariable name.
Dim dataType as integer  ' Data type of the rVariable.
Dim numElems as integer  ' Number of elements (of data type).
Dim recVary as integer  ' Record variance.
Dim dimVarys(CDF_MAX_DIMS) as integer  ' Dimension variances (allocate to allow the
                                           ' maximum number of dimensions).

try
...
status = CDFVarInquire (id, CDFgetVarNum (id,"HEAT_FLUX"), varName, dataType, _
numElems, recVary, dimVarys)
...
catch ex as Exception
...
end try

4.3.113 CDFvarNum

integer CDFvarNum(  ' out -- Variable number.
    id as long,  ' in -- CDF identifier.
    varName as string)  ' in -- Variable name.

CDFvarNum is used to determine the number associated with a given variable name. If the variable is found, CDFvarNum returns its variable number - which will be equal to or greater than zero (0). If an error occurs (e.g., the variable does not exist in the CDF), an error code (of type Int) is returned. Error codes are less than zero (0). The returned variable number should be used in the functions of the same variable type, rVariable or zVariable. If it is an rVariable, functions dealing with rVariables should be used. Similarly, functions for zVariables should be used for zVariables.

The arguments to CDFvarNum are defined as follows:

id          The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopen.

varName     The name of the variable to search. This may be at most CDF_VAR_NAME_LEN256 characters. Variable names are case-sensitive.

4.3.113.1. Example(s)

In the following example CDFvarNum is used as an embedded function call when inquiring about an rVariable.

...  
...  

dim id as long  ' CDF identifier.

dim status as integer  ' Returned status code.

14 A legacy CDF function. It used to handle only rVariables. It has been extended to include zVariables. While it is still available in V3.1, CDFgetVarNum is the preferred function for it.
try

    status = CDFvarInquire (id, CDFvarNum (id,"LATITUDE"), varName, dataType, _
    numElements, recVariance, dimVariances)

    .

catch ex as Exception

    ...

end try

In this example the rVariable named LATITUDE was inquired. Note that if LATITUDE did not exist in the CDF, the call to CDFgetVarNum would have returned an error code. Passing that error code to CDFvarInquire as an rVariable number would have resulted in CDFvarInquire also returning an error code. Also note that the name written into varName is already known (LATITUDE). In some cases the rVariable names will be unknown - CDFvarInquire would be used to determine them. CDFvarInquire is described in Section 4.3.112.

4.3.114 CDFvarPut15

integer CDFvarPut(
    id as long,
    varNum as integer,
    recNum as integer,
    indices as integer(),
    value as TYPE)

CDFvarPut writes a single data value to an rVariable. CDFvarPut may be used to write more than one value with a single call.

The arguments to CDFvarPut are defined as follows:

id
The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopen.

varNum
The rVariable number to which to write. This number may be determined with a call to CDFvarNum.

recNum
The record number at which to write.

indices
The dimension indices within the specified record at which to write. Each element of indices specifies the corresponding dimension index. For 0-dimensional variables, this argument is ignored (but must be present).

value
The data value to be written to the rVariable.

TYPE -- VB value/string type

15 A legacy CDF function, handling rVariables only. While it is still available in V3.1, CDFputrVarData is the preferred function for it.
value The data value to write.

### 4.3.114.1. Example(s)

The following example will write two data values (1\textsuperscript{st} and 5\textsuperscript{th} elements) of a 2-dimensional rVariable (2 by 3) named MY_VAR to record number 0.

```plaintext
dim id as long  ' CDF identifier.
dim varNum as integer  ' rVariable number.
dim recNum as integer  ' The record number.
Dim indices(2) as integer  ' The dimension indices.
Dim value1 as double, value2 as double  ' The data values.

try
    varNum = CDFgetVarNum (id, "MY_VAR")
    recNum = 0
    indices(0) = 0
    indices(1) = 0
    value1 = 10.1
    status = CDFvarPut (id, varNum, recNum, indices, value1)
    indices(0) = 1
    indices(1) = 1
    value2 = 20.2
    status = CDFvarPut (id, varNum, recNum, indices, value2)

catch ex as Exception
    ...
end try
```

### 4.3.115 CDFvarRename\textsuperscript{16}

```plaintext
type CDFvarRename
    id as long,  ' out -- Completion status code.
    varNum as integer,  ' in -- CDF identifier.
    varName as string  ' in -- rVariable number.
end type

CDFvarRename is used to rename an existing rVariable. A variable (rVariable or zVariable) name must be unique.

The arguments to CDFvarRename are defined as follows:
```
\textsuperscript{16} A legacy CDF function, handling rVariables only. While it is still available in V3.1, CDFrenameVar is the preferred function for it.
id The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopen.

varNum The rVariable number to rename. This number may be determined with a call to CDFvarNum.

varName The new rVariable name. The maximum length of the new name is CDF_VAR_NAME_LEN256 characters. Variable names are case-sensitive.

4.3.115.1. Example(s)

In the following example the rVariable named TEMPERATURE is renamed to TMP (if it exists). Note that if CDFvarNum returns a value less than zero (0) then that value is not an rVariable number but rather a warning/error code.

```plaintext
... 
... 
dim id as long
Dim status as integer
Dim varNum as integer
...
try
.... varNum = CDFvarNum (id, "TEMPERATURE")
...
} 
catch ex as Exception
...
end try
```

4.4 Attributes/Entries

This section provides functions that are related to CDF attributes or attribute entries. An attribute is identified by its name or an number in the CDF. Before you can perform any operation on an attribute or attribute entry, the CDF in which it resides must be opened.

4.4.1 CDFattrCreate\(^{17}\)

```plaintext
integer CDFattrCreate(
    id as long,
    attrName as string,
    attrScope as integer,
    attrNum as integer)
```

\(^{17}\) Same as CDFcreateAttr.
CDFattrCreate creates an attribute in the specified CDF. An attribute with the same name must not already exist in the CDF.

The arguments to CDFattrCreate are defined as follows:

- **id**: The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopen.
- **attrName**: The name of the attribute to create. This may be at most CDF_ATTR_NAME_LEN256 characters. Attribute names are case-sensitive.
- **attrScope**: The scope of the new attribute. Specify one of the scopes described in Section 2.13.
- **attrNum**: The number assigned to the new attribute. This number must be used in subsequent CDF function calls when referring to this attribute. An existing attribute's number may be determined with the CDFgetAttrNum function.

### 4.4.1.1. Example(s)

The following example creates two attributes. The TITLE attribute is created with global scope - it applies to the entire CDF (most likely the title of the data set stored in the CDF). The Units attribute is created with variable scope - each entry describes some property of the corresponding variable (in this case the units for the data).

```r
try
  ...
  status = CDFattrCreate (id, "TITLE", TITLEattrScope, TITLEattrNum)
  status = CDFattrCreate (id, UNITSattrName, VARIABLE_SCOPE, UNITSattrNum)
  ...
catch ex as Exception
  ...
end try
```

### 4.4.2   CDFattrEntryInquire

```r
integer CDFattrEntryInquire(  
  ' out -- Completion status code.
```
id as long,
attrNum as integer,
entryNum as integer,
dataType as integer,
numElements as integer)  

CDFAttrEntryInquire is used to inquire about a specific attribute entry. To inquire about the attribute in general, use CDFattrInquire. CDFAttrEntryInquire would normally be called before calling CDFattrGet in order to determine the data type and number of elements (of that data type) for an entry. This would be necessary to correctly allocate enough memory to receive the value read by CDFattrGet.

The arguments to CDFattrEntryInquire are defined as follows:

- **id**
  - The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopen.

- **attrNum**
  - The attribute number for which to inquire an entry. This number may be determined with a call to CDFattrNum (see Section 4.4.5).

- **entryNum**
  - The entry number to inquire. If the attribute is global in scope, this is simply the gEntry number and has meaning only to the application. If the attribute is variable in scope, this is the number of the associated rVariable (the rVariable being described in some way by the rEntry).

- **dataType**
  - The data type of the specified entry. The data types are defined in Section 2.6.

- **NumElements**
  - The number of elements of the data type. For character data types (CDF_CHAR and CDF_UCHAR), this is the number of characters in the string (An array of characters). For all other data types this is the number of elements in an array of that data type.

### 4.4.2.1. Example(s)

The following example returns each entry for an attribute. Note that entry numbers need not be consecutive - not every entry number between zero (0) and the maximum entry number must exist. For this reason NO_SUCH_ENTRY is an expected error code. Note also that if the attribute has variable scope, the entry numbers are actually rVariable numbers.

```plaintext
try
    dim id as long
    Dim status as integer
    Dim attrN as integer
    Dim entryN as integer
    Dim attrName as string
    Dim attrScope as integer
    Dim maxEntry as integer
    Dim dataType as integer
    Dim numElems as integer
    
    CDFAttrEntryInquire
```
...attrN = CDFgetAttrNum (id, "TMP")
status = CDFattrInquire (id, attrN, attrName, attrScope, maxEntry)

for entryN = 0 to maxEntry
    status = CDFattrEntryInquire (id, attrN, entryN, dataType, numElems)
    next entryN
...
}
catch ex as Exception
...end try

4.4.3 CDFattrGet

integer CDFattrGet(
    id as long,
    integer attrNum,
    integer entryNum,
    value as TYPE)

CDFattrGet is used to read an attribute entry from a CDF. In most cases it will be necessary to call CDFattrEntryInquire before calling CDFattrGet in order to determine the data type and number of elements (of that data type) for the entry.

The arguments to CDFattrGet are defined as follows:

id The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopen.

attrNum The attribute number. This number may be determined with a call to CDFattrNum (Section 4.4.5).

entryNum The entry number. If the attribute is global in scope, this is simply the gEntry number and has meaning only to the application. If the attribute is variable in scope, this is the number of the associated rVariable (the rVariable being described in some way by the rEntry).

value The value read. This buffer must be large enough to hold the value. The method CDFattrEntryInquire would be used to determine the entry data type and number of elements (of that data type). The value is read from the CDF and placed into memory at address value.

18 A legacy CDF function. While it is still available in V3.1, CDFgetAttrgEntry or CDFgetAttrrEntry is the preferred function for it.
4.4.3.1. Example(s)

The following example displays the value of the UNITS attribute for the rEntry corresponding to the PRES_LVL rVariable (but only if the data type is CDF_CHAR).

```plaintext
dim id as long
  ' CDF identifier.
dim status as integer
  ' Returned status code.
Dim attrN as integer
  ' Attribute number.
Dim entryN as integer
  ' Entry number.
Dim dataType as integer
  ' Data type.
Dim numElems as integer
  ' Number of elements (of data type).

try
  ...
  attrN = CDFattrNum (id, "UNITS")
  entryN = CDFvarNum (id, "PRES_LVL")
    ' The rEntry number is the rVariable number.
  status = CDFattrEntryInquire (id, attrN, entryN, dataType, numElems)
  if dataType = CDF_CHAR then
    dim buffer as string
    status = CDFattrGet (id, attrN, entryN, buffer)
  end if
  catch ex as Exception
    ...
end try
```

4.4.4 CDFattrInquire

```plaintext
integer CDFattrInquire(id as long,
  attrNum as integer,
  attrName as string,
  attrScope as integer,
  maxEntry as integer)
  ' out -- Completion status code.
  ' in -- CDF identifier.
  ' in -- Attribute number.
  ' out -- Attribute name.
  ' out -- Attribute scope.
  ' out -- Maximum gEntry/rEntry number.
```

CDFattrInquire is used to inquire about the specified attribute. To inquire about a specific attribute entry, use CDFattrEntryInquire.

The arguments to CDFattrInquire are defined as follows:

- `id` The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopen.

---

19 A legacy function. While it is still available in V3.1, CDFinquireAttr is the preferred function for it.
attrNum The number of the attribute to inquire. This number may be determined with a call to CDFattrNum (see Section 4.4.5).

attrName The attribute's name. This string length is limited to CDF_ATTR_NAME_LEN256.

attrScope The scope of the attribute. Attribute scopes are defined in Section 2.13.

maxEntry For gAttributes this is the maximum gEntry number used. For vAttributes this is the maximum rEntry number used. In either case this may not correspond with the number of entries (if some entry numbers were not used). If no entries exist for the attribute, then a value of -1 will be passed back.

4.4.4.1. Example(s)

The following example displays the name of each attribute in a CDF. The number of attributes in the CDF is first determined using the method CDFinquire. Note that attribute numbers start at zero (0) and are consecutive.

```pascal
try
  status = CDFinquire (id, numDims, dimSizes, encoding, majority, maxRec, numVars, numAttrs)
  for attrN = 0 to (numAttrs-1)
    status = CDFattrInquire (id, attrN, attrName, attrScope, maxEntry)
  next attrN
catch ex as Exception
  ...
end try
```

189
4.4.5  CDFattrNum\textsuperscript{20}

\begin{verbatim}
integer CDFattrNum(
  id as long,
  attrName as string)
\end{verbatim}

CDFattrNum is used to determine the attribute number associated with a given attribute name. If the attribute is found, CDFattrNum returns its number - which will be equal to or greater than zero (0). If an error occurs (e.g., the attribute name does not exist in the CDF), an error code (of type Int) is returned. Error codes are less than zero (0).

The arguments to CDFattrNum are defined as follows:

\begin{itemize}
  \item \texttt{id} The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopen.
  \item \texttt{attrName} The name of the attribute for which to search. This may be at most \texttt{CDF_ATT_NAME_LEN256} characters. Attribute names are case-sensitive.
\end{itemize}

CDFattrNum may be used as an embedded function call when an attribute number is needed.

4.4.5.1. Example(s)

In the following example the attribute named pressure will be renamed to PRESSURE with CDFattrNum being used as an embedded function call. Note that if the attribute pressure did not exist in the CDF, the call to CDFattrNum would have returned an error code. Passing that error code to CDFattrRename as an attribute number would have resulted in CDFattrRename also returning an error code.

\begin{verbatim}
... try ...
  status = CDFattrRename (id,  CDFattrNum (id, "pressure"),  "PRESSURE")
  ... catch ex as Exception ...
end try
\end{verbatim}

4.4.6  CDFattrPut

\begin{verbatim}
integer CDFattrPut(
  id as long,
)\end{verbatim}

\textsuperscript{20} A legacy CDF function. While it is still available in V3.1, CDFgetAttrNum is the preferred function for it.
CDFattrPut is used to write an entry to a global or rVariable attribute in a CDF. The entry may or may not already exist. If it does exist, it is overwritten. The data type and number of elements (of that data type) may be changed when overwriting an existing entry.

The arguments to CDFattrPut are defined as follows:

- **id**
  - The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopen.

- **attrNum**
  - The attribute number. This number may be determined with a call to CDFgetAttrNum.

- **entryNum**
  - The entry number. If the attribute is global in scope, this is simply the gEntry number and has meaning only to the application. If the attribute is variable in scope, this is the number of the associated rVariable (the rVariable being described in some way by the rEntry).

- **dataType**
  - The data type of the specified entry. Specify one of the data types defined in Section 2.6.

- **numElements**
  - The number of elements of the data type. For character data types (CDF_CHAR and CDF_UCHAR), this is the number of characters in the string (an array of characters). For all other data types this is the number of elements in an array of that data type.

- **value**
  - The value(s) to write. The entry value is written to the CDF from memory address value.

### 4.4.6.1. Example(s)

The following example writes two attribute entries. The first is to gEntry number zero (0) of the gAttribute TITLE. The second is to the variable scope attribute VALIDs for the rEntry that corresponds to the rVariable TMP.

```vbnet
' CDF identifier.
' Returned status code.
' Entry string length.
' Entry number.
' Number of elements (of data type).
' Value of TITLE attribute, entry number 0.
' Value(s) of VALIDs attribute,
' rEntry for rVariable TMP.

dim id as long
Dim status as integer
Dim TITLE_LEN as integer = 10
Dim entryNum as integer
Dim numElements as integer
Dim title as string = "CDF title."
Dim TMPvalids() as short = {15,30}

entryNum = 0
```
try
    status = CDFattrPut (id, CDFgetAttrNum (id,"TITLE"), entryNum, CDF_CHAR, TITLE_LEN, title)
    numElements = 2
    status = CDFattrPut (id, CDFgetAttrNum (id,"VALIDs"), CDFgetVarNum (id,"TMP"), _
                        CDF_INT2, numElements, TMPvalids)
    catch ex as Exception
        ...
    end try

4.4.7 CDFattrRename

integer CDFattrRename(
    id as long,
    attrNum as integer,
    attrName as string)

CDFattrRename is used to rename an existing attribute. An attribute with the new name must not already exist in the CDF.

The arguments to CDFattrRename are defined as follows:

id            The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopen.
attrNum        The number of the attribute to rename. This number may be determined with a call to CDFattrNum (see Section 4.4.5).
attrName       The new attribute name. This may be at most CDF_ATTR_NAME_LEN256 characters. Attribute names are case-sensitive.

4.4.7.1. Example(s)

In the following example the attribute named LAT is renamed to LATITUDE.

    dim id as long
    Dim status as integer
    try
        status = CDFattrRename (id, CDFgetAttrNum (id,"LAT"), "LATITUDE")

21 A legacy CDF function. While it is still available in V3.1, CDFrenameAttr is the preferred function for it.
catch ex as Exception
...
end try

4.4.8 CDFconfirmAttrExistence

integer CDFconfirmAttrExistence( id as long, attrName as string)    

CDFconfirmAttrExistence confirms whether an attribute exists for the given attribute name in a CDF. If the attribute doesn’t exist, the informational status code, NO_SUCH_ATTR, is returned and no exception is thrown.

The arguments to CDFconfirmAttrExistence are defined as follows:

id The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

attrName The attribute name to check.

4.4.8.1. Example(s)

The following example checks whether an attribute by the name of “ATTR_NAME1” is in a CDF.

dim id as long
Dim status as integer

try
    status = CDFconfirmAttrExistence (id, “ATTR_NAME1”)  
    if status = NO_SUCH_ATTR then
        ....
        end if
    catch ex as Exception
        ...
    end try
### 4.4.9 CDFconfirmgEntryExistence

```plaintext
integer CDFconfirmgEntryExistence(
  id as long,
  attrNum as integer,
  entryNum as integer)
```

CDFconfirmgEntryExistence confirms the existence of the specified entry (gEntry), in a global attribute from a CDF. If the gEntry does not exist, the informational status code NO_SUCH_ENTRY will be returned and no exception is thrown.

The arguments to CDFconfirmgEntryExistence are defined as follows:

- **id**: The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **attrNum**: The (global) attribute number.
- **entryNum**: The gEntry number.

#### 4.4.9.1. Example(s)

The following example checks the existence of a gEntry numbered 1 for attribute “MY_ATTR” in a CDF.

```plaintext
dim id as long
Dim status as integer
dim attrNum as integer
Dim entryNum as integer

try
  attrNum = CDFgetAttrNum (id, “MY_ATTR”)  
  entryNum = 1
  status = CDFconfirmgEntryExistence (id, attrNum, entryNum)
  if status = NO_SUCH_ENTRY then UserStatusHandler (status)

```

### 4.4.10 CDFconfirmrEntryExistence

```plaintext
integer CDFconfirmrEntryExistence(
  id as long,
  attrNum as integer,
  entryNum as integer)
```

CDFconfirmrEntryExistence confirms the existence of the specified entry (rEntry), in a global attribute from a CDF. If the gEntry does not exist, the informational status code NO_SUCH_ENTRY will be returned and no exception is thrown.

The arguments to CDFconfirmrEntryExistence are defined as follows:

- **id**: The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **attrNum**: The (global) attribute number.
- **entryNum**: The rEntry number.
CDFconfirmrEntryExistence confirms the existence of the specified entry (rEntry), corresponding to an rVariable, in a variable attribute from a CDF. If the rEntry does not exist, the informational status code NO_SUCH_ENTRY will be returned and no exception is thrown.

The arguments to CDFconfirmrEntryExistence are defined as follows:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.</td>
</tr>
<tr>
<td>attrNum</td>
<td>The variable attribute number.</td>
</tr>
<tr>
<td>entryNum</td>
<td>The rEntry number.</td>
</tr>
</tbody>
</table>

### 4.4.10.1. Example(s)

The following example checks the existence of an rEntry, corresponding to rVariable “MY_VAR”, for attribute “MY_ATTR” in a CDF.

```plaintext
try
  attrNum = CDFgetAttrNum(id, “MY_ATTR”)
  entryNum = CDFgetVarNum(id, “MY_VAR”)
  status = CDFconfirmrEntryExistence(id, attrNum, entryNum)
  if status = NO_SUCH_ENTRY then UserStatusHandler(status)
end try
```

### 4.4.11 CDFconfirmzEntryExistence

integer CDFconfirmzEntryExistence(id as long, attrNum as integer, entryNum as integer)
CDFconfirmzEntryExistence confirms the existence of the specified entry (zEntry), corresponding to a zVariable, in a variable attribute from a CDF. If the zEntry does not exist, the informational status code NO_SUCH_ENTRY will be returned and no exception is thrown.

The arguments to CDFconfirmzEntryExistence are defined as follows:

- \textit{id} \quad The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- \textit{attrNum} \quad The (variable) attribute number.
- \textit{entryNum} \quad The zEntry number.

### 4.4.11.1. Example(s)

The following example checks the existence of the zEntry corresponding to zVariable “MY_VAR” for the variable attribute “MY_ATTR” in a CDF.

```plaintext
.
.
.
dim id as long
Dim status as integer
dim varNum as integer
dim entryNum as integer
.
.
try
    ...
    attrNum = CDFgetAttrNum (id, “MY_ATTR”)
    entryNum = CDFgetVarNum (id, “MY_VAR”)
    status = CDFconfirmzEntryExistence (id, attrNum, entryNum)
    if status = NO_SUCH_ENTRY then UserStatusHandler (status)
.
catch ex as Exception
    ...
end try
.
```

### 4.4.12  CDFcreateAttr

```plaintext
integer CDFcreateAttr(
    id as long,
    attrName as string,
    attrScope as integer,
    attrNum as integer)
```

CDFcreateAttr creates an attribute with the specified scope in a CDF. It is identical to the method CDFattrCreate. An attribute with the same name must not already exist in the CDF.
The arguments to CDFcreateAttr are defined as follows:

- **id**: The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **attrName**: The name of the attribute to create. This may be at most CDF_ATTR_NAME_LEN256 characters. Attribute names are case-sensitive.
- **attrScope**: The scope of the new attribute. Specify one of the scopes described in Section 2.13.
- **attrNum**: The number assigned to the new attribute. This number must be used in subsequent CDF function calls when referring to this attribute. An existing attribute's number may be determined with the CDFgetAttrNum function.

### 4.4.12.1. Example(s)

The following example creates two attributes. The TITLE attribute is created with global scope - it applies to the entire CDF (most likely the title of the data set stored in the CDF). The Units attribute is created with variable scope - each entry describes some property of the corresponding variable (in this case the units for the data).

```plaintext
try
    status = CDFcreateAttr (id, "TITLE", TITLEattrScope, TITLEattrNum)
    status = CDFcreateAttr (id, UNITSattrName, VARIABLE_SCOPE, UNITSattrNum)
...catch ex as Exception
    ...
end try
```

### 4.4.13 CDFdeleteAttr

integer CDFdeleteAttr(
    id as long,
    attrNum as integer)

CDFdeleteAttr deletes the specified attribute from a CDF.

The arguments to CDFdeleteAttr are defined as follows:

- **id**: CDF identifier.
- **attrNum**: Returned status code.
- **attrName**: Name of "Units" attribute.
- **attrScope**: "Units" attribute number.
- **attrNum**: "TITLE" attribute number.
- **attrNum**: "TITLE" attribute scope.
id The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

attrNum The attribute number to be deleted.

4.4.13.1. Example(s)

The following example deletes an existing attribute named MY_ATTR from a CDF.

```vba
    try
        attrNum = CDFgetAttrNum (id, “MY_ATTR”)  ‘ CDF identifier.
        status = CDFdeleteAttr (id, attrNum)  ‘ Returned status code.
    catch ex as Exception  ‘ Attribute number.
        …
    end try
```

4.4.14 CDFdeleteAttrgEntry

integer CDFdeleteAttrgEntry(
    id as long,  ‘ out -- Completion status code.
    attrNum as integer,  ‘ in -- CDF identifier.
    entryNum as integer)  ‘ in -- Attribute identifier.

CDFdeleteAttrgEntry deletes the specified entry (gEntry) in a global attribute from a CDF.

The arguments to CDFdeleteAttrgEntry are defined as follows:

id The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

attrNum The global attribute number from which to delete an attribute entry.

entryNum The gEntry number to delete.
4.4.14.1. Example(s)

The following example deletes the entry number 5 from an existing global attribute MY_ATTR in a CDF.

```vba
    .
    .
    dim id as long ' CDF identifier.
    Dim status as integer ' Returned status code.
    dim varNum as integer ' Attribute number.
    dim entryNum as integer ' gEntry number.
    .
    try
        ....
        attrNum = CDFgetAttrNum (id, "MY_ATTR")
        entryNum = 5
        status = CDFdeleteAttrgEntry (id, attrNum, entryNum)
    .
    catch ex as Exception
        ...
    end try
    .
```

4.4.15 CDFdeleteAttrrEntry

```vba
integer CDFdeleteAttrrEntry( id as long, ' out -- Completion status code.
attrNum as integer, ' in -- CDF identifier.
entryNum as integer) ' in -- Attribute identifier.
    ' in -- rEntry identifier.
```

CDFdeleteAttrrEntry deletes the specified entry (rEntry), corresponding to an rVariable, in an (variable) attribute from a CDF.

The arguments to CDFdeleteAttrrEntry are defined as follows:

- **id** The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **attrNum** The (variable) attribute number.
- **entryNum** The rEntry number.

4.4.15.1. Example(s)

The following example deletes the entry corresponding to rVariable “MY_VAR1” from the variable attribute “MY_ATTR” in a CDF.
try
    ....
    attrNum = CDFgetAttrNum (id, “MY_ATTR”)
    entryNum = CDFgetVarNum (id, “MY_VAR1”)
    status = CDFdeleteAttrrEntry (id, attrNum, entryNum)
    catch ex as Exception
    ...
end try

4.4.16 CDFdeleteAttrzEntry

integer CDFdeleteAttrzEntry(id as long, attrNum as integer, entryNum as integer)

CDFdeleteAttrzEntry deletes the specified entry (zEntry), corresponding to a zVariable, in an (variable) attribute from a CDF.

The arguments to CDFdeleteAttrzEntry are defined as follows:

id The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

attrNum The identifier of the variable attribute.

entryNum The zEntry number to be deleted that is the zVariable number.

4.4.16.1. Example(s)

The following example deletes the variable attribute entry named MY_ATTR that is attached to the zVariable MY_VAR1.
Dim status as integer  
Dim attrNum as integer  
Dim entryNum as integer  

try
    ....
    attrNum = CDFgetAttrNum (id, “MY_ATTR”)  
    entryNum = CDFgetVarNum (id, “MY_VAR1”)  
    status = CDFdeleteAttrzEntry (id, attrNum, entryNum)

catch ex as Exception
    ...
end try

4.4.17  CDFgetAttrgEntry

integer CDFgetAttrgEntry (  
id as long,  
attrNum as integer,  
entryNum as integer,  
value as TYPE)

This method is identical to the method CDFattrGet.  CDFgetAttrgEntry is used to read a global attribute entry from a CDF.  In most cases it will be necessary to call CDFinquireAttrgEntry before calling CDFgetAttrgEntry in order to determine the data type and number of elements (of that data type) for the entry.

The arguments to CDFgetAttrgEntry are defined as follows:

id         The identifier of the CDF.  This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
attrNum    The attribute number.  This number may be determined with a call to CDFgetAttrNum.
entryNum   The global attribute entry number.
value      The value read.

4.4.17.1. Example(s)

The following example displays the value of the global attribute called HISTORY.

Dim id as long  
Dim status as integer  

' CDF identifier.
' Returned status code.
Dim attrN as integer  
Dim entryN as integer  
Dim dataType as integer  
Dim numElems as integer  
Dim buffer as Object  
' Attribute number.
' Entry number.
' Data type.
' Number of elements (of data type).
' Buffer to receive value.

try
....
attrN  =  CDFattrNum (id, "HISTORY")
entryN  =  0
status =  CDFinquireAttrgEntry (id, attrN, entryN, dataType, numElems)
status =  CDFgetAttrgEntry (id, attrN, entryN, buffer)
if dataType  =  CDF_CHAR then
' buffer is a string

end if
.
catch ex as Exception
...
end try
.

4.4.18 CDFgetAttrgEntryDataType

integer CDFgetAttrgEntryDataType (  
id as long,
attrNum as integer,
entryNum as integer,
dataType as integer)
' out -- Completion status code.
' in -- CDF identifier.
' in -- Attribute identifier.
' in -- gEntry number.
' out -- gEntry data type.

CDFgetAttrgEntryDataType returns the data type of the specified global attribute and gEntry number in a CDF. The data types are described in Section 2.6.

The arguments to CDFgetAttrgEntryDataType are defined as follows:

id The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
attrNum The global attribute number.
entryNum The gEntry number.
dataType The data type of the gEntry.

4.4.18.1. Example(s)

The following example gets the data type for the gEntry numbered 2 from the global attribute “MY_ATTR” in a CDF.
try
    attrNum = CDFgetAttrNum (id, “MY_ATTR”)
    entryNum = 2
    status = CDFgetAttrgEntryDataType (id, attrNum, entryNum, dataType)
    ...
catch ex as Exception
    ...
end try

4.4.19 CDFgetAttrgEntryNumElements

integer CDFgetAttrgEntryNumElements (id as long, attrNum as integer, entryNum as integer, numElems as integer) · out -- Completion status code.
             · in -- CDF identifier.
             · in -- Attribute identifier.
             · in -- gEntry number.
             · out -- gEntry’s number of elements.

CDFgetAttrgEntryNumElements returns the number of elements of the specified global attribute and gentry number in a CDF.

The arguments to CDFgetAttrgEntryNumElements are defined as follows:

id The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
attrNum The identifier of the global attribute.
entryNum The gEntry number.
numElems The number of elements of the gEntry.

4.4.19.1. Example(s)

The following example gets the number of elements from the gEntry numbered 2 from the global attribute “MY_ATTR” in a CDF.
dim id as long
Dim status as integer
dim attrNum as integer
dim entryNum as integer
dim numElements as integer

try
    ....
    attrNum = CDFgetAttrNum (id, "MY_ATTR")
    entryNum = 2
    status = CDFgetAttrEntryNumElements (id, attrNum, entryNum, numElements)

catch ex as Exception
    ...
end try

4.4.20 CDFgetAttrMaxEntry

integer CDFgetAttrMaxEntry ( id as long,
attrNum as integer,
maxEntry as integer)

CDFgetAttrMaxEntry returns the last entry number of the specified global attribute in a CDF.

The arguments to CDFgetAttrMaxEntry are defined as follows:

id The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

attrNum The identifier of the global attribute.

maxEntry The last gEntry number.

4.4.20.1. Example(s)

The following example gets the last entry number from the global attribute “MY_ATTR” in a CDF.

...
try
....
attrNum = CDFgetAttrNum (id, “MY_ATTR”)
status = CDFgetAttrMaxrEntry (id, attrNum, maxEntry)
.
catch ex as Exception
...
end try
.

4.4.21 CDFgetAttrMaxrEntry

integer CDFgetAttrMaxrEntry (id as long,
attrNum as integer,
maxEntry as integer)

CDFgetAttrMaxrEntry returns the last rEntry number (rVariable number) to which the given variable attribute is attached.

The arguments to CDFgetAttrMaxrEntry are defined as follows:

- id: The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- attrNum: The identifier of the variable attribute.
- maxEntry: The last rEntry number (rVariable number) to which attrNum is attached.

4.4.21.1. Example(s)

The following example gets the last entry, corresponding to the last rVariable number, from the variable attribute “MY_ATTR” in a CDF.

try
....
attrNum = CDFgetAttrNum (id, “MY_ATTR”)
status = CDFgetAttrMaxEntry (id, attrNum, maxEntry)

catch ex as Exception
  ...
end try

4.4.22  CDFgetAttrMaxzEntry

integer CDFgetAttrMaxzEntry (  
id as long,  
attrNum as integer,  
maxEntry as integer)
  ‘ out -- Completion status code.
  ‘ in -- CDF identifier.
  ‘ in -- Attribute identifier.
  ‘ out -- The maximum zEntry number.

CDFgetAttrMaxzEntry returns the last entry number, corresponding to the last zVariable number, to which the given variable attribute is attached.

The arguments to CDFgetAttrMaxzEntry are defined as follows:

  id     The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
          
  attrNum The identifier of the variable attribute.
          
  maxEntry The last zEntry number (zVariable number) to which attrNum is attached.

4.4.22.1. Example(s)

The following example gets the last entry, corresponding to the last zVariable number, attached to the variable attribute MY_ATTR in a CDF.

dim id as long
Dim status as integer
dim attrNum as integer
dim maxEntry as integer

try
  ...
  attrNum = CDFgetAttrNum (id, “MY_ATTR”)  
  status = CDFgetAttrMaxzEntry (id, attrNum, maxEntry)

catch ex as Exception
4.4.23 CDFgetAttrName

integer CDFgetAttrName (id as long, attrNum as integer, attrName as string)  ' out -- Completion status code.

CDFgetAttrName gets the name of the specified attribute (by its number) in a CDF.

The arguments to CDFgetAttrName are defined as follows:

  id          The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
  attrNum     The identifier of the attribute.
  attrName    The name of the attribute.

4.4.23.1. Example(s)

The following example retrieves the name of the attribute number 2, if it exists, in a CDF.

        ....
    attrNum = 2
    try
        ....
        status = CDFgetAttrName (id, attrNum, attrName)
    catch ex as Exception
        ....
    end try

### 4.4.24 CDFgetAttrNum

integer CDFgetAttrNum ( 
  id as long, 
  attrName as string) 
  `out -- Attribute number.
  `in -- CDF identifier.
  `in -- The attribute name.

CDFgetAttrNum is used to determine the attribute number associated with a given attribute name. If the attribute is found, CDFgetAttrNum returns its number - which will be equal to or greater than zero (0). If an error occurs (e.g., the attribute name does not exist in the CDF), an error code (of type Int) is returned. Error codes are less than zero (0).

The arguments to CDFgetAttrNum are defined as follows:

- **id**
  - The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

- **attrName**
  - The name of the attribute for which to search. This may be at most CDF_ATTR_NAME_LEN256 characters. Attribute names are case-sensitive.

CDFgetAttrNum may be used as an embedded function call when an attribute number is needed.

#### 4.4.24.1. Example(s)

In the following example the attribute named pressure will be renamed to PRESSURE with CDFgetAttrNum being used as an embedded function call. Note that if the attribute pressure did not exist in the CDF, the call to CDFgetAttrNum would have returned an error code. Passing that error code to CDFattrRename as an attribute number would have resulted in CDFattrRename also returning an error code.

```plaintext
try
  status = CDFrenameAttr (id, CDFgetAttrNum (id,"pressure"), "PRESSURE")
catch ex as Exception
  ...
end try
```

### 4.4.25 CDFgetAttrEntry

integer CDFgetAttrEntry ( 
  id as long, 
  attrNum as integer, 
  entryNum as integer, 
  `out -- Completion status code.
  `in -- CDF identifier.
  `in -- Attribute identifier.
  `in -- Entry number.
This method is identical to the method CDFattrGet. CDFgetAttrrEntry is used to read an rVariable attribute entry from a CDF. In most cases it will be necessary to call CDFinquireAttrrEntry before calling CDFgetAttrrEntry in order to determine the data type and number of elements (of that data type) for the entry.

The arguments to CDFgetAttrrEntry are defined as follows:

- **id**
  - The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

- **attrNum**
  - The attribute number. This number may be determined with a call to CDFgetAttrNum.

- **entryNum**
  - The rVariable attribute entry number that is the rVariable number from which the attribute is read.

- **value**
  - The entry value read.

### 4.4.25.1. Example(s)

The following example displays the value of the UNITS attribute for the rEntry corresponding to the PRES_LVL rVariable (but only if the data type is CDF_CHAR).

```plaintext
dim id as long
Dim status as integer
Dim attrN as integer
Dim entryN as integer
Dim dataType as integer
Dim numElems as integer
try
...
attrN = CDFattrNum (id, "UNITS")
entryN = CDFvarNum (id, "PRES_LVL") ' The rEntry number is the rVariable number.
status = CDFinquireAttrrEntry (id, attrN, entryN, out dataType, out numElems)
if dataType = CDF_CHAR then
    Dim buffer as string
    status = CDFgetAttrrEntry (id, attrN, entryN, buffer)
end if.
catch ex as Exception
...
end try
```

' out -- Entry data.
' TYPE -- VB value/string type or object
4.4.26 CDFgetAttrrEntryDataType

integer CDFgetAttrrEntryDataType ( id as long,
attrNum as integer,
entryNum as integer,
dataType as integer)

CDFgetAttrrEntryDataType returns the data type of the rEntry from an (variable) attribute in a CDF. The data types are described in Section 2.6.

The arguments to CDFgetAttrrEntryDataType are defined as follows:

- **id**  The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **attrNum**  The identifier of the variable attribute.
- **entryNum**  The rEntry number.
- **dataType**  The data type of the rEntry.

4.4.26.1. Example(s)

The following example gets the data type for the entry of rVariable “MY_VAR1” in the (variable) attribute “MY_ATTR” in a CDF.

```
try
    attrNum = CDFgetAttrNum (id, "MY_ATTR")
    entryNum = CDFgetVarNum (id, "MY_VAR1")
    status = CDFgetAttrrEntryDataType (id, attrNum, entryNum, dataType)
    ...
catch ex as Exception
    ...
end try
```
### 4.4.27 CDFgetAttrrEntryNumElements

```c
integer CDFgetAttrrEntryNumElements (  
    id as long,  
    attrNum as integer,  
    startRec as integer,  
    numElems as integer)  
    
    ' out -- Completion status code.  
    ' in -- CDF identifier.  
    ' in -- Attribute identifier.  
    ' in -- rEntry number.  
    ' out -- rEntry’s number of elements.
```

CDFgetAttrrEntryNumElements returns the number of elements of the rEntry from an (variable) attribute in a CDF.

The arguments to CDFgetAttrrEntryNumElements are defined as follows:

- **id**: The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **attrNum**: The identifier of the variable attribute.
- **entryNum**: The rEntry number.
- **numElems**: The number of elements of the rEntry.

#### 4.4.27.1. Example(s)

The following example gets the number of elements for the entry of rVariable “MY_VAR1” in the (variable) attribute “MY_ATTR” in a CDF.

```c
try
    attrNum = CDFgetAttrNum (id, “MY_ATTR”)  
    entryNum = CDFgetVarNum (id, “MY_VAR1”)  
    status = CDFgetAttrrEntryNumElements (id, attrNum, entryNum, numElems)  

    numElements = status  

catch ex as Exception  
    …
end try
```
4.4.28  CDFgetAttrScope

integer CDFgetAttrScope (  
id as long,  
attrNum as integer,  
attrScope as integer)  

CDFgetAttrScope returns the attribute scope (GLOBAL_SCOPE or VARIABLE_SCOPE) of the specified attribute in a CDF. Refer to Section 2.13 for the description of the attribute scopes.

The arguments to CDFgetAttrScope are defined as follows:

    id           The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
    attrNum      The attribute number.
    attrScope    The scope of the attribute.

4.4.28.1. Example(s)

The following example gets the scope of the attribute “MY_ATTR” in a CDF.

```
  try
    attrNum = CDFgetAttrNum (id, “MY_ATTR”)  
    status = CDFgetAttrScope (id, attrNum, attrScope)
  catch ex as Exception
    …
  end try
```

4.4.29  CDFgetAttrzEntry

integer CDFgetAttrzEntry(  

    ‘ out -- Completion status code.

    ‘ in -- CDF identifier.
    ‘ in -- Attribute number.
    ‘ out -- Attribute scope.
CDFgetAttrzEntry is used to read zVariable’s attribute entry. In most cases it will be necessary to call CDFinquireAttrzEntry before calling this method in order to determine the data type and number of elements (of that data type) for the entry.

The arguments to CDFgetAttrzEntry are defined as follows:

- **id**
  - The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

- **attrNum**
  - The variable attribute number. This number may be determined with a call to CDFgetAttrNum.

- **entryNum**
  - The variable attribute entry number that is the zVariable number from which the attribute entry is read.

- **value**
  - The entry value read.

### 4.4.29.1. Example(s)

The following example displays the value of the UNITS attribute for the PRES_LVL zVariable (but only if the data type is CDF_CHAR).

```plaintext
dim id as long  ' CDF identifier.
dim status as integer  ' Returned status code.
dim attrN as integer  ' Attribute number.
dim entryN as integer  ' Entry number.
dim dataType as integer  ' Data type.
dim numElems as integer  ' Number of elements (of data type).

try
  attrN  =  CDFgetAttrNum  (id,  "UNITS")  ' The zEntry number is the zVariable number.
  entryN  =  CDFgetVarNum  (id,  "PRES_LVL")  ' The zEntry number is the zVariable number.
  status =  CDFinquireAttrzEntry  (id,  attrN,  entryN,  dataType,  numElems)
  if dataType  =  CDF_CHAR then
    dim buffer as string
    status =  CDFgetAttrzEntry  (id,  attrN,  entryN,  buffer)
  end if
  catch ex as Exception
    ...
  end try
```
4.4.30 CDFgetAttrzEntryDataType

integer CDFgetAttrzEntryDataType ( id as long, attrNum as integer, entryNum as integer, dataType as integer)  

CDFgetAttrzEntryDataType returns the data type of the zEntry for the specified variable attribute in a CDF. The data types are described in Section 2.6.

The arguments to CDFgetAttrzEntryDataType are defined as follows:

id The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
attrNum The identifier of the variable attribute.
entryNum The zEntry number that is the zVariable number.
dataType The data type of the zEntry.

4.4.30.1. Example(s)

The following example gets the data type of the attribute named MY_ATTR for the zVariable MY_VAR1 in a CDF.

dim id as long
Dim status as integer
dim attrNum as integer
dim entryNum as integer
dim dataType as integer

try
    attrNum = CDFgetAttrNum (id, “MY_ATTR”)
    entryNum = CDFgetVarNum (id, “MY_VAR1”)
    status = CDFgetAttrzEntryDataType (id, attrNum, entryNum, dataType)

catch ex as Exception
    …
end try
4.4.31  CDFgetAttrzEntryNumElements

integer CDFgetAttrzEntryNumElements (id as long, attrNum as integer, entryNum as integer, numElems as integer)  

CDFgetAttrzEntryNumElements returns the number of elements of the zEntry for the specified variable attribute in a CDF.

The arguments to CDFgetAttrzEntryNumElements are defined as follows:

- **id**: The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **attrNum**: The identifier of the variable attribute.
- **entryNum**: The zEntry number that is the zVariable number.
- **numElems**: The number of elements of the zEntry.

4.4.31.1. Example(s)

The following example returns the number of elements for attribute named MY_ATTR for the zVariable MY_VAR1 in a CDF:

```
dim id as long
Dim status as integer
dim attrNum as integer
dim entryNum as integer
dim numElements as integer

try
    attrNum = CDFgetAttrNum (id, “MY_ATTR”)
    entryNum = CDFgetVarNum (id, “MY_VAR1”)
    status = CDFgetAttrzEntryNumElements (id, attrNum, entryNum, out numElements)

catch ex as Exception
    ...
end try
```
4.4.32 CDFgetNumAttrgEntries

integer CDFgetNumAttrgEntries (  
    id as long,  
    attrNum as integer,  
    entries as integer)  

    ' out -- Completion status code.  
    ' in -- CDF identifier.  
    ' in -- Attribute number.  
    ' out -- Total gEntries.

CDFgetNumAttrgEntries returns the total number of entries (gEntries) written for the specified global attribute in a CDF.

The arguments to CDFgetNumAttrgEntries are defined as follows:

id  
    The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

attrNum  
    The attribute number.

entries  
    Number of gEntries for attrNum.

4.4.32.1. Example(s)

The following example retrieves the total number of gEntries for the global attribute MY_ATTR in a CDF.

    
    dim status as integer  
    dim id as long  
    Dim attrNum as integer  
    Dim numEntries as integer  
    Dim i as integer  

    try  
      attrNum = CDFgetAttrNum (id, “MUY_ATTR”)  
      status = CDFgetNumAttrgEntries (id, attrNum, numEntries)  
      for i=0 to (numEntries-1)  
          ' process an entry  
          next i  
    catch ex as Exception  
      ...  
    end try  

4.4.33  CDFgetNumAttributes

integer CDFgetNumAttributes (  
id as long,  
numAttrs as integer)  

CDFgetNumAttributes returns the total number of global and variable attributes in a CDF.

The arguments to CDFgetNumAttributes are defined as follows:

id  
The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

numAttrs  
The total number of global and variable attributes.

4.4.33.1. Example(s)

The following example returns the total number of global and variable attributes in a CDF.

```plaintext

dim status as integer  
dim id as long  
dim numAttrs as integer  

try  
    ....  
    status = CDFgetNumAttributes (id, out numAttrs)  

catch ex as Exception  
    ...  
end try  
```

4.4.34  CDFgetNumAttrrEntries

integer CDFgetNumAttrrEntries (  
id as long,  
attrNum as integer)  

CDFgetNumAttrrEntries returns the total number of global and variable attributes in a CDF.

The arguments to CDFgetNumAttrrEntries are defined as follows:

id  
The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

attrNum  
The attribute number.
CDFgetNumAttrEntries returns the total number of entries (rEntries) written for the rVariables in the specified (variable) attribute of a CDF.

The arguments to CDFgetNumAttrEntries are defined as follows:

- **id** The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **attrNum** The attribute number.
- **entries** Total rEntries.

### 4.4.34.1. Example(s)

The following example returns the total number of rEntries from the variable attribute “MY_ATTR” in a CDF.

```c
... dim status as integer
    dim id as long
    dim attrNum as integer
    dim entries as integer
    ...
    try
        attrNum = CDFgetAttrNum (id, “MY_ATTR”)  ' Returned status code.
        status = CDFgetNumAttrEntries (id, attrNum, entries)  ' Attribute number.
        ' Number of entries.
    ...
    catch ex as Exception
        ...
    end try
```

### 4.4.35 CDFgetNumAttrzEntries

integer CDFgetNumAttrzEntries (  
    id as long,  
    attrNum as integer,  
    entries as integer)  
    ' out -- Completion status code.  
    ' in -- CDF identifier.  
    ' in -- Attribute number.  
    ' out -- Total zEntries.

CDFgetNumAttrzEntries returns the total number of entries (zEntries) written for the zVariables in the specified variable attribute in a CDF.
The arguments to CDFgetNumAttrzEntries are defined as follows:

- **id**: The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **attrNum**: The attribute number.
- **entries**: Total zEntries.

### 4.4.35.1. Example(s)

The following example returns the total number of zEntries for the variable attribute MY_ATTR in a CDF.

```
....
dim status as integer
  ' Returned status code.
dim id as long
  ' CDF identifier.
dim attrNum as integer
  ' Attribute number.
dim entries as integer
  ' Number of entries.
.
try
  ....
  attrNum = CDFgetAttrNum (id, "MY_ATTR")
  status = CDFgetNumAttrzEntries (id, attrNum, entries)
.
catch ex as Exception
  ...
end try
```

### 4.4.36 CDFgetNumgAttributes

```
integer CDFgetNumgAttributes ( id as long,
numAttrs as integer)  
  ' out -- Completion status code.
  ' in -- CDF identifier.
  ' out -- Total number of global attributes.
```

CDFgetNumgAttributes returns the total number of global attributes in a CDF.

The arguments to CDFgetNumgAttributes are defined as follows:

- **id**: The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **numAttrs**: The number of global attributes.
4.4.36.1. Example(s)

The following example returns the total number of global attributes in a CDF.

```plaintext
dim status as integer
    ' Returned status code.
dim id as long
    ' CDF identifier.
dim numAttrs as integer
    ' Number of global attributes.

try
    ....
    status = CDFgetNumgAttributes (id, numAttrs)

catch ex as Exception
    ...
end try
```

4.4.37  CDFgetNumvAttributes

```plaintext
integer CDFgetNumvAttributes (                          ' out -- Completion status code.
    id as long,                                    ' in --  CDF identifier.
    numAttrs as integer)                           ' out --  Total number of variable attributes.
```

CDFgetNumvAttributes returns the total number of variable attributes in a CDF.

The arguments to CDFgetNumvAttributes are defined as follows:

- `id` The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- `numAttr` The number of variable attributes.

4.4.37.1. Example(s)

The following example returns the total number of variable attributes of a CDF.


dim status as integer       ' Returned status code.
dim id as long             ' CDF identifier.
dim numAttrs as integer   ' Number of variable attributes.

try
....
    status = CDFgetNumvAttributes (id, numAttrs)
catch ex as Exception
    ...
end try

4.4.38  CDFinquireAttr

integer CDFinquireAttr(    ' out -- Completion status code.
    id as long,       ' in -- CDF identifier.
    attrNum as integer,  ' in -- Attribute number.
    attrName as string,  ' out -- Attribute name.
    attrScope as integer, ' out -- Attribute scope.
    maxgEntry as integer,  ' out -- Maximum gEntry number.
    maxrEntry as integer,  ' out -- Maximum rEntry number.
    maxzEntry as integer)  ' out -- Maximum zEntry number.

CDFinquireAttr is used to inquire information about the specified attribute. This method expands the method CDFattrInquire to provide an extra information about zEntry if the attribute has a variable scope.

The arguments to CDFinquireAttr are defined as follows:

id               The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
attrNum           The attribute number to inquire. This number may be determined with a call to CDFgetAttrNum.
attrName          The attribute's name that corresponds to attrNum. This string length is limited to CDF_ATTR_NAME_LEN256.
attrScope         The scope of the attribute (GLOBAL_SCOPE or VARIABLE_SCOPE). Attribute scopes are defined in Section 2.13.
maxgEntry         For vAttributes, this value of this field is -1 as it doesn't apply to global attribute entry (gEntry). For gAttributes, this is the maximum entry (gentry) number used. This number may not correspond with the number of entries (if some entry numbers were not used). If no entries exist for the attribute, then the value of -1 is returned.
maxrEntry
For gAttributes, this value of this field is -1 as it doesn’t apply to rVariable attribute entry (rEntry). For vAttributes, this is the maximum rVariable attribute entry (rEntry) number used. This number may not correspond with the number of entries (if some entry numbers were not used). If no entries exist for the attribute, then the value of -1 is returned.

maxzEntry
For gAttributes, this value of this field is -1 as it doesn’t apply to zVariable attribute entry (zEntry). For vAttributes, this is the maximum zVariable attribute entry (zEntry) number used. This may not correspond with the number of entries (if some entry numbers were not used). If no entries exist for the attribute, then the value of -1 is returned.

4.4.38.1. Example(s)

The following example displays the name of each attribute in a CDF. The number of attributes in the CDF is first determined by calling the method CDFinquireCDF. Note that attribute numbers start at zero (0) and are consecutive.

```
try
   dim id as long
   Dim status as integer
   Dim numDims as integer
   Dim dimSizes() as integer
   Dim encoding as integer
   Dim majority as integer
   Dim maxRec as integer
   Dim numVars as integer
   Dim numAttrs as integer
   Dim attrN as integer
   Dim attrName as string
   Dim attrScope as integer
   Dim maxgEntry as integer
   Dim maxrEntry as integer
   Dim maxzEntry as integer

   status = CDFinquireCDF (id, numDims, dimSizes, encoding, majority, maxRec, numVars, numAttrs)
   for attrN = 0 to (numAttrs-1)
      status = CDFinquireAttr (id, attrN, attrName, attrScope, maxgEntry, maxrEntry, maxzEntry)
   next attrN
   catch ex as Exception
      ...
   end try
```
4.4.39  CDFinquireAttrgEntry

integer CDFinquireAttrgEntry (  
  id as long,  
  attrNum as integer,  
  entryNum as integer,  
  dataType as integer,  
  numElements as integer)  
  '  out -- Completion status code.  
  '  in -- CDF identifier.  
  '  in -- attribute number.  
  '  in -- Entry number.  
  '  out -- Data type.  
  '  out -- Number of elements (of the data type).

This method is identical to CDFattrEntryInquire. CDFinquireAttrgEntry is used to inquire information about a global attribute entry.

The arguments to CDFinquireAttrgEntry are defined as follows:

id          The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
attrNum     The attribute number to inquire. This number may be determined with a call to CDFgetAttrNum.
entryNum    The entry number to inquire.
dataType     The data type of the specified entry. The data types are defined in Section 2.6.
numElements The number of elements of the data type. For character data types (CDF_CHAR and CDF_UCHAR), this is the number of characters in the string. For all other data types this is the number of elements in an array of that data type.

4.4.39.1. Example(s)

The following example returns each entry for a global attribute named TITLE. Note that entry numbers need not be consecutive - not every entry number between zero (0) and the maximum entry number must exist. For this reason NO_SUCH_ENTRY is an expected error code.

```
...  

dim id as long  
Dim status as integer  
Dim attrN as integer  
Dim entryN as integer  
Dim attrName as string  
Dim attrScope as integer  
Dim maxEntry as integer  
Dim dataType as integer  
Dim numElems as integer  

try  
....  
attrN = CDFgetAttrNum (id, "TITLE")  
status = CDFattrInquire (id, attrN, attrName, attrScope, maxEntry)
```
for entry\(N = 0\) to maxEntry
   status = CDFinquireAttrEntry (id, attr\(N\), entry\(N\), dataType, numElements)
   · process entries
   ·
next entry\(N\)
catch ex as Exception
   ...
end try

### 4.4.40 CDFinquireAttrrEntry

integer CDFinquireAttrrEntry (id as long, attrNum as integer, entryNum as integer, dataType as integer, numElements as integer)  

This method is identical to the method CDFattrEntryInquire. CDFinquireAttrrEntry is used to inquire about an rVariable’s attribute entry.

The arguments to CDFinquireAttrrEntry are defined as follows:

- **id**
  - The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

- **attrNum**
  - The attribute number to inquire. This number may be determined with a call to CDFgetAttribute.

- **entryNum**
  - The entry number to inquire. This is the rVariable number (the rVariable being described in some way by the rEntry).

- **dataType**
  - The data type of the specified entry. The data types are defined in Section 2.6.

- **numElements**
  - The number of elements of the data type. For character data types (CDF_CHAR and CDF_UCHAR), this is the number of characters in the string. For all other data types this is the number of elements in an array of that data type.

### 4.4.40.1. Example(s)

The following example determines the data type of the “UNITS” attribute for the rVariable “Temperature”, then retrieves and displays the value of the UNITS attribute.
dim id as long
Dim status as integer
Dim attrN as integer
Dim entryN as integer
Dim dataType as integer
Dim numElems as integer
.
.
try
....
attrN = CDFgetAttrNum (id, "UNITS")
entryN = CDFgetVarNum (id, "Temperature")
status = CDFinquireAttrEntry (id, attrN, entryN, dataType, numElems)
if dataType = CDF_CHAR then
    dim buffer as string
    status = CDFgetAttrEntry (id, attrN, entryN, buffer)
end if
.
catch ex as Exception
...
end try
.

4.4.41 CDFinquireAttrzEntry

integer CDFinquireAttrzEntry ( id as long,
attrNum as integer,
entryNum as integer,
dataType as integer,
umElements as integer)  
  * out -- Completion status code.
  * in -- CDF identifier.
  * in -- (Variable) Attribute number.
  * in -- zEntry number.
  * out -- Data type.
  * out -- Number of elements (of the data type).

CDFinquireAttrzEntry is used to inquire about a zVariable’s attribute entry.

The arguments to CDFinquireAttrzEntry are defined as follows:

id          The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
attrNum     The (variable) attribute number for which to inquire an entry. This number may be determined with a call to CDFgetAttrNum (see Section 4.4.24).
entryNum    The entry number to inquire. This is the zVariable number (the zVariable being described in some way by the zEntry).
dataType    The data type of the specified entry. The data types are defined in Section 2.6.
umElements The number of elements of the data type. For character data types (CDF_CHAR and CDF_UCHAR), this is the number of characters in the string. For all other data types this is the number of elements in an array of that data type.
4.4.41.1. Example(s)

The following example determines the data type of the UNITS attribute for the zVariable Temperature, then retrieves and displays the value of the UNITS attribute.

```vba
try
    attrN  =  CDFgetAttrNum (id, "UNITS")
    entryN = CDFgetVarNum (id, "Temperature")
    status = CDFinquireAttrzEntry (id, attrN, entryN, dataType, numElems)
    if dataType = CDF_CHAR then
        dim buffer as string
        status =  CDFgetAttrzEntry  (id, attrN, entryN, buffer)
    end if
    catch ex as Exception
    ...
end try
```

4.4.42  CDFputAttrgEntry

```vba
integer CDFputAttrgEntry(  
id as long,
attrNum as integer,
entryNum as integer,
dataType as integer,
numElements as integer,
value as TYPE)
```

CDFputAttrgEntry is used to write a global attribute entry. The entry may or may not already exist. If it does exist, it is overwritten. The data type and number of elements (of that data type) may be changed when overwriting an existing entry. A global attribute can have one or more attribute entries.

The arguments to CDFputAttrgEntry are defined as follows:
id

The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

attrNum

The attribute number. This number may be determined with a call to CDFgetAttrNum.

entryNum

The attribute entry number.

dataType

The data type of the specified entry. Specify one of the data types defined in Section 2.6.

numElements

The number of elements of the data type. For character data types (CDF_CHAR and CDF_UCHAR), this is the number of characters in the string (An array of characters). For all other data types this is the number of elements in an array of that data type.

value

The value(s) to write. The entry value is written to the CDF from memory address value.

4.4.42.1. Example(s)

The following example writes a global attribute entry to the global attribute called TITLE.

```vba
dim id as long
Dim status as integer
Dim entryNum as integer
Dim title as string = "CDF title."

entryNum = 0
try
    ....
    status = CDFputAttrgEntry (id, CDFgetAttrNum (id,"TITLE"), entryNum, CDF_CHAR, title.Length, title)
    ....
catch ex as Exception
    ...
end try
```

4.4.43 CDFputAttrrEntry

integer CDFputAttrrEntry( id as long, attrNum as integer, entryNum as integer, dataType as integer, numElems as integer, value as TYPE)  

' out -- Completion status code.
' in -- CDF identifier.
' in -- Attribute number.
' in -- Attribute entry number.
' in -- Data type.
' in -- Number of elements.
' in -- tribute entry value.
This method is identical to the method CDFattrPut. CDFputAttrrEntry is used to write rVariable’s attribute entry. The entry may or may not already exist. If it does exist, it is overwritten. The data type and number of elements (of that data type) may be changed when overwriting an existing entry.

The arguments to CDFputAttrrEntry are defined as follows:

- **id**
  The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

- **attrNum**
  The attribute number. This number may be determined with a call to CDFgetAttrNum.

- **entryNum**
  The attribute entry number that is the rVariable number to which this attribute entry belongs.

- **dataType**
  The data type of the specified entry. Specify one of the data types defined in Section 2.6.

- **numElements**
  The number of elements of the data type. For character data types (CDF_CHAR and CDF_UCHAR), this is the number of characters in the string (An array of characters). For all other data types this is the number of elements in an array of that data type.

- **value**
  The value(s) to write. The entry value is written to the CDF from memory address value.

### 4.4.43.1. Example(s)

The following example writes to the variable scope attribute VALIDs for the entry, of two elements, that corresponds to the rVariable TMP.

```vbnet
dim id as long
Dim status as integer
Dim entryNum as integer
Dim numElements as integer
Dim TMPvalids() as short = {15,30} ‘Value(s) of VALIDs attribute,
‘ rEntry for rVariable TMP.

numElements = 2
try
....
status = CDFputAttrrEntry (id, CDFgetAttrNum (id,"VALIDs"), CDFgetVarNum (id,"TMP"), _
CDF_INT2, numElements, TMPvalids)
} try

catch ex as Exception
...
end try
```
4.4.44  CDFputAttrzEntry

integer CDFputAttrzEntry(  
  id as long,  
  attrNum as integer,  
  entryNum as integer,  
  dataType as integer,  
  numElements as integer,  
  value as TYPE)  

' out -- Completion status code.  
' in -- CDF identifier.  
' in -- Attribute number.  
' in -- Attribute entry number.  
' in -- Data type of this entry.  
' in -- Number of elements in the entry (of the data type)  
' in -- Attribute entry value.  
' TYPE -- VB value/string type.

CDFputAttrzEntry is used to write zVariable’s attribute entry. The entry may or may not already exist. If it does exist, it is overwritten. The data type and number of elements (of that data type) may be changed when overwriting an existing entry.

The arguments to CDFputAttrzEntry are defined as follows:

id  The identifier of the CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.

attrNum  The (variable) attribute number. This number may be determined with a call to CDFgetAttrNum (see Section 4.4.24).

entryNum  The entry number that is the zVariable number to which this attribute entry belongs.

dataType  The data type of the specified entry. Specify one of the data types defined in Section 2.6.

numElements  The number of elements of the data type. For character data types (CDF_CHAR and CDF_UCHAR), this is the number of characters in the string (An array of characters). For all other data types this is the number of elements in an array of that data type.

value  The value(s) to write. The entry value is written to the CDF from memory address value.

4.4.44.1. Example(s)

The following example writes a zVariable’s attribute entry. The entry has two elements (that is two values for non-CDF_CHAR type). The zEntry in the variable scope attribute VALIDs corresponds to the zVariable TMP.

\[
\begin{align*}
\text{dim id as long} & \quad \text{CDF identifier.} \\
\text{Dim status as integer} & \quad \text{Returned status code.} \\
\text{Dim numElements as integer} & \quad \text{Number of elements (of data type).} \\
\text{Dim TMPvalids() as short } = \{15,30\} & \quad \text{Value(s) of VALIDs attribute,}
\end{align*}
\]
numElements = 2
try
    status = CDFputAttrEntry (id, CDFgetAttrNum (id,"VALIDs"), CDFgetVarNum (id,"TMP"),
    CDF_INT2, numElements, TMPvalids)
catch ex as Exception
    ...
end try

4.4.45 CDFrenameAttr

integer CDFrenameAttr(
id as long,
attrNum as integer,
attrName as string)

This method is identical to method CDFattrRename. CDFrenameAttr renames an existing attribute.

4.4.45.1. Example(s)

In the following example the attribute named LAT is renamed to LATITUDE.

dim id as long
Dim status as integer

try
    status = CDFrenameAttr (id, CDFgetAttrNum (id,"LAT"), "LATITUDE")
catch ex as Exception
    ...
end try

4.4.46 CDFsetAttrEntryDataSpec
integer CDFsetAttrgEntryDataSpec (  
  id as long, 
  attrNum as integer, 
  entryNum as integer, 
  dataType as integer)  
  
  ‘ out -- Completion status code.  
  ‘ in -- CDF identifier.  
  ‘ in -- Attribute number.  
  ‘ in -- gEntry number.  
  ‘ in -- Data type.

CDFsetAttrgEntryDataSpec respecifies the data type of a gEntry of a global attribute in a CDF. The new and old data type must be equivalent. Refer to the CDF User’s Guide for descriptions of equivalent data types.

The arguments to CDFsetAttrgEntryDataSpec are defined as follows:

- **id**: The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **attrNum**: The global attribute number.
- **entryNum**: The gEntry number.
- **dataType**: The new data type.

### 4.4.46.1. Example(s)

The following example modifies the third entry’s (entry number 2) data type of the global attribute MY_ATTR in a CDF. It will change its original data type from CDF_INT2 to CDF_UINT2.

```plaintext
...  
dim id as long  
Dim status as integer  
Dim entryNum as integer  
Dim dataType as integer  
...  
entryNum = 2  
dataType = CDF_UINT2  
numElems = 1  
try  
....  
status = CDFsetAttrgEntryDataSpec (id, CDFgetAttrNum (id, “MY_ATTR”), entryNum, dataType)  
...  
catch ex as Exception  
...  
end try  
...
```

### 4.4.47  CDFsetAttrrEntryDataSpec

integer CDFsetAttrrEntryDataSpec (  
  id as long, 
  attrNum as integer, 
  entryNum as integer, 
  dataType as integer)  
  
  ‘ out -- Completion status code.
CDFsetAttrEntryDataSpec respecifies the data specification (data type and number of elements) of an rEntry of a
variable attribute in a CDF. The new and old data type must be equivalent, and the number of elements must not be
changed. Refer to the CDF User’s Guide for descriptions of equivalent data types.

The arguments to CDFsetAttrEntryDataSpec are defined as follows:

id                  The identifier of the current CDF. This identifier must have been initialized by a call to
                      CDFcreate (or CDFcreateCDF) or CDFopenCDF.
attrNum              The variable attribute number.
entryNum             The rEntry number.
dataType             The new data type.
numElements          The new number of elements.

4.4.47.1. Example(s)

The following example modifies the data specification for an rEntry, corresponding to rVariable “MY_VAR”, in the
variable attribute “MY_ATTR” in a CDF. It will change its original data type from CDF_INT2 to CDF_UINT2.

\[
\begin{align*}
\text{dim id as long} & \quad \text{CDF identifier.} \\
\text{Dim status as integer} & \quad \text{Returned status code.} \\
\text{Dim dataType as integer} & \quad \text{Data type and number of elements.} \\
\text{Dim numElements as integer} & \\
\end{align*}
\]

\[
\text{dataType} = \text{CDF_UINT2} \quad \text{numElems} = 1 \\
\text{try} \\
\text{status} = \text{CDFsetAttrEntryDataSpec (id, CDFgetAttrNum (id, “MY_ATTR”),} \\
\text{CDFgetVarNum (id, “MY_VAR”), dataType, numElems)} \\
\text{catch ex as Exception} \\
\text{end try}
\]
4.4.48  CDFsetAttrScope

integer CDFsetAttrScope (  
id as long,  
attrNum as integer,  
scope as integer)  

CDFsetAttrScope respecifies the scope of an attribute in a CDF. Specify one of the scopes described in Section 2.13. Global-scoped attributes will contain only gEntries, while variable-scoped attributes can hold rEntries and zEntries.

The arguments to CDFsetAttrScope are defined as follows:

  id                   The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
  attrNum              The attribute number.
  scope                The new attribute scope. The value should be either VARIABLE_SCOPE or GLOBAL_SCOPE.

4.4.48.1. Example(s)

The following example changes the scope of the global attribute named MY_ATTR to a variable attribute (VARIABLE_SCOPE).

```
   dim id as long
   Dim scope as integer

   scope = VARIABLE_SCOPE
   try
     status = CDFsetAttrScope (id, CDFgetAttrNum (id, “MY_ATTR”), scope)
   catch ex as Exception
     ...  
   end try
```

4.4.49  CDFsetAttrzEntryDataSpec

integer CDFsetAttrzEntryDataSpec (  
id as long,  
attrNum as integer,  

formal parameters:  
  `out` -- Completion status code.  
  `in` -- CDF identifier.  
  `in` -- Attribute number.  

DFsetAttrzEntryDataSpec (  
id as long,  
attrNum as integer,  
scope as integer)  

CDFsetAttrzEntryDataSpec respecifies the scope of an attribute in a CDF. Specify one of the scopes described in Section 2.13. Global-scoped attributes will contain only gEntries, while variable-scoped attributes can hold rEntries and zEntries.

The arguments to CDFsetAttrzEntryDataSpec are defined as follows:

  id                   The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
  attrNum              The attribute number.
  scope                The new attribute scope. The value should be either VARIABLE_SCOPE or GLOBAL_SCOPE.
entryNum as integer, ' in -- zEntry number.
dataType as integer) ' in -- Data type.

CDFsetAttrzEntryDataSpec modifies the data type of a zEntry of a variable attribute in a CDF. The new and old data type must be equivalent. Refer to the CDF User’s Guide for the description of equivalent data types.

The arguments to CDFsetAttrzEntryDataSpec are defined as follows:

- **id**: The identifier of the current CDF. This identifier must have been initialized by a call to CDFcreate (or CDFcreateCDF) or CDFopenCDF.
- **attrNum**: The variable attribute number.
- **entryNum**: The zEntry number that is the zVariable number.
- **dataType**: The new data type.

### 4.49.1. Example(s)

The following example respecifies the data type of the attribute entry of the attribute named MY_ATTR that is associated with the zVariable MY_VAR. It will change its original data type from CDF_INT2 to CDF_UINT2.

```plaintext
dim id as long ' CDF identifier.
Dim status as integer ' Returned status code.
dim dataType as integer ' Data type

try
    dataType = CDF_UINT2
    numElems = 1
    status = CDFsetAttrzEntryDataSpec (id, CDFgetAttrNum (id, “MY_ATTR”),
        CDFgetVarNum (id, “MY_VAR”), dataType)

    .
    .
catch ex as Exception
    .
end try
```
Chapter 5

5 Interpreting CDF Status Codes

Most CDF APIs return a status code of type int. The symbolic names for these codes are defined in CDFException.cs and should be used in your applications rather than using the true numeric values. Appendix A explains each status code. When the status code returned from a CDF API is tested, the following rules apply.

- \( \text{status} > \text{CDF\_OK} \) Indicates successful completion but some additional information is provided. These are informational codes.
- \( \text{status} = \text{CDF\_OK} \) Indicates successful completion.
- \( \text{CDF\_WARN} < \text{status} < \text{CDF\_OK} \) Indicates that the function completed but probably not as expected. These are warning codes.
- \( \text{status} < \text{CDF\_WARN} \) Indicates that the function did not complete. These for most cases are error codes, thus an exception might be thrown.

The following example shows how you could check the status code returned from CDF functions.

```plaintext
dim status as integer
.
try
  status = CDFfunction(...) ' any CDF function returning integer
.
catch ex as Exception
  ....
end try
```

In your own status handler you can take whatever action is appropriate to the application. An example status handler follows. Note that no action is taken in the status handler if the status is CDF_OK.

```plaintext
dim status as integer = ex.GetCurrentStatus()
dim errorMsg as string = ex.GetStatusMsg(status)
```

Explanations for all CDF status codes are available to your applications through the method CDFerror. CDFerror encodes in a text string an explanation of a given status code.
Chapter 6

6 EPOCH Utility Routines

Several functions exist that compute, decompose, parse, and encode CDF_EPOCH and CDF_EPOCH16 values. These functions may be called by applications using the CDF_EPOCH and CDF_EPOCH16 data types and are included in the CDF library. The Concepts chapter in the CDF User's Guide describes EPOCH values. All these APIs are defined as static methods in CDFAPIs class. The date/time components for CDF_EPOCH and CDF_EPOCH16 are UTC-based, without leap seconds.

The CDF_EPOCH and CDF_EPOCH16 data types are used to store time values referenced from a particular epoch. For CDF that epoch values for CDF_EPOCH and CDF_EPOCH16 are 01-Jan-0000 00:00:00.000 and 01-Jan-0000 00:00:00.000.000.000, respectively.

6.1 computeEPOCH

double computeEPOCH( 
  year as integer, 
  month as integer, 
  day as integer, 
  hour as integer, 
  minute as integer, 
  second as integer, 
  msec as integer) 

  ' out -- CDF_EPOCH value returned.
  ' in -- Year (AD, e.g., 1994).
  ' in -- Month (1-12).
  ' in -- Day (1-31).
  ' in -- Hour (0-23).
  ' in -- Minute (0-59).
  ' in -- Second (0-59).
  ' in -- Millisecond (0-999).

computeEPOCH calculates a CDF_EPOCH value given the individual components. If an illegal component is detected, the value returned will be ILLEGAL_EPOCH_VALUE.

NOTE: There are two variations on how computeEPOCH may be used. If the month argument is 0 (zero), then the day argument is assumed to be the day of the year (DOY) having a range of 1 through 366. Also, if the hour, minute, and second arguments are all 0 (zero), then the msec argument is assumed to be the millisecond of the day having a range of 0 through 86400000.
6.2  EPOCHbreakdown

void EPOCHbreakdown(
epoch as double,  \*in -- The CDF_EPOCH value.
year as integer,  \*out -- Year (AD, e.g., 1994).
month as integer,  \*out -- Month (1-12).
day as integer,  \*out -- Day (1-31).
hour as integer,  \*out -- Hour (0-23).
minute as integer,  \*out -- Minute (0-59).
second as integer,  \*out -- Second (0-59).
msec as integer)  \*out -- Millisecond (0-999).

EPOCHbreakdown decomposes a CDF_EPOCH value into the individual components.

6.3  toEncodeEPOCH

string toEncodeEPOCH(
epoch as double)  \*out -- Encode date/time string.
\*in -- The CDF_EPOCH value.

string toEncodeEPOCH(
epoch as double,
style as int)  \*out -- Encode date/time string.
\*in -- The encoding style.

string[] toEncodeEPOCH(
epochs as double[])  \*out -- Encode date/time strings.
\*in -- The CDF_EPOCH values.

string[] toEncodeEPOCH(
epochs as double[],
style as int)  \*out -- Encode date/time strings.
\*in -- The encoding style.

toEncodeEPOCH encodes a CDF_EPOCH value(s) into a date/time character string(s) in one of the standard forms. The style is between the value 0 and 4. With style 0, it is similar to calling encodeEPOCH. With style 1, 2 3 and 4, it is similar to calling encodeEPOCH1, encodeEPOCH2, encodeEPOCH3 and encodeEPOCH4, respectively. Without style, the default style, 4, is used. Refer the following sections to see what a standard date/time string looks like for each style.

6.4  encodeEPOCH

void encodeEPOCH(
epoch as double  \*in -- The CDF_EPOCH value.
epString as string)  \*out -- The standard date/time string.

encodeEPOCH encodes a CDF_EPOCH value into the standard date/time character string. The format of the string is dd-mmm-yyyy hh:mm:ss.ccc where dd is the day of the month (1-31), mmm is the month (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, or Dec), yyyy is the year, hh is the hour (0-23), mm is the minute (0-59), ss is the second (0-59), and ccc is the millisecond (0-999).
6.5  **encodeEPOCH1**

```c
void encodeEPOCH1(
    epoch as double  /* in -- The CDF_EPOCH value. */
    epString as string)   /* out -- The alternate date/time string. */
```

encodeEPOCH1 encodes a CDF_EPOCH value into an alternate date/time character string. The format of the string is `yyyyymmdd.tttttt`, where `yyyy` is the year, `mm` is the month (1-12), `dd` is the day of the month (1-31), and `ttttt` is the fraction of the day (e.g., 5000000 is 12 o'clock noon).

6.6  **encodeEPOCH2**

```c
void encodeEPOCH2(
    epoch as double  /* in -- The CDF_EPOCH value. */
    epString as string)  /* out -- The alternate date/time string. */
```

encodeEPOCH2 encodes a CDF_EPOCH value into an alternate date/time character string. The format of the string is `yyyyymmdthhmmss` where `yyyy` is the year, `mm` is the month (1-12), `dd` is the day of the month (1-31), `hh` is the hour (0-23), `mm` is the minute (0-59), and `ss` is the second (0-59).

6.7  **encodeEPOCH3**

```c
void encodeEPOCH3(
    epoch as double  /* in -- The CDF_EPOCH value. */
    epString as string)  /* out -- The alternate date/time string. */
```

encodeEPOCH3 encodes a CDF_EPOCH value into an alternate date/time character string. The format of the string is `yyyy-mm-ddThh:mm:ss.cccZ` where `yyyy` is the year, `mm` is the month (1-12), `dd` is the day of the month (1-31), `hh` is the hour (0-23), `mm` is the minute (0-59), `ss` is the second (0-59), and `ccc` is the millisecond (0-999).

6.8  **encodeEPOCH4**

```c
void encodeEPOCH4(
    epoch as double  /* in -- The CDF_EPOCH value. */
    epString as string)  /* out -- The ISO 8601 date/time string. */
```

encodeEPOCH4 encodes a CDF_EPOCH value into an alternate, ISO 8601 date/time character string. The format of the string is `yyyy-mm-ddThh:mm:ss.ccc` where `yyyy` is the year, `mm` is the month (1-12), `dd` is the day of the month (1-31), `hh` is the hour (0-23), `mm` is the minute (0-59), `ss` is the second (0-59), and `ccc` is the millisecond (0-999).
6.9 encodeEPOCHx

```c
void encodeEPOCHx(
    epoch as double,  
    format as string,  
    encoded as string)
```

encodeEPOCHx encodes a CDF_EPOCH value into a custom date/time character string. The format of the encoded string is specified by a format string.

The format string consists of EPOCH components, which are encoded, and text that is simply copied to the encoded custom string. Components are enclosed in angle brackets and consist of a component token and an optional width. The syntax of a component is: `<token[.width]>`. If the optional width contains a leading zero, then the component will be encoded with leading zeroes (rather than leading blanks).

The supported component tokens and their default widths are as follows.

<table>
<thead>
<tr>
<th>Token</th>
<th>Meaning</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>dom</td>
<td>Day of month (1-31)</td>
<td><code>&lt;dom.0&gt;</code></td>
</tr>
<tr>
<td>doy</td>
<td>Day of year (001-366)</td>
<td><code>&lt;doy.03&gt;</code></td>
</tr>
<tr>
<td>month</td>
<td>Month (<code>Jan', </code>Feb',..., `Dec')</td>
<td><code>&lt;month&gt;</code></td>
</tr>
<tr>
<td>mm</td>
<td>Month (1,2,...,12)</td>
<td><code>&lt;mm.0&gt;</code></td>
</tr>
<tr>
<td>year</td>
<td>Year (4-digit)</td>
<td><code>&lt;year.04&gt;</code></td>
</tr>
<tr>
<td>yr</td>
<td>Year (2-digit)</td>
<td><code>&lt;yr.02&gt;</code></td>
</tr>
<tr>
<td>hour</td>
<td>Hour (00-23)</td>
<td><code>&lt;hour.02&gt;</code></td>
</tr>
<tr>
<td>min</td>
<td>Minute (00-59)</td>
<td><code>&lt;min.02&gt;</code></td>
</tr>
<tr>
<td>sec</td>
<td>Second (00-59)</td>
<td><code>&lt;sec.02&gt;</code></td>
</tr>
<tr>
<td>fos</td>
<td>Fraction of second.</td>
<td><code>&lt;fos.3&gt;</code></td>
</tr>
<tr>
<td>fod</td>
<td>Fraction of day.</td>
<td><code>&lt;fod.8&gt;</code></td>
</tr>
</tbody>
</table>

Note that a width of zero indicates that as many digits as necessary should be used to encode the component. The `<month>` component is always encoded with three characters. The `<fos>` and `<fod>` components are always encoded with leading zeroes.

If a left angle bracket is desired in the encoded string, then simply specify two left angle brackets (`<<`) in the format string (character stuffing).

For example, the format string used to encode the standard EPOCH date/time character string (see Section 6.3) would be.

```
<dom.02><month><year> <hour>:<min>:<sec>.<fos>
```

6.10 toParseEPOCH

```c
double toParseEPOCH(
    epString as string)
```

`out -- The CDF_EPOCH value.
`in -- The date/time string.

```c
double[] toParseEPOCH(
    epStrings as string[])`  
```

`out -- The CDF_EPOCH values.
`in -- The date/time strings.`
toParseEPOCH parses an encoded, standard date/time character string(s) and returns a CDF_EPOCH value(s). The format of the string is that produced by one of the encoding functions, e.g., toEncodeEPOCH, encodeEPOCH, encodeEPOCH1, etc. If an illegal field is detected in the string, the value returned will be ILLEGAL_EPOCH_VALUE.

6.11 parseEPOCH

double parseEPOCH(
    epString as string)

    ' out -- CDF_EPOCH value.
    ' in -- The standard date/time string.

parseEPOCH parses a standard date/time character string and returns a CDF_EPOCH value. The format of the string is that produced by the encodeEPOCH method described in Section 6.3. If an illegal field is detected in the string the value returned will be ILLEGAL_EPOCH_VALUE.

6.12 parseEPOCH1

double parseEPOCH1(
    epString as string)

    ' out -- CDF_EPOCH value.
    ' in -- The alternate date/time string.

parseEPOCH1 parses an alternate date/time character string and returns a CDF_EPOCH value. The format of the string is that produced by the encodeEPOCH1 method described in Section 6.5. If an illegal field is detected in the string the value returned will be ILLEGAL_EPOCH_VALUE.

6.13 parseEPOCH2

double parseEPOCH2(
    epString as string)

    ' out -- CDF_EPOCH value.
    ' in -- The alternate date/time string.

parseEPOCH2 parses an alternate date/time character string and returns a CDF_EPOCH value. The format of the string is that produced by the encodeEPOCH2 method described in Section 6.6. If an illegal field is detected in the string the value returned will be ILLEGAL_EPOCH_VALUE.

6.14 parseEPOCH3

double parseEPOCH3(
    epString as string)

    ' out -- CDF_EPOCH value.
    ' in -- The alternate date/time string.
parseEPOCH3 parses an alternate date/time character string and returns a CDF_EPOCH value. The format of the string is that produced by the encodeEPOCH3 method described in Section 6.7. If an illegal field is detected in the string the value returned will be ILLEGAL_EPOCH_VALUE.

### 6.15 parseEPOCH4

double parseEPOCH4(
    epString as string)

    ' out -- CDF_EPOCH value.
    ' in -- The alternate date/time string.

parseEPOCH3 parses an alternate, ISO 8601 date/time character string and returns a CDF_EPOCH value. The format of the string is that produced by the encodeEPOCH3 method described in Section 6.8. If an illegal field is detected in the string the value returned will be ILLEGAL_EPOCH_VALUE.

### 6.16 computeEPOCH16

double computeEPOCH16(
    year as integer,
    month as integer,
    day as integer,
    hour as integer,
    minute as integer,
    second as integer,
    msec as integer,
    microsec as integer,
    nanosec as integer,
    picosec as integer,
    epoch as double())

    ' out -- status code returned.
    ' in -- Year (AD, e.g., 1994).
    ' in -- Month (1-12).
    ' in -- Day (1-31).
    ' in -- Hour (0-23).
    ' in -- Minute (0-59).
    ' in -- Second (0-59).
    ' in -- Millisecond (0-999).
    ' in -- Microsecond (0-999).
    ' in -- Nanosecond (0-999).
    ' in -- Picossecond (0-999).
    ' out -- CDF_EPOCH16 value

computeEPOCH16 calculates a CDF_EPOCH16 value given the individual components. If an illegal component is detected, the value returned will be ILLEGAL_EPOCH_VALUE.

### 6.17 EPOCH16breakdown

void EPOCH16breakdown(
    epoch as double(),
    year as integer,
    month as integer,
    day as integer,
    hour as integer,
    minute as integer,
    second as integer,
    msec as integer,
    microsec as integer,
    nanosec as integer,
    picosec as integer,
    ' in -- The CDF_EPOCH16 value.
    ' out -- Year (AD, e.g., 1994).
    ' out -- Month (1-12).
    ' out -- Day (1-31).
    ' out -- Hour (0-23).
    ' out -- Minute (0-59).
    ' out -- Second (0-59).
    ' out -- Millisecond (0-999).
    ' out -- Microsecond (0-999).
    ' out -- Nanosecond (0-999).
EPOCH16breakdown decomposes a CDF_EPOCH16 value into the individual components.

### 6.18 toEncodeEPOCH16

```c
string toEncodeEPOCH16(
    epoch as double[],
    style as int)  

string toEncodeEPOCH16(
    epoch as double[],
    year as int)  

string toEncodeEPOCH16(
    epoch as double[],
    month as int)  

string toEncodeEPOCH16(
    epoch as double[],
    day as int)  

string toEncodeEPOCH16(
    epoch as double[],
    hour as int)  

string toEncodeEPOCH16(
    epoch as double[],
    minute as int)  

string toEncodeEPOCH16(
    epoch as double[],
    second as int)  

string toEncodeEPOCH16(
    epoch as double[],
    millisecond as int)  

string toEncodeEPOCH16(
    epoch as double[],
    microsecond as int)  

string toEncodeEPOCH16(
    epoch as double[],
    nanosecond as int)  

string toEncodeEPOCH16(
    epoch as double[],
    picosecond as int)  
```

`toEncodeEPOCH16` encodes a CDF_EPOCH16 value, a two-double array, into a date/time character string in one of the standard forms. The style is between the value 0 and 4. With style 0, it is similar to calling encodeEPOCH16. With style 1, 2, 3, and 4, it is similar to calling encodeEPOCH16_1, encodeEPOCH16_2, encodeEPOCH16_3, and encodeEPOCH16_4, respectively. Without style, the default style, 4, is used. Refer the following sections to see what a date/time string looks like for each style.

### 6.19 encodeEPOCH16

```c
void encodeEPOCH16(
    epoch as double(),
    epString as string)  
```

`encodeEPOCH16` encodes a CDF_EPOCH16 value into the standard date/time character string. The format of the string is `dd-mm-yyyy hh:mm:ss.mmm:uuu:nnn:ppp` where `dd` is the day of the month (1-31), `mmm` is the month (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, or Dec), `yyyy` is the year, `hh` is the hour (0-23), `mm` is the minute (0-59), `ss` is the second (0-59), `mmm` is the millisecond (0-999), `uuu` is the microsecond (0-999), `nnn` is the nanosecond (0-999), and `ppp` is the picosecond (0-999).

### 6.20 encodeEPOCH16_1

```c
void encodeEPOCH16_1(
    epoch as double(),
    epString as string)  
```

`encodeEPOCH16_1` encodes a CDF_EPOCH16 value into an alternate date/time character string. The format of the string is `yyyyymmdd.tttt1111111`, where `yyyy` is the year, `mm` is the month (1-12), `dd` is the day of the month (1-31), and `tttttttttttt` is the fraction of the day (e.g., 500000000000000 is 12 o'clock noon).
6.21  encodeEPOCH16_2

void encodeEPOCH16_2(
  epoch as double(),
  epString as string)
  ' in -- The CDF_EPOCH16 value.
  ' out -- The date/time string.

encodeEPOCH16_2 encodes a CDF_EPOCH16 value into an alternate date/time character string. The format of the string is yyyymoddhhmss where yyyy is the year, mo is the month (1-12), dd is the day of the month (1-31), hh is the hour (0-23), mm is the minute (0-59), and ss is the second (0-59).

6.22  encodeEPOCH16_3

void encodeEPOCH16_3(
  epoch as double(),
  epString as string)
  ' in -- The CDF_EPOCH16 value.
  ' out -- The alternate date/time string.

encodeEPOCH16_3 encodes a CDF_EPOCH16 value into an alternate date/time character string. The format of the string is yyyy-mo-ddThh:mm:ss.mmm:uuu:nnn:pppZ where yyyy is the year, mo is the month (1-12), dd is the day of the month (1-31), hh is the hour (0-23), mm is the minute (0-59), ss is the second (0-59), mmm is the millisecond (0-999), uuu is the microsecond (0-999), nnn is the nanosecond (0-999), and ppp is the picosecond (0-999).

6.23  encodeEPOCH16_4

void encodeEPOCH16_4(
  epoch as double(),
  epString as string)
  ' in -- The CDF_EPOCH16 value.
  ' out -- The alternate date/time string.

encodeEPOCH16_4 encodes a CDF_EPOCH16 value into an alternate, ISO 8601 date/time character string. The format of the string is yyyy-mo-ddThh:mm:ss.mmmuuunnnppp where yyyy is the year, mo is the month (1-12), dd is the day of the month (1-31), hh is the hour (0-23), mm is the minute (0-59), ss is the second (0-59), mmm is the millisecond (0-999), uuu is the microsecond (0-999), nnn is the nanosecond (0-999), and ppp is the picosecond (0-999).

6.24  encodeEPOCH16_x

void encodeEPOCH16_x(
  epoch as double(),
  format as string
  encoded as string)
  ' in -- The CDF_EPOCH16 value.
  ' in -- The format string.
  ' out -- The date/time string.

encodeEPOCH16_x encodes a CDF_EPOCH16 value into a custom date/time character string. The format of the encoded string is specified by a format string.
The format string consists of EPOCH components, which are encoded, and text that is simply copied to the encoded custom string. Components are enclosed in angle brackets and consist of a component token and an optional width. The syntax of a component is: `<token[,width]>`. If the optional width contains a leading zero, then the component will be encoded with leading zeroes (rather than leading blanks).

The supported component tokens and their default widths are as follows...

<table>
<thead>
<tr>
<th>Token</th>
<th>Meaning</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>dom</td>
<td>Day of month (1-31)</td>
<td>&lt;dom.0&gt;</td>
</tr>
<tr>
<td>doy</td>
<td>Day of year (001-366)</td>
<td>&lt;doy.03&gt;</td>
</tr>
<tr>
<td>month</td>
<td>Month ('Jan', 'Feb', ..., 'Dec')</td>
<td>&lt;month&gt;</td>
</tr>
<tr>
<td>mm</td>
<td>Month (1,2,...,12)</td>
<td>&lt;mm.0&gt;</td>
</tr>
<tr>
<td>year</td>
<td>Year (4-digit)</td>
<td>&lt;year.04&gt;</td>
</tr>
<tr>
<td>yr</td>
<td>Year (2-digit)</td>
<td>&lt;yr.02&gt;</td>
</tr>
<tr>
<td>hour</td>
<td>Hour (00-23)</td>
<td>&lt;hour.02&gt;</td>
</tr>
<tr>
<td>min</td>
<td>Minute (00-59)</td>
<td>&lt;min.02&gt;</td>
</tr>
<tr>
<td>sec</td>
<td>Second (00-59)</td>
<td>&lt;sec.02&gt;</td>
</tr>
<tr>
<td>msc</td>
<td>Millisecond (000-999)</td>
<td>&lt;msc.3&gt;</td>
</tr>
<tr>
<td>usc</td>
<td>Microsecond (000-999)</td>
<td>&lt;usc.3&gt;</td>
</tr>
<tr>
<td>nsc</td>
<td>Nanosecond (000-999)</td>
<td>&lt;nsc.3&gt;</td>
</tr>
<tr>
<td>psc</td>
<td>Picosecond (000-999)</td>
<td>&lt;psc.3&gt;</td>
</tr>
<tr>
<td>fos</td>
<td>Fraction of second.</td>
<td>&lt;fos.12&gt;</td>
</tr>
<tr>
<td>fod</td>
<td>Fraction of day.</td>
<td>&lt;fod.8&gt;</td>
</tr>
</tbody>
</table>

Note that a width of zero indicates that as many digits as necessary should be used to encoded the component. The `<month>` component is always encoded with three characters. The `<fos>` and `<fod>` components are always encoded with leading zeroes.

If a left angle bracket is desired in the encoded string, then simply specify two left angle brackets (<<) in the format string (character stuffing).

For example, the format string used to encode the standard EPOCH date/time character string would be...

```
<dom.02><month><year> <hour>;<min>;<sec>.<msc>.<usc>.<nsc>.<psc>.<fos>
```

### 6.25 toParseEPOCH16

double[] toParseEPOCH16(  
    epString as string)  

`toParseEPOCH16` parses a encoded, standard date/time character string and returns a CDF_EPOCH16 value, a two-double array. The format of the string is that produced by one of the encoding functions, e.g., toEncodeEPOCH16, encodeEPOCH16, encodeEPOCH16_1, etc. If an illegal field is detected in the string, the value returned will be ILLEGAL_EPOCH_VALUE.

### 6.26 parseEPOCH16

double parseEPOCH16(  

`parseEPOCH16`
parseEPOCH16 parses a standard date/time character string and returns a CDF_EPOCH16 value. The format of the string is that produced by the encodeEPOCH16 function. If an illegal field is detected in the string the value returned will be ILLEGAL_EPOCH_VALUE.

### 6.27 parseEPOCH16_1

double parseEPOCH16_1(
  epString as string,
  epoch as double())

parseEPOCH16_1 parses an alternate date/time character string and returns a CDF_EPOCH16 value. The format of the string is that produced by the encodeEPOCH16_1 function. If an illegal field is detected in the string the value returned will be ILLEGAL_EPOCH_VALUE.

### 6.28 parseEPOCH16_2

double parseEPOCH16_2(
  epString as string,
  epoch as double())

parseEPOCH16_2 parses an alternate date/time character string and returns a CDF_EPOCH16 value. The format of the string is that produced by the encodeEPOCH16_2 function. If an illegal field is detected in the string the value returned will be ILLEGAL_EPOCH_VALUE.

### 6.29 parseEPOCH16_3

double parseEPOCH16_3(
  epString as string,
  epoch as double())

parseEPOCH16_3 parses an alternate date/time character string and returns a CDF_EPOCH16 value. The format of the string is that produced by the encodeEPOCH16_3 function. If an illegal field is detected in the string the value returned will be ILLEGAL_EPOCH_VALUE.

### 6.30 parseEPOCH16_4
double parseEPOCH16_4(
  epString as string,
  epoch as double())

parseEPOCH16_4 parses an alternate date/time character string and returns a CDF_EPOCH16 value. The format of the string is that produced by the encodeEPOCH16_3 function. If an illegal field is detected in the string the value returned will be ILLEGAL_EPOCH_VALUE.

6.31 EPOCHtoUnixTime

double EPOCHtoUnixTime(
  epoch as double)
double EPOCHtoUnixTime(
  epochs as double())

EPOCHtoUnixTime converts an epoch time(s) in CDF_EPOCH type into a Unix time(s). A CDF_EPOCH epoch, a double, is milliseconds from 0000-01-01T00:00:00.000 while Unix time, also a double, is seconds from 1970-01-01T00:00:00.000. The Unix time can have sub-second, with a time resolution of microseconds, in its fractional part.

6.32 UnixTimetoEPOCH

double UnixTimetoEPOCH ( 
  unixTime as double)
double UnixTimetoEPOCH ( 
  unixTimes as double())

UnixTimetoEPOCH converts a Unix time(s) to an epoch time(s) in CDF_EPOCH. A CDF_EPOCH epoch, a double, is milliseconds from 0000-01-01T00:00:00.000 while Unix time, also a double, is seconds from 1970-01-01T00:00:00.000. The Unix time can have sub-second, with a time resolution of microseconds, in its fractional part. Converting the Unix time to EPOCH will only keep the resolution to milliseconds.

6.33 EPOCH16toUnixTime

double EPOCH16toUnixTime(
  epoch as double())

ePOCH16toUnixTime converts an epoch time in CDF_EPOCH16 type, a two-double array, to a Unix time. A CDF_EPOCH16 epoch is picoseconds from 0000-01-01T00:00:00.000.000.000.000.000, while Unix time, a double, is
seconds from 1970-01-01T00:00:00.000. The Unix time can have sub-second, with a time resolution of microseconds, in its fractional part. **Note:** As CDF_EPOCH16 has much higher time resolution, sub-microseconds portion of its time might get lost during the conversion.

### 6.34 UnixTimetoEPOCH16

```c
double() UnixTimetoEPOCH16 (unixTimes as double)
```

UnixTimetoEPOCH16 converts a Unix time to an epoch time in CDF_EPOCH16. A CDF_EPOCH16 epoch, a two-double array, is picoseconds from 0000-01-01T00:00:00.000.000.000.000.000, while Unix time, also a double, is seconds from 1970-01-01T00:00:00.000. The Unix time can have sub-second, with a time resolution of microseconds, in its fractional part. Sub-microseconds will be filled with 0’s when converting from Unix time to EPOCH16.
7 TT2000 Utility Routines

Several functions exist that compute, decompose, parse, and encode CDF_TIME_TT2000 values. These functions may be called by applications using the CDF_TIME_TT2000 data type and is included in the CDF library. The Concepts chapter in the CDF User’s Guide describes TT2000 values. All these APIs are defined as static methods in CDFAPIs class. The date/time components for CDF_TIME_TT2000 are UTC-based, with leap seconds.

The CDF_TIME_TT2000 data type is used to store time values referenced from J2000 (2000-01-01T12:00:00.000000000). For CDF, values in CDF_TIME_TT2000 are nanoseconds from J2000 with leap seconds included. TT2000 data can cover years between 1707 and 2292.

7.1 computeTT2000

computeTT2000 is a overloaded function.

```c
long computeTT2000(
    year as double,
    month as double,
    day as double)

long computeTT2000(
    year as double,
    month as double,
    day as double,
    hour as double)

long computeTT2000(
    year as double,
    month as double,
    day as double,
    hour as double,
    minute as double)

long computeTT2000(
    year as double,
    month as double,
    day as double,
    hour as double,
    minute as double,
    second as double)
```

- `out` -- CDF_TIME_TT2000 value.
- `in` -- Year (AD, e.g., 1994).
- `in` -- Month (1-12).
- `in` -- Day (1-31).
- `in` -- Hour (0-23).
- `in` -- Minute (0-59).
- `in` -- Second (0-59 or 0-60 if leap second).
computeTT2000 calculates a CDF\_TIME\_TT2000 value given the individual, UTC-based date/time components. If an illegal component is detected, the value returned will be ILLEGAL\_TT2000\_VALUE. The day component can be presented in day of the month or day of the year (DOY). If DOY form is used, the month component must have a value(s) of one (1).

**NOTE:** Even though this overloaded function uses double for all its parameter fields, all but the very last parameter can not have a non-zero fractional part for simplifying the computation. An exception will be thrown if the rule is not followed. For example, this call is allowed:

```
dm tt2000 as long = computeTT2000(2010.0, 10.0, 10.5)
```

But, this call will fail:

```
dim tt2000 as long = computeTT2000(2010.0, 10.0, 10.5, 12.5)
```

### 7.2 TT2000breakdown

```c
void TT2000breakdown(
    tt2000 as long,
    year as double,
    month as double,
    day as double,
    hour as double,
    minute as double,
    second as double,
    msec as double,
    usec as double
)
```
TT2000breakdown decomposes a CDF_TIME_TT2000 value into the individual components.

### 7.3 toEncodeTT2000

String toEncodeTT2000(
epoch as long)

String toEncodeTT2000(
epoch as long,
style as int)

String() toEncodeTT2000(
epochs as long())

String() toEncodeTT2000(
epochs as long(),
style as int)

toEncodeTT2000 encodes a CDF_TIME_TT2000 value(s) into a date/time character string(s) in one of the standard forms. The style is between the value 0 and 4. Without style, the default style is used, which is style 3. Refer the following section to see what a date/time string looks like for each style.

### 7.4 encodeTT2000

encodeTT2000 is a overloaded function.

void encodeTT2000(
tt2000 as long
EpString as string)

void encodeTT2000(
tt2000 as long
epString as string,
style as int)

encodeTT2000 encodes a CDF_TIME_TT2000 value into one of the standard date/time UTC character strings. Without the style, the default style of 3 is used, which makes the string in ISO 8601 format: `yyyy-mm-ddT hh:mm:ss.mmmuuunnn` where yyyy is the year (1707-2292), mm is the month (01-12), dd is the day of the month (1-31), hh is the hour (0-23), mm is the minute (0-59), ss is the second (0-59 or 0-60 if leap second), mmm is the millisecond (0-999), uuu is the microsecond (0-999), and nnn is the nanosecond (0-999).

For a style of value 0, the encoded UTC string is `DD-Mon-YYYY hh:mm:ss.mmmuuunnn`, where DD is the day of the month (1-31), Mon is the month (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, or Dec), YYYY is the year, hh is the hour (0-23), mm is the minute (0-59 or 0-60 if leap second), ss is the second (0-59), mmm is the millisecond (0-999), uuu is the microsecond (0-999), and nnn is the nanosecond (0-999). The encoded string has a length of `TT2000_0_STRING_LEN (30)`.
For a style of value 1, the encoded UTC string is `YYYYMMDD.tttttttt`, where YYYY is the year, MM is the month (1-12) DD is the day of the month (1-31), and tttttttt is sub-day (0-999999999). The encoded string has a length of `TT2000_1_STRING_LEN` (19).

For a style of value 2, the encoded UTC string is `YYYYMMDDhhmmss`, where YYYY is the year, MM is the month (1-12) DD is the day of the month (1-31), hh is the hour (0-23), mm is the minute (0-59), and ss is the second (0-59 or 0-60 if leap second). The encoded string has a length of `TT2000_2_STRING_LEN` (14).

For a style of value 3, the encoded UTC string is `YYYY-MM-DDThh:mm:ss.mmmuuunnn`, where YYYY is the year, MM is the month (1-12), DD is the day of the month (1-31), hh is the hour (0-23), mm is the minute (0-59 or 0-60 if leap second), ss is the second (0-59), mmm is the millisecond (0-999), uuu is the microsecond (0-999), and nnn is the nanosecond (0-999). The encoded string has a length of `TT2000_3_STRING_LEN` (29).

For a style of value 4, the encoded UTC string is similar to style 3, with an addition of “Z” appended to the end. The encoded string has a length of `TT2000_4_STRING_LEN` (30).

### 7.5 toParseTT2000

```cpp
long toParseTT2000(
    epString as string)

out -- CDF_TIME_TT2000 value.

in -- The standard date/time string.
```

`toParseTT2000` parses an encoded date/time character string(s) and returns a `CDF_TIME_TT2000` value(s). The format of the string is that produced by the `toEncodeTT2000` or `encodeTT2000` method described in Section 6.3 or 7.4. If an illegal field is detected in the string, the value(s) returned will be `ILLEGAL_TT2000_VALUE`.

### 7.6 parseTT2000

```cpp
long parseTT2000(
    epString as string)

out -- CDF_TIME_TT2000 value.

in -- The encoded date/time string.
```

`parseTT2000` parses an encoded date/time character string and returns a `CDF_TIME_TT2000` value. The format of the string is that produced by the `encodeTT2000` method described in Section 7.3 or 7.4. If an illegal field is detected in the string the value returned will be `ILLEGAL_TT2000_VALUE`.

### 7.7 CDFgetLastDateinLeapSecondsTable

```cpp
void CDFgetLastDateinLeapSecondsTable(
    year as integer)

out -- The year.
```

`CDFgetLastDateinLeapSecondsTable`
month as integer  ` out -- The month.
day as integer)  ` out -- The day.

CDFgetLastDateinLeapSecondsTable returns the last entry in the leap second table used by the CDF processing. This
date comes from the leap second table, either through an external text file, or the hard-coded table in the library code.
This information can tell whether the leap second table is up-to-date.

7.8 TT2000toUnixTime

double TT2000toUnixTime(
epoch as long)  ` in -- The Unix time value.
` in -- The TT2000 epoch value.

double() TT2000toUnixTime(
epochs as long())  ` in -- The Unix time values.
` in -- The TT2000 epoch values.

TT2000toUnixTime converts epoch time(s) in CDF_TIME_TT2000 (TT2000) type into Unix time(s). A
CDF_TIME_TT2000 epoch, a 8-byte integer, is nanoseconds from J2000 with leap seconds, while Unix time, a double,
is seconds from 1970-01-01T00:00:00.000. The Unix time can have sub-second, with a time resolution of
microseconds, in its fractional part. **Note:** As CDF_TIME_TT2000 has much higher time resolution, sub-microseconds
portion of its time might get lost during the conversion. Also, TT2000’s leap seconds will get lost during conversion.

7.9 UnixTimetoTT2000

long UnixTimetoTT2000 (
epoch as double)  ` in -- The TT2000 epoch value.
` in -- The Unix time value.

long() UnixTimetoTT2000 (
epochs as double())  ` in -- The TT2000 epoch values.
` in -- The Unix time values.

UnixTimetoTT2000 converts Unix time(s) into epoch time(s) in CDF_TIME_TT2000 (TT2000) type. A Unix time, a
double, is seconds from 1970-01-01T00:00:00.000 while a CDF_TIME_TT2000 epoch, a 8-byte integer, is
nanoseconds from J2000 with leap seconds. The Unix time can have sub-second, with a time resolution of
microseconds, in its fractional part. Sub-microseconds will be filled with 0’s when converting from Unix time to
TT2000.
8 CDF Utility Methods

Several methods are created that are mainly used to decipher the strings and their corresponding constant values or vice versa. All these APIs are defined as static methods in CDFUtils class. The constant values are defined in CDFConstants class.

8.1 CDFFileExists

boolean CDFFileExists(
    filename as string)

CDFFileExists method checks whether a CDF file by the given file name, with or without the .cdf extension, exists. Even the file exists, CDFFileExists will not be able to verify whether it is a valid one. (Use CDFopen to validate it).

8.2 CDFgetChecksumValue

integer CDFgetChecksumValue(
    checksum as string)

CDFgetChecksumValue method returns the corresponding file checksum type value, based on the passed string. The file checksum types and their values are as follows:

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NONE</td>
<td>NO_CHECKSUM (0)</td>
</tr>
<tr>
<td>MD5</td>
<td>MD5_CHECKSUM (1)</td>
</tr>
<tr>
<td>OTHER</td>
<td>OTHER_CHECKSUM</td>
</tr>
</tbody>
</table>

8.3 CDFgetCompressionTypeValue

integer CDFgetCompressionTypeValue(
    compressionType as string)

CDFgetCompressionTypeValue method returns the corresponding compression type value, based on the passed string. The compression types and values are as follows:

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NONE</td>
<td>NO_COMPRESSION (0)</td>
</tr>
</tbody>
</table>
8.4 CDFgetDataTypeValue

integer CDFgetDataTypeValue(
  dataType as string)

CDFgetDataTypeValue method returns the corresponding data type value, based on the passed string. The data types and their values are as follows:

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDF_BYTE</td>
<td>CDF_BYTE (41)</td>
</tr>
<tr>
<td>CDF_CHAR</td>
<td>CDF_CHAR (51)</td>
</tr>
<tr>
<td>CDF_UCHAR</td>
<td>CDF_UCHAR (52)</td>
</tr>
<tr>
<td>CDF_INT1</td>
<td>CDF_INT1 (1)</td>
</tr>
<tr>
<td>CDF_UINT1</td>
<td>CDF_UINT1 (11)</td>
</tr>
<tr>
<td>CDF_INT2</td>
<td>CDF_INT2 (2)</td>
</tr>
<tr>
<td>CDF_UINT2</td>
<td>CDF_UINT2 (12)</td>
</tr>
<tr>
<td>CDF_INT4</td>
<td>CDF_INT4 (4)</td>
</tr>
<tr>
<td>CDF_UINT4</td>
<td>CDF_UINT4 (14)</td>
</tr>
<tr>
<td>CDF_INT8</td>
<td>CDF_INT8 (8)</td>
</tr>
<tr>
<td>CDF_REAL4</td>
<td>CDF_REAL4 (21)</td>
</tr>
<tr>
<td>CDF_FLOAT</td>
<td>CDF_FLOAT (44)</td>
</tr>
<tr>
<td>CDF_REAL8</td>
<td>CDF_REAL8 (22)</td>
</tr>
<tr>
<td>CDF_DOUBLE</td>
<td>CDF_DOUBLE (45)</td>
</tr>
<tr>
<td>CDF_EPOCH</td>
<td>CDF_EPOCH (31)</td>
</tr>
<tr>
<td>CDF_EPOCH16</td>
<td>CDF_EPOCH16 (32)</td>
</tr>
<tr>
<td>CDF_TIME_TT2000</td>
<td>CDF_TIME_TT2000 (33)</td>
</tr>
</tbody>
</table>

8.5 CDFgetDecodingValue

integer CDFgetDecodingValue(
  decoding as string)

CDFgetDecodingValue method returns the corresponding data decoding value, based on the passed string. The data decodings and their values are as follows:

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NETWORK</td>
<td>NETWORK_DECODING (1)</td>
</tr>
<tr>
<td>SUN</td>
<td>SUN_DECODING (2)</td>
</tr>
<tr>
<td>VAX</td>
<td>VAX_DECODING (3)</td>
</tr>
<tr>
<td>DECASTATION</td>
<td>DECASTATION_DECODING (4)</td>
</tr>
<tr>
<td>SGi</td>
<td>SGi_DECODING (5)</td>
</tr>
<tr>
<td>IBMPC</td>
<td>IBMPC_DECODING (6)</td>
</tr>
<tr>
<td>IBMRS</td>
<td>IBMRS_DECODING (7)</td>
</tr>
</tbody>
</table>
8.6  CDFgetEncodingValue

integer CDFgetEncodingValue(
    encoding as string)

CDFgetEncodingValue method returns the corresponding data encoding value, based on the passed string. The data encodings and their values are as follows:

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NETWORK</td>
<td>NETWORK_ENCODING (1)</td>
</tr>
<tr>
<td>SUN</td>
<td>SUN_ENCODING (2)</td>
</tr>
<tr>
<td>VAX</td>
<td>VAX_ENCODING (3)</td>
</tr>
<tr>
<td>DECSTATION</td>
<td>DECSTATION_ENCODING (4)</td>
</tr>
<tr>
<td>SGi</td>
<td>SGi_ENCODING (5)</td>
</tr>
<tr>
<td>IBMPC</td>
<td>IBMPC_ENCODING (6)</td>
</tr>
<tr>
<td>IBMRS</td>
<td>IBMRS_ENCODING (7)</td>
</tr>
<tr>
<td>HOST</td>
<td>HOST_ENCODING (8)</td>
</tr>
<tr>
<td>PPC</td>
<td>PPC_ENCODING (9)</td>
</tr>
<tr>
<td>HP</td>
<td>HP_ENCODING (11)</td>
</tr>
<tr>
<td>NeXT</td>
<td>NeXT_ENCODING (12)</td>
</tr>
<tr>
<td>ALPHAOSF1</td>
<td>ALPHAOSF1_ENCODING (13)</td>
</tr>
<tr>
<td>ALPHAVMSd</td>
<td>ALPHAVMSd_ENCODING (14)</td>
</tr>
<tr>
<td>ALPHAVMSg</td>
<td>ALPHAVMSg_ENCODING (15)</td>
</tr>
<tr>
<td>ALPHAVMSi</td>
<td>ALPHAVMSi_ENCODING (16)</td>
</tr>
</tbody>
</table>

8.7  CDFgetFormatValue

integer CDFgetFormatValue(
    format as string)

CDFgetFormatValue method returns the corresponding file format value, based on the passed string. The file formats and their values are as follows:

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SINGLE'</td>
<td>SINGLE_FILE (1)</td>
</tr>
<tr>
<td>MULTI</td>
<td>MULTI_FILE (2)</td>
</tr>
</tbody>
</table>
### 8.8 CDFgetMajorityValue

```c
integer CDFgetMajorityValue(
    majority as string)
```

CDFgetMajorityValue method returns the corresponding file majority value, based on the passed string. The file majorities and their values are as follows:

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROW</td>
<td>ROW_MAJOR (1)</td>
</tr>
<tr>
<td>COLUMN</td>
<td>COLUMN_MAJOR (2)</td>
</tr>
</tbody>
</table>

### 8.9 CDFgetSparseRecordValue

```c
integer CDFgetSparseRecordValue(
    sparseRecord as string)
```

CDFgetSparseRecordValue method returns the corresponding sparse record value, based on the passed string. The sparse records types and their values are as follows:

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NONE</td>
<td>NO_SPARSERECORDS (0)</td>
</tr>
<tr>
<td>PAD</td>
<td>PAD_SPARSERECORDS (1)</td>
</tr>
<tr>
<td>PREV</td>
<td>PREV_SPARSERECORDS (2)</td>
</tr>
</tbody>
</table>

### 8.10 CDFgetStringChecksum

```c
string CDFgetStringChecksum(
    checksum as integer)
```

CDFgetStringChecksum method returns the corresponding file checksum string, based on the passed type. The file checksum types and their values are the same as those defined in CDFgetChecksumValue method.

### 8.11 CDFgetStringCompressionType

```c
string CDFgetStringCompressionType(
    compressionType as integer)
```

CDFgetStringCompressionType method returns the corresponding compression type string, based on the passed type. The file checksum types and their values are the same as those defined in CDFgetCompressionTypeValue method.
8.12 **CDFgetStringDataType**

```c
string CDFgetStringDataType(  
    dataType as integer)  

' out  -- The data type string.
' in   -- The data type.
```

CDFgetStringDataType method returns the corresponding data type string, based on the passed type. The data types and their values are the same as those in CDFgetDataTypeValue method:

8.13 **CDFgetStringDecoding**

```c
string CDFgetStringDecoding(  
    decoding as integer)  

' out  -- The decoding string.
' in   -- The data decoding type.
```

CDFgetStringDecoding method returns the corresponding data decoding string, based on the passed type. The data decodings and their values are as same as those defined in CDFgetDecodingValue:

8.14 **CDFgetStringEncoding**

```c
string CDFgetStringEncoding(  
    encoding as integer)  

' out  -- The encoding string.
' in   -- The data encoding type.
```

CDFgetStringEncoding method returns the corresponding data encoding string, based on the passed type. The data encodings and their values are the same as those defined in CDFgetEncodingValue method:

8.15 **CDFgetStringFormat**

```c
string CDFgetStringFormat(  
    format as integer)  

' out  -- The format string.
' in   -- The file format type.
```

CDFgetStringFormat method returns the corresponding file format string, based on the passed type. The file formats and their values are the same as those defined in CDFgetFormatValue method:

8.16 **CDFgetStringMajority**

```c
string CDFgetStringMajority(  
    majority as integer)  

' out  -- The majority string.
' in   -- The data majority type.
```

CDFgetStringMajority method returns the corresponding file majority string, based on the passed type. The file majorities and their values are the same as those defined in CDFgetMajorityValue method.
8.17 CDFgetStringSparseRecord

string CDFgetStringSparseRecord(
    sparseRecord as integer)

    ' out  -- The sparse record string.
    ' in   -- The sparse record type.

CDFgetStringSparseRecord method returns the corresponding sparse record string, based on the passed type. The sparse records types and their values are the same as those defined in CDFgetSparseRecordValue method.
9  CDF Exception Methods

Several methods in the CDFexception class can be used to check what happens when an exception is thrown by the
CDFAPIs, and react to it if necessary. All these APIs are defined as static methods. CDFException inherits from VB’s
Exception class.

9.1  CDFgetCurrentStatus

integer CDFgetCurrentStatus()
   {' out -- The status.

CDFgetCurrentStatus method returns the status when an exception is detected. The status value should be a negative
value. Chapter 5 covers all possible status codes. Use the following CDFgetStatusMsg method to decipher what the
status means.

9.2  CDFgetStatusMsg

string CDFgetStatusMsg(
   {' out -- The descriptive message.
   status as integer)  {' in -- The exception status.

CDFgetStatusMsg method returns the descriptive information of the passed status.
Appendix A

A.1 Introduction

A status code is returned from most CDF functions. The CDFConstants class contains the numerical values (constants) for each of the status codes (and for any other constants referred to in the explanations). The method CDFerror can be used within a program to inquire the explanation text for a given status code.

There are three classes of status codes: informational, warning, and error. The purpose of each is as follows:

- **Informational**: Indicates success but provides some additional information that may be of interest to an application.
- **Warning**: Indicates that the method completed but possibly not as expected.
- **Error**: Indicates that a fatal error occurred and the function aborted.

Status codes fall into classes as follows:

- Error codes < CDF_WARN < Warning codes < CDF_OK < Informational codes

CDF_OK indicates an unqualified success (it should be the most commonly returned status code). CDF_WARN is simply used to distinguish between warning and error status codes.

A.2 Status Codes and Messages

The following list contains an explanation for each possible status code. Whether a particular status code is considered informational, a warning, or an error is also indicated.

- **ATTR_EXISTS**: Named attribute already exists - cannot create or rename. Each attribute in a CDF must have a unique name. Note that trailing blanks are ignored by the CDF library when comparing attribute names. [Error]

- **ATTR_NAME_TRUNC**: Attribute name truncated to CDF_ATTR_NAME_LEN256 characters. The attribute was created but with a truncated name. [Warning]

- **BAD_ALLOCATE_RECS**: An illegal number of records to allocate for a variable was specified. For RV variables the number must be one or greater. For NRV variables the number must be exactly one. [Error]

- **BAD_ARGUMENT**: An illegal/undefined argument was passed. Check that all arguments are properly declared and initialized. [Error]
BAD_ATTR_NAME  Illegal attribute name specified. Attribute names must contain at least one character, and each character must be printable. [Error]

BAD_ATTR_NUM  Illegal attribute number specified. Attribute numbers must be zero (0) or greater for C applications and one (1) or greater for Fortran applications. [Error]

BAD_BLOCKING_FACTOR  An illegal blocking factor was specified. Blocking factors must be at least zero (0). [Error]

BAD_CACHESIZE  An illegal number of cache buffers was specified. The value must be at least zero (0). [Error]

BAD_CDF_EXTENSION  An illegal file extension was specified for a CDF. In general, do not specify an extension except possibly for a single-file CDF that has been renamed with a different file extension or no file extension. [Error]

BAD_CDF_ID  CDF identifier is unknown or invalid. The CDF identifier specified is not for a currently open CDF. [Error]

BAD_CDF_NAME  Illegal CDF name specified. CDF names must contain at least one character, and each character must be printable. Trailing blanks are allowed but will be ignored. [Error]

BAD_INT  Unknown CDF status code received. The CDF library does not use the status code specified. [Error]

BAD_CHECKSUM  An illegal checksum mode received. It is invalid or currently not supported. [Error]

BAD_COMPRESSION_PARM  An illegal compression parameter was specified. [Error]

BAD_DATA_TYPE  An unknown data type was specified or encountered. The CDF data types are defined in CDFConstants class for VB applications. [Error]

BAD_DECODING  An unknown decoding was specified. The CDF decodings are defined in CDFConstants class for VB applications. [Error]

BAD_DIM_COUNT  Illegal dimension count specified. A dimension count must be at least one (1) and not greater than the size of the dimension. [Error]

BAD_DIM_INDEX  One or more dimension index is out of range. A valid value must be specified regardless of the dimension variance. Note also that the combination of dimension index, count, and interval must not specify an element beyond the end of the dimension. [Error]

BAD_DIM_INTERVAL  Illegal dimension interval specified. Dimension intervals must be at least one (1). [Error]

BAD_DIM_SIZE  Illegal dimension size specified. A dimension size must be at least one (1). [Error]

---

22 The status code BAD_BLOCKING_FACTOR was previously named BAD_EXTEND_RECS.
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAD_ENCODING</td>
<td>Unknown data encoding specified. The CDF encodings are defined in CDFConstants class for VB applications. [Error]</td>
</tr>
<tr>
<td>BAD_ENTRY_NUM</td>
<td>Illegal attribute entry number specified. Entry numbers must be at least zero (0) for VB applications. [Error]</td>
</tr>
<tr>
<td>BAD_FNC_OR_ITEM</td>
<td>The specified function or item is illegal. Check that the proper number of arguments are specified for each operation being performed. [Error]</td>
</tr>
<tr>
<td>BAD_FORMAT</td>
<td>Unknown format specified. The CDF formats are defined in CDFConstants class for VB applications. [Error]</td>
</tr>
<tr>
<td>BAD_INITIAL_RECS</td>
<td>An illegal number of records to initially write has been specified. The number of initial records must be at least one (1). [Error]</td>
</tr>
<tr>
<td>BAD_MAJORITY</td>
<td>Unknown variable majority specified. The CDF variable majorities are defined in CDFConstants class for VB applications. [Error]</td>
</tr>
<tr>
<td>BAD_MALLOC</td>
<td>Unable to allocate dynamic memory - system limit reached. Contact CDF User Support if this error occurs. [Error]</td>
</tr>
<tr>
<td>BAD_NEGtoPOSfp0_MODE</td>
<td>An illegal -0.0 to 0.0 mode was specified. The -0.0 to 0.0 modes are defined in CDFConstants class for VB applications. [Error]</td>
</tr>
<tr>
<td>BAD_NUM_DIMS</td>
<td>The number of dimensions specified is out of the allowed range. Zero (0) through CDF_MAX_DIMS dimensions are allowed. If more are needed, contact CDF User Support. [Error]</td>
</tr>
<tr>
<td>BAD_NUM_ELEMS</td>
<td>The number of elements of the data type is illegal. The number of elements must be at least one (1). For variables with a non-character data type, the number of elements must always be one (1). [Error]</td>
</tr>
<tr>
<td>BAD_NUM_VARS</td>
<td>Illegal number of variables in a record access operation. [Error]</td>
</tr>
<tr>
<td>BAD_READONLY_MODE</td>
<td>Illegal read-only mode specified. The CDF read-only modes are defined in CDFConstants class for VB applications. [Error]</td>
</tr>
<tr>
<td>BAD_REC_COUNT</td>
<td>Illegal record count specified. A record count must be at least one (1). [Error]</td>
</tr>
<tr>
<td>BAD_REC_INTERVAL</td>
<td>Illegal record interval specified. A record interval must be at least one (1). [Error]</td>
</tr>
<tr>
<td>BAD_REC_NUM</td>
<td>Record number is out of range. Record numbers must be at least zero (0) for C applications and at least one (1) for Fortran applications. Note that a valid value must be specified regardless of the record variance. [Error]</td>
</tr>
<tr>
<td>BAD_SCOPE</td>
<td>Unknown attribute scope specified. The attribute scopes are defined in CDFConstants class for VB applications. [Error]</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>BAD_SCRATCH_DIR</td>
<td>An illegal scratch directory was specified. The scratch directory must be writable and accessible (if a relative path was specified) from the directory in which the application has been executed. [Error]</td>
</tr>
<tr>
<td>BAD_SPARSEARRAYS_PARM</td>
<td>An illegal sparse arrays parameter was specified. [Error]</td>
</tr>
<tr>
<td>BAD_VAR_NAME</td>
<td>Illegal variable name specified. Variable names must contain at least one character and each character must be printable. [Error]</td>
</tr>
<tr>
<td>BAD_VAR_NUM</td>
<td>Illegal variable number specified. Variable numbers must be zero (0) or greater for VB applications. [Error]</td>
</tr>
<tr>
<td>BAD_zMODE</td>
<td>Illegal zMode specified. The CDF zModes are defined in CDFConstants class for VB applications. [Error]</td>
</tr>
<tr>
<td>CANNOT_ALLOCATE_RECORDS</td>
<td>Records cannot be allocated for the given type of variable (e.g., a compressed variable). [Error]</td>
</tr>
<tr>
<td>CANNOT_CHANGE</td>
<td>Because of dependencies on the value, it cannot be changed. Some possible causes of this error follow:</td>
</tr>
<tr>
<td></td>
<td>1. Changing a CDF's data encoding after a variable value (including a pad value) or an attribute entry has been written.</td>
</tr>
<tr>
<td></td>
<td>2. Changing a CDF's format after a variable has been created or if a compressed single-file CDF.</td>
</tr>
<tr>
<td></td>
<td>3. Changing a CDF's variable majority after a variable value (excluding a pad value) has been written.</td>
</tr>
<tr>
<td></td>
<td>4. Changing a variable's data specification after a value (including the pad value) has been written to that variable or after records have been allocated for that variable.</td>
</tr>
<tr>
<td></td>
<td>5. Changing a variable's record variance after a value (excluding the pad value) has been written to that variable or after records have been allocated for that variable.</td>
</tr>
<tr>
<td></td>
<td>6. Changing a variable's dimension variances after a value (excluding the pad value) has been written to that variable or after records have been allocated for that variable.</td>
</tr>
<tr>
<td></td>
<td>7. Writing “initial” records to a variable after a value (excluding the pad value) has already been written to that variable.</td>
</tr>
<tr>
<td></td>
<td>8. Changing a variable's blocking factor when a compressed variable and a value (excluding the pad value) has been written or when a variable with sparse records and a value has been accessed.</td>
</tr>
<tr>
<td></td>
<td>9. Changing an attribute entry's data specification where the new specification is not equivalent to the old specification.</td>
</tr>
<tr>
<td><strong>Error Code</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>---------------</td>
<td>----------------</td>
</tr>
<tr>
<td>CANNOT_COMPRESS</td>
<td>The CDF or variable cannot be compressed. For CDFs, this occurs if the CDF has the multi-file format. For variables, this occurs if the variable is in a multi-file CDF, values have been written to the variable, or if sparse arrays have already been specified for the variable. [Error]</td>
</tr>
<tr>
<td>CANNOT_SPARSEARRAYS</td>
<td>Sparse arrays cannot be specified for the variable. This occurs if the variable is in a multi-file CDF, values have been written to the variable, or if compression has already been specified for the variable. [Error]</td>
</tr>
<tr>
<td>CANNOT_SPARSERECORDS</td>
<td>Sparse records cannot be specified for the variable. This occurs if the variable is in a multi-file CDF, values have been written to the variable, or records have been allocated for the variable. [Error]</td>
</tr>
<tr>
<td>CDF_CLOSE_ERROR</td>
<td>Error detected while trying to close CDF. Check that sufficient disk space exists for the dotCDF file and that it has not been corrupted. [Error]</td>
</tr>
<tr>
<td>CDF_CREATE_ERROR</td>
<td>Cannot create the CDF specified - error from file system. Make sure that sufficient privilege exists to create the dotCDF file in the disk/directory location specified and that an open file quota has not already been reached. [Error]</td>
</tr>
<tr>
<td>CDF_DELETE_ERROR</td>
<td>Cannot delete the CDF specified - error from file system. Insufficient privileges exist to delete the CDF file(s). [Error]</td>
</tr>
<tr>
<td>CDF_EXISTS</td>
<td>The CDF named already exists - cannot create it. The CDF library will not overwrite an existing CDF. [Error]</td>
</tr>
<tr>
<td>CDF_INTERNAL_ERROR</td>
<td>An unexpected condition has occurred in the CDF library. Report this error to CDFsupport. [Error]</td>
</tr>
<tr>
<td>CDF_NAME_TRUNC</td>
<td>CDF file name truncated to CDF_PATHNAME_LEN characters. The CDF was created but with a truncated name. [Warning]</td>
</tr>
<tr>
<td>CDF_OK</td>
<td>Function completed successfully.</td>
</tr>
<tr>
<td>CDF_OPEN_ERROR</td>
<td>Cannot open the CDF specified - error from file system. Check that the dotCDF file is not corrupted and that sufficient privilege exists to open it. Also check that an open file quota has not already been reached. [Error]</td>
</tr>
<tr>
<td>CDF_READ_ERROR</td>
<td>Failed to read the CDF file - error from file system. Check that the dotCDF file is not corrupted. [Error]</td>
</tr>
<tr>
<td>CDF_WRITE_ERROR</td>
<td>Failed to write the CDF file - error from file system. Check that the dotCDF file is not corrupted. [Error]</td>
</tr>
<tr>
<td>CHECKSUM_ERROR</td>
<td>The data integrity verification through the checksum failed. [Error]</td>
</tr>
<tr>
<td>CHECKSUM_NOT_ALLOWED</td>
<td>The checksum is not allowed for old versioned files. [Error]</td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>COMPRESSION_ERROR</td>
<td>An error occurred while compressing a CDF or block of variable records. This is an internal error in the CDF library. Contact CDF User Support. [Error]</td>
</tr>
<tr>
<td>CORRUPTED_V2_CDF</td>
<td>This Version 2 CDF is corrupted. An error has been detected in the CDF's control information. If the CDF file(s) are known to be valid, please contact CDF User Support. [Error]</td>
</tr>
<tr>
<td>DECOMPRESSION_ERROR</td>
<td>An error occurred while decompressing a CDF or block of variable records. The most likely cause is a corrupted dotCDF file. [Error]</td>
</tr>
<tr>
<td>DID_NOT_COMPRESS</td>
<td>For a compressed variable, a block of records did not compress to smaller than their uncompressed size. They have been stored uncompressed. This can result If the blocking factor is set too low or if the characteristics of the data are such that the compression algorithm chosen is unsuitable. [Informational]</td>
</tr>
<tr>
<td>EMPTY_COMPRESSED_CDF</td>
<td>The compressed CDF being opened is empty. This will result if a program, which was creating/modifying, the CDF abnormally terminated. [Error]</td>
</tr>
<tr>
<td>END_OF_VAR</td>
<td>The sequential access current value is at the end of the variable. Reading beyond the end of the last physical value for a variable is not allowed (when performing sequential access). [Error]</td>
</tr>
<tr>
<td>FORCED_PARAMETER</td>
<td>A specified parameter was forced to an acceptable value (rather than an error being returned). [Warning]</td>
</tr>
<tr>
<td>IBM_PC_OVERFLOW</td>
<td>An operation involving a buffer greater than 64k bytes in size has been specified for PCs running 16-bit DOS/Windows 3.*. [Error]</td>
</tr>
<tr>
<td>ILLEGAL_EPOCH_VALUE</td>
<td>Illegal component is detected in computing an epoch value or an illegal epoch value is provided in decomposing an epoch value. [Error]</td>
</tr>
<tr>
<td>ILLEGAL_FOR_SCOPE</td>
<td>The operation is illegal for the attribute's scope. For example, only gEntries may be written for gAttributes - not rEntries or zEntries. [Error]</td>
</tr>
<tr>
<td>ILLEGAL_IN_ZMODE</td>
<td>The attempted operation is illegal while in zMode. Most operations involving rVariables or rEntries will be illegal. [Error]</td>
</tr>
<tr>
<td>ILLEGAL_ON_V1_CDF</td>
<td>The specified operation (i.e., opening) is not allowed on Version 1 CDFs. [Error]</td>
</tr>
<tr>
<td>MULTI_FILE_FORMAT</td>
<td>The specified operation is not applicable to CDFs with the multi-file format. For example, it does not make sense to inquire indexing statistics for a variable in a multi-file CDF (indexing is only used in single-file CDFs). [Informational]</td>
</tr>
<tr>
<td>NA_FOR_VARIABLE</td>
<td>The attempted operation is not applicable to the given variable. [Warning]</td>
</tr>
</tbody>
</table>
NEGATIVE_FP_ZERO: One or more of the values read/written are -0.0  (An illegal value on VAXes and DEC Alphas running OpenVMS).  [Warning]

NO_ATTR_SELECTED: An attribute has not yet been selected.  First select the attribute on which to perform the operation.  [Error]

NO_CDF_SELECTED: A CDF has not yet been selected.  First select the CDF on which to perform the operation.  [Error]

NO_DELETE_ACCESS: Deleting is not allowed  (read-only access).  Make sure that delete access is allowed on the CDF file(s).  [Error]

NO_ENTRY_SELECTED: An attribute entry has not yet been selected.  First select the entry number on which to perform the operation.  [Error]

NO_MORE_ACCESS: Further access to the CDF is not allowed because of a severe error.  If the CDF was being modified, an attempt was made to save the changes made prior to the severe error.  in any event, the CDF should still be closed.  [Error]

NO_PADVALUE_SPECIFIED: A pad value has not yet been specified.  The default pad value is currently being used for the variable.  The default pad value was returned.  [Informational]

NO_STATUS_SELECTED: A CDF status code has not yet been selected.  First select the status code on which to perform the operation.  [Error]

NO_SUCH_ATTR: The named attribute was not found.  Note that attribute names are case-sensitive.  [Error]

NO_SUCH_CDF: The specified CDF does not exist.  Check that the file name specified is correct.  [Error]

NO_SUCH_ENTRY: No such entry for specified attribute.  [Error]

NO_SUCH_RECORD: The specified record does not exist for the given variable.  [Error]

NO_SUCH_VAR: The named variable was not found.  Note that variable names are case-sensitive.  [Error]

NO_VAR_SELECTED: A variable has not yet been selected.  First select the variable on which to perform the operation.  [Error]

NO_VARS_IN_CDF: This CDF contains no rVariables.  The operation performed is not applicable to a CDF with no rVariables.  [Informational]

NO_WRITE_ACCESS: Write access is not allowed on the CDF file(s).  Make sure that the CDF file(s) have the proper file system privileges and ownership.  [Error]

NOT_A_CDF: Named CDF is corrupted or not actually a CDF.  Contact CDF User Support if you are sure that the specified file is a CDF that should be readable by the CDF distribution being used.  [Error]

NOT_A_CDF_OR_NOT_SUPPORTED: This can occur if an older CDF distribution is being used to read a CDF created by a more recent CDF distribution.  Contact CDF
User Support if you are sure that the specified file is a CDF that should be readable by the CDF distribution being used. CDF is backward compatible but not forward compatible. [Error]

**PRECEEDING_RECORDS_ALLOCATED**
Because of the type of variable, records preceding the range of records being allocated were automatically allocated as well. [Informational]

**READ_ONLY_DISTRIBUTION**
Your CDF distribution has been built to allow only read access to CDFs. Check with your system manager if you require write access. [Error]

**READ_ONLY_MODE**
The CDF is in read-only mode - modifications are not allowed. [Error]

**SCRATCH_CREATE_ERROR**
Cannot create a scratch file - error from file system. If a scratch directory has been specified, ensure that it is writeable. [Error]

**SCRATCH_DELETE_ERROR**
Cannot delete a scratch file - error from file system. [Error]

**SCRATCH_READ_ERROR**
Cannot read from a scratch file - error from file system. [Error]

**SCRATCH_WRITE_ERROR**
Cannot write to a scratch file - error from file system. [Error]

**SINGLE_FILE_FORMAT**
The specified operation is not applicable to CDFs with the single-file format. For example, it does not make sense to close a variable in a single-file CDF. [Informational]

**SOME_ALREADY_ALLOCATED**
Some of the records being allocated were already allocated. [Informational]

**TOO_MANY_PARMS**
A type of sparse arrays or compression was encountered having too many parameters. This could be caused by a corrupted CDF or if the CDF was created/modified by a CDF distribution more recent than the one being used. [Error]

**TOO_MANY_VARS**
A multi-file CDF on a PC may contain only a limited number of variables because of the 8.3 file naming convention of MS-DOS. This consists of 100 rVariables and 100 zVariables. [Error]

**UNKNOWN_COMPRESSION**
An unknown type of compression was specified or encountered. [Error]

**UNKNOWN_SPARSENESS**
An unknown type of sparseness was specified or encountered. [Error]

**UNSUPPORTED_OPERATION**
The attempted operation is not supported at this time. [Error]

**VAR_ALREADY_CLOSED**
The specified variable is already closed. [Informational]

**VAR_CLOSE_ERROR**
Error detected while trying to close variable file. Check that sufficient disk space exists for the variable file and that it has not been corrupted. [Error]

**VAR_CREATE_ERROR**
An error occurred while creating a variable file in a multi-file CDF. Check that a file quota has not been reached. [Error]
VAR_DELETE_ERROR  An error occurred while deleting a variable file in a multi-file CDF. Check that sufficient privilege exist to delete the CDF files. [Error]

VAR_EXISTS  Named variable already exists - cannot create or rename. Each variable in a CDF must have a unique name (rVariables and zVariables can not share names). Note that the CDF library when comparing variable names ignores trailing blanks. [Error]

VAR_NAME_TRUNC  Variable name truncated to CDF_VAR_NAME_LEN256 characters. The variable was created but with a truncated name. [Warning]

VAR_OPEN_ERROR  An error occurred while opening variable file. Check that sufficient privilege exists to open the variable file. Also make sure that the associated variable file exists. [Error]

VAR_READ_ERROR  Failed to read variable as requested - error from file system. Check that the associated file is not corrupted. [Error]

VAR_WRITE_ERROR  Failed to write variable as requested - error from file system. Check that the associated file is not corrupted. [Error]

VIRTUAL_RECORD_DATA  One or more of the records are virtual (never actually written to the CDF). Virtual records do not physically exist in the CDF file(s) but are part of the conceptual view of the data provided by the CDF library. Virtual records are described in the Concepts chapter in the CDF User's Guide. [Informational]
Appendix B

B.1 VB-CDF APIs

The APIs that have the TYPE symbol use a general form for dealing with data, either variable value(s) or attribute entry, in various data type for input and output. TYPE can be specified either in VB basic value or string type (scalar or array) for writing out and reading from a CDF. The VB base Object class can also be used to represent a data object reading from a CDF, which will be a scalar or array of value or string type.

integer CDFattrCreate (id, attrName, attrScope, attrNum)
  id as long                  ' in
  attrName as string         ' in
  attrScope as integer       ' in
  attrNum as integer         ' in

integer CDFattrEntryInquire (id, attrNum, entryNum, dataType, numElements)
  id as long                  ' in
  attrNum as integer          ' in
  entryNum as integer         ' in
  dataType as integer         ' in
  numElements as integer      ' out

integer CDFattrGet (id, attrNum, entryNum, value)
  id as long                  ' in
  attrNum as integer          ' in
  entryNum as integer         ' in
  value as TYPE               ' out

integer CDFattrInquire (id, attrNum, attrName, attrScope, maxEntry)
  id as long                  ' in
  attrNum as integer          ' in
  attrName as string          ' out
  attrScope as integer        ' out
  maxEntry as integer         ' out

integer CDFattrNum (id, attrName)
  id as long                  ' in
  attrName as string          ' in

integer CDFattrPut (id, attrNum, entryNum, dataType, numElements, value)
  id as long                  ' in
  attrNum as integer          ' in
  entryNum as integer         ' in
  dataType as integer         ' in
  numElements as integer      ' in
value as **TYPE**

integer CDFAttrRename (id, attrNum, attrName)  
id as long  
attrNum as integer  
attrName as string

integer CDFclose (id)  
id as long

integer CDFcloseCDF (id)  
id as long

integer CDFcloserVar (id, varNum)  
id as long  
varNum as integer

integer CDFclosezVar (id, varNum)  
id as long  
varNum as integer

integer CDFconfirmAttrExistence (id, attrName)  
id as long  
attrName as string

integer CDFconfirmgEntryExistence (id, attrNum, entryNum)  
id as long  
attrNum as integer  
entryNum as integer

integer CDFconfirmrEntryExistence (id, attrNum, entryNum)  
id as long  
attrNum as integer  
entryNum as integer

integer CDFconfirmrVarExistence (id, varNum)  
id as long  
varNum as integer

integer CDFconfirmrVarPadValueExistence (id, varNum)  
id as long  
varNum as integer

integer CDFconfirmzEntryExistence (id, attrNum, entryNum)  
id as long  
attrNum as integer  
entryNum as integer

integer CDFconfirmzVarExistence (id, varNum)  
id as long  
varNum as integer

integer CDFconfirmzVarPadValueExistence (id, varNum)  
id as long  
varNum as integer
integer CDFcreate  (CDFname, numDims, dimSizes, encoding, majority, id)
CDFname as string  ' in
numDims as integer  ' in
dimSizes as integer()  ' in
encoding as integer  ' in
majority as integer  ' in
id as long  ' out

integer CDFcreateAttr  (id, attrName, scope, attrNum)
id as long  ' in
attrName as string  ' in
scope as integer  ' in
attrNum as integer  ' out

integer CDFcreateCDF  (CDFname, id)
CDFname as string  ' in
id as long  ' out

integer CDFcreateVar  (id, varName, dataType, numElements, recVary, dimVarys, varNum)
id as long  ' in
varName as string  ' in
dataType as integer  ' in
numElements as integer  ' in
recVary as integer  ' in
dimVarys as integer()  ' in
varNum as integer  ' out

integer CDFcreatezVar  (id, varName, dataType, numElements, numDims, dimSizes, recVary, dimVarys, varNum)
id as long  ' in
varName as string  ' in
dataType as integer  ' in
numElements as integer  ' in
numDims as integer  ' in
dimSizes as integer()  ' in
recVary as integer  ' in
dimVarys as integer()  ' in
varNum as integer  ' out

integer CDFdelete  (id)
id as long  ' in

integer CDFdeleteAttr  (id, attrNum)
id as long  ' in
attrNum as integer  ' in

integer CDFdeleteAttrEntry  (id, attrNum, entryNum)
id as long  ' in
attrNum as integer  ' in
entryNum as integer  ' in

integer CDFdeleteAttrEntry  (id, attrNum, entryNum)
id as long  ' in
attrNum as integer  ' in
entryNum as integer  ' in

integer CDFdeleteAttrzEntry  (id, attrNum, entryNum)
id as long
attrNum as integer
entryNum as integer

integer CDFdeleteCDF (id)
id as long

integer CDFdeleterVar (id, varNum)
id as long
varNum as integer

integer CDFdeleterVarRecords (id, varNum, startRec, endRec)
id as long
varNum as integer
startRec as integer
endRec as integer

integer CDFdeleterVarRecordsRenumber (id, varNum, startRec, endRec)
id as long
varNum as integer
startRec as integer
endRec as integer

integer CDFdeletezVar (id, varNum)
id as long
varNum as integer

integer CDFdeletezVarRecords (id, varNum, startRec, endRec)
id as long
varNum as integer
startRec as integer
endRec as integer

integer CDFdeletezVarRecordsRenumber (id, varNum, startRec, endRec)
id as long
varNum as integer
startRec as integer
endRec as integer

integer CDFdoc (id, version, release, text)
id as long
version as integer
release as integer
text as string

integer CDFerror (status, message)
status as integer
message as string

integer CDFgetAttrgEntry (id, attrNum, entryNum, value)
id as long
attrNum as integer
entryNum as integer
value as TYPE

integer CDFgetAttrgEntryDataType (id, attrNum, entryNum, dataType)
id as long  ' in
attrNum as integer  ' in
dataType as integer  ' in
numElems as integer  ' out

integer CDFgetAttrgEntry (id, attrNum, entryNum)  ' out
id as long  ' in
attrNum as integer  ' in
entryNum as integer  ' in

integer CDFgetAttrMaxgEntry (id, attrNum, entryNum)  ' out
id as long  ' in
attrNum as integer  ' in
entryNum as integer  ' in

integer CDFgetAttrMaxrEntry (id, attrNum, entryNum)  ' out
id as long  ' in
attrNum as integer  ' in
entryNum as integer  ' in

integer CDFgetAttrMaxzEntry (id, attrNum, entryNum)  ' out
id as long  ' in
attrNum as integer  ' in
entryNum as integer  ' in

integer CDFgetAttrName (id, attrNum, attrName)  ' out
id as long  ' in
attrNum as integer  ' in
attrName as string  ' in

integer CDFgetAttrNum (id, attrName)  ' in
id as long  ' in
attrName as string  ' in

integer CDFgetAttrrEntry (id, attrNum, entryNum, value)  ' out
id as long  ' in
attrNum as integer  ' in
entryNum as integer  ' in
value as TYPE

integer CDFgetAttrrEntryDataType (id, attrNum, entryNum, dataType)  ' out
id as long  ' in
attrNum as integer  ' in
entryNum as integer  ' in
dataType as integer  ' in

integer CDFgetAttrrEntryNumElements (id, attrNum, entryNum, numElems)  ' out
id as long  ' in
attrNum as integer  ' in
entryNum as integer  ' in
numElems as integer  ' in
scope as integer

integer CDFgetAttrzEntry (id, attrNum, entryNum, value)
  id as long ' in
  attrNum as integer ' in
  entryNum as integer ' in
  value as TYPE ' out

integer CDFgetAttrzEntryDataType (id, attrNum, entryNum, dataType)
  id as long ' in
  attrNum as integer ' in
  entryNum as integer ' in
  dataType as integer ' out

integer CDFgetAttrzEntryNumElements (id, attrNum, entryNum, numElems)
  id as long ' in
  attrNum as integer ' in
  entryNum as integer ' in
  numElems as integer ' out

integer CDFgetCacheSize (id, numBuffers)
  id as long ' in
  numBuffers as integer ' out

integer CDFgetChecksum (id, checksum)
  id as long ' in
  checksum as integer ' out

integer CDFgetCompression (id, compType, compParms, compPercent)
  id as long ' in
  compType as integer ' out
  compParms as integer ' out
  compPercent as integer ' out

integer CDFgetCompressionCacheSize (id, numBuffers)
  id as long ' in
  numBuffers as integer ' out

integer CDFgetCompressionInfo (cdfName, compType, compParms, compSize, uncompSize)
  cdfName as string ' in
  compType as integer ' out
  compParms as integer() ' out
  compSize as long ' out
  uncompSize as long ' out

integer CDFgetCopyright (id, copyright)
  id as long ' in
  copyright as string ' out

integer CDFgetDataTypeSize (dataType, numBytes)
  dataType as integer ' in
  numBytes as integer ' out

integer CDFgetDecoding (id, decoding)
  id as long ' in
  decoding as integer ' out
integer CDFgetEncoding (id, encoding)  
id as long  
encoding as integer  

integer CDFgetFileBackward ()  

integer CDFgetFormat (id, format)  
id as long  
format as integer  

integer CDFgetLibraryCopyright (copyright)  
copyright as string  

integer CDFgetLibraryVersion (version, release, increment, subIncrement)  
version as integer  
release as integer  
increment as integer  
subIncrement as string  

integer CDFgetLeapSecondLastUpdated (id, lastUpdated)  
id as long  
lastUpdate as integer  

integer CDFgetMajority (id, majority)  
id as long  
majority as integer  

integer CDFgetMaxWrittenRecNums (id, maxRecrVars, maxReczVars)  
id as long  
maxRecrVars as integer  
maxReczVars as integer  

integer CDFgetName (id, name)  
id as long  
name as string  

integer CDFgetNegtoPosfp0Mode (id, negtoPosfp0)  
id as long  
negtoPosfp0 as integer  

integer CDFgetNumAttrgEntries (id, attrNum, entries)  
id as long  
attrNum as integer  
entries as integer  

integer CDFgetNumAttributes (id, numAttrs)  
id as long  
umAttrs as integer  

integer CDFgetNumAttrrEntries (id, attrNum, entries)  
id as long  
attrNum as integer  
entries as integer  

integer CDFgetNumAttrzEntries (id, attrNum, entries)
id as long
attrNum as integer
entries as integer

integer CDFgetNumAttributes (id, numAttrs)
   id as long ' in
   numAttrs as integer ' in

integer CDFgetNumrVars (id, numrVars)
   id as long ' in
   numrVars as integer ' out

integer CDFgetNumvAttributes (id, numrVars)
   id as long ' in
   numrVars as integer ' out

integer CDFgetNumzVars (id, numrVars)
   id as long ' in
   numzVars as integer ' out

integer CDFgetReadOnlyMode (id, mode)
   id as long ' in
   mode as integer ' out

integer CDFgetrVarAllocRecords (id, varNum, allocRecs)
   id as long ' in
   varNum as integer ' in
   allocRecs as integer ' out

integer CDFgetrVarBlockingFactor (id, varNum, bf)
   id as long ' in
   varNum as integer ' in
   bf as integer ' out

integer CDFgetrVarCacheSize (id, varNum, numBuffers)
   id as long ' in
   varNum as integer ' in
   numBuffers as integer ' out

integer CDFgetrVarCompression (id, varNum, cType, cParms, cPercent)
   id as long ' in
   varNum as integer ' in
   compType as integer ' out
   cParms as integer() ' out
   cPercent as integer ' out

integer CDFgetrVarData (id, varNum, recNum, indices, value)
   id as long ' in
   varNum as integer ' in
   recNum as integer ' in
   indices as integer() ' in
   value as TYPE ' out

integer CDFgetrVarDataType (id, varNum, dataType)
   id as long ' in
   varNum as integer ' in
integer CDFgetrVarsDimSizes (id, varNum, dimSizes)
id as long ' in
varNum as integer ' in
dimSizes as integer() ' out

integer CDFgetrVarDimVariances (id, varNum, dimVarys)
id as long ' in
varNum as integer ' in
dimVarys as integer() ' out

integer CDFgetrVarInfo (id, varNum, dataType, numElems, numDims, dimSizes)
id as long ' in
varNum as integer ' in
dataType as integer ' out
numElems as integer ' out
numDims as integer ' out
dimSizes as integer() ' out

integer CDFgetrVarMaxAllocRecNum (id, varNum, maxRec)
id as long ' in
varNum as integer ' in
maxRec as integer ' out

integer CDFgetrVarMaxWrittenRecNum (id, varNum, maxRec)
id as long ' in
varNum as integer ' in
maxRec as integer ' out

integer CDFgetrVarName (id, varNum, varName)
id as long ' in
varNum as integer ' in
varName as string ' out

integer CDFgetrVarsNumDims (id, varNum, numDims)
id as long ' in
varNum as integer ' in
numDims as integer ' out

integer CDFgetrVarNumElements (id, varNum, numElems)
id as long ' in
varNum as integer ' in
numElems as integer ' out

integer CDFgetrVarNumRecsWritten (id, varNum, numRecs)
id as long ' in
varNum as integer ' in
numRecs as integer ' out

integer CDFgetrVarPadValue (id, varNum, padValue)
id as long ' in
varNum as integer ' in
padValue as TYPE ' out

integer CDFgetrVarRecordData (id, varNum, recNum, buffer)
id as long
varNum as integer
recNum as integer
buffer as TYPE

integer CDFgetrVarRecVariance (id, varNum, recVary)
id as long
varNum as integer
recVary as integer

integer CDFgetrVarReservePercent (id, varNum, percent)
id as long
varNum as integer
percent as integer

integer CDFgetrVarsDimSizes (id, dimSizes)
id as long
dimSizes as integer()

integer CDFgetrVarSeqData (id, varNum, value)
id as long
varNum as integer
value as TYPE

integer CDFgetrVarSeqPos (id, varNum, recNum, indices)
id as long
varNum as integer
recNum as integer
indices as integer()

integer CDFgetrVarsMaxWrittenRecNum (id, recNum)
id as long
recNum as integer

integer CDFgetrVarsNumDims (id, numDims)
id as long
numDims as integer

integer CDFgetrVarSparseRecords (id, varNum, sRecords)
id as long
varNum as integer
sRecords as integer

integer CDFgetStageCacheSize (id, numBuffers)
id as long
numBuffers as integer

integer CDFgetStatusText (status, text)
status as integer
text as string

integer CDFgetValidate ()

integer CDFgetVarNum (id, varName)
id as long
varName as string
integer CDFgetVersion (id, version, release, increment)
id as long ' in
version as integer ' out
release as integer ' out
increment as integer ' out

integer CDFgetzMode (id, zMode)
id as long ' in
zMode as integer ' out

integer CDFgetzVarAllocRecords (id, varNum, allocRecs)
id as long ' in
varNum as integer ' in
allocRecs as integer ' out

integer CDFgetzVarBlockingFactor (id, varNum, bf)
id as long ' in
varNum as integer ' in
bf as integer ' out

integer CDFgetzVarCacheSize (id, varNum, numBuffers)
id as long ' in
varNum as integer ' in
numBuffers as integer ' out

integer CDFgetzVarCompression (id, varNum, cType, cParms, cPercent)
id as long ' in
varNum as integer ' in
compType as integer ' out
cParms as integer() ' out
cPercent as integer ' out

integer CDFgetzVarData (id, varNum, recNum, indices, value)
id as long ' in
varNum as integer ' in
recNum as integer ' in
indices as integer() ' in
value as TYPE ' out

integer CDFgetzVarDataType (id, varNum, dataType)
id as long ' in
varNum as integer ' in
dataType as integer ' out

integer CDFgetzVarDimSizes (id, varNum, dimSizes)
id as long ' in
varNum as integer ' in
dimSizes as integer() ' out

integer CDFgetzVarDimVariances (id, varNum, dimVarys)
id as long ' in
varNum as integer ' in
dimVarys as integer() ' out

integer CDFgetzVarInfo (id, varNum, dataType, numElems, numDims, dimSizes)
id as long ' in
varNum as integer ' in
dataType as integer ' out
numElems as integer ' out
numDims as integer ' out
dimSizes as integer() ' out

integer CDFgetzVarMaxAllocRecNum (id, varNum, maxRec)
id as long ' in
varNum as integer ' in
maxRec as integer ' out

integer CDFgetzVarMaxWrittenRecNum (id, varNum, maxRec)
id as long ' in
varNum as integer ' in
maxRec as integer ' out

integer CDFgetzVarName (id, varNum, varName)
id as long ' in
varNum as integer ' in
varName as string ' out

integer CDFgetzVarNumDims (id, varNum, numDims)
id as long ' in
varNum as integer ' in
numDims as integer ' out

integer CDFgetzVarNumElements (id, varNum, numElems)
id as long ' in
varNum as integer ' in
numElems as integer ' out

integer CDFgetzVarNumRecsWritten (id, varNum, numRecs)
id as long ' in
varNum as integer ' in
numRecs as integer ' out

integer CDFgetzVarPadValue (id, varNum, padValue)
id as long ' in
varNum as integer ' in
padValue as TYPE ' out

integer CDFgetzVarRecordData (id, varNum, recNum, data)
id as long ' in
varNum as integer ' in
recNum as integer ' in
data as TYPE ' out

integer CDFgetzVarRecVariance (id, varNum, recVary)
id as long ' in
varNum as integer ' in
recVary as integer ' out

integer CDFgetzVarReservePercent (id, varNum, percent)
id as long ' in
varNum as integer ' in
percent as integer

integer CDFgetzVarSeqData (id, varNum, value)
   id as long   ' in
   varNum as integer   ' in
   value as TYPE   ' out

integer CDFgetzVarSeqPos (id, varNum, recNum, indices)
   id as long   ' in
   varNum as integer   ' in
   recNum as integer   ' out
   indices as integer()   ' out

integer CDFgetzVarsMaxWrittenRec (id, recNum)
   id as long   ' in
   recNum as integer   ' out

integer CDFgetzVarSparseRecords (id, varNum, sRecords)
   id as long   ' in
   varNum as integer   ' in
   sRecords as integer   ' out

integer CDFhyperGetrVarData (id, varNum, recNum, recCount, recInterval, indices, counts, intervals, buffer)
   id as long   ' in
   varNum as integer   ' in
   recNum as integer   ' in
   recCount as integer   ' in
   recInterval as integer   ' in
   indices as integer()   ' in
   counts as integer()   ' in
   intervals as integer()   ' in
   buffer as TYPE   ' out

integer CDFhyperGetzVarData (id, varNum, recNum, recCount, recInterval, indices, counts, intervals, buffer)
   id as long   ' in
   varNum as integer   ' in
   recNum as integer   ' in
   recCount as integer   ' in
   recInterval as integer   ' in
   indices as integer()   ' in
   counts as integer()   ' in
   intervals as integer()   ' in
   buffer as TYPE   ' out

integer CDFhyperPutrVarData (id, varNum, recNum, recCount, recInterval, indices, counts, intervals, buffer)
   id as long   ' in
   varNum as integer   ' in
   recNum as integer   ' in
   recCount as integer   ' in
   recInterval as integer   ' in
   indices as integer()   ' in
   counts as integer()   ' in
   intervals as integer()   ' in
   buffer as TYPE   ' in

integer CDFhyperPutzVarData (id, varNum, recNum, recCount, recInterval, indices, counts, intervals, data)

285
id as long
varNum as integer
recNum as integer
recCount as integer
recInterval as integer
indices as integer()
counts as integer()
intervals as integer()
data as TYPE

integer CDFinquire (id, numDims, dimSizes, encoding, majority, maxRec, numVars, numAttrs)
id as long
numDims as integer
dimSizes as integer()
encoding as integer
majority as integer
maxRec as integer
numVars as integer
numAttrs as integer

integer CDFinquireAttr (id, attrNum, attrName, attrScope, maxgEntry, maxrEntry, maxzEntry)
id as long
attrNum as integer
attrName as string
attrScope as integer
maxgEntry as integer
maxrEntry as integer
maxzEntry as integer

integer CDFinquireAttrgEntry (id, attrNum, entryNum, dataType, numElems)
id as long
attrNum as integer
entryNum as integer
dataType as integer
numElems as integer

integer CDFinquireAttrrEntry (id, attrNum, entryNum, dataType, numElems)
id as long
attrNum as integer
entryNum as integer
dataType as integer
numElems as integer

integer CDFinquireAttrzEntry (id, attrNum, entryNum, dataType, numElems)
id as long
attrNum as integer
entryNum as integer
dataType as integer
numElems as integer

integer CDFinquireCDF (id, numDims, dimSizes, encoding, majority, maxRec, numrVars, maxzRec, numzVars, numAttrs)
id as long
numDims as integer
dimSizes as integer()
encoding as integer
majority as integer 'out
maxrRec as integer 'out
numrVars as integer 'out
maxzRec as integer 'out
numzVars as integer 'out
numAttrs as integer 'out

integer CDFinquirerVar (id, varNum, varName, dataType, numElems, numDims, dimSizes, recVary, dimVarys)
id as long 'in
varNum as integer 'in
varName as string 'out
dataType as integer 'out
numElems as integer 'out
numDims as integer 'out
dimSizes as integer() 'out
recVary as integer 'out
dimVarys as integer() 'out

integer CDFinquirezVar (id, varNum, varName, dataType, numElems, numDims, dimSizes, recVary, dimVarys)
id as long 'in
varNum as integer 'in
varName as string 'out
dataType as integer 'out
numElems as integer 'out
numDims as integer 'out
dimSizes as integer() 'out
recVary as integer 'out
dimVarys as integer() 'out

integer CDFopen (CDFname, id)
CDFname as string 'in
id as long 'out

integer CDFopenCDF (CDFname, id)
CDFname as string 'in
id as long 'out

integer CDFselectCDF (id)
id as long 'in

integer CDFputAttrgEntry (id, attrNum, entryNum, dataType, numElems, value)
id as long 'in
attrNum as integer 'in
entryNum as integer 'in
dataType as integer 'in
numElems as integer 'in
value as TYPE 'in

integer CDFputAttrrEntry (id, attrNum, entryNum, dataType, numElems, value)
id as long 'in
attrNum as integer 'in
entryNum as integer 'in
dataType as integer 'in
numElems as integer 'in
value as TYPE 'in
integer CDFputAttrEntry (id, attrNum, entryNum, dataType, numElems, value)
  id as long
  attrNum as integer
  entryNum as integer
  dataType as integer
  numElems as integer
  value as TYPE

integer CDFputrVarData (id, varNum, recNum, indices, value)
  id as long
  varNum as integer
  recNum as integer
  indices as integer()
  value as TYPE

integer CDFputrVarPadValue (id, varNum, padValue)
  id as long
  varNum as integer
  padValue as TYPE

integer CDFputrVarRecordData (id, varNum, recNum, values)
  id as long
  varNum as integer
  recNum as integer
  values as TYPE

integer CDFputrVarSeqData (id, varNum, value)
  id as long
  varNum as integer
  value as TYPE

integer CDFputzVarData (id, varNum, recNum, indices, value)
  id as long
  varNum as integer
  recNum as integer
  indices as integer()
  value as TYPE

integer CDFputzVarPadValue (id, varNum, padValue)
  id as long
  varNum as integer
  padValue as TYPE

integer CDFputzVarRecordData (id, varNum, recNum, values)
  id as long
  varNum as integer
  recNum as integer
  values as TYPE

integer CDFputzVarSeqData (id, varNum, value)
  id as long
  varNum as integer
  value as TYPE

integer CDFrenameAttr (id, attrNum, attrName)
  id as long
attrNum as integer ' in
attrName as string ' in

integer CDFrenamerVar (id, varNum, varName)
id as long ' in
varNum as integer ' in
varName as string ' in

integer CDFrenamezVar (id, varNum, varName)
id as long ' in
varNum as integer ' in
varName as string ' in

integer CDFselect (id)
id as long ' in

integer CDFselectCDF (id)
id as long ' in

integer CDFsetAttrgEntryDataSpec (id, attrNum, entryNum, dataType)
id as long ' in
attrNum as integer ' in
entryNum as integer ' in
dataType as integer ' in

integer CDFsetAttrrEntryDataSpec (id, attrNum, entryNum, dataType)
id as long ' in
attrNum as integer ' in
entryNum as integer ' in
dataType as integer ' in

integer CDFsetAttrScope (id, attrNum, scope)
id as long ' in
attrNum as integer ' in
scope as integer ' in

integer CDFsetAttrzEntryDataSpec (id, attrNum, entryNum, dataType)
id as long ' in
attrNum as integer ' in
entryNum as integer ' in
dataType as integer ' in

integer CDFsetCacheSize (id, numBuffers)
id as long ' in
numBuffers as integer ' in

integer CDFsetChecksum (id, checksum)
id as long ' in
checksum as integer ' in

integer CDFsetCompression (id, compressionType, compressionParms)
id as long ' in
compressionType as integer ' in
compressionParms as integer() ' in

integer CDFsetCompressionCacheSize (id, numBuffers)
id as long in
numBuffers as integer in

integer CDFsetDecoding (id, decoding) in
id as long in
decoding as integer in

integer CDFsetEncoding (id, encoding) in
id as long in
encoding as integer in

void CDFsetFileBackward (mode) in
mode as integer in

integer CDFsetFormat (id, format) in
id as long in
format as integer in

integer CDFsetLeapSecondLastUpdated (id, lastUpdated) in
id as long in
lastUpdated as integer in

integer CDFsetMajority (id, majority) in
id as long in
majority as integer in

integer CDFsetNegtoPosfp0Mode (id, negtoPosfp0) in
id as long in
negtoPosfp0 as integer in

integer CDFsetReadOnlyMode (id, readOnly) in
id as long in
readOnly as integer in

integer CDFsetrVarAllocBlockRecords (id, varNum, firstRec, lastRec) in
id as long in
varNum as integer in
firstRec as integer in
lastRec as integer in

integer CDFsetrVarAllocRecords (id, varNum, numRecs) in
id as long in
varNum as integer in
numRecs as integer in

integer CDFsetrVarBlockingFactor (id, varNum, bf) in
id as long in
varNum as integer in
bf as integer in

integer CDFsetrVarCacheSize (id, varNum, numBuffers) in
id as long in
varNum as integer in
numBuffers as integer in

integer CDFsetrVarCompression (id, varNum, compressionType, compressionParms) in
id as long
varNum as integer
compressionType as integer
compressionParms as integer()

integer CDFsetrVarDataSpec (id, varNum, dataType)
id as long
varNum as integer
dataType as integer

integer CDFsetrVarDimVariances (id, varNum, dimVarys)
id as long
varNum as integer
dimVarys as integer()

integer CDFsetrVarInitialRecs (id, varNum, initialRecs)
id as long
varNum as integer
initialRecs as integer

integer CDFsetrVarRecVariance (id, varNum, recVary)
id as long
varNum as integer
recVary as integer

integer CDFsetrVarReservePercent (id, varNum, reservePercent)
id as long
varNum as integer
reservePercent as integer

integer CDFsetrVarsCacheSize (id, numBuffers)
id as long
numBuffers as integer

integer CDFsetrVarSeqPos (id, varNum, recNum, indices)
id as long
varNum as integer
recNum as integer
indices as integer()

integer CDFsetrVarSparseRecords (id, varNum, sRecords)
id as long
varNum as integer
sRecords as integer

integer CDFsetStageCacheSize (id, numBuffers)
id as long
numBuffers as integer

void CDFsetValidate (mode)
mode as integer

integer CDFsetzMode (id, zMode)
id as long
zMode as integer
integer CDFsetzVarAllocBlockRecords (id, varNum, firstRec, lastRec)
id as long
varNum as integer
firstRec as integer
lastRec as integer

integer CDFsetzVarAllocRecords (id, varNum, numRecs)
id as long
varNum as integer
numRecs as integer

integer CDFsetzVarBlockingFactor (id, varNum, bf)
id as long
varNum as integer
bf as integer

integer CDFsetzVarCacheSize (id, varNum, numBuffers)
id as long
varNum as integer
numBuffers as integer

integer CDFsetzVarCompression (id, varNum, compressionType, compressionParms)
id as long
varNum as integer
compressionType as integer
compressionParms as integer()

integer CDFsetzVarDataSpec (id, varNum, dataType)
id as long
varNum as integer
dataType as integer

integer CDFsetzVarDimVariances (id, varNum, dimVarys)
id as long
varNum as integer
dimVarys as integer()

integer CDFsetzVarInitialRecs (id, varNum, initialRecs)
id as long
varNum as integer
initialRecs as integer

integer CDFsetzVarRecVariance (id, varNum, recVary)
id as long
varNum as integer
recVary as integer

integer CDFsetzVarReservePercent (id, varNum, reservePercent)
id as long
varNum as integer
reservePercent as integer

integer CDFsetzVarsCacheSize (id, numBuffers)
id as long
numBuffers as integer
integer CDFsetzVarSeqPos (id, varNum, recNum, indices)
   id as long  ' in
   varNum as integer  ' in
   recNum as integer  ' in
   indices as integer()  ' in

integer CDFsetzVarSparseRecords (id, varNum, sRecords)
   id as long  ' in
   varNum as integer  ' in
   sRecords as integer  ' in

integer CDFvarClose (id, varNum)
   id as long  ' in
   varNum as integer  ' in

integer CDFvarCreate (id, varName, dataType, numElements, recVariance, dimVariances, varNum)
   id as long  ' in
   varName as string  ' in
   dataType as integer  ' in
   numElements as integer  ' in
   recVariance as integer  ' in
   dimVariances as integer()  ' in
   varNum as integer  ' out

integer CDFvarGet (id, varNum, recNum, indices, value)
   id as long  ' in
   varNum as integer  ' in
   recNum as integer  ' in
   indices as integer()  ' in
   value as TYPE  ' out

integer CDFvarHyperGet (id, varNum, recStart, recCount, recInterval, indices, counts, intervals, buffer)
   id as long  ' in
   varNum as integer  ' in
   recStart as integer  ' in
   recCount as integer  ' in
   recInterval as integer  ' in
   indices as integer()  ' in
   counts as integer()  ' in
   intervals as integer()  ' in
   buffer as TYPE  ' out

integer CDFvarHyperPut (id, varNum, recStart, recCount, recInterval, indices, counts, intervals, buffer)
   id as long  ' in
   varNum as integer  ' in
   recStart as integer  ' in
   recCount as integer  ' in
   recInterval as integer  ' in
   indices as integer()  ' in
   counts as integer()  ' in
   intervals as integer()  ' in
   buffer as TYPE  ' in

integer CDFvarInquire (id, varName, dataType, numElements, recVariance, dimVariances)
   id as long  ' in
   varNum as integer  ' in
varName as string  
dataType as integer  
numElements as integer  
recVariance as integer  
dimVariances as integer()  

integer CDFvarNum (id, varName)  
id as long  
varName as string  

integer CDFvarPut (id, varNum, recNum, indices, value)  
id as long  
varNum as integer  
recNum as integer  
indices as integer()  
value as TYPE  

integer CDFvarRename (id, varNum, varName)  
id as long  
varNum as integer  
varName as string
B.2 EPOCH Utility Methods

double computeEPOCH (year, month, day, hour, minute, second, msec)  
  year as integer  
  month as integer  
  day as integer  
  hour as integer  
  minute as integer  
  second as integer  
  msec as integer  

void EPOCHbreakdown (epoch, year, month, day, hour, minute, second, msec)  
  epoch as double  
  year as integer  
  month as integer  
  day as integer  
  hour as integer  
  minute as integer  
  second as integer  
  msec as integer  

string toEncodeEPOCH (epoch)  
  epoch as double  

string toEncodeEPOCH (epoch, style)  
  epoch as double  
  style as integer  

string() toEncodeEPOCH (epoch)  
  epoch as double()  

string() toEncodeEPOCH (epoch, style)  
  epoch as double()  
  style as integer  

void encodeEPOCH (epoch, epString)  
  epoch as double  
  epString as string  

void encodeEPOCH1 (epoch, epString)  
  epoch as double  
  epString as string  

void encodeEPOCH2 (epoch, epString)  
  epoch as double  
  epString as string  

void encodeEPOCH3 (epoch, epString)  
  epoch as double  
  epString as string  

void encodeEPOCH4 (epoch, epString)  
  epoch as double  

epString as string

void encodeEPOCHx (epoch, format, epString)
epoch as double
format as string
epString as string

double toParseEPOCH (epString)
epString as string

double() toParseEPOCH (epString)
epString as string()

double parseEPOCH (epString)
epString as string

double parseEPOCH1 (epString)
epString as string

double parseEPOCH2 (epString)
epString as string

double parseEPOCH3 (epString)
epString as string

double parseEPOCH4 (epString)
epString as string

double computeEPOCH16 (year, month, day, hour, minute, second, msec, microsec, nanosec, picosec)
year as integer
month as integer
day as integer
hour as integer
minute as integer
second as integer
msec as integer
microsec as integer
nanosec as integer
picosec as integer
epoch as double()

double EPOCH16breakdown (epoch, year, month, day, hour, minute, second, msec, microsec, nanosec, picosec)
epoch as double()
year as integer
month as integer
day as integer
hour as integer
minute as integer
second as integer
msec as integer
microsec as integer
nanosec as integer
picosec as integer
epoch as double()
string toEncodeEPOCH16 (epoch, style)
epoch as double()
style as integer

void encodeEPOCH16 (epoch, epString)
epoch as double()
epString as string

void encodeEPOCH16_1 (epoch, epString)
epoch as double()
epString as string

void encodeEPOCH16_2 (epoch, epString)
epoch as double()
epString as string

void encodeEPOCH16_3 (epoch, epString)
epoch as double()
epString as string

void encodeEPOCH16_4 (epoch, epString)
epoch as double()
epString as string

void encodeEPOCH16_x (epoch, format, epString)
epoch as double()
format as string
epString as string

double() toParseEPOCH16 (epString)
epString as string

double parseEPOCH16 (epString, epoch)
epString as string
epoch as double()

double parseEPOCH16_1 (epString)
epString as string
epoch as double()

double parseEPOCH16_2 (epString)
epString as string
epoch as double()

double parseEPOCH16_3 (epString)
epString as string
epoch as double()

double parseEPOCH16_4 (epString)
epString as string
epoch as double()

long computeTT2000 (year, month, day)
year as double
long computeTT2000 (year, month, day, hour)
  year as double  ' in
  month as double ' in
  day as double   ' in
  hour as double  ' in

long computeTT2000 (year, month, day, hour, minute)
  year as double  ' in
  month as double ' in
  day as double   ' in
  hour as double  ' in
  minute as double ' in

long computeTT2000 (year, month, day, hour, minute, second)
  year as double  ' in
  month as double ' in
  day as double   ' in
  hour as double  ' in
  minute as double ' in
  second as double ' in

long computeTT2000 (year, month, day, hour, minute, second, msec)
  year as double  ' in
  month as double ' in
  day as double   ' in
  hour as double  ' in
  minute as double ' in
  second as double ' in
  msec as double  ' in

long computeTT2000 (year, month, day, hour, minute, second, msec, usec)
  year as double  ' in
  month as double ' in
  day as double   ' in
  hour as double  ' in
  minute as double ' in
  second as double ' in
  msec as double  ' in
  usec as double  ' in

long computeTT2000 (year, month, day, hour, minute, second, msec, usec, nsec)
  year as double  ' in
  month as double ' in
  day as double   ' in
  hour as double  ' in
  minute as double ' in
  second as double ' in
  msec as double  ' in
  usec as double  ' in
  nsec as double  ' in

void TT2000breakdown (epoch, year, month, day, hour, minute, second, msec, usec, nsec)
  epoch as long  ' in
year as double  
month as double  
day as double  
hour as double  
minute as double  
second as double  
msec as double  
usec as double  
nsec as double  

string toEncodeTT2000 (epoch)  
epoch as long

string toEncodeTT2000 (epoch, style)  
epoch as long
style as integer

string() toEncodeTT2000 (epoch)  
epoch as long()

string() toEncodeTT2000 (epoch, style)  
epoch as long()
style as integer

void encodeTT2000 (epoch, epString, style)  
epoch as long
epString as string
style as string

long toParseTT2000 (epString)  
epString as string

long() toParseTT2000 (epString)  
epString as string()

long parseTT2000 (epString)  
epString as string

void CDFgetLastDateinLeapSecondsTable (year, month, day)  
year as integer
month as integer
day as integer

double EPOCHtoUnixTime (epoch)  
epoch as double

double() EPOCHtoUnixTime (epoch)  
epoch as double()

double UnixTimetoEPOCH (unixTime)  
unixTime as double

double() UnixTimetoEPOCH (unixTime)  
unixTime as double()

double EPOCH16toUnixTime (epoch)  

301
epoch as double()  ' in

double()  UnixTimetoEPOCH16 (unixTime)
unixTime as double  ' in

double()  TT2000toUnixTime (epoch)
epoch as long  ' in

double()  TT2000toUnixTime (epoch)
epoch as long()  ' in

long  UnixTimetoTT2000 (unixTime)
unixTime as double  ' in

long()  UnixTimetoTT2000 (unixTime)
unixTime as double()  ' in
### B.3 CDF Utility Methods

- boolean CDFFileExists (fileName)
  - filename as string

- integer CDFgetChecksumValue(checksum)
  - fileName as string

- integer CDFgetCompressionTypeValue(compressionType)
  - compressionType as string

- integer CDFgetDataTypeValue(dataType)
  - dataType as string

- integer CDFgetDecodingValue(decoding)
  - decoding as string

- integer CDFgetEncodingValue(encoding)
  - encoding as string

- integer CDFgetFormatValue(format)
  - format as string

- integer CDFgetMajorityValue(majority)
  - majority as string

- integer CDFgetSparseRecordValue(sparseRecord)
  - sparseRecord as string

- string CDFgetStringChecksum(checksum)
  - checksum as integer

- string CDFgetStringCompressionType(compressionType)
  - compressionType as integer

- string CDFgetStringDataType(dataType)
  - dataType as integer

- string CDFgetStringDecoding(decoding)
  - decoding as integer

- string CDFgetStringEncoding(encoding)
  - encoding as integer

- string CDFgetStringFormat(format)
  - format as integer

- string CDFgetStringMajority(majority)
  - majority as integer

- string CDFgetStringSparseRecord(sparseRecord)
  - sparseRecord as integer
B.4  CDF Exception Methods

integer CDFgetCurrentStatus ()

string  CDFgetStatusMsg(status)
status as integer
## Index

<table>
<thead>
<tr>
<th>Code</th>
<th>Page</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALPHAOSF1_DECODING</td>
<td>9</td>
<td>zVariable entry deleting</td>
<td>200</td>
</tr>
<tr>
<td>ALPHAOSF1_ENCODING</td>
<td>8</td>
<td>Attributes entries global entry</td>
<td>202</td>
</tr>
<tr>
<td>ALPHAVMSd_DECODING</td>
<td>9</td>
<td>data type inquiring</td>
<td></td>
</tr>
<tr>
<td>ALPHAVMSd_ENCODING</td>
<td>8</td>
<td>Attributes entries global entry</td>
<td></td>
</tr>
<tr>
<td>ALPHAVMSg_DECODING</td>
<td>9</td>
<td>number of elements</td>
<td></td>
</tr>
<tr>
<td>ALPHAVMSg_ENCODING</td>
<td>8</td>
<td>inquiring</td>
<td></td>
</tr>
<tr>
<td>ALPHAVMSi_DECODING</td>
<td>9</td>
<td>Attributes entries global entry</td>
<td></td>
</tr>
<tr>
<td>ALPHAVMSi_ENCODING</td>
<td>8</td>
<td>last entry number inquiring</td>
<td></td>
</tr>
<tr>
<td>ARM_BIG_DECODING</td>
<td>10</td>
<td>Attributes entries zVariable entry</td>
<td>204</td>
</tr>
<tr>
<td>ARM_BIG_ENCODING</td>
<td>8</td>
<td>last entry number inquiring</td>
<td></td>
</tr>
<tr>
<td>ARM_LITTLE_DECODING</td>
<td>10</td>
<td>Attributes entries zVariable entry</td>
<td>205</td>
</tr>
<tr>
<td>ARM_LITTLE_ENCODING</td>
<td>8</td>
<td>inquiring</td>
<td></td>
</tr>
<tr>
<td>attribute</td>
<td>188</td>
<td>Attributes entries rVariable entry</td>
<td>206</td>
</tr>
<tr>
<td>inquiring</td>
<td></td>
<td>last entry number inquiring</td>
<td></td>
</tr>
<tr>
<td>number</td>
<td>190</td>
<td>Attributes entries rVariable entry</td>
<td></td>
</tr>
<tr>
<td>renaming</td>
<td>192</td>
<td>inquiring</td>
<td></td>
</tr>
<tr>
<td>attributes</td>
<td>193</td>
<td>Attributes entries rVariable entry</td>
<td></td>
</tr>
<tr>
<td>checking existence</td>
<td></td>
<td>inquiring</td>
<td></td>
</tr>
<tr>
<td>creation</td>
<td>184, 196</td>
<td>Attributes entries rVariable entry</td>
<td></td>
</tr>
<tr>
<td>entries</td>
<td></td>
<td>last entry number inquiring</td>
<td></td>
</tr>
<tr>
<td>global entry</td>
<td>194</td>
<td>Attributes entries rVariable entry</td>
<td></td>
</tr>
<tr>
<td>checking existence</td>
<td></td>
<td>inquiring</td>
<td></td>
</tr>
<tr>
<td>inquiring</td>
<td>185</td>
<td>Attributes entries rVariable entry</td>
<td></td>
</tr>
<tr>
<td>reading</td>
<td>187</td>
<td>inquiring</td>
<td></td>
</tr>
<tr>
<td>writing</td>
<td>190</td>
<td>Attributes entries rVariable entry</td>
<td></td>
</tr>
<tr>
<td>naming</td>
<td>14, 185, 197</td>
<td>Attributes entries rVariable entry</td>
<td></td>
</tr>
<tr>
<td>inquiring</td>
<td>189</td>
<td>Attributes entries rVariable entry</td>
<td></td>
</tr>
<tr>
<td>number of</td>
<td></td>
<td>name inquiring</td>
<td>207</td>
</tr>
<tr>
<td>inquiring</td>
<td>52</td>
<td>Attributes number inquiring</td>
<td>208</td>
</tr>
<tr>
<td>scopes</td>
<td></td>
<td>Attributes entries rVariable entry</td>
<td></td>
</tr>
<tr>
<td>constants</td>
<td>12</td>
<td>inquiring</td>
<td></td>
</tr>
<tr>
<td>GLOBAL_SCOPE</td>
<td>12</td>
<td>Attributes entries rVariable entry</td>
<td></td>
</tr>
<tr>
<td>VARIABLE_SCOPE</td>
<td>13</td>
<td>inquiring</td>
<td></td>
</tr>
<tr>
<td>inquiring</td>
<td>188, 221</td>
<td>Attributes entries rVariable entry</td>
<td></td>
</tr>
<tr>
<td>Attributes entries</td>
<td></td>
<td>data type inquiring</td>
<td>210</td>
</tr>
<tr>
<td>global entry</td>
<td>198</td>
<td>Attributes entries global entry</td>
<td></td>
</tr>
<tr>
<td>deleting</td>
<td>201</td>
<td>data type inquiring</td>
<td></td>
</tr>
<tr>
<td>Attributes deleting</td>
<td>197</td>
<td>inquiring</td>
<td></td>
</tr>
<tr>
<td>entries</td>
<td></td>
<td>Attributes entries global entry</td>
<td></td>
</tr>
<tr>
<td>rVariable entry</td>
<td>194</td>
<td>number of elements inquiring</td>
<td></td>
</tr>
<tr>
<td>checking existence</td>
<td></td>
<td>Attributes scope inquiring</td>
<td>211</td>
</tr>
<tr>
<td>zVariable entry</td>
<td>195</td>
<td>Attributes entries global entry</td>
<td></td>
</tr>
<tr>
<td>checking existence</td>
<td></td>
<td>inquiring</td>
<td></td>
</tr>
<tr>
<td>Attributes entries</td>
<td></td>
<td>Attributes entries zVariable entry</td>
<td>212</td>
</tr>
<tr>
<td>rVariable entry</td>
<td></td>
<td>inquiring</td>
<td></td>
</tr>
<tr>
<td>deleting</td>
<td>199</td>
<td>Attributes entries zVariable entry</td>
<td></td>
</tr>
<tr>
<td>Attributes entries</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
selecting read-only constants
READONLYoff 13 READONLYon 13 selecting zMode constants zMODEoff 13 zMODEon1 13 zMODEon2 13 selecting CDF setNegtoPosfp0Mode 67 CDF_ATTR_NAME_LEN256 14 CDF_BYTE 6 CDF_CHAR 6 CDF_COPYRIGHT_LEN 14 CDF_DOUBLE 7 CDF_EPOCH 7 CDF_EPOCH16 7 CDF_FLOAT 7 CDF_INT1 6 CDF_INT2 6 CDF_INT4 6 CDF_INT8 7 CDF_MAX_DIMS 13 CDF_MAX_PARMS 14 CDF_OK 6 CDF_PATHNAME_LEN 14 CDF_REAL4 7 CDF_REAL8 7 CDF_STATUSTEXT_LEN 14 CDF_TIME_TT2000 7 CDF_UCHAR 6 CDF_UINT1 6 CDF_UINT2 6 CDF_UINT4 7 CDF_VAR_NAME_LEN256 14 CDF_WARN 6 CDFatCreate 184 CDFattrEntryInquire 185 CDFattrGet 187 CDFattrInquire 188 CDFattrNum 190 CDFattrPut 190 CDFattrRename 192 CDFatClose 29 CDFatCloseCDF 30 CDFatCloseVar 71 CDFatClosezVar 72 CDFatConfirmAttrExistence 193 CDFatConfirmEntryExistence 194 CDFatConfirmVarExistence 73 CDFatConfirmVarPadValueExistence 73 CDFatConfirmzEntryExistence 195 CDFatConfirmzVarExistence 74 CDFatConfirmzVarPadValueExistence 75 CDFatCreate 31 CDFatCreateAttr 196 CDFatCreateCDF 32 CDFatCreateVar 76 CDFatDelete 33 CDFatDeleteAttr 197 CDFatDeleteAttrgEntry 198 CDFatDeleteAttrrEntry 199 CDFatDeleteAttrzEntry 200 CDFatDeleteCDF 34 CDFatDeleteVar 79 CDFatDeleteVarRecords 80, 81 CDFatDeletezVar 82 CDFatDeletezVarRecords 83, 84 CDFattrCreate 184 CDFattrEntryInquire 185 CDFattrGet 187 CDFattrInquire 188 CDFattrNum 208 CDFattrPut 190 CDFattrRename 207 CDFattrScope 212 CDFattrMaxEntry 205 CDFattrMaxzEntry 206 CDFattrProfileName 207 CDFattrProfileNum 208 CDFattrProfileEntry 209 CDFattrProfileEntryDataType 210 CDFattrProfileMaxEntry 212 CDFattrProfileMaxzEntry 212 CDFattrProfileName 215 CDFattrProfileCacheSize 216 CDFattrProfileChecksumValue 255 CDFattrProfileChecksum 37 CDFattrProfileCompression 38 CDFattrProfileCompressionCacheSize 39 CDFattrProfileCompressionInfo 40 CDFattrProfileCompressioTypeValue 255 CDFattrProfileCopyright 41 CDFattrProfileCurrentStatus 261 CDFattrProfileDataType 265 CDFattrProfileDataTypeSize 26 CDFattrProfileDataTypeValue 256 CDFattrProfileDecoding 42 CDFattrProfileDecodingValue 256 CDFattrProfileEncoding 42 CDFattrProfileEncodingValue 257 CDFattrProfileFileBackward 43 CDFattrProfileFormat 44, 45 CDFattrProfileFormatValue 257 CDFattrProfileLastDateinLeapSecondsTable 252, 253 CDFattrProfileLibraryCopyright 27 CDFattrProfileLibraryVersion 27 CDFattrProfileMajority 46 CDFattrProfileMajorityValue 258 CDFattrProfileMaxWrittenRecNums 85 CDFattrProfileName 46 CDFattrProfileNumAttrgEntries 216 CDFattrProfileNumAttributes 217 CDFattrProfileNumAttrrEntries 217
ALPHAVMSi_ENCODING 8
ARM_BIG.Encoding 8
ARM_LITTLE.Encoding 8
DECSTATION.Encoding 8
HOST.Encoding 8
HP.Encoding 8
IA64VMSd.Encoding 9
IA64VMSg.Encoding 9
IA64VMSi.Encoding 8
IBMPC.Encoding 8
IBMRS.Encoding 8
MAC.Encoding 8
NETWORK.Encoding 8
NeXT.Encoding 8
SGI.Encoding 8
SUN.Encoding 8
VAX.Encoding 8
default 8
format
constants
MULTI_FILE 6
SINGLE_FILE 6
default 6
naming 14, 31, 32
overwriting 31, 32
version 35
inquiring
CDFs
inquiring
cache size
compression
inquiring 39
CDFs
decoding
inquiring
42
CDFs
decoding
inquiring
42
CDFs
file backard
inquiring
43
CDFs
format
inquiring
44
CDFs
format
inquiring
45
CDFs
majority
inquiring
46
CDFs
name
inquiring
46
CDFs
-0.0 to 0.0 mode
inquiring
47
CDFs
read-only mode
inquiring
48
CDFs
stage
cache size
inquiring
49
CDFs
validation
50
CDFs
inquiring
50
CDFs
version
50
CDFs
inquiring
51
CDFs
encoding
inquiring
52
CDFs
inquiring
53
CDFs
naming
55
CDFs
naming
56
CDFs
naming
56
CDFs
cache size
resetting
59
CDFs
checksum
resetting
60
CDFs
compression
resetting
60
CDFs
decoding
resetting
62
CDFs
encoding
resetting
63
CDFs
File Backward
resetting
64
CDFs
format
resetting
64
CDFs
format
resetting
65
CDFs
-0.0 to 0.0 Mode
resetting
67
CDFs
read-only mode
resetting
68
CDFs
stage
cache size
resetting
68
CDFs
validation
resetting
69
CDFs
zMode
resetting
70
CDFs
record numbers
maximum written
85
zVariables and rVariables
CDFs
rVariables  
  number of rVariables  
inquiring  
86
CDFs
zVariables  
  number of zVariables  
inquiring  
87
CDFs
global attributes  
  number of  
inquiring  
219
CDFs
variable attributes  
  number of  
inquiring  
220
CDFselect  
57
CDFselectCDF  
58
CDFsetAttrEntryDataSpec  
230
CDFsetAttrEntryDataSpec  
231
CDFsetAttrScope  
233
CDFsetAttrzEntryDataSpec  
233
CDFsetChecksum  
60
CDFsetCompression  
60
CDFsetCompressionCacheSize  
61
CDFsetDecoding  
62
CDFsetEncoding  
63
CDFsetFileBackward  
64
CDFsetFormat  
64, 65
CDFsetMajority  
66
CDFsetReadOnlyMode  
68
CDFsetVarAllocBlockRecords  
150
CDFsetVarAllocRecords  
151
CDFsetVarBlockingFactor  
152
CDFsetVarCacheSize  
153
CDFsetVarCompression  
154
CDFsetVarDataSpec  
155
CDFsetVarDimVariances  
156
CDFsetVarInitialRecs  
157
CDFsetVarRecVariance  
158
CDFsetVarReservePercent  
158
CDFsetVarSeqPos  
159
CDFsetVarSparseRecords  
160
CDFsetVarStageCacheSize  
161
CDFsetVarStringChecksum  
162
CDFsetVarStringCompressionType  
163
CDFsetVarStringDataSpec  
164
CDFsetVarStringDecoding  
165
CDFsetVarStringEncoding  
166
CDFsetVarStringFormat  
167
CDFsetVarStringMajority  
168
CDFsetVarStringSparseRecord  
169
CDFUtils  
255
CDFFileExists  
255
CDFgetChecksumValue  
255
CDFgetCompressionTypeValue  
255
CDFgetDataTypeValue  
256
CDFgetDecodingValue  
256
CDFgetEncodingValue  
257
CDFgetFormatValue  
257
CDFgetMajorityValue  
258
CDFgetSparseRecordValue  
258
CDFgetStringChecksum  
258
CDFgetStringCompressionType  
258
CDFgetStringDataType  
259
CDFgetStringDecoding  
259
CDFgetStringEncoding  
259
CDFgetStringFormat  
259
CDFgetStringMajority  
259
CDFgetStringSparseRecord  
260
Checking  
37, 60
Classes  
1
Closing  
71
rVar in a multi-file CDF  
72
zVar in a multi-file CDF  
10
COLUMN_MAJOR  
2
compiling  
11
types/parameters  
26
data types  
6
| CDF_BYTE     | 6   | parseEPOCH16_4 | 245, 246, 247 |
| CDF_CHAR     | 6   | parseEPOCH2   | 240           |
| CDF_DOUBLE   | 7   | parseEPOCH3   | 240           |
| CDF_EPOCH    | 7   | parseEPOCH4   | 241           |
| CDF_EPOCH16  | 7   | EPOCH16breakdown | 241          |
| CDF_FLOAT    | 7   | EPOCHbreakdown | 237           |
| CDF_INT1     | 6   | Equivalent data types | 22          |
| CDF_UINT8    | 7   | CDF           |               |
| CDF_REAL4    | 7   | -0.0 to 0.0 mode |               |
| CDF_REAL8    | 7   | attribute      |               |
| CDF_TIME_TT2000 | 7 | name           |               |
| CDF_UCHAR    | 6   | get            | 207           |
| CDF_UINT1    | 6   | scope          | 212           |
| CDF_UINT2    | 6   | get            |               |
| CDF_UINT4    | 7   | checksum       |               |
| CDF_UINT4    | 7   | set60          |               |
| DECSTATION_DECODING | 9 | compression    | 39            |
| DECSTATION_ENCODING | 8 | get            |               |
| dimensions   | 13  | compression cache size |         |
| encodeEPOCH  | 237, 242, 251 | set62         |               |
| encodeEPOCH1 | 238  | Copyright      |               |
| encodeEPOCH16 | 242 | get            | 41            |
| encodeEPOCH16_1 | 242 | decoding       |               |
| encodeEPOCH16_2 | 243 | get            | 42, 43        |
| encodeEPOCH16_3 | 243 | encoding       |               |
| encodeEPOCH16_4 | 243 | set63          |               |
| encodeEPOCH16_x | 243 | file backward  |               |
| encodeEPOCH2  | 238  | set64          |               |
| encodeEPOCH3  | 238  | global attribute|               |
| encodeEPOCH4  | 238  | entry          |               |
| encodeEPOCHx  | 239  | data type      | 202           |
| encodeTT2000  | 251  | get            | 201           |
| EPOCH        |      | get            |               |
| computing    | 236, 241 | entry        |               |
| decomposing  | 237, 241 | number of elements |         |
| encoding     | 237, 238, 239, 242, 243, 251 | get            | 203           |
| parsing      | 239, 240, 241, 244, 245, 246, 247 | number of entries |         |
| utility routines | 236 | get            | 216           |
| computeEPOCH  | 236  | inquiring      | 54            |
| computeEPOCH16 | 241  | number of attributes |         |
| encodeEPOCH   | 236  | get            | 217           |
| encodeEPOCH1  | 238  | read-only mode | set68         |
| encodeEPOCH16 | 238  | rVariable attribute |         |
| encodeEPOCH16_1 | 242 | entry          |               |
| encodeEPOCH16_2 | 243 | get            | 209           |
| encodeEPOCH16_3 | 243 | entry          |               |
| encodeEPOCH16_4 | 243 | data type      | 210           |
| encodeEPOCH16_x | 243 | get            |               |
| encodeEPOCH2  | 238  | stage cache size| set69         |
| encodeEPOCH3  | 238  | validate       |               |
| encodeEPOCH4  | 238  | set70          |               |
| encodeEPOCHx  | 239  | validation     |               |
| EPOCH16breakdown | 241 | set70          | 50            |
| EPOCHbreakdown | 237  | validation     |               |
| parseEPOCH    | 240  | get            | 51            |
| parseEPOCH1   | 240  | zMode          |               |
| parseEPOCH16  | 239, 244 | get            | 52            |
| parseEPOCH16_1 | 245  |                |               |
| parseEPOCH16_2 | 245  | get            |               |
| parseEPOCH16_3 | 245  | set70          |               |
CDF
  cache size
    get 37
  checksum
    get 38
  close
  create 33
  delete 34
CDF
  compression cache size
    get 40
CDF
  compression information
    get 40
CDF
  file backward
    get 44
CDF
  format
    get 44
CDF
  format
    get 45
CDF
  majority
    get 46
CDF
  name
    get 47
CDF
  -0.0 to 0.0 mode
    get 47
CDF
  read-only mode
    get 48
CDF
  cache buffer size
    get 49
CDF
  open 57
CDF
  select
  CDF
  select 58
CDF
  cache size
    set 59
CDF
  compression
    set 61
CDF
  decoding
    set 62
CDF
  format
    set 65
CDF
  format
    set 66
CDF
  majority
    set 66
CDF
  rVar
    close 71
CDF
  zVar
    close 72
CDF
  rVariable
    existence
    confirm 73
CDF
  rVariable
    pad value existence
    confirm 74
CDF
  zVariable
    existence
    confirm 75
CDF
  zVariable
    pad value existence
    confirm 76
CDF
  rVariable
    create 77
CDF
  zVariable
    create 79
CDF
  rVariable
    delete 80
CDF
  rVariable
    data records
    delete 81
CDF
  rVariable
    data records
    delete 82
CDF
  zVariable
    delete 83
CDF
  zVariable
    data records
    delete 84
CDF
  zVariable
    data records
    delete 85
CDF
  max record numbers
    zVariables and rVariables
    get 86
CDF
  number of rVariables
    get 86
CDF
  number of zVariables
    get 87
CDF
  rVariable
number of records allocated
get
88
CDF
rVariable
blocking factor
get
89
CDF
cache size
get
90
CDF
compression
get
91
CDF
variable data
get
92
CDF
data type
get
93
CDF
dimension variances
get
94
CDF
information
get
95
CDF
maximum number of records allocated
get
96
CDF
maximum record number
get
96
CDF
name
get
97
CDF
number of elements
get
98
CDF
number of records written
get
99
CDF
pad value
get
100
CDF
record data
get
101
CDF
record variance
get
102
CDF
rVariable
compression reserve percentage
get
103
CDF
dimension sizes
get
103
CDF
data value
get
104
CDF
read position
get
105
CDF
maximum record number
get
106
CDF
dimensionality
get
107
CDF
sparse record type
get
108
CDF
Variable number
get
109
CDF
number of records allocated
get
110
CDF
blocking factor
get
111
CDF
cache size
get
112
CDF
compression
get
113
CDF
variable data
get
114
CDF
data type
get
115
CDF
dimension sizes
get
116
CDF
dimension variances
CDF
  rVariable
  information
  get
CDF
  zVariable
  maximum number of records allocated
  get
CDF
  zVariable
  maximum record number
  get
CDF
  zVariable
  name
  get
CDF
  zVariable
  dimensionality
  get
CDF
  zVariable
  number of elements
  get
CDF
  zVariable
  number of records written
  get
CDF
  zVariable
  pad value
  get
CDF
  zVariable
  record data
  get
CDF
  zVariable
  record variance
  get
CDF
  zVariable
  compression reserve percentage
  get
CDF
  zVariable
  data value
  get
CDF
  zVariable
  read position
  get
CDF
  Variables
  maximum record number
  get
CDF
  zVariable
  sparse record type
  get
CDF
  rVariable
  multiple values or records
  get
CDF
  zVariable
  multiple values or records
  get
CDF
  rVariable
  data values
  write
CDF
  zVariable
  data values
  write
CDF
  rVariable
  inquire
CDF
  zVariable
  inquire
CDF
  rVariable
  data value
  write
CDF
  rVariable
  pad value
  set
CDF
  rVariable
  record data
  write
CDF
  rVariable
  data value
  write
CDF
  zVariable
  data value
  write
CDF
  zVariable
  pad value
  set
CDF
  zVariable
  record data
  write
CDF
  zVariable
  data value
  write
CDF
  zVariable
  data value
  serial write
CDF
  zVariable
  data value
  serial write
CDF
  rVariable
  rename
CDF
  zVariable
  rename
CDF
  rVariable
  data records
deleting CDF get CDF Copyright library version data type size rVariable data inquiring attribute entry attribute number CDF error code explanation text rVariable variable number interpreting status codes opening CDF reading attribute entry rVariable values hyper renaming attribute rVariable status handler writing attribute gEntry rEntry rVariable multiple records/values rVariable Exception handling Fixed statement getAttrEntryNumElements getAttrMaxgEntry GLOBAL_SCOPE HOST_DECODING HOST_ENCODING HP_DECODING HP_ENCODING IA64VMSd_DECODING IA64VMSd_ENCODING IA64VMSg_DECODING IA64VMSg_ENCODING IA64VMSi_DECODING IA64VMSi_ENCODING IBMPc_DECODING IBMPc_ENCODING IBMRs_DECODING IBMRs_ENCODING id 5 inquiring CDF information Interface Leap Seconds Library error text
resetting 158
rVariables 159
cache size 160
resetting 161
rVariables 162
sequential position 163
resetting 164
rVariables 165
sparse records type 166
resetting 167
rVariables 168
close 169
rVariables 170
creation 171
rVariables 172
reading 173
single value 174
rVariables 175
hyper read 176
multiple values or records 177
rVariables 178
hyper put 179
multiple values or records 180
rVariables 181
writing 182
single value 183
rVariables 184
renaming 185
sample programs 186
SGi_DECODING 187
SGi_ENCODING 188
SINGLE_FILE 189
sparse arrays 190
types 191
sparse records 192
types 193
status 194
status codes 195
constants 196
CDF_OK 197
CDF_WARN 198
error 199
explanation text 200
inquiring 201
max length 202
informational 203
interpreting 204
warning 205
SUN_DECODING 206
SUN_ENCODING 207
TT2000 208
computing 209
decomposing 210
encoding 211
info 212
parsing 213
utility routines 214
CDFgetLastDateinLeapSecondsTable 215
computeTT2000 216
encodeTT2000 217
parseTT2000 218
TT2000breakdown 219
VARIABLE_SCOPE 13
variables 11
compression 10
types/parameters 10
data specification 10
data type 10
inquiring 10
number of elements 10
dimensionality 10
inquiring 10
majority 10
considering 10
constants 10
COLUMN_MAJOR 10
ROW_MAJOR 10
maximum records 10
name 10
inquiring 180
naming 14
max length 14
records 12
sparse 12
sparse arrays 12
types 12
variable number 12
VAX_DECODING 9
VAX_ENCODING 8
VB-CDF Interface 19, 25
zMODEoff 13
zMODEon1 13
zMODEon2 13
zVariables 11
VARY 11
NOVARY 11
variables 108
VARY 11
VAX_DECODING 9
VAX_ENCODING 8
check existence 74
creation 78
deleting 82
checking existence 75
zVariables 109
record numbers 75
allocated records 110
zVariables 110
cache size 110
blocking factor 109
zVariables 109
inquiring

zVariables
  compression
    inquiring
  zVariables
    reading data
  zVariables
    data type
      inquiring
  zVariables
    dimension sizes
      inquiring
  zVariables
    dimension variances
      inquiring
  zVariables
    information
      inquiring
  zVariables
    record numbers
      maximum allocated record
        inquiring
  zVariables
    record numbers
      maximum written record
        inquiring
  zVariables
    name
      inquiring
  zVariables
    dimensionality
      inquiring
  zVariables
    number of elements
      inquiring
  zVariables
    record numbers
      written records
        inquiring
  zVariables
    pad value
      inquiring
  zVariables
    reading
      one record
  zVariables
    record variance
      inquiring
  zVariables
    compression
      reserve percentage
        inquiring
  zVariables
    sequential data
      reading one value
  zVariables
    sequential position
      inquiring
  zVariables
    record numbers
      written records
        maximum

rVariables and zVariables
  zVariables
    sparse records type
  zVariables
    inquiring
  zVariables
    reading
      multiple values or records
  zVariables
    writing
      multiple values or records
  zVariables
    inquiring
  zVariables
    writing
    single data
  zVariables
    pad value
      resetting
  zVariables
    writing
    record data
  zVariables
    writing
    sequential data
  zVariables
    renaming
  zVariables
    records
      allocation
      resetting
  zVariables
    records
      allocation
      resetting
  zVariables
    blocking factor
      resetting
  zVariables
    cache size
      resetting
  zVariables
    compression
      resetting
  zVariables
    data specification
      resetting
  zVariables
    dimension variances
      resetting
  zVariables
    records
      writing initially
      resetting
  zVariables
    record variance
      resetting
  zVariables
    compression
      reserve percentage
        resetting
  zVariables
    cache size
      resetting
  zVariables
    sequential position
resetting
zVariables

resetting

sparse records type
resetting