All Classes

Packages

gsfc.nssdc.cdf
gsfc.nssdc.cdf.util

Overview Package Class Tree Deprecated Index Help

Prev Next Frames No Frames

Packages

Package	Description
gsfc.nssdc.cdf	
gsfc.nssdc.cdf.util	

Overview Package Class Tree Deprecated Index Help

Prev Next Frames No Frames

All Classes

Attribute
CDF
CDFConstants
CDFData
CDFDelegate
CDFException
CDFNativeLibrary
CDFObject
CDFTools
CDFTT2000
CDFUtils
Entry
Epoch
Epoch16
EpochNative
Variable

All Classes

Attribute
CDF
CDFConstants
CDFData
CDFDelegate
CDFException
CDFNativeLibrary
CDFObject
CDFTools
CDFTT2000
CDFUtils
Entry
Epoch
Epoch16
EpochNative
Variable

gsfc.nssdc.cdf

Interfaces

CDFConstants
CDFDelegate
CDFObject

Classes

Attribute
CDF
CDFData
CDFNativeLibrary
CDFTools
Entry
Variable

Exceptions

CDFException

gsfc.nssdc.cdf.util

Classes

CDFTT2000
CDFUtils
Epoch
Epoch16
EpochNative

gsfc.nssdc.cdf

Class Attribute

java.lang.Object
gsfc.nssdc.cdf.Attribute

All Implemented Interfaces:

CDFConstants, CDFObject

```
public class Attribute
extends java.lang.Object
implements CDFConstants, CDFObject
```

This class contains the methods that are associated with either global or variable attributes.

Version:

1.0, 2.0 03/18/05 Selection of current CDF and attribute are done as part of operations passed to JNI. JNI call is synchronized so only one process is allowed in a JVM, due to multi-thread safety. The select method will never be called., 3.3 01/31/11 Reset the vector size to match the maximum entry in stead of its expanded capacity in getEntries method.

See Also:

CDF, CDFException, Entry, Variable

Field Summary

Fields inherited from interface gsfc.nssdc.cdf.CDFConstants

AHUFF_COMPRESSION, ALPHAOSF1_DECODING, ALPHAOSF1_ENCODING, ALPHAVMSd_DECODING, ALPHAVMSd_ENCODING, ALPHAVMSg_DECODING, ALPHAVMSg_ENCODING, ALPHAVMSi_DECODING, ALPHAVMSi_ENCODING, ARM_BIG_DECODING, ARM_BIG_ENCODING, ARM_LITTLE_DECODING, ARM_LITTLE_ENCODING, ATTR, ATTR_EXISTENCE, ATTR_EXISTS, ATTR_MAXgENTRY, ATTR_MAXrENTRY, ATTR_MAXzENTRY, ATTR_NAME, ATTR_NAME_TRUNC, ATTR_NUMBER, ATTR_NUMgENTRIES, ATTR_NUMrENTRIES, ATTR_NUMzENTRIES, ATTR_SCOPE, BACKWARD, BACKWARDFILEoff, BACKWARDFILEon, BAD_ALLOCATE_RECS, BAD_ARGUMENT, BAD_ATTR_NAME, BAD_ATTR_NUM, BAD_BLOCKING_FACTOR, BAD_CACHE_SIZE, BAD_CDF_EXTENSION, BAD_CDF_ID, BAD_CDF_NAME, BAD_CDFSTATUS, BAD_CHECKSUM, BAD_COMPRESSION_PARM, BAD_DATA_TYPE, BAD_DECODING, BAD_DIM_COUNT, BAD_DIM_INDEX, BAD_DIM_INTERVAL, BAD_DIM_SIZE, BAD_ENCODING, BAD_ENTRY_NUM, BAD_FNC_OR_ITEM, BAD_FORMAT, BAD_INITIAL_RECS, BAD_MAJORITY, BAD_MALLOC, BAD_NEGtoPOSfp0_MODE, BAD_NUM_DIMS, BAD_NUM_ELEMS, BAD_NUM_STRINGS, BAD_NUM_VARS, BAD_READONLY_MODE, BAD_REC_COUNT, BAD_REC_INTERVAL, BAD_REC_NUM, BAD_SCOPE, BAD_SCRATCH_DIR, BAD_SPARSEARRAYS_PARM, BAD_VAR_NAME, BAD_VAR_NUM, BAD_zMODE, BADDATE_LEAPSECOND_UPDATED, BeginUnixTimeEPOCH, BeginUnixTimeEPOCH16, BLOCKINGFACTOR_TOO_LARGE, BLOCKINGFACTOR_TOO_SMALL, BLOCKINGFACTOR_TOO_SMALL2, CANNOT_ALLOCATE_RECORDS, CANNOT_CHANGE, CANNOT_COMPRESS, CANNOT_CONVERT_WIDECHAR, CANNOT_COPY, CANNOT_INSERT_RECORDS, CANNOT_SPARSEARRAYS, CANNOT_SPARSERECORDS, CDF, CDF_ACCESS, CDF_ATTR_NAME_LEN, CDF_ATTR_NAME_LEN256, CDF_BYTE, CDF_CACHESIZE, CDF_CHAR, CDF_CHECKSUM, CDF_CLOSE_ERROR, CDF_COMPRESSION, CDF_COPYRIGHT, CDF_COPYRIGHT_LEN, CDF_CREATE_ERROR, CDF_DECODING, CDF_DELETE_ERROR, CDF_DOUBLE, CDF_ENCODING, CDF_EPOCH, CDF_EPOCH16, CDF_EXISTS, CDF_FLOAT, CDF_FORMAT, CDF_INCREMENT, CDF_INFO, CDF_INT1, CDF_INT2, CDF_INT4, CDF_INT8, CDF_INTERNAL_ERROR, CDF_LEAPSECONDLASTUPDATED, CDF_MAJORITY, CDF_MAX_DIMS, CDF_MAX_PARMS, CDF_MIN_DIMS, CDF_NAME, CDF_NAME_TRUNC, CDF_NEGtoPOSfp0_MODE, CDF_NUMATTRS, CDF_NUMgATTRS, CDF_NUMrVARS, CDF_NUMvATTRS, CDF_NUMzVARS, CDF_OK, CDF_OPEN_ERROR, CDF_PATHNAME_LEN, CDF_READ_ERROR, CDF_READONLY_MODE, CDF_REAL4, CDF_REAL8, CDF_RELEASE, CDF_SAVE_ERROR, CDF_SCRATCHDIR, CDF_STATUS, CDF_STATUSTEXT_LEN, CDF_TIME_TT2000, CDF_UCHAR, CDF_UINT1, CDF_UINT2, CDF_UINT4, CDF_VAR_NAME_LEN, CDF_VAR_NAME_LEN256, CDF_VERSION, CDF_WARN, CDF_WRITE_ERROR, CDF_zMODE, CDFwithSTATS, CHECKSUM, CHECKSUM_ERROR, CHECKSUM_NOT_ALLOWED, CLOSE, COLUMN_MAJOR, COMPRESS_CACHESIZE, COMPRESSION_ERROR, CONFIRM, CORRUPTED_V2_CDF, CORRUPTED_V3_CDF, CREATE, CURgENTRY_EXISTENCE, CURrENTRY_EXISTENCE, CURzENTRY_EXISTENCE, DATATYPE_MISMATCH, DATATYPE_SIZE, DECOMPRESSION_ERROR, DECSTATION_DECODING, DECSTATION_ENCODING, DEFAULT_BYTE_PADVALUE, DEFAULT_CHAR_PADVALUE, DEFAULT_DOUBLE_PADVALUE, DEFAULT_EPOCH_PADVALUE, DEFAULT_EPOCH16_PADVALUE, DEFAULT_FLOAT_PADVALUE, DEFAULT_INT1_PADVALUE, DEFAULT_INT2_PADVALUE, DEFAULT_INT4_PADVALUE, DEFAULT_INT8_PADVALUE, DEFAULT_REAL4_PADVALUE, DEFAULT_REAL8_PADVALUE, DEFAULT_TT2000_PADVALUE, DEFAULT_UCHAR_PADVALUE, DEFAULT_UINT1_PADVALUE, DEFAULT_UINT2_PADVALUE, DEFAULT_UINT4_PADVALUE, DELETE, DID_NOT_COMPRESS, EMPTY_COMPRESSED_CDF, END_OF_VAR, EPOCH_STRING_LEN, EPOCH_STRING_LEN_EXTEND, EPOCH1_STRING_LEN, EPOCH1_STRING_LEN_EXTEND, EPOCH2_STRING_LEN, EPOCH2_STRING_LEN_EXTEND, EPOCH3_STRING_LEN,

EPOCH3_STRING_LEN_EXTEND, EPOCH4_STRING_LEN, EPOCH4_STRING_LEN_EXTEND, EPOCHx_FORMAT_MAX, EPOCHx_STRING_MAX, FILLED_TT2000_VALUE, FORCED_PARAMETER, FUNCTION_NOT_SUPPORTED, gENTRY_, gENTRY_DATA_, gENTRY_DATASPEC_, gENTRY_DATATYPE_, gENTRY_EXISTENCE_, gENTRY_NUMELEMS_, GET_, GETCDFCHECKSUM_, GETCDFFILEBACKWARD_, GETCDFVALIDATE_, GETLEAPSECONDSENVVAR_, GLOBAL_SCOPE, GZIP_COMPRESSION, HOST_DECODING, HOST_ENCODING, HP_DECODING, HP_ENCODING, HUFF_COMPRESSION, IA64VMSd_DECODING, IA64VMSd_ENCODING, IA64VMSg_DECODING, IA64VMSg_ENCODING, IA64VMSi_DECODING, IA64VMSi_ENCODING, IBM_PC_OVERFLOW, IBMPC_DECODING, IBMPC_ENCODING, IBMRS_DECODING, IBMRS_ENCODING, ILLEGAL_EPOCH_FIELD, ILLEGAL_EPOCH_VALUE, ILLEGAL_FOR_SCOPE, ILLEGAL_IN_zMODE, ILLEGAL_ON_V1_CDF, ILLEGAL_TT2000_VALUE, IS_A_NETCDF, LIB_COPYRIGHT_, LIB_INCREMENT_, LIB_RELEASE_, LIB_subINCREMENT_, LIB_VERSION_, MAC_DECODING, MAC_ENCODING, MD5_CHECKSUM, MULTI_FILE, MULTI_FILE_FORMAT, NA_FOR_VARIABLE, NEGATIVE_FP_ZERO, NEGtoPOSfp0off, NEGtoPOSfp0on, NETWORK_DECODING, NETWORK_ENCODING, NeXT_DECODING, NeXT_ENCODING, NO_ATTR_SELECTED, NO_CDF_SELECTED, NO_CHECKSUM, NO_COMPRESSION, NO_DELETE_ACCESS, NO_ENTRY_SELECTED, NO_MORE_ACCESS, NO_PADVALUE_SPECIFIED, NO_SPARSEARRAYS, NO_SPARSERECORDS, NO_STATUS_SELECTED, NO_SUCH_ATTR, NO_SUCH_CDF, NO_SUCH_ENTRY, NO_SUCH_RECORD, NO_SUCH_VAR, NO_VAR_SELECTED, NO_VARS_IN_CDF, NO_WRITE_ACCESS, NONE_CHECKSUM, NOT_A_CDF, NOT_A_CDF_OR_NOT_SUPPORTED, NOVARY, NULL_, OPEN_, OPTIMAL_ENCODING_TREES, OTHER_CHECKSUM, PAD_SPARSERECORDS, PPC_DECODING, PPC_ENCODING, PRECEEDING_RECORDS_ALLOCATED, PREV_SPARSERECORDS, PUT_, READ_ONLY_DISTRIBUTION, READ_ONLY_MODE, READONLYoff, READONLYon, rENTRY_, rENTRY_DATA_, rENTRY_DATASPEC_, rENTRY_DATATYPE_, rENTRY_EXISTENCE_, rENTRY_NAME_, rENTRY_NUMELEMS_, rENTRY_NUMSTRINGS_, rENTRY_STRINGSDATA_, RLE_COMPRESSION, RLE_OF_ZEROs, ROW_MAJOR, rVAR_, rVAR_ALLOCATEBLOCK_, rVAR_ALLOCATEDFROM_, rVAR_ALLOCATEDTO_, rVAR_ALLOCATERECS_, rVAR_BLOCKINGFACTOR_, rVAR_CACHESIZE_, rVAR_COMPRESSION_, rVAR_DATA_, rVAR_DATASPEC_, rVAR_DATATYPE_, rVAR_DIMVARYS_, rVAR_EXISTENCE_, rVAR_HYPERDATA_, rVAR_INITIALRECS_, rVAR_MAXallocREC_, rVAR_MAXREC_, rVAR_NAME_, rVAR_nINDEXENTRIES_, rVAR_nINDEXLEVELS_, rVAR_nINDEXRECORDS_, rVAR_NUMallocRECS_, rVAR_NUMBER_, rVAR_NUMELEMS_, rVAR_NUMRECS_, rVAR_PADVALUE_, rVAR_RECORDS_, rVAR_RECORDS_RENUMBER_, rVAR_RECVARY_, rVAR_RESERVEPERCENT_, rVAR_SEQDATA_, rVAR_SEQPOS_, rVAR_SPARSEARRAYS_, rVAR_SPARSERECORDS_, rVARs_CACHESIZE_, rVARs_DIMCOUNTS_, rVARs_DIMINDICES_, rVARs_DIMINTERVALS_, rVARs_DIMSIZES_, rVARs_MAXREC_, rVARs_NUMDIMS_, rVARs_RECCOUNT_, rVARs_RECDATA_, rVARs_RECINTERVAL_, rVARs_RECNUMBER_, SAVE_, SCRATCH_CREATE_ERROR, SCRATCH_DELETE_ERROR, SCRATCH_READ_ERROR, SCRATCH_WRITE_ERROR, SELECT_, SGI_DECODING, SGi_ENCODING, SINGLE_FILE, SINGLE_FILE_FORMAT, SOME_ALREADY_ALLOCATED, STAGE_CACHESIZE_, STATUS_TEXT_, STRING_NOT_UTF8_ENCODING, STRINGDELIMITER, SUN_DECODING, SUN_ENCODING, TOO_MANY_PARMS, TOO_MANY_VARS, TRY_TO_READ_NONSTRING_DATA, TT2000_0_STRING_LEN, TT2000_1_STRING_LEN, TT2000_2_STRING_LEN, TT2000_3_STRING_LEN, TT2000_4_STRING_LEN, TT2000_CDF_MAYNEEDUPDATE, TT2000_TIME_ERROR, TT2000_USED_OUTDATED_TABLE, UNABLE_TO_PROCESS_CDF, UNKNOWN_COMPRESSION, UNKNOWN_SPARSENESS, UNSUPPORTED_OPERATION, VALIDATE_, VALIDATEFILEoff, VALIDATEFILEon, VAR_ALREADY_CLOSED, VAR_CLOSE_ERROR, VAR_CREATE_ERROR, VAR_DELETE_ERROR, VAR_EXISTS, VAR_NAME_TRUNC, VAR_OPEN_ERROR, VAR_READ_ERROR, VAR_SAVE_ERROR, VAR_WRITE_ERROR, VARIABLE_SCOPE, VARY, VAX_DECODING, VAX_ENCODING, VIRTUAL_RECORD_DATA, zENTRY_, zENTRY_DATA_, zENTRY_DATASPEC_, zENTRY_DATATYPE_, zENTRY_EXISTENCE_, zENTRY_NAME_, zENTRY_NUMELEMS_, zENTRY_NUMSTRINGS_, zENTRY_STRINGSDATA_, ZLIB_COMPRESS_ERROR, ZLIB_UNCOMPRESS_ERROR, zMODEoff, zMODEon1, zMODEon2, zVAR_, zVAR_ALLOCATEBLOCK_, zVAR_ALLOCATEDFROM_, zVAR_ALLOCATEDTO_, zVAR_ALLOCATERECS_, zVAR_BLOCKINGFACTOR_, zVAR_CACHESIZE_, zVAR_COMPRESSION_, zVAR_DATA_, zVAR_DATASPEC_, zVAR_DATATYPE_, zVAR_DIMCOUNTS_, zVAR_DIMINDICES_, zVAR_DIMINTERVALS_, zVAR_DIMSIZES_, zVAR_DIMVARYS_, zVAR_EXISTENCE_, zVAR_HYPERDATA_, zVAR_INITIALRECS_, zVAR_MAXallocREC_, zVAR_MAXREC_, zVAR_NAME_, zVAR_nINDEXENTRIES_, zVAR_nINDEXLEVELS_, zVAR_nINDEXRECORDS_, zVAR_NUMallocRECS_, zVAR_NUMBER_, zVAR_NUMDIMS_, zVAR_NUMELEMS_, zVAR_NUMRECS_, zVAR_PADVALUE_, zVAR_RECCOUNT_, zVAR_RECINTERVAL_, zVAR_RECNUMBER_, zVAR_RECORDS_, zVAR_RECORDS_RENUMBER_, zVAR_RECVARY_, zVAR_RESERVEPERCENT_, zVAR_SEQDATA_, zVAR_SEQPOS_, zVAR_SPARSEARRAYS_, zVAR_SPARSERECORDS_, zVARs_CACHESIZE_, zVARs_MAXREC_, zVARs_RECDATA_, zVARs_RECNUMBER_

Method Summary

Methods

Modifier and Type	Method and Description
static **Attribute**	**create**(**CDF** myCDF, java.lang.String name, long scope) Creates a new attribute in the given CDF.
void	**delete**() Deletes this attribute.
void	**deleteEntry**(long entryID) Deletes an attribute entry for the given entry number.
void	**deleteEntry**(**Variable** var) Deletes the attribute entry for the given variable.
java.util.Vector	**getEntries**() Gets all the entries defined for this attribute.
Entry	**getEntry**(long entryID) Gets the attribute entry for the given entry number.
Entry	**getEntry**(**Variable** var) Gets the attribute entry for the given variable.
long	**getEntryID**(**Entry** entry)

	Gets the entry id for the given entry.
long	**getID**() Gets the attribute ID of this attribute.
long	**getMaxEntryNumber**() Gets the largest Entry number for this attribute.
CDF	**getMyCDF**() Gets the CDF object to which this attribute belongs.
java.lang.String	**getName**() Gets the name of this attribute.
long	**getNumEntries**() Gets the number of entries in this attribute.
long	**getScope**() Gets the scope of this attribute.
void	**rename**(java.lang.String newName) Renames the current attribute.
java.lang.String	**toString**() Gets the name of this attribute.

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Method Detail

create

```
public static Attribute create(CDF myCDF,
               java.lang.String name,
               long scope)
                        throws CDFException
```

Creates a new attribute in the given CDF. Attributes and attribute entries are used to describe information about a CDF file and the variables in the file. Any number of attributes may be stored in a CDF file.

The following example creates a global attribute called 'Project' and a variable attribute called 'VALIDMIN':

```
Attribute project, validMin;

project  = Attribute.create(cdf, "Project", GLOBAL_SCOPE);
validMin = Attribute.create(cdf, "VALIDMIN", VARIABLE_SCOPE);
```

Parameters:

myCDF - the CDF object to which this attribute belongs

name - the name of the attribute to be created

scope - the attribute's scope - it should be either GLOBAL_SCOPE or VARIABLE_SCOPE

Returns:

Attribute an Attribute object in the CDF.

Throws:

CDFException - if a problem occurred in creating an attribute

delete

```
public void delete()
            throws CDFException
```

Deletes this attribute.

Note: When an attribute is deleted all the entries for attribute are deleted as well. Also, all attributes that follow the deleted attribute will be renumbered immediately (their IDs will be decremented by one). This can cause confusion when using a loop to delete attributes. The following is incorrect and will result in every other attribute being deleted:

```
     Vector attrs = cdf.getAttributes();
     int n = attrs.size();
     for (int i = 0 i < n; i++)
         ((Attribute)attrs.getElementAt(i)).delete();


 Two possible workarounds are:


     Vector attrs = cdf.getAttributes();
     int n = attrs.size();
     for (int i = n-1; i >= 0; i--)
         ((Attribute)attrs.getElementAt(i)).delete();

     and

     Vector attrs = cdf.getAttributes();
     int n = attrs.size();
     for (int i = 0 i < n; i++)
         ((Attribute)attrs.getElementAt(0)).delete();
```

Specified by:

`delete` in interface `CDFObject`

Throws:

`CDFException` - if there is a problem deleting the attribute

getEntry

```
public Entry getEntry(long entryID)
               throws CDFException
```

Gets the attribute entry for the given entry number.

The following example retrieves the first entry of the global attribute 'project'. Please note that a global attribute can have multiple entries (whereas, a variable attribute has only one entry for a particular attribute), and attribute id starts at 0, not 1.

```
     Entry  tEntry = project.getEntry(0L)
```

Parameters:

`entryID` - the entry number from which an attribute entry is retrieved

Returns:

Entry the Entry object

Throws:

`CDFException` - if an error occurred getting an entry (i.e. invalid entryID, no attribute entry for entryID)

getEntry

```
public Entry getEntry(Variable var)
               throws CDFException
```

Gets the attribute entry for the given variable.

The following example retrieves the 'longitude' variable entry associate with the attribute 'validMin':

```
vEntry = validMin.getEntry(longitude);
```

Parameters:

`var` - the variable from which an attribute entry is retrieved

Returns:

Entry the Entry object

Throws:

`CDFException` - if an error occurred getting a variable attribute entry (e.g. non-existent variable, no attribute entry for this variable, etc.)

deleteEntry

```
public void deleteEntry(long entryID)
                throws CDFException
```

Deletes an attribute entry for the given entry number.

The following example deletes the first and second entries of the global attribute 'Project':

```
project.deleteEntry(0L);
project.deleteEntry(1L);
```

The following example deletes the 'longitude' variable entry associated with the attribute 'validMin':

```
validMin.deleteEntry(longitude.getID());
```

Parameters:

`entryID` - the ID of the entry to be deleted

Throws:

`CDFException` - if there was a porblem deleting the entry

deleteEntry

```
public void deleteEntry(Variable var)
                throws CDFException
```

Deletes the attribute entry for the given variable.

The following example deletes the 'longitude' variable entry associated with the attribute 'validMin':

```
validMin.deleteEntry(longitude);
```

Parameters:

`var` - the variable from which the attribute entry is deleted

Throws:

CDFException - if there was a porblem deleting the entry

getEntries

```
public java.util.Vector getEntries()
                           throws CDFException
```

Gets all the entries defined for this attribute. A global attribute can have multiple entries. Whereas, a variable attribute has only one entry for a particular attribute.

Returns:

all the entries (one or more) defined for a global attribute or all variable entries for this attribute. The number of returned entries vector for a variable attribute may be less than the number of variables, as not all variables have an entry for the attribute.

Throws:

CDFException - if there was a porblem getting the entries

getEntryID

```
public long getEntryID(Entry entry)
```

Gets the entry id for the given entry.

Parameters:

entry - the entry from which an entry id is retrieved

Returns:

the entry id for the given entry

rename

```
public void rename(java.lang.String newName)
            throws CDFException
```

Renames the current attribute.

Specified by:

rename in interface CDFObject

Parameters:

newName - the new attribute name

Throws:

CDFException - if there was a problem renaming the attribute

getNumEntries

```
public long getNumEntries()
```

Gets the number of entries in this attribute.

Returns:

the number of entries in this attribute

getMaxEntryNumber

```
public long getMaxEntryNumber()
```

Gets the largest Entry number for this attribute.

Returns:

the largest Entry number for this attribute

getID

```
public long getID()
```

Gets the attribute ID of this attribute.

Returns:

the attribute id of this attribute

getMyCDF

```
public CDF getMyCDF()
```

Gets the CDF object to which this attribute belongs.

Returns:

the CDF object to which this attribute belongs

getName

```
public java.lang.String getName()
```

Gets the name of this attribute.

Specified by:

`getName` in interface `CDFObject`

Returns:

the name of this attribute

toString

```
public java.lang.String toString()
```

Gets the name of this attribute.

Overrides:

`toString` in class `java.lang.Object`

Returns:

the name of this attribute

getScope

```
public long getScope()
```

Gets the scope of this attribute.

Returns:

If the attribute is a global attribute, GLOBAL_SCOPE is returned. If the attribute is a variable attribute, VARIABLE_SCOPE is returned.

Overview Package Class Tree Deprecated Index Help

Prev Class **Next Class** Frames No Frames All Classes

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

gsfc.nssdc.cdf

Class CDF

java.lang.Object
    gsfc.nssdc.cdf.CDF

All Implemented Interfaces:

CDFConstants, CDFObject

```
public class CDF
extends java.lang.Object
implements CDFObject, CDFConstants
```

The CDF class is the main class used to interact with a CDF file.

Notes:

- All files are placed in zMODE 2 upon opening or creation
- Variable attributes are handled slightly differently from C.
    - Each variable has a java.util.Vector of attributes.
    - This vector contains only those vAttributes that have a z entry for this variable.
    - Therefore, the index for a given variable Attribute may not be the same for another variable.

CDF dataType	Java dataType	Read/Write
CDF_BYTE	java.lang.Byte	Y/Y
CDF_INT1	java.lang.Byte	Y/Y
CDF_UINT1	java.lang.Short	Y/Y
CDF_INT2	java.lang.Short	Y/Y
CDF_UINT2	java.lang.Integer	Y/Y
CDF_INT4	java.lang.Integer	Y/Y
CDF_UINT4	java.lang.Long	Y/Y
CDF_INT8	java.lang.Long	Y/Y
CDF_FLOAT	java.lang.Float	Y/Y
CDF_REAL4	java.lang.Float	Y/Y
CDF_DOUBLE	java.lang.Double	Y/Y
CDF_REAL8	java.lang.Double	Y/Y
CDF_EPOCH	java.lang.Double	Y/Y
CDF_EPOCH16	java.lang.Double[	Y/Y
CDF_TIME_TT2000	java.lang.Long	Y/Y
CDF_CHAR	java.lang.String	Y/Y
CDF_UCHAR	java.lang.String	Y/Y

Version:

1.0, 2.0 03/18/05 Selection of current attribute is done as part of operations passed to JNI. JNI call is synchronized so only one process is allowed in a JVM, due to multi-thread safety. The select method will never be called. Sync'd the CDF (id) for every JNI calls.

See Also:

Attribute, CDFException, Variable

Field Summary

Fields inherited from interface gsfc.nssdc.cdf.CDFConstants

AHUFF_COMPRESSION, ALPHAOSF1_DECODING, ALPHAOSF1_ENCODING, ALPHAVMSd_DECODING, ALPHAVMSd_ENCODING, ALPHAVMSg_DECODING, ALPHAVMSg_ENCODING, ALPHAVMSi_DECODING, ALPHAVMSi_ENCODING, ARM_BIG_DECODING, ARM_BIG_ENCODING, ARM_LITTLE_DECODING, ARM_LITTLE_ENCODING, ATTR_, ATTR_EXISTENCE_, ATTR_EXISTS, ATTR_MAXgENTRY_, ATTR_MAXrENTRY_, ATTR_MAXzENTRY_, ATTR_NAME_, ATTR_NAME_TRUNC, ATTR_NUMBER_, ATTR_NUMgENTRIES_, ATTR_NUMrENTRIES_, ATTR_NUMzENTRIES_, ATTR_SCOPE_, BACKWARD_, BACKWARDFILEoff, BACKWARDFILEon, BAD_ALLOCATE_RECS, BAD_ARGUMENT, BAD_ATTR_NAME, BAD_ATTR_NUM, BAD_BLOCKING_FACTOR, BAD_CACHE_SIZE, BAD_CDF_EXTENSION, BAD_CDF_ID, BAD_CDF_NAME, BAD_CDFSTATUS, BAD_CHECKSUM, BAD_COMPRESSION_PARM, BAD_DATA_TYPE, BAD_DECODING, BAD_DIM_COUNT, BAD_DIM_INDEX, BAD_DIM_INTERVAL, BAD_DIM_SIZE, BAD_ENCODING, BAD_ENTRY_NUM, BAD_FNC_OR_ITEM, BAD_FORMAT, BAD_INITIAL_RECS, BAD_MAJORITY, BAD_MALLOC, BAD_NEGtoPOSfp0_MODE, BAD_NUM_DIMS, BAD_NUM_ELEMS, BAD_NUM_STRINGS, BAD_NUM_VARS, BAD_READONLY_MODE, BAD_REC_COUNT, BAD_REC_INTERVAL, BAD_REC_NUM, BAD_SCOPE, BAD_SCRATCH_DIR, BAD_SPARSEARRAYS_PARM, BAD_VAR_NAME, BAD_VAR_NUM, BAD_zMODE, BADDATE_LEAPSECOND_UPDATED, BeginUnixTimeEPOCH, BeginUnixTimeEPOCH16, BLOCKINGFACTOR_TOO_LARGE, BLOCKINGFACTOR_TOO_SMALL, BLOCKINGFACTOR_TOO_SMALL2, CANNOT_ALLOCATE_RECORDS, CANNOT_CHANGE, CANNOT_COMPRESS, CANNOT_CONVERT_WIDECHAR, CANNOT_COPY, CANNOT_INSERT_RECORDS, CANNOT_SPARSEARRAYS, CANNOT_SPARSERECORDS, CDF_, CDF_ACCESS_, CDF_ATTR_NAME_LEN, CDF_ATTR_NAME_LEN256, CDF_BYTE, CDF_CACHESIZE_, CDF_CHAR, CDF_CHECKSUM_, CDF_CLOSE_ERROR, CDF_COMPRESSION_, CDF_COPYRIGHT_, CDF_COPYRIGHT_LEN, CDF_CREATE_ERROR, CDF_DECODING_, CDF_DELETE_ERROR, CDF_DOUBLE, CDF_ENCODING_, CDF_EPOCH, CDF_EPOCH16, CDF_EXISTS, CDF_FLOAT, CDF_FORMAT_, CDF_INCREMENT_, CDF_INFO_, CDF_INT1, CDF_INT2, CDF_INT4, CDF_INT8, CDF_INTERNAL_ERROR, CDF_LEAPSECONDLASTUPDATED_, CDF_MAJORITY_, CDF_MAX_DIMS, CDF_MAX_PARMS, CDF_MIN_DIMS, CDF_NAME_, CDF_NAME_TRUNC, CDF_NEGtoPOSfp0_MODE_, CDF_NUMATTRS_, CDF_NUMgATTRS_, CDF_NUMrVARS_, CDF_NUMvATTRS_, CDF_NUMzVARS_, CDF_OK, CDF_OPEN_ERROR, CDF_PATHNAME_LEN, CDF_READ_ERROR, CDF_READONLY_MODE_, CDF_REAL4, CDF_REAL8, CDF_RELEASE_, CDF_SAVE_ERROR, CDF_SCRATCHDIR_, CDF_STATUS_, CDF_STATUSTEXT_LEN, CDF_TIME_TT2000, CDF_UCHAR, CDF_UINT1, CDF_UINT2, CDF_UINT4, CDF_VAR_NAME_LEN, CDF_VAR_NAME_LEN256, CDF_VERSION_, CDF_WARN, CDF_WRITE_ERROR, CDF_zMODE_, CDFwithSTATS_, CHECKSUM_, CHECKSUM_ERROR, CHECKSUM_NOT_ALLOWED, CLOSE_, COLUMN_MAJOR, COMPRESS_CACHESIZE_, COMPRESSION_ERROR, CONFIRM_, CORRUPTED_V2_CDF, CORRUPTED_V3_CDF, CREATE_, CURgENTRY_EXISTENCE_, CURrENTRY_EXISTENCE_, CURzENTRY_EXISTENCE_, DATATYPE_MISMATCH, DATATYPE_SIZE_, DECOMPRESSION_ERROR, DECSTATION_DECODING, DECSTATION_ENCODING, DEFAULT_BYTE_PADVALUE, DEFAULT_CHAR_PADVALUE, DEFAULT_DOUBLE_PADVALUE, DEFAULT_EPOCH_PADVALUE, DEFAULT_EPOCH16_PADVALUE, DEFAULT_FLOAT_PADVALUE, DEFAULT_INT1_PADVALUE, DEFAULT_INT2_PADVALUE, DEFAULT_INT4_PADVALUE, DEFAULT_INT8_PADVALUE, DEFAULT_REAL4_PADVALUE, DEFAULT_REAL8_PADVALUE, DEFAULT_TT2000_PADVALUE, DEFAULT_UCHAR_PADVALUE, DEFAULT_UINT1_PADVALUE, DEFAULT_UINT2_PADVALUE, DEFAULT_UINT4_PADVALUE, DELETE_, DID_NOT_COMPRESS, EMPTY_COMPRESSED_CDF, END_OF_VAR, EPOCH_STRING_LEN, EPOCH_STRING_LEN_EXTEND, EPOCHI_STRING_LEN, EPOCH1_STRING_LEN_EXTEND, EPOCH2_STRING_LEN, EPOCH2_STRING_LEN_EXTEND, EPOCH3_STRING_LEN, EPOCH3_STRING_LEN_EXTEND, EPOCH4_STRING_LEN, EPOCH4_STRING_LEN_EXTEND, EPOCHx_FORMAT_MAX, EPOCHx_STRING_MAX, FILLED_TT2000_VALUE, FORCED_PARAMETER, FUNCTION_NOT_SUPPORTED, gENTRY_, gENTRY_DATA_, gENTRY_DATASPEC_, gENTRY_DATATYPE_, gENTRY_EXISTENCE_, gENTRY_NUMELEMS_, GET_, GETCDFCHECKSUM_, GETCDFFILEBACKWARD_, GETCDFVALIDATE_, GETLEAPSECONDSENVVAR_, GLOBAL_SCOPE, GZIP_COMPRESSION, HOST_DECODING, HOST_ENCODING, HP_DECODING, HP_ENCODING, HUFF_COMPRESSION, IA64VMSd_DECODING, IA64VMSd_ENCODING, IA64VMSg_DECODING, IA64VMSg_ENCODING, IA64VMSi_DECODING, IA64VMSi_ENCODING, IBM_PC_OVERFLOW, IBMPC_DECODING, IBMPC_ENCODING, IBMRS_DECODING, IBMRS_ENCODING, ILLEGAL_EPOCH_FIELD, ILLEGAL_EPOCH_VALUE, ILLEGAL_FOR_SCOPE, ILLEGAL_IN_zMODE, ILLEGAL_ON_V1_CDF, ILLEGAL_TT2000_VALUE, IS_A_NETCDF, LIB_COPYRIGHT_, LIB_INCREMENT_, LIB_RELEASE_, LIB_subINCREMENT_, LIB_VERSION_, MAC_DECODING, MAC_ENCODING, MD5_CHECKSUM, MULTI_FILE, MULTI_FILE_FORMAT, NA_FOR_VARIABLE, NEGATIVE_FP_ZERO, NEGtoPOSfp0off, NEGtoPOSfp0on, NETWORK_DECODING, NETWORK_ENCODING, NeXT_DECODING, NeXT_ENCODING, NO_ATTR_SELECTED, NO_CDF_SELECTED, NO_CHECKSUM, NO_COMPRESSION, NO_DELETE_ACCESS, NO_ENTRY_SELECTED, NO_MORE_ACCESS, NO_PADVALUE_SPECIFIED, NO_SPARSEARRAYS, NO_SPARSERECORDS, NO_STATUS_SELECTED, NO_SUCH_ATTR, NO_SUCH_CDF, NO_SUCH_ENTRY, NO_SUCH_RECORD, NO_SUCH_VAR, NO_VAR_SELECTED, NO_VARS_IN_CDF, NO_WRITE_ACCESS, NONE_CHECKSUM, NOT_A_CDF, NOT_A_CDF_OR_NOT_SUPPORTED, NOVARY, NULL_, OPEN_, OPTIMAL_ENCODING_TREES, OTHER_CHECKSUM, PAD_SPARSERECORDS, PPC_DECODING, PPC_ENCODING, PRECEEDING_RECORDS_ALLOCATED, PREV_SPARSERECORDS, PUT_, READ_ONLY_DISTRIBUTION, READ_ONLY_MODE, READONLYoff, READONLYon, rENTRY_, rENTRY_DATA_, rENTRY_DATASPEC_, rENTRY_DATATYPE_, rENTRY_EXISTENCE_, rENTRY_NAME_, rENTRY_NUMELEMS_, rENTRY_NUMSTRINGS_, rENTRY_STRINGSDATA_, RLE_COMPRESSION, RLE_OF_ZEROs, ROW_MAJOR, rVAR_, rVAR_ALLOCATEBLOCK_, rVAR_ALLOCATEDFROM_, rVAR_ALLOCATEDTO_, rVAR_ALLOCATERECS_, rVAR_BLOCKINGFACTOR_, rVAR_CACHESIZE_, rVAR_COMPRESSION_, rVAR_DATA_, rVAR_DATASPEC_, rVAR_DATATYPE_, rVAR_DIMVARYS_, rVAR_EXISTENCE_, rVAR_HYPERDATA_, rVAR_INITIALRECS_, rVAR_MAXallocREC_, rVAR_MAXREC_, rVAR_NAME_, rVAR_nINDEXENTRIES_, rVAR_nINDEXLEVELS_, rVAR_nINDEXRECORDS_, rVAR_NUMallocRECS_, rVAR_NUMBER_, rVAR_NUMELEMS_, rVAR_NUMRECS_, rVAR_PADVALUE_, rVAR_RECORDS_, rVAR_RECORDS_RENUMBER_, rVAR_RECVARY_, rVAR_RESERVEPERCENT_, rVAR_SEQDATA_, rVAR_SEQPOS_, rVAR_SPARSEARRAYS_, rVAR_SPARSERECORDS_, rVARs_CACHESIZE_, rVARs_DIMCOUNTS_, rVARs_DIMINDICES_, rVARs_DIMINTERVALS_, rVARs_DIMSIZES_, rVARs_MAXREC_, rVARs_NUMDIMS_, rVARs_RECCOUNT_, rVARs_RECDATA_, rVARs_RECINTERVAL_, rVARs_RECNUMBER_, SAVE_, SCRATCH_CREATE_ERROR, SCRATCH_DELETE_ERROR, SCRATCH_READ_ERROR, SCRATCH_WRITE_ERROR, SELECT_, SGI_DECODING, SGi_ENCODING, SINGLE_FILE, SINGLE_FILE_FORMAT, SOME_ALREADY_ALLOCATED, STAGE_CACHESIZE_, STATUS_TEXT_, STRING_NOT_UTF8_ENCODING, STRINGDELIMITER, SUN_DECODING, SUN_ENCODING, TOO_MANY_PARMS, TOO_MANY_VARS, TRY_TO_READ_NONSTRING_DATA, TT2000_0_STRING_LEN, TT2000_1_STRING_LEN, TT2000_2_STRING_LEN, TT2000_3_STRING_LEN, TT2000_4_STRING_LEN, TT2000_CDF_MAYNEEDUPDATE, TT2000_TIME_ERROR, TT2000_USED_OUTDATED_TABLE, UNABLE_TO_PROCESS_CDF, UNKNOWN_COMPRESSION, UNKNOWN_SPARSENESS, UNSUPPORTED_OPERATION, VALIDATE_, VALIDATEFILEoff, VALIDATEFILEon, VAR_ALREADY_CLOSED, VAR_CLOSE_ERROR, VAR_CREATE_ERROR, VAR_DELETE_ERROR, VAR_EXISTS, VAR_NAME_TRUNC, VAR_OPEN_ERROR, VAR_READ_ERROR, VAR_SAVE_ERROR, VAR_WRITE_ERROR, VARIABLE_SCOPE, VARY, VAX_DECODING, VAX_ENCODING, VIRTUAL_RECORD_DATA, zENTRY_,

zENTRY_DATA_, zENTRY_DATASPEC_, zENTRY_DATATYPE_, zENTRY_EXISTENCE_, zENTRY_NAME_, zENTRY_NUMELEMS_, zENTRY_NUMSTRINGS_, zENTRY_STRINGSDATA_, ZLIB_COMPRESS_ERROR, ZLIB_UNCOMPRESS_ERROR, zMODEoff, zMODEon1, zMODEon2, zVAR_, zVAR_ALLOCATEBLOCK_, zVAR_ALLOCATEDFROM_, zVAR_ALLOCATEDTO_, zVAR_ALLOCATERECS_, zVAR_BLOCKINGFACTOR_, zVAR_CACHESIZE_, zVAR_COMPRESSION_, zVAR_DATA_, zVAR_DATASPEC_, zVAR_DATATYPE_, zVAR_DIMCOUNTS_, zVAR_DIMINDICES_, zVAR_DIMINTERVALS_, zVAR_DIMSIZES_, zVAR_DIMVARYS_, zVAR_EXISTENCE_, zVAR_HYPERDATA_, zVAR_INITIALRECS_, zVAR_MAXallocREC_, zVAR_MAXREC_, zVAR_NAME_, zVAR_nINDEXENTRIES_, zVAR_nINDEXLEVELS_, zVAR_nINDEXRECORDS_, zVAR_NUMallocRECS_, zVAR_NUMBER_, zVAR_NUMDIMS_, zVAR_NUMELEMS_, zVAR_NUMRECS_, zVAR_PADVALUE_, zVAR_RECCOUNT_, zVAR_RECINTERVAL_, zVAR_RECNUMBER_, zVAR_RECORDS_, zVAR_RECORDS_RENUMBER_, zVAR_RECVARY_, zVAR_RESERVEPERCENT_, zVAR_SEQDATA_, zVAR_SEQPOS_, zVAR_SPARSEARRAYS_, zVAR_SPARSERECORDS_, zVARs_CACHESIZE_, zVARs_MAXREC_, zVARs_RECDATA_, zVARs_RECNUMBER_

Method Summary

Methods

Modifier and Type	Method and Description
void	**close**() Closes this CDF file.
long	**confirmCDFCacheSize**() Gets the CDF cache size (the number of 512-byte cache buffers) set for this CDF.
long	**confirmCompressCacheSize**() Gets the number of 512-byte cache buffers being used for the compression scratch file (for the current CDF).
long	**confirmDecoding**() Gets the CDF decoding method defined for this CDF.
long	**confirmNegtoPosfp0**() Gets the -0.0 to 0.0 translation flag set for this CDF.
long	**confirmReadOnlyMode**() Gets the value of the read-only mode flag set for this CDF file.
long	**confirmStageCacheSize**() Gets the number of 512-byte cache buffers defined for the staging scratch file.
long	**confirmzMode**() Gets the zMode set for this CDF.
static **CDF**	**create**(java.lang.String path) Creates a CDF file in the current directory.
static **CDF**	**create**(java.lang.String path, int flag) **Deprecated.** *Use setFileBackward(long) method to set the file backward flag and create(String) to create file instead.*
void	**delete**() Deletes this CDF file.
void	**finalize**() Do the necessary cleanup when garbage collector reaps it.
Attribute	**getAttribute**(long attrNum) Gets the attribute for the given attribute number.
Attribute	**getAttribute**(java.lang.String attrName) Gets the attribute for the given attribute name.
long	**getAttributeID**(java.lang.String attrName) Gets the id of the given attribute.
java.util.Vector	**getAttributes**() Gets all the global and variable attributes defined for this CDF.
long	**getChecksum**()

	Gets the checksum method, if any, applied to the CDF.
static long	**getChecksumEnvVar**() Gets the value of the CDF_CHECKSUM environment variable.
java.lang.String	**getCompression**() Gets the string representation of the compression type and parameters defined for this CDF.
long[]	**getCompressionParms**() Gets the compression parameters set for this CDF.
long	**getCompressionPct**() Gets the compression percentage set for this CDF.
long	**getCompressionType**() Gets the compression type set for this CDF.
java.lang.String	**getCopyright**() Gets the CDF copyright statement for this CDF.
CDFDelegate	**getDelegate**() This is a placeholder for future expansions/extensions.
long	**getEncoding**() Gets the encoding method defined for this CDF.
static boolean	**getFileBackward**() Gets the file backward flag.
static int	**getFileBackwardEnvVar**() Gets the value of the CDF_FILEBACKWARD environment variable.
long	**getFormat**() Gets the CDF format defined for this CDF.
java.util.Vector	**getGlobalAttributes**() Gets the global attributes defined for this CDF.
long	**getID**() Gets the id of this CDF file.
long	**getLeapSecondLastUpdated**() Gets the date that the CDF has the last leap second updated.
static java.lang.String	**getLeapSecondsTableEnvVar**() Gets the the CDF_LEAPSECONDSTABLE (or CDF$LEAPSECONDSTABLE on VMS) environment variable.
static java.lang.String	**getLibraryCopyright**() Retrieve library copyright information associated with the CDF library.
static java.lang.String	**getLibraryVersion**() Retrieve library version/release/increment/sub_increment information associated with the CDF library.
long	**getMajority**() Gets the variable majority defined for this CDF.
java.lang.String	**getName**() Gets the name of this CDF.
long	**getNumAttrs**() Gets the total number of global and variable attributes in this CDF.
long	**getNumGattrs**() Gets the number of global attributes in this CDF.
long	**getNumRvars**() Gets the number of r variables.
long	**getNumVars**() Gets the number of Z variables defined for this CDF.
long	**getNumVattrs**() Gets the number of variable attributes in this CDF.

long	**getNumZvars**() Gets the number of z variables in this CDF file.
java.util.Vector	**getOrphanAttributes**() Gets the variable attributes defined for this CDF that are not associated with any variables.
java.util.Vector	**getRecord**(long recNum, long[] varIDs) Retrieves a logical record that consists of single variable record(s) from an arbitrary number of CDF variables.
java.util.Vector	**getRecord**(long recNum, long[] varIDs, long[] status) Retrieves a logical record that consists of single variable record(s) from an arbitrary number of CDF variables.
java.util.Vector	**getRecord**(long recNum, java.lang.String[] strVars) Retrieves a logical record that consists of single variable record(s) from an arbitrary number of CDF variables.
java.util.Vector	**getRecord**(long recNum, java.lang.String[] strVars, long[] status) Retrieves a logical record that consists of single variable record(s) from an arbitrary number of CDF variables.
long	**getStatus**() Gets the status of the most recent CDF JNI/library function call.
static java.lang.String	**getStatusText**(long statusCode) Gets the status text of the most recent CDF JNI/library function call.
static boolean	**getValidate**() Gets the file validation mode.
Variable	**getVariable**(long varNum) Gets the variable object for the given variable number.
Variable	**getVariable**(java.lang.String varName) Gets the variable object for the given variable name.
java.util.Vector	**getVariableAttributes**() Gets the variable attributes defined for this CDF.
long	**getVariableID**(java.lang.String varName) Gets the ID of the given variable.
java.util.Vector	**getVariables**() Gets the z variables defined for this CDF.
java.lang.String	**getVersion**() Gets the CDF library version that was used to create this CDF (e.g. 2.6.7, etc.).
static **CDF**	**open**(java.lang.String path) Open a CDF file for read/write, the default mode for opening a CDF.
static **CDF**	**open**(java.lang.String path, long readOnly) Open a CDF file.
void	**putRecord**(long recNum, long[] varIDs, java.util.Vector myData) Writes a logical record that consists of single variable record(s) from an arbitrary number of CDF variables.
void	**putRecord**(long recNum, long[] varIDs, java.util.Vector myData, long[] status) Writes a logical record that consists of single variable record(s) from an arbitrary number of CDF variables.
void	**putRecord**(long recNum, java.lang.String[] strVars, java.util.Vector myData) Writes a logical record that consists of single variable record(s) from an arbitrary number of CDF variables.
void	**putRecord**(long recNum, java.lang.String[] strVars, java.util.Vector myData, long[] status) Writes a logical record that consists of single variable record(s) from an arbitrary number of CDF variables.
java.util.Map	**readCDF**()

	Gets the full CDF information.
java.util.Map	**readCDF**(boolean encoding) Gets the full CDF information.
java.util.Map	**readCDF**(boolean encoding, boolean basic, boolean globals, boolean varinfo) Gets the specified information from a CDF.
java.util.Map	**readCDF**(boolean encoding, boolean basic, boolean globals, boolean varinfo, boolean varmeta, boolean vardata) Gets the specified information from a CDF.
java.util.Map	**readCDF**(boolean encoding, boolean basic, boolean globals, boolean varinfo, boolean varmeta, boolean vardata, boolean noentry) Gets the specified information from a CDF.
java.util.Map	**readCDFGlobalAttributes**() Gets the global attributes and their entries defined for this CDF.
java.util.Map	**readCDFGlobalAttributes**(boolean encoding) Gets the global attributes and their entries defined for this CDF.
java.util.Map	**readCDFGlobalAttributes**(long attrNum) Gets a specified global attribute's entry(ies) defined for this CDF.
java.util.Map	**readCDFGlobalAttributes**(long[] attrNums) Gets a specified list of global attributes and their entries defined for this CDF.
java.util.Map	**readCDFGlobalAttributes**(java.lang.Object attrs, boolean encoding) Gets a list of specified global attributes and their entries defined for this CDF.
java.util.Map	**readCDFGlobalAttributes**(java.lang.String attrName) Gets a specified global attribute's entry(ies) defined for this CDF.
java.util.Map	**readCDFGlobalAttributes**(java.lang.String[] attrNames) Gets a specified list of global attributes and their entries defined for this CDF.
java.util.Map	**readCDFInfo**() Gets the basic information about this CDF.
java.util.Map	**readCDFNoEntryAttributes**() Gets a map of global attributes and variable attributes with no entry data.
java.util.Map	**readCDFVariables**() Reads all variables' information in the CDF.
java.util.Map	**readCDFVariablesData**() Reads all variables' data in the CDF.
java.util.Map	**readCDFVariablesMetaData**() Reads all variables' metadata in the CDF.
java.util.Map	**readCDFVariablesSpec**() Reads all variables' specifications in the CDF.
void	**rename**(java.lang.String path) Renames the current CDF.
void	**save**() Saves this CDF file without closing.
void	**selectCDFCacheSize**(long cacheSize) Defines the number of 512-byte cache buffers to be used for the dotCDF file (for the current CDF).
void	**selectCompressCacheSize**(long compressCacheSize) Sets the number of 512-byte cache buffers to be used for the compression scratch file (for the current CDF).
void	**selectDecoding**(long decoding) Defines the CDF decoding method to be used for this CDF.
void	**selectNegtoPosfp0**(long negtoPosfp0) Defines whether to translate -0.0 to 0.0 for reading or writing.
void	**selectReadOnlyMode**(long readOnly) Sets the desired read-only mode.

void	**selectStageCacheSize**(long stageCacheSize) Sets the number of 512-byte cache buffers to be used for the staging scratch file (for the current CDF).
void	**setChecksum**(long checksum) Specifies the checksum option applied to the CDF.
void	**setCompression**(long cType, long[] cParms) Sets the compression type and parameters for this CDF.
void	**setDelegate**(**CDFDelegate** delegate) This is a placeholder for future expansions/extensions.
void	**setEncoding**(long encoding) Defines the encoding method to be used for this CDF.
static void	**setFileBackward**(long flag) Sets the file backward flag so that when a new CDF file is created, it will be created in either in the older V2.7 version or the current library version, i.e., V3.*.
void	**setFormat**(long format) Specifies the format of this CDF.
void	**setInfoWarningOff**() Sets the informational (status code < 0) or warning messages (status code between -1 and -2000) coming from the CDF JNI/library function off.
void	**setInfoWarningOn**() Sets the informational (status code < 0) or warning messages (status code between -1 and -2000) coming from the CDF JNI/library function on.
void	**setLeapSecondLastUpdated**(long lastUpdated) Set the leap second last updated for this CDF.
void	**setMajority**(long majority) Sets the variable majority for this CDF.
static void	**setValidate**(long mode) Sets the file validation mode so that when a CDF file is open, it will be validated accordingly.
java.lang.String	**toString**() Gets the name of this CDF.
long	**verifyChecksum**() Verifies the data integrity of the CDF file from its checksum.

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Method Detail

create

```
public static CDF create(java.lang.String path)
                  throws CDFException
```

Creates a CDF file in the current directory. By default, a single-file CDF is created and it's the preferred format. However, if the user wants to create a multi-file CDF, its file format needs to be changed as following:

```
CDF cdf = null;
cdf = CDF.create("test");
cdf.setFormat(MULTI_FILE);
```

For the single-file format CDF, the above example would have created a single-file CDF called 'test.cdf'. See Chapter 1 of the CDF

User's Guide for more information about the file format options. **Notes:**
The newly created file will be of the same version as the CDF library, as a V3.*. To create a backward file, i.e., V2.7, there are two options that can be used. Use the static method setFileBackward to set the backward flag. The following example will create backward file for test1.cdf and test2.cdf, but a V3.* file for test3.cdf.

```
CDF cdf1, cdf2, cdf3;
CDF.setFileBackward(BACKWARDFILEon);
cdf1 = CDF.create("test1");
cdf2 = CDF.create("test2");
CDF.setFileBackward(BACKWARDFILEoff);
cdf3 = CDF.create("test3");
```

Alternatively, use an environment variable to control the backward file creation. The environment variable CDF_FILEBACKWARD on Unix or Windows or CDF$FILEBACKWARD on Open/VMS is used. When it is set to TRUE, a V2.7 file(s) will be created automatically. In the following example, both test1.cdf and test2.cdf will be V2.7 if environment variable CDF_FILEBACKWARD (or CDF$FILEBACKWARD) is TRUE.

```
CDF cdf1 = CDF.create("test1");
CDF cdf2 = CDF.create("test2");
```

Parameters:

`path` - the full pathname of the CDF file to be created

Returns:

the newly created CDF file/object

Throws:

`CDFException` - if there was a problem creating a CDF file

create

```
public static CDF create(java.lang.String path,
         int flag)
                  throws CDFException
```

Deprecated. *Use setFileBackward(long) method to set the file backward flag and create(String) to create file instead.*
Creates a CDF file in the current directory. By default, a single-file CDF is created and it's the preferred format. The following example will create a CDF file:

```
CDF cdf = null;
cdf = CDF.create("test", 0);
```

For the single-file format CDF, the above example would have created a single-file CDF called 'test.cdf'. The newly created file will be of the same version as the CDF library, To create a backward file, i.e., V2.7, use a differnt argument for the flag.

```
CDF cdf;
cdf = CDF.create("test", 1);
```

Parameters:

`path` - the full pathname of the CDF file to be created

`flag` - the file backward indicator flag. Passed 0 if a file of current library version is to be created. Not 0 if a backward is to be created.

Returns:

the newly created CDF file/object

Throws:

`CDFException` - if there was a problem creating a CDF file

open

```
public static CDF open(java.lang.String path)
                throws CDFException
```

Open a CDF file for read/write, the default mode for opening a CDF. If the user wants only to read the file, the file must be opened in read-only mode as following:

```
CDF cdf = CDF.open(fileName, READONLYon);
```

Note: Opening a file with read/write mode will cause the checksum signature to be recomputed every time the file is closed. (NEW) Each open CDF file is subjected to a default data validating process (some overheads) that will perform sanity checks. To overwrite it, you can use one of two ways: 1) Use the CDF static method setValidate before calling the open method: CDF.setValidate (VALIDATEFILEoff); 2) Set the environment variable CDF_VALIDATE (CDF$VALIDATE on VMS) to no

Parameters:

`path` - the full pathname of the CDF file to be opened

Returns:

the CDF object that represents the CDF file the user requested for opening

Throws:

`CDFException` - if there was a major problem opening a file Not always that a CDFException will be thrown. It is good pratice to check the status from this open method to see if some others problems, e.g., invalid checksum is being detected, may occur. Use something like the followings

```
if (cdf.getStatus() != CDF_OK)
{
  if (cdf.getStatus() == CHECKSUM_ERROR)
    ......
}
```

where cdf is the returned object from the open method. It is up to each individual to determine whether to continue to use a CDF file with an error like checksum.

open

```
public static CDF open(java.lang.String path,
       long readOnly)
                throws CDFException
```

Open a CDF file. A CDF file can be opened in read-only or read/write mode. If a file is opened in read-only mode, the user can only read values out of the file. Any operation other than reading data will throw a CDFException. If the user wants to modify the contents of a file, the file must be opened in read/write mode as following:

```
CDF cdf = CDF.open(fileName, READONLYoff);
```

Parameters:

`path` - the full pathname of the CDF file to be opened

`readOnly` - read-only flag that should be one the following:

- READONLYon - opens the file in read only mode.
- READONLYoff - opens the file in read/write mode

Returns:

the CDF object that represents the CDF file the user requested for opening

Throws:

CDFException - if there was a major problem opening a file Not always that a CDFException will be thrown. It is good pratice to check the status from this open method to see if some others problems, e.g., invalid checksum is being detected, may occur. Use something like the followings

```
if (cdf.getStatus() != CDF_OK)
{
  if (cdf.getStatus() == CHECKSUM_ERROR)
    ......
}
```

where cdf is the returned object from the open method. It is up to each individual to determine whether to continue to use a CDF file with an error like checksum.

getLibraryVersion

```
public static java.lang.String getLibraryVersion()
                                          throws CDFException
```

Retrieve library version/release/increment/sub_increment information associated with the CDF library.

Returns:

The library version in string

Throws:

CDFException - If there was a problem retrieving the information associated with this CDF file

getLibraryCopyright

```
public static java.lang.String getLibraryCopyright()
                                            throws CDFException
```

Retrieve library copyright information associated with the CDF library.

Returns:

the string for CDF copy right note

Throws:

CDFException - If there was a problem retrieving the information associated with this CDF file

close

```
public void close()
           throws CDFException
```

Closes this CDF file. It is essential that a CDF that has been created or modified by an application be closed before the program exits. If the CDF is not closed, the file will be corrupted and unreadable. This is because the cache buffers maintained by the CDF library will not have been written to the CDF file(s).

The following example closes a CDF file:

```
cdf.close();
```

Throws:

CDFException - if there was a problem closing the CDF file

getID

```
public long getID()
```

Gets the id of this CDF file.

Returns:

the id of this CDF file

getEncoding

```
public long getEncoding()
```

Gets the encoding method defined for this CDF.

Returns:

The encoding method defined for this CDF file. One of the encoding methods described in the setEncoding method is returned.

setEncoding

```
public void setEncoding(long encoding)
                 throws CDFException
```

Defines the encoding method to be used for this CDF. A CDF's data encoding affects how its attribute entry and variable data values are stored. By default, attribute entry and variable data values passed into the CDF library are always stored using the host machine's native encoding. For example, if a CDF file is created without specifying what encoding method should be should on a IBM PC, the IBMPC_ENCODING method is used. This method becomes useful if someone wants to create a CDF file that will be read on a machine that is different from the machine the CDF file was created. A CDF with any of the supported encodings may be read from and written to any supported computer. See section 2.2.8 of the CDF User's Guide for a detailed description of the encodings listed below.

Parameters:

`encoding` - the encoding method to be used for this CDF that should be one of the following:

- HOST_ENCODING
- NETWORK_ENCODING
- SUN_ENCODING
- VAX_ENCODING
- DECSTATION_ENCODING
- SGi_ENCODING
- IBMPC_ENCODING
- IBMRS_ENCODING
- PPC_ENCODING
- HP_ENCODING
- NeXT_ENCODING
- ALPHAOSF1_ENCODING
- ALPHAVMSd_ENCODING
- ALPHAVMSg_ENCODING
- ALPHAVMSi_ENCODING

Throws:

`CDFException` - if there was a problem setting the requested encoding method

selectDecoding

```
public void selectDecoding(long decoding)
                    throws CDFException
```

Defines the CDF decoding method to be used for this CDF. 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 any number of times while the CDF is open. Selecting a decoding does not affect how the values are store in the CDF file(s) - only how the values are decoded by the CDF library.

Parameters:

`decoding` - the decoding method to be used for this CDF that should be one of the following:

- HOST_DECODING - this is the default decoding
- NETWORK_DECODING
- SUN_DECODING
- VAX_DECODING
- DECSTATION_DECODING
- SGi_DECODING
- IBMPC_DECODING
- IBMRS_DECODING
- PPC_DECODING
- HP_DECODING
- NeXT_DECODING
- ALPHAOSF1_DECODING
- ALPHAVMSd_DECODING
- ALPHAVMSg_DECODING
- ALPHAVMSi_DECODING

Throws:

`CDFException` - if there was a problem selecting the requested decoding method

confirmDecoding

```
public long confirmDecoding()
                     throws CDFException
```

Gets the CDF decoding method defined for this CDF.

Returns:

The decoding method set for this CDF file. One of the decoding methods defined in the selectDecoding method is returned.

Throws:

`CDFException` - if there was a problem getting the decoding method set for this CDF file

selectCDFCacheSize

```
public void selectCDFCacheSize(long cacheSize)
                        throws CDFException
```

Defines the number of 512-byte cache buffers to be used for the dotCDF file (for the current CDF). The concepts Chapter in the CDF User's Guide describes the caching scheme used by the CDF library.

Parameters:

`cacheSize` - the number of 512-byte cache buffers

Throws:

CDFException - if there was a problem setting the CDF cache size

confirmCDFCacheSize

```
public long confirmCDFCacheSize()
                          throws CDFException
```

Gets the CDF cache size (the number of 512-byte cache buffers) set for this CDF.

Returns:

the number of 512-byte cache buffers set for this CDF

Throws:

CDFException - if there was a problem getting the CDF cache size

selectNegtoPosfp0

```
public void selectNegtoPosfp0(long negtoPosfp0)
                       throws CDFException
```

Defines whether to translate -0.0 to 0.0 for reading or writing. Negative floating-point zero (-0.0) is legal on computers that use IEEE 754 floating-point representation (e.g. most UNIX-based computers and the PC) but is illegal on VAXes and DEC alphas running OpenVMS operating system. If this mode disabled, a warning (NEGATIVE_FP_ZERO) is returned when -0.0 is read from a CDF (and the decoding is that of a VAX or DEC Alpha running OpenVMS) or written to a CDF (and the encoding is that of a VAX or DEC Alpha running i OpenVMS).

Parameters:

negtoPosfp0 - flag to translate -0.0 to 0.0 (NEGtoPOSfp0on = on, NEGtoPOSfp0off = off)

Throws:

CDFException - if there was a problem setting the -0.0 to 0.0 translation flag

confirmNegtoPosfp0

```
public long confirmNegtoPosfp0()
                         throws CDFException
```

Gets the -0.0 to 0.0 translation flag set for this CDF.

Returns:

flag to translate -0.0 to 0.0 (NEGtoPOSfp0on = on, NEGtoPOSfp0off = off)

Throws:

CDFException - if there was a problem getting the value of the -0.0 to 0.0 translation flag

getFormat

```
public long getFormat()
```

Gets the CDF format defined for this CDF.

Returns:

the format of this CDF (SINGLE_FILE = single-file CDF, MULTI_FILE = multi-file CDF)

setFormat

```
public void setFormat(long format)
               throws CDFException
```

Specifies the format of this CDF. A CDF's format can't be changed once any variables are created. See section 1.4 of the CDF User's Guide for more detailed information about the file format options.

Parameters:

`format` - the CDF file format to be used that should be one of the following:
- SINGLE_FILE - This is the default. The CDF consists of only one file.
- MULTI_FILE - The CDF consists of one header file for control and attribute data and one additional file for each variable in the CDF.

Throws:

`CDFException` - if there was a problem setting a file format

getVersion

```
public java.lang.String getVersion()
```

Gets the CDF library version that was used to create this CDF (e.g. 2.6.7, etc.).

Returns:

the CDF libray version number that was used to create this CDF

getMajority

```
public long getMajority()
```

Gets the variable majority defined for this CDF.

Returns:

the variable majority defined for this CDF (ROW_MAJOR = row major, COLUMN_MAJOR = column major)

setMajority

```
public void setMajority(long majority)
                 throws CDFException
```

Sets the variable majority for this CDF. The variable majority of a CDF describes how variable values within each variable array (record) are stored. Each variable in a CDF has the same majority.

Parameters:

`majority` - The majority to be used in storing data (ROW_MAJOR = row major, COLUMN_MAJOR = column major)

Throws:

`CDFException` - if a problem occurred in setting a majority

getNumAttrs

```
public long getNumAttrs()
```

Gets the total number of global and variable attributes in this CDF.

Returns:

the total number of global and variable attributes in this CDF

getNumGattrs

```
public long getNumGattrs()
```

Gets the number of global attributes in this CDF.

Returns:

the number of global attributes in this CDF file

getNumVattrs

```
public long getNumVattrs()
```

Gets the number of variable attributes in this CDF. Since r variables are not supported by the CDF Java APIs, the number of z variables is always returned.

Returns:

the number of variable attributes in this CDF file

getNumRvars

```
public long getNumRvars()
```

Gets the number of r variables. Zero is returned since r variables are not supported. Z variables can do everything r variables can do plus more.

Returns:

the number of r variables in this CDF file

getNumZvars

```
public long getNumZvars()
```

Gets the number of z variables in this CDF file.

Returns:

the number of z variables in this CDF file

getCopyright

```
public java.lang.String getCopyright()
```

Gets the CDF copyright statement for this CDF.

Returns:

the CDF copyright statement

selectReadOnlyMode

```
public void selectReadOnlyMode(long readOnly)
                        throws CDFException
```

Sets the desired read-only mode. See the description of the read-only flag defined in the open method in this class for details. Caveat: Arbitrary changing the read-only mode to READONLYon while doing writing/updating will cause a problem to the file if the checksum bit is turned on (as the checksum signature may not get updated and a warning for data integrity will be issued when the file is open later).

Parameters:

`readOnly` - read-only flag (READONLYon = on, READONLYoff = off)

Throws:

`CDFException` - if a problem occurred in setting a flag

confirmReadOnlyMode

```
public long confirmReadOnlyMode()
                         throws CDFException
```

Gets the value of the read-only mode flag set for this CDF file.

Returns:

read-only flag (READONLYon = on, READONLYoff = off)

Throws:

`CDFException` - if a problem occurred in getting the value of the read-only flag set for this CDF file

getCompressionType

```
public long getCompressionType()
```

Gets the compression type set for this CDF.

Returns:

the compression type set for this CDF - one of the following is returned:

- NO_COMPRESSION - no compresssion
- RLE_COMPRESSION - Run-length compression
- HUFF_COMPRESSION - Huffman compression
- AHUFF_COMPRESSION - Adaptive Huffman compression
- GZIP_COMPRESSION - Gnu's "zip" compression

getCompressionPct

```
public long getCompressionPct()
```

Gets the compression percentage set for this CDF.

Returns:

the compression percentage set for this CDF.

getCompressionParms

```
public long[] getCompressionParms()
```

Gets the compression parameters set for this CDF. See the description of the setCompression method in this class for more information.

Returns:

the compression parameter set for this CDF

setCompression

```
public void setCompression(long cType,
                  long[] cParms)
                    throws CDFException
```

Sets the compression type and parameters for this CDF.

Parameters:

`cType` - the compression type to be applied to this CDF that should be one of the following:

- NO_COMPRESSION - no compresssion
- RLE_COMPRESSION - Run-length compression. Currently, only the run-length encoding of zeros is supported. The compression parameter must be set to RLE_OF_ZEROs.
- HUFF_COMPRESSION - Huffman compression. Currently, only optimal encoding trees are supported. The compression parameter must be set to OPTIMAL_ENCODING_TREES.
- AHUFF_COMPRESSION - Adaptive Huffman compression. Currently, only optimal encoding trees are supported. The compression parameter must be set to OPTIMAL_ENCODING_TREES.
- GZIP_COMPRESSION - Gnu's "zip" compression. The compression parameter may range from 1 to 9. 1 provides the least compression and requires less execution time. 9 provides the most compression but requires the most execution time.

`cParms` - Compression parameter. There is only one parameter for all the compression methods described above.

Throws:

`CDFException` - if a problem occurred in setting the compression type and parameters

getCompression

```
public java.lang.String getCompression()
                                throws CDFException
```

Gets the string representation of the compression type and parameters defined for this CDF.

Returns:

the string representation of the compression type and parameters (e.g. GZIP.9, RLE.0, etc.) defined for this CDF

Throws:

`CDFException` - if a problem occurred in getting the compression type and parameters set for this CDF

confirmzMode

```
public long confirmzMode()
                  throws CDFException
```

Gets the zMode set for this CDF.

Returns:

'zMODEon2' is always returned since it is the only mode supported by the CDF Java APIs.

Throws:

`CDFException` - if a problem occurred in getting the zmode set for this CDF file

selectCompressCacheSize

```
public void selectCompressCacheSize(long compressCacheSize)
                             throws CDFException
```

Sets the number of 512-byte cache buffers to be used for the compression scratch file (for the current CDF). The Concepts Chapter in the CDF User's Guide describes the caching scheme used by the CDF library.

Parameters:

`compressCacheSize` - the number of 512-byte cache buffers to be used

Throws:

`CDFException` - if a problem occurs in setting the cache size

confirmCompressCacheSize

```
public long confirmCompressCacheSize()
                              throws CDFException
```

Gets the number of 512-byte cache buffers being used for the compression scratch file (for the current CDF).

Returns:

the number of 512-byte cache buffers being used

Throws:

`CDFException` - if a problem occurs in getting the cache size defined

selectStageCacheSize

```
public void selectStageCacheSize(long stageCacheSize)
                          throws CDFException
```

Sets the number of 512-byte cache buffers to be used for the staging scratch file (for the current CDF). The Concepts Chapter in the CDF User's Guide describes the caching scheme used by the CDF library.

Parameters:

`stageCacheSize` - the Number of cache buffers to be used

Throws:

`CDFException` - if a problem occurs in setting the cache size

confirmStageCacheSize

```
public long confirmStageCacheSize()
                            throws CDFException
```

Gets the number of 512-byte cache buffers defined for the staging scratch file.

Returns:

the number of 512-byte cache buffers defined for the staging scratch file

Throws:

`CDFException` - if a problem occurs in getting the number of cache buffers defined for the staging scratch file

getName

```
public java.lang.String getName()
```

Gets the name of this CDF.

Specified by:

`getName` in interface `CDFObject`

Returns:

the name of this CDF

rename

```
public void rename(java.lang.String path)
```

Renames the current CDF. It's here because CDF.java implements the CDFObject interface that defines three methods: rename, delete, getname. This method doesn't do anything now, but it will be refined to rename a single-CDF and multi-CDF files in the future.

Specified by:

`rename` in interface `CDFObject`

Parameters:

`path` - the new CDF name to be renamed to

delete

```
public void delete()
            throws CDFException
```

Deletes this CDF file.

Specified by:

`delete` in interface `CDFObject`

Throws:

`CDFException` - if a problem occurs in deleting this CDF file

save

```
public void save()
```

```
        throws CDFException
```

Saves this CDF file without closing. There are times the users will have to save the contents of a CDF file before some operations can be performed. For example, a CDF file must be saved first before records can be deleted properly for variables that are defined to have sparse and/or compressed records.

Throws:

`CDFException` - if there was a problem saving the contents of this CDF file

setFileBackward

```
public static void setFileBackward(long flag)
                            throws CDFException
```

Sets the file backward flag so that when a new CDF file is created, it will be created in either in the older V2.7 version or the current library version, i.e., V3.*. It only works for V3.* library. Setting this flag will overwrite environment varibale CDF_FILEBACKWARD (or CDF$FILEBACKWARD on OpenVMS) if it is set. All CDF files created after this static method call will be affected.

Parameters:

`flag` - The flag indicates whether to create a new CDF(s) in the backward version. BACKWARDFILEon means a backward file(s) is to be created and BACKWARDFILEoff means a V3.* file(s) is to be created.

Throws:

`CDFException` - if there was a problem

getFileBackward

```
public static boolean getFileBackward()
```

Gets the file backward flag.

Returns:

The flag indicating whether the CDF file was created in the older V2.7 version. It is only applicable for V3.* library. Returns true if backward files are to be created, false otherwise.

getFileBackwardEnvVar

```
public static int getFileBackwardEnvVar()
                                 throws CDFException
```

Gets the value of the CDF_FILEBACKWARD environment variable.

Returns:

1 if the environment variable is set to true, 0 if not set or set to anything else.

Throws:

`CDFException` - if there was a problem

getLeapSecondsTableEnvVar

```
public static java.lang.String getLeapSecondsTableEnvVar()
                                                  throws CDFException
```

Gets the the CDF_LEAPSECONDSTABLE (or CDF$LEAPSECONDSTABLE on VMS) environment variable.

Returns:

the string the environment variable is set to, null if not set

Throws:

`CDFException` - if there was a problem

getChecksumEnvVar

```
public static long getChecksumEnvVar()
                                throws CDFException
```

Gets the value of the CDF_CHECKSUM environment variable.

Returns:

1 if the environment variable is set to MD5, 0 if not set or set to anything else.

Throws:

`CDFException` - if there was a problem

setValidate

```
public static void setValidate(long mode)
                        throws CDFException
```

Sets the file validation mode so that when a CDF file is open, it will be validated accordingly. Setting this flag will overwrite environment varibale CDF_VALIDATE (or CDF$VALIDATE on OpenVMS) if it is set. If none is set, the default will have the files validated when open. All CDF files open after this static method call will be applied.

Parameters:

`mode` - The mode indicates whether to validate CDF(s) while open. **VALIDATEFILEon** means all files are subjected to ve validated. **VALIDATEFILEoff** means no data validation will be performed.

Throws:

`CDFException` - if there was a problem

getValidate

```
public static boolean getValidate()
```

Gets the file validation mode.

Returns:

The mode indicating whether the CDF file is to be validated when it is open. Returns true if it will be validated, false otherwise.

getStatus

```
public long getStatus()
```

Gets the status of the most recent CDF JNI/library function call. This value can be examined and appropriate action can be taken.

The following example sends a signal to the JNI code to write a single data to the current CDF. JNI in turn performs the requested

operation. It then checks to see whether the requested operation was successfully performed or not.

```
variable.putSingleData(recNum, dimIndicies, data);
long status = cdf.getStatus();
if (status != CDF_OK) {
    String statusText = CDF.getStatusText(status);
    System.out.println ("status = "+statusText);
}
```

Returns:

the status of the most recent CDF JNI/library function call

getStatusText

```
public static java.lang.String getStatusText(long statusCode)
```

Gets the status text of the most recent CDF JNI/library function call.

The following example shows how to obtain the text representation of the status code returned from the getStatus method:

```
long status = cdf.getStatus();
if (status != CDF_OK) {
    String statusText = CDF.getStatusText(status);
    System.out.println ("status = "+statusText);
}
```

Parameters:

`statusCode` - status code to be translated

Returns:

she string representation of the passed status code

setInfoWarningOff

```
public void setInfoWarningOff()
```

Sets the informational (status code < 0) or warning messages (status code between -1 and -2000) coming from the CDF JNI/library function off. This is the default when a file is opened or created.

setInfoWarningOn

```
public void setInfoWarningOn()
```

Sets the informational (status code < 0) or warning messages (status code between -1 and -2000) coming from the CDF JNI/library function on.

toString

```
public java.lang.String toString()
```

Gets the name of this CDF.

Overrides:

`toString` in class `java.lang.Object`

Returns:

the name of this CDF

finalize

```
public void finalize()
             throws java.lang.Throwable
```

Do the necessary cleanup when garbage collector reaps it.

Overrides:

`finalize` in class `java.lang.Object`

Throws:

`java.lang.Throwable` - if there was a problem doing cleanup

getDelegate

```
public CDFDelegate getDelegate()
```

This is a placeholder for future expansions/extensions.

Returns:

CDFDelegate object

setDelegate

```
public void setDelegate(CDFDelegate delegate)
```

This is a placeholder for future expansions/extensions.

Parameters:

`delegate` - The CDFDelegate executing the command

getAttributeID

```
public long getAttributeID(java.lang.String attrName)
```

Gets the id of the given attribute.

Parameters:

`attrName` - the name of the attribute to check

Returns:

the id of the named attribute if it exists, -1 otherwise

getAttribute

```
public Attribute getAttribute(long attrNum)
                       throws CDFException
```

Gets the attribute for the given attribute number.

Note: The attrNum may not necessarily correspond to the attribute number stored in the CDF file.

Parameters:

`attrNum` - the attribute number to get

Returns:

the Attribute object that corresponds to the requested attribute number

Throws:

`CDFException` - if the supplied attribute number does not exist

getAttribute

```
public Attribute getAttribute(java.lang.String attrName)
                       throws CDFException
```

Gets the attribute for the given attribute name.

The following example retrieves the attribute named "ValidMin":

```
Attribute validMin = cdf.getAttribute("ValidMin");
```

Parameters:

`attrName` - the name of the attribute to get

Returns:

the Attribute object that corresponds to the requested attribute name

Throws:

`CDFException` - if the supplied attribute name does not exist

getAttributes

```
public java.util.Vector getAttributes()
```

Gets all the global and variable attributes defined for this CDF. The following example retrieves all the global and variable attributes:

```
Vector attr = cdf.getAttributes();
```

Returns:

a vector that contains the global and variable attributes defined in this CDF

getGlobalAttributes

```
public java.util.Vector getGlobalAttributes()
```

Gets the global attributes defined for this CDF.

Returns:

A vector that contains the global attributes defined in this CDF

readCDFInfo

```
public java.util.Map readCDFInfo()
```

Gets the basic information about this CDF. The returned map has the following form: "CDFinfo": { "Format": "SINGLE", "Version": "3.8.0", "FileName": "test", "NumVars": 21, "Majority": "ROW" ... ... }

Returns:

A Map object that contains the basic information about the CDF.

readCDFGlobalAttributes

```
public java.util.Map readCDFGlobalAttributes()
```

Gets the global attributes and their entries defined for this CDF. Note: Each attribute's entry, an object, can be a single data or an array of values. The epoch data type into encoded date/time string.

Returns:

A Map object that contains the global attributes, as the keys, and entries, as the values. All are of object type. Each entry is also a map, which consists of entry number as the key and entry data as the value.

readCDFGlobalAttributes

```
public java.util.Map readCDFGlobalAttributes(boolean encoding)
```

Gets the global attributes and their entries defined for this CDF. Note: Each attribute's entry, an object, can be a single data or an array of values. The returned map has the following form: "GLOBAL_ATTR_NAME1": { "0": ..., "1": ... }, "GLOBAL_ATTR_NAME2": { "0": ..., "1": [4294967295, 2147483648] } "GLOBAL_ATTR_NAME3": { "0": ..., "1": ..., ... { ... ...

Parameters:

`encoding` - A true/false flag to indicate whether to convert epoch data type into encoded date/time string.

Returns:

A Map object that contains the global attributes, as the keys, and entries, as the values. All are of object type. Each entry is also a map, which consists of entry number as the key and entry data as the value.

readCDFGlobalAttributes

```
public java.util.Map readCDFGlobalAttributes(java.lang.String attrName)
```

Gets a specified global attribute's entry(ies) defined for this CDF. Note: Each attribute's entry, an object, can be a single data or an array of values. The epoch data type into encoded date/time string.

Parameters:

`attrName` - Aglobal attribute name

Returns:

A Map object that contains the global attributes, as the keys, and entries, as the values. All are of object type. Each entry is also a map, which consists of entry number as the key and entry data as the value.

readCDFGlobalAttributes

```
public java.util.Map readCDFGlobalAttributes(java.lang.String[] attrNames)
```

Gets a specified list of global attributes and their entries defined for this CDF. Note: Each attribute's entry, an object, can be a single data or an array of values. The epoch data type into encoded date/time string.

Parameters:

`attrNames` - An array of global attribute names

Returns:

A Map object that contains the global attributes, as the keys, and entries, as the values. All are of object type. Each entry is also a map, which consists of entry number as the key and entry data as the value.

readCDFGlobalAttributes

```
public java.util.Map readCDFGlobalAttributes(long attrNum)
```

Gets a specified global attribute's entry(ies) defined for this CDF. Note: Each attribute's entry, an object, can be a single data or an array of values. The epoch data type into encoded date/time string.

Parameters:

`attrNum` - A global attribute number

Returns:

A Map object that contains the global attributes, as the keys, and entries, as the values. All are of object type. Each entry is also a map, which consists of entry number as the key and entry data as the value.

readCDFGlobalAttributes

```
public java.util.Map readCDFGlobalAttributes(long[] attrNums)
```

Gets a specified list of global attributes and their entries defined for this CDF. Note: Each attribute's entry, an object, can be a single data or an array of values. The epoch data type into encoded date/time string.

Parameters:

`attrNums` - An array of global attribute numbers

Returns:

A Map object that contains the global attributes, as the keys, and entries, as the values. All are of object type. Each entry is also a map, which consists of entry number as the key and entry data as the value.

readCDFGlobalAttributes

```
public java.util.Map readCDFGlobalAttributes(java.lang.Object attrs,
                                             boolean encoding)
```

Gets a list of specified global attributes and their entries defined for this CDF. Note: Each attribute's entry, an object, can be a single data or an array of values.

Parameters:

`attrs` - An array of attribute names of numbers

`encoding` - A true/false flag to indicate whether to convert epoch data type into encoded date/time string.

Returns:

A Map object that contains the global attributes, as the keys, and entries, as the values. All are of object type. Each entry is also a map, which consists of entry number as the key and entry data as the value.

readCDFNoEntryAttributes

```
public java.util.Map readCDFNoEntryAttributes()
```

Gets a map of global attributes and variable attributes with no entry data.

Returns:

A Map object that contains up to itwo elements, one for global attributes and one for variable attributes. The value is the name(s) of the attributes() that has no entry data.

readCDFVariables

```
public java.util.Map readCDFVariables()
                              throws CDFException
```

Reads all variables' information in the CDF. The returned map has the following form: "Variables": { "Var_Name1": { "VarSpec": { "NumElements": 1, ... } "VarMetaData" : { "VALIDMIN": 0 ... } "VarData" : { ... } } "Var_Name2": { "VarSpec": { "NumElements": 1, ... } "VarMetaData" : { "VALIDMIN": 0 ... } "VarData" : { ... } } ... ... }

Returns:

A Map object that contains the basic variable information, its metadata and data from all variables

Throws:

`CDFException` - If there was a problem retrieving the variable information from this CDF file

readCDFVariablesSpec

```
public java.util.Map readCDFVariablesSpec()
                                  throws CDFException
```

Reads all variables' specifications in the CDF. The returned map has the following form: "Variables": { "Var_Name1": { "VarSpec": { "NumElements": 1, "PadValue": "0000-01-01T00:00:00.000", "DataType": "CDF_EPOCH", "NumDims": 0 ... ... } } "Var_Name2": { "VarSpec": { "NumElements": 1, "PadValue": "0000-01-01T00:00:00.000", "DataType": "CDF_EPOCH", "NumDims": 0 ... ... } } ... ... }

Returns:

A Map object that contains the basic information for all variables

Throws:

`CDFException` - If there was a problem retrieving the variable information from this CDF file

readCDFVariablesMetaData

```
public java.util.Map readCDFVariablesMetaData()
                                       throws CDFException
```

Reads all variables' metadata in the CDF. The returned map has the following form: "Variables": { "Var_Name1": { "VarMetaData": { "VALIDMIN": "2010-01-01T00:00:00.000", "VALIMAX": ... ... } } "Var_Name2": { "VarMetaData": { "VALIMIN": 1, "VALIMAX": 10, ... } } ... }

Returns:

A Map object that contains the metadata for all variables

Throws:

`CDFException` - If there was a problem retrieving the variable information from this CDF file

readCDFVariablesData

```
public java.util.Map readCDFVariablesData()
                                    throws CDFException
```

Reads all variables' data in the CDF. The returned map has the following form: "Variables": { "Var_Name1": { "VarData": { ... ... } } "Var_Name2": { "VarData": { ... ... } } ... }

Returns:

A Map object that contains the data from all variables

Throws:

`CDFException` - If there was a problem retrieving the variable information from this CDF file

readCDF

```
public java.util.Map readCDF()
```

Gets the full CDF information. Note: Each attribute's entry, an object, can be a single data or an array of values. CDF Epoch type data will be encoded in date/time string.

Returns:

A Map object that contains the CDF basic information, its global attributes and variables' specification, metadata and data.

readCDF

```
public java.util.Map readCDF(boolean encoding)
```

Gets the full CDF information. Note: Each attribute's entry, an object, can be a single data or an array of values. CDF Epoch type data will be encoded in date/time string.

Parameters:

`encoding` - A true/false flag indicates whether to encode epoch data type from metadata to date/time string.

Returns:

A Map object that contains the CDF basic information, its global attributes and variables' specification, metadata and data.

readCDF

```
public java.util.Map readCDF(boolean encoding,
                     boolean basic,
                     boolean globals,
                     boolean varinfo)
```

Gets the specified information from a CDF. Note: Each attribute's entry, an object, can be a single data or an array of values.

Parameters:

`encoding` - A true/false flag indicates whether to encode epoch data type from metadata to date/time string.

`basic` - A true/false flag indicates whether to read the basic information about the CDF

`globals` - A true/false flag indicates whether to read the global attributes in the CDF

`varinfo` - A true/false flag indicates whether to read each variable's information.

Returns:

A Map object that contains the CDF basic information, its global attributes and/or all variables' specification, metadata and data.

readCDF

```
public java.util.Map readCDF(boolean encoding,
                     boolean basic,
                     boolean globals,
                     boolean varinfo,
                     boolean varmeta,
                     boolean vardata)
```

Gets the specified information from a CDF. Note: Each attribute's entry, an object, can be a single data or an array of values.

Parameters:

`encoding` - A true/false flag indicates whether to encode epoch data type from metadata to date/time string.

`basic` - A true/false flag indicates whether to read the basic information about the CDF

`globals` - A true/false flag indicates whether to read the global attributes in the CDF

`varinfo` - A true/false flag indicates whether to read each variable's basic information.

`varmeta` - A true/false flag indicates whether to read each variable's metadata.

`vardata` - A true/false flag indicates whether to read each variable's data.

Returns:

A Map object that contains the CDF basic information, its global attributes and all variables' metadata and data.

readCDF

```
public java.util.Map readCDF(boolean encoding,
                     boolean basic,
                     boolean globals,
                     boolean varinfo,
                     boolean varmeta,
                     boolean vardata,
                     boolean noentry)
```

Gets the specified information from a CDF. Note: Each attribute's entry, an object, can be a single data or an array of values.

```
 Map acdf = cdf.readCDF(true, true, true, true, true, true);

 for (Object key : acdf.keySet()) {

   // May contain 3 key/value pairs: keys include "CDFinfo",
   // "GlobalAttributes" and "Variables".

   // Get key: "Variables"'s value -- a Map

   if (((String)key).equals("Variables")) {
     Map vars = (Map) acdf.get(key);
     if (vars == null) continue;
     for (Object varName : vars.keySet()) {
       // each key/value pair is variable name and its info
       System.out.println("** Variable: "+varName);
       Map var = (Map) vars.get(varName);
       // Each variable's info includes upto 3 key/value mappings:
       // "VarDef", "VarMetaData" and "VarData".
```

```
       for (Object key1 : var.keySet()) {
         if (key1.equals("VarSpec")) {
           // Key: "VarSpec" has Variable specification
           Map specs = var.get(key1);
           for (Object key2 : specs.keySet()) {
             ....
             ....
           }
         } else if (key1.equals("VarMetaData")) {
           // Key: "VarMeataData" has variable metadata
           Map meta = var.get(key1);
           for (Object key2 : meta.keySet()) {
             ....
             ....
           }
         } else if ((key1.equals("VarData")) {
           // Key: "VarData" has variable data
           Object data = var.get(key1);
           ....
           ....
         } else {
           error....
         }
       }
     }
   } else { // other keys: "CDFinfo", "GloblaAttributes" and their values
     System.out.println(key + " - \n   " + acdf.get(key));
   }
 }
```

Parameters:

`encoding` - A true/false flag indicates whether to encode epoch data type from metadata to date/time string.

`basic` - A true/false flag indicates whether to read the basic information about the CDF

`globals` - A true/false flag indicates whether to read the global attributes in the CDF

`varinfo` - A true/false flag indicates whether to read each variable's basic information.

`varmeta` - A true/false flag indicates whether to read each variable's metadata.

`vardata` - A true/false flag indicates whether to read each variable's data.

`noentry` - A true/false flag indicates whether to collect attributes that have no entries

Returns:

A Map object that contains the CDF basic information, its global attributes and all variables' metadata and data.

getVariableAttributes

`public java.util.Vector getVariableAttributes()`

Gets the variable attributes defined for this CDF.

Returns:

A vector that contains the variable attributes defined in this CDF

getOrphanAttributes

`public java.util.Vector getOrphanAttributes()`

Gets the variable attributes defined for this CDF that are not associated with any variables.

Returns:

A vector that contains the empty variable attributes defined in this CDF.

getVariableID

```
public long getVariableID(java.lang.String varName)
```

Gets the ID of the given variable.

Parameters:

`varName` - the name of the variable to check

Returns:

-1 if the variable does not exist. The variable id if the variable does exist.

getVariable

```
public Variable getVariable(long varNum)
                     throws CDFException
```

Gets the variable object for the given variable number.

Parameters:

`varNum` - variable number from which the variable is retrieved

Returns:

the variable object that corresponds to the variable id

Throws:

`CDFException` - if the supplied variable number does not exist

getVariable

```
public Variable getVariable(java.lang.String varName)
                     throws CDFException
```

Gets the variable object for the given variable name.

The following example retrieves a variable called "Longitude":

```
     Variable longitude = cdf.getVariable("Longitude");
```

Parameters:

`varName` - the variable name to get

Returns:

the variable object that corresponds to the variable name

Throws:

`CDFException` - if the supplied variable name does not exist

getVariables

```
public java.util.Vector getVariables()
```

Gets the z variables defined for this CDF.

Note: Since all CDFs opened or created with the CDFJava APIs are placed into zMODE 2, there are no rVarialbles. All variables are treated as zVariables.

Returns:

a Vector containing all the z variables defined in this CDF

getNumVars

```
public long getNumVars()
```

Gets the number of Z variables defined for this CDF.

Returns:

the numer of variables **Note:** Since all CDFs opened or create with the CDFJava APIs are placed into zMODE 2, there are no rVarialbles. All variables are treated as zVariables.

getRecord

```
public java.util.Vector getRecord(long recNum,
                         java.lang.String[] strVars)
                           throws CDFException
```

Retrieves a logical record that consists of single variable record(s) from an arbitrary number of CDF variables. This is a convenient method for reading one or more variables' data in a single call, instead of reading individual variable's data one at a time.

Parameters:

`recNum` - the record number to retrieve data from

`strVars` - the variable (array of variable names) to retrieve data from

Returns:

the requested record in a Java vector that contains the variables' data.
The first object in the vector corresponds to the first variable's record, the second object in the vector corresponds to the second variable's record, and so on.

Throws:

`CDFException` - if there was a problem getting a record

Note: A virtual variable record is returned if the given record does not exist. Any error during data retrieval will cause the process to stop (an exception thrown) and thus nothing (a null object) will be returned.

getRecord

```
public java.util.Vector getRecord(long recNum,
                         java.lang.String[] strVars,
                         long[] status)
                           throws CDFException
```

Retrieves a logical record that consists of single variable record(s) from an arbitrary number of CDF variables. This is a convenient method for reading one or more variables' data in a single call, instead of reading individual variable's data one at a time.

Parameters:

`recNum` - the record number to retrieve data from

`strVars` - the variable (array of variable names) to retrieve data from

`status` - the individual status (array of statuses) for reading each variable record

Returns:

the requested record in a Java vector that contains the variables' data.
The first object in the vector corresponds to the first variable's record, the second object in the vector corresponds to the second variable's record, and so on.

Throws:

`CDFException` - if there was a problem getting a record

Note: A virtual variable record is returned if the given record does not exist. Any error during data retrieval will cause the process to stop (an exception thrown) and thus nothing (a null object) will be returned.

The following example reads the 2nd record from Longitude and Temperature and prints their contents.

```
String[] strVars = {"Longitude", "Temperature"};
Vector record;
long[] status = new long[2];
record = cdf.getRecord(1L, strVars, status);

// Check the contents of the 'status' array - optional

// var: Longitude - data type: CDF_UINT2, dimensionality: 1:[3]
System.out.print ("    2nd record of Longitude -- ");
for (int i=0; i < 3; i++)
     System.out.print (((int[])record.elementAt(0))[i]+" ");
System.out.println ("");

// var: Temperature -- data type: CDF_REAL4, dimensionality: 1:[3]
System.out.print ("    2nd record of Temperature -- ");
for (int i=0; i < 3; i++)
     System.out.print (((float[])record.elementAt(1))[i]+" ");
System.out.println ("");
```

getRecord

```
public java.util.Vector getRecord(long recNum,
                          long[] varIDs)
                            throws CDFException
```

Retrieves a logical record that consists of single variable record(s) from an arbitrary number of CDF variables. This is a convenient method for reading one or more variables' data in a single call, instead of reading individual variable's data one at a time.

Parameters:

`recNum` - the record number to retrieve data from

`varIDs` - the variable IDs (array of variable IDs) to retrieve data from

Returns:

the requested record in a Java vector that contains the variables' data.
The first object in the vector corresponds to the first variable's record, the second object in the vector corresponds to the second variable's record, and so on.

Throws:

`CDFException` - if there was a problem getting a record

Note: A virtual variable record is returned if the given record does not exist. Any error during data retrieval will cause the process to stop (an exception thrown) and thus nothing (a null object) will be returned.

getRecord

```
public java.util.Vector getRecord(long recNum,
                        long[] varIDs,
                        long[] status)
                          throws CDFException
```

Retrieves a logical record that consists of single variable record(s) from an arbitrary number of CDF variables. This is a convenient method for reading one or more variables' data in a single call, instead of reading individual variable's data one at a time.

Parameters:

`recNum` - the record number to retrieve data from

`varIDs` - the variable IDs (array of variable IDs) to retrieve data from

`status` - the individual status (array of statuses) for reading each variable record

Returns:

the requested record in a Java vector that contains the variables' data.
The first object in the vector corresponds to the first variable's record, the second object in the vector corresponds to the second variable's record, and so on.

Throws:

`CDFException` - if there was a problem getting a record

Note: A virtual variable record is returned if the given record does not exist. Any error during data retrieval will cause the process to stop (an exception thrown) and thus nothing (a null object) will be returned.

The following example reads the 2nd record from Longitude (varIds[0]) and Temperature (varIDs[1]) and prints their contents.

```
long[] varIDs = {2, 10};    // Obtained from Variable.getID()
Vector record;
long[] status = new long[2];
record = cdf.getRecord(1L, varIDs, status);

// Check the contents of the 'status' array - optional

// var: Longitude - data type: CDF_UINT2, dimensionality: 1:[3]
System.out.print ("    2nd record of Longitude -- ");
for (int i=0; i < 3; i++)
     System.out.print (((int[])record.elementAt(0))[i]+" ");
System.out.println ("");

// var: Temperature - data type: CDF_REAL4, dimensionality: 1:[3]
System.out.print ("    2nd record of Temperature -- ");
for (int i=0; i < 3; i++)
     System.out.print (((float[])record.elementAt(1))[i]+" ");
System.out.println ("");
```

putRecord

```
public void putRecord(long recNum,
             java.lang.String[] strVars,
             java.util.Vector myData)
               throws CDFException
```

Writes a logical record that consists of single variable record(s) from an arbitrary number of CDF variables. This is a convenient method for writing one or more variables' data in a single call, instead of writing individual variable's data one at a time.

Parameters:

`recNum` - the record number to write data to

`strVars` - the variable (array of variable names) to write data to

`myData` - a Java vector that contains the variables' data.

The first object in the vector corresponds to the first variable's record, the second object in the vector corresponds to the second variable's record, and so on.

Throws:

`CDFException` - if there was a problem writing the record for any of the variables

Note: Any error during the data writing will cause the process to stop (an exception thrown) and thus the operation will not be completed. Nothing will be done if the element counts of parameters don't match.

putRecord

```
public void putRecord(long recNum,
             java.lang.String[] strVars,
             java.util.Vector myData,
             long[] status)
               throws CDFException
```

Writes a logical record that consists of single variable record(s) from an arbitrary number of CDF variables. This is a convenient method for writing one or more variables' data in a single call, instead of writing individual variable's data one at a time.

Parameters:

`recNum` - the record number to write data to

`strVars` - the variable (array of variable names) to write data to

`myData` - a Java vector that contains the variables' data.
The first object in the vector corresponds to the first variable's record, the second object in the vector corresponds to the second variable's record, and so on.

`status` - the individual status (array of statuses) for writing each variable record

Throws:

`CDFException` - if there was a problem writing the record for any of the variables

Note: Any error during the data writing will cause the process to stop (an exception thrown) and thus the operation will not be completed. Nothing will be done if the element counts of parameters don't match.

The following example writes the contents of a record (which consists of two CDF variables - Longitude and Temperature) to record number 2.

```
 String[] strVars = {"Longitude",     // variable names in CDF
                     "Temperature"};

// Longitude -- data type: CDF_UINT2 dimensionality: 1:[3]
int[] longitude_data = {333, 444, 555};

// Temperature -- data type: CDF_FLOAT dimensionality: 0:[]
Float temperature_data = new Float((float)999.99);

Vector record = new Vector();
record.add(longitude_data);
record.add(temperature_data);

cdf.putRecord(1L, strVars, record);  // Write a record to record #2
```

putRecord

```
public void putRecord(long recNum,
             long[] varIDs,
             java.util.Vector myData)
               throws CDFException
```

Writes a logical record that consists of single variable record(s) from an arbitrary number of CDF variables. This is a convenient

method for writing one or more variables' data in a single call, instead of writing individual variable's data one at a time.

Parameters:

`recNum` - the record number to write data to

`varIDs` - the variable IDs (array of variable IDs) to write data to

`myData` - a Java vector that contains the variables' data.
The first object in the vector corresponds to the first variable's record, the second object in the vector corresponds to the second variable's record, and so on.

Throws:

`CDFException` - if there was a problem writing the record for any of the variables

Note: Any error during the data writing will cause the process to stop (an exception thrown) and thus the operation will not be completed. Nothing will be done if the element counts of parameters don't match.

putRecord

```
public void putRecord(long recNum,
             long[] varIDs,
             java.util.Vector myData,
             long[] status)
               throws CDFException
```

Writes a logical record that consists of single variable record(s) from an arbitrary number of CDF variables. This is a convenient method for writing one or more variables' data in a single call, instead of writing individual variable's data one at a time.

Parameters:

`recNum` - the record number to write data to

`varIDs` - the variable IDs (array of variable IDs) to write data to

`myData` - a Java vector that contains the variables' data.
The first object in the vector corresponds to the first variable's record, the second object in the vector corresponds to the second variable's record, and so on.

`status` - the individual status (array of statuses) for writing each variable record

Throws:

`CDFException` - if there was a problem writing the record for any of the variables

Note: Any error during the data writing will cause the process to stop (an exception thrown) and thus the operation will not be completed. Nothing will be done if the element counts of parameters don't match.

The following example writes the contents of a record (which consists of two CDF variables - Longitude and Temperature) by using variable IDs (instead of variable names) to record number 2.

```
long[] varIDs = {3, 9};   // Can be obtained from variable.getID()

// Longitude -- data type: CDF_UINT2 dimensionality: 1:[3]
int[] longitude_data = {333, 444, 555};

// Temperature -- data type: CDF_FLOAT dimensionality: 0:[]
Float temperature_data = new Float((float)999.99);

Vector record = new Vector();
record.add(longitude_data);
record.add(temperature_data);

cdf.putRecord(1L, varIDs, record);  // Write a record to record #2
```

setChecksum

```
public void setChecksum(long checksum)
                 throws CDFException
```

Specifies the checksum option applied to the CDF.

Parameters:

`checksum` - the checksum option to be used for this CDF. Currently, other than NO_CHECKSUM option, only MD5_CHECKSUM (using MD5 checksum algorithm) is supported.

Throws:

`CDFException` - if there was a problem with the passed option, setting the checksum or other vital infomation from this CDF file.

getChecksum

```
public long getChecksum()
```

Gets the checksum method, if any, applied to the CDF.

Returns:

the checksum method used for this CDF. Currently, it returns NONE_CHECKSUM (0) if no checksum is used; MD5_CHECKSUM (1) if MD5 method is used;

setLeapSecondLastUpdated

```
public void setLeapSecondLastUpdated(long lastUpdated)
                              throws CDFException
```

Set the leap second last updated for this CDF. Normally, a new CDF is created, this field is set automatically. Only to call this method if the CDF is based on an existing one.

Parameters:

`lastUpdated` - The last date that the leap second was updated. It is in YYYYMMDD form.

Throws:

`CDFException` - If a problem occurred in setting the desired field.

getLeapSecondLastUpdated

```
public long getLeapSecondLastUpdated()
```

Gets the date that the CDF has the last leap second updated.

Returns:

the date in YYYYMMDD that the last leap second was updated when the CDF is based (relavant to TT2000 variables).

verifyChecksum

```
public long verifyChecksum()
                   throws CDFException
```

Verifies the data integrity of the CDF file from its checksum.

Returns:

The status of data integrity check through its checksum. it should return CDF_OK if the integrity check is fine. Or, it may return a value of CHECKSUM_ERROR indicating the data integrity was compromised. Or, it may return other CDF error if it has problem reading the CDF data filed(s). No need to use this method as when the file is open, its data integrity is automatically checked with the used checksum method.

Throws:

`CDFException` - if there was a problem getting the checksum or other vital infomation from this CDF file

gsfc.nssdc.cdf

Interface CDFConstants

All Known Implementing Classes:

Attribute, CDF, CDFData, CDFException, CDFTools, CDFTT2000, CDFUtils, Entry, Epoch, Epoch16, Variable

---

`public interface` **`CDFConstants`**

This class defines the constants used by the CDF library and CDF Java APIs, and it mimics the cdf.h include file from the cdf distribution.

Version:

1.0

Field Summary

Fields

Modifier and Type	Field and Description
static long	AHUFF_COMPRESSION
static long	ALPHAOSF1_DECODING
static long	ALPHAOSF1_ENCODING
static long	ALPHAVMSd_DECODING
static long	ALPHAVMSd_ENCODING
static long	ALPHAVMSg_DECODING
static long	ALPHAVMSg_ENCODING
static long	ALPHAVMSi_DECODING
static long	ALPHAVMSi_ENCODING
static long	ARM_BIG_DECODING
static long	ARM_BIG_ENCODING
static long	ARM_LITTLE_DECODING
static long	ARM_LITTLE_ENCODING
static long	ATTR_
static long	ATTR_EXISTENCE_
static long	ATTR_EXISTS
static long	ATTR_MAXgENTRY_
static long	ATTR_MAXrENTRY_
static long	ATTR_MAXzENTRY_
static long	ATTR_NAME_
static long	ATTR_NAME_TRUNC
static long	ATTR_NUMBER_

static long	ATTR_NUMgENTRIES_
static long	ATTR_NUMrENTRIES_
static long	ATTR_NUMzENTRIES_
static long	ATTR_SCOPE_
static long	BACKWARD_
static long	BACKWARDFILEoff
static long	BACKWARDFILEon
static long	BAD_ALLOCATE_RECS
static long	BAD_ARGUMENT
static long	BAD_ATTR_NAME
static long	BAD_ATTR_NUM
static long	BAD_BLOCKING_FACTOR
static long	BAD_CACHE_SIZE
static long	BAD_CDF_EXTENSION
static long	BAD_CDF_ID
static long	BAD_CDF_NAME
static long	BAD_CDFSTATUS
static long	BAD_CHECKSUM
static long	BAD_COMPRESSION_PARM
static long	BAD_DATA_TYPE
static long	BAD_DECODING
static long	BAD_DIM_COUNT
static long	BAD_DIM_INDEX
static long	BAD_DIM_INTERVAL
static long	BAD_DIM_SIZE
static long	BAD_ENCODING
static long	BAD_ENTRY_NUM
static long	BAD_FNC_OR_ITEM
static long	BAD_FORMAT
static long	BAD_INITIAL_RECS
static long	BAD_MAJORITY
static long	BAD_MALLOC
static long	BAD_NEGtoPOSfp0_MODE
static long	BAD_NUM_DIMS
static long	BAD_NUM_ELEMS
static long	BAD_NUM_STRINGS
static long	BAD_NUM_VARS
static long	BAD_READONLY_MODE
static long	BAD_REC_COUNT
static long	BAD_REC_INTERVAL
static long	BAD_REC_NUM
static long	BAD_SCOPE
static long	BAD_SCRATCH_DIR
static long	BAD_SPARSEARRAYS_PARM

static long	BAD_VAR_NAME
static long	BAD_VAR_NUM
static long	BAD_zMODE
static long	BADDATE_LEAPSECOND_UPDATED
static double	BeginUnixTimeEPOCH
static double	BeginUnixTimeEPOCH16
static long	BLOCKINGFACTOR_TOO_LARGE
static long	BLOCKINGFACTOR_TOO_SMALL
static long	BLOCKINGFACTOR_TOO_SMALL2
static long	CANNOT_ALLOCATE_RECORDS
static long	CANNOT_CHANGE
static long	CANNOT_COMPRESS
static long	CANNOT_CONVERT_WIDECHAR
static long	CANNOT_COPY
static long	CANNOT_INSERT_RECORDS
static long	CANNOT_SPARSEARRAYS
static long	CANNOT_SPARSERECORDS
static long	CDF_
static long	CDF_ACCESS_
static long	CDF_ATTR_NAME_LEN
static long	CDF_ATTR_NAME_LEN256
static long	CDF_BYTE
static long	CDF_CACHESIZE_
static long	CDF_CHAR
static long	CDF_CHECKSUM_
static long	CDF_CLOSE_ERROR
static long	CDF_COMPRESSION_
static long	CDF_COPYRIGHT_
static long	CDF_COPYRIGHT_LEN
static long	CDF_CREATE_ERROR
static long	CDF_DECODING_
static long	CDF_DELETE_ERROR
static long	CDF_DOUBLE
static long	CDF_ENCODING_
static long	CDF_EPOCH
static long	CDF_EPOCH16
static long	CDF_EXISTS
static long	CDF_FLOAT
static long	CDF_FORMAT_
static long	CDF_INCREMENT_
static long	CDF_INFO_
static long	CDF_INT1
static long	CDF_INT2
static long	CDF_INT4

static long	CDF_INT8
static long	CDF_INTERNAL_ERROR
static long	CDF_LEAPSECONDLASTUPDATED_
static long	CDF_MAJORITY_
static long	CDF_MAX_DIMS
static long	CDF_MAX_PARMS
static long	CDF_MIN_DIMS
static long	CDF_NAME_
static long	CDF_NAME_TRUNC
static long	CDF_NEGtoPOSfp0_MODE_
static long	CDF_NUMATTRS_
static long	CDF_NUMgATTRS_
static long	CDF_NUMrVARS_
static long	CDF_NUMvATTRS_
static long	CDF_NUMzVARS_
static long	CDF_OK
static long	CDF_OPEN_ERROR
static long	CDF_PATHNAME_LEN
static long	CDF_READ_ERROR
static long	CDF_READONLY_MODE_
static long	CDF_REAL4
static long	CDF_REAL8
static long	CDF_RELEASE_
static long	CDF_SAVE_ERROR
static long	CDF_SCRATCHDIR_
static long	CDF_STATUS_
static long	CDF_STATUSTEXT_LEN
static long	CDF_TIME_TT2000
static long	CDF_UCHAR
static long	CDF_UINT1
static long	CDF_UINT2
static long	CDF_UINT4
static long	CDF_VAR_NAME_LEN
static long	CDF_VAR_NAME_LEN256
static long	CDF_VERSION_
static long	CDF_WARN
static long	CDF_WRITE_ERROR
static long	CDF_zMODE_
static long	CDFwithSTATS_
static long	CHECKSUM_
static long	CHECKSUM_ERROR
static long	CHECKSUM_NOT_ALLOWED
static long	CLOSE_
static long	COLUMN_MAJOR

static long	COMPRESS_CACHESIZE_
static long	COMPRESSION_ERROR
static long	CONFIRM_
static long	CORRUPTED_V2_CDF
static long	CORRUPTED_V3_CDF
static long	CREATE_
static long	CURgENTRY_EXISTENCE_
static long	CURrENTRY_EXISTENCE_
static long	CURzENTRY_EXISTENCE_
static long	DATATYPE_MISMATCH
static long	DATATYPE_SIZE_
static long	DECOMPRESSION_ERROR
static long	DECSTATION_DECODING
static long	DECSTATION_ENCODING
static byte	DEFAULT_BYTE_PADVALUE
static char	DEFAULT_CHAR_PADVALUE
static double	DEFAULT_DOUBLE_PADVALUE
static double	DEFAULT_EPOCH_PADVALUE
static double	DEFAULT_EPOCH16_PADVALUE
static float	DEFAULT_FLOAT_PADVALUE
static byte	DEFAULT_INT1_PADVALUE
static short	DEFAULT_INT2_PADVALUE
static int	DEFAULT_INT4_PADVALUE
static long	DEFAULT_INT8_PADVALUE
static float	DEFAULT_REAL4_PADVALUE
static double	DEFAULT_REAL8_PADVALUE
static long	DEFAULT_TT2000_PADVALUE
static char	DEFAULT_UCHAR_PADVALUE
static short	DEFAULT_UINT1_PADVALUE
static int	DEFAULT_UINT2_PADVALUE
static long	DEFAULT_UINT4_PADVALUE
static long	DELETE_
static long	DID_NOT_COMPRESS
static long	EMPTY_COMPRESSED_CDF
static long	END_OF_VAR
static long	EPOCH_STRING_LEN
static long	EPOCH_STRING_LEN_EXTEND
static long	EPOCH1_STRING_LEN
static long	EPOCH1_STRING_LEN_EXTEND
static long	EPOCH2_STRING_LEN
static long	EPOCH2_STRING_LEN_EXTEND
static long	EPOCH3_STRING_LEN
static long	EPOCH3_STRING_LEN_EXTEND
static long	EPOCH4_STRING_LEN

static long	EPOCH4_STRING_LEN_EXTEND
static long	EPOCHx_FORMAT_MAX
static long	EPOCHx_STRING_MAX
static long	FILLED_TT2000_VALUE
static long	FORCED_PARAMETER
static long	FUNCTION_NOT_SUPPORTED
static long	gENTRY_
static long	gENTRY_DATA_
static long	gENTRY_DATASPEC_
static long	gENTRY_DATATYPE_
static long	gENTRY_EXISTENCE_
static long	gENTRY_NUMELEMS_
static long	GET_
static long	GETCDFCHECKSUM_
static long	GETCDFFILEBACKWARD_
static long	GETCDFVALIDATE_
static long	GETLEAPSECONDSENVVAR_
static long	GLOBAL_SCOPE
static long	GZIP_COMPRESSION
static long	HOST_DECODING
static long	HOST_ENCODING
static long	HP_DECODING
static long	HP_ENCODING
static long	HUFF_COMPRESSION
static long	IA64VMSd_DECODING
static long	IA64VMSd_ENCODING
static long	IA64VMSg_DECODING
static long	IA64VMSg_ENCODING
static long	IA64VMSi_DECODING
static long	IA64VMSi_ENCODING
static long	IBM_PC_OVERFLOW
static long	IBMPC_DECODING
static long	IBMPC_ENCODING
static long	IBMRS_DECODING
static long	IBMRS_ENCODING
static long	ILLEGAL_EPOCH_FIELD
static double	ILLEGAL_EPOCH_VALUE
static long	ILLEGAL_FOR_SCOPE
static long	ILLEGAL_IN_zMODE
static long	ILLEGAL_ON_V1_CDF
static long	ILLEGAL_TT2000_VALUE
static long	IS_A_NETCDF
static long	LIB_COPYRIGHT_
static long	LIB_INCREMENT_

static long	LIB_RELEASE_
static long	LIB_subINCREMENT_
static long	LIB_VERSION_
static long	MAC_DECODING
static long	MAC_ENCODING
static long	MD5_CHECKSUM
static long	MULTI_FILE
static long	MULTI_FILE_FORMAT
static long	NA_FOR_VARIABLE
static long	NEGATIVE_FP_ZERO
static long	NEGtoPOSfp0off
static long	NEGtoPOSfp0on
static long	NETWORK_DECODING
static long	NETWORK_ENCODING
static long	NeXT_DECODING
static long	NeXT_ENCODING
static long	NO_ATTR_SELECTED
static long	NO_CDF_SELECTED
static long	NO_CHECKSUM
static long	NO_COMPRESSION
static long	NO_DELETE_ACCESS
static long	NO_ENTRY_SELECTED
static long	NO_MORE_ACCESS
static long	NO_PADVALUE_SPECIFIED
static long	NO_SPARSEARRAYS
static long	NO_SPARSERECORDS
static long	NO_STATUS_SELECTED
static long	NO_SUCH_ATTR
static long	NO_SUCH_CDF
static long	NO_SUCH_ENTRY
static long	NO_SUCH_RECORD
static long	NO_SUCH_VAR
static long	NO_VAR_SELECTED
static long	NO_VARS_IN_CDF
static long	NO_WRITE_ACCESS
static long	NONE_CHECKSUM
static long	NOT_A_CDF
static long	NOT_A_CDF_OR_NOT_SUPPORTED
static long	NOVARY
static long	NULL_
static long	OPEN_
static long	OPTIMAL_ENCODING_TREES
static long	OTHER_CHECKSUM
static long	PAD_SPARSERECORDS

static long	PPC_DECODING
static long	PPC_ENCODING
static long	PRECEEDING_RECORDS_ALLOCATED
static long	PREV_SPARSERECORDS
static long	PUT_
static long	READ_ONLY_DISTRIBUTION
static long	READ_ONLY_MODE
static long	READONLYoff
static long	READONLYon
static long	rENTRY_
static long	rENTRY_DATA_
static long	rENTRY_DATASPEC_
static long	rENTRY_DATATYPE_
static long	rENTRY_EXISTENCE_
static long	rENTRY_NAME_
static long	rENTRY_NUMELEMS_
static long	rENTRY_NUMSTRINGS_
static long	rENTRY_STRINGSDATA_
static long	RLE_COMPRESSION
static long	RLE_OF_ZEROs
static long	ROW_MAJOR
static long	rVAR_
static long	rVAR_ALLOCATEBLOCK_
static long	rVAR_ALLOCATEDFROM_
static long	rVAR_ALLOCATEDTO_
static long	rVAR_ALLOCATERECS_
static long	rVAR_BLOCKINGFACTOR_
static long	rVAR_CACHESIZE_
static long	rVAR_COMPRESSION_
static long	rVAR_DATA_
static long	rVAR_DATASPEC_
static long	rVAR_DATATYPE_
static long	rVAR_DIMVARYS_
static long	rVAR_EXISTENCE_
static long	rVAR_HYPERDATA_
static long	rVAR_INITIALRECS_
static long	rVAR_MAXallocREC_
static long	rVAR_MAXREC_
static long	rVAR_NAME_
static long	rVAR_nINDEXENTRIES_
static long	rVAR_nINDEXLEVELS_
static long	rVAR_nINDEXRECORDS_
static long	rVAR_NUMallocRECS_
static long	rVAR_NUMBER_

static long	rVAR_NUMELEMS_
static long	rVAR_NUMRECS_
static long	rVAR_PADVALUE_
static long	rVAR_RECORDS_
static long	rVAR_RECORDS_RENUMBER_
static long	rVAR_RECVARY_
static long	rVAR_RESERVEPERCENT_
static long	rVAR_SEQDATA_
static long	rVAR_SEQPOS_
static long	rVAR_SPARSEARRAYS_
static long	rVAR_SPARSERECORDS_
static long	rVARs_CACHESIZE_
static long	rVARs_DIMCOUNTS_
static long	rVARs_DIMINDICES_
static long	rVARs_DIMINTERVALS_
static long	rVARs_DIMSIZES_
static long	rVARs_MAXREC_
static long	rVARs_NUMDIMS_
static long	rVARs_RECCOUNT_
static long	rVARs_RECDATA_
static long	rVARs_RECINTERVAL_
static long	rVARs_RECNUMBER_
static long	SAVE_
static long	SCRATCH_CREATE_ERROR
static long	SCRATCH_DELETE_ERROR
static long	SCRATCH_READ_ERROR
static long	SCRATCH_WRITE_ERROR
static long	SELECT_
static long	SGi_DECODING
static long	SGi_ENCODING
static long	SINGLE_FILE
static long	SINGLE_FILE_FORMAT
static long	SOME_ALREADY_ALLOCATED
static long	STAGE_CACHESIZE_
static long	STATUS_TEXT_
static long	STRING_NOT_UTF8_ENCODING
static java.lang.String	STRINGDELIMITER
static long	SUN_DECODING
static long	SUN_ENCODING
static long	TOO_MANY_PARMS
static long	TOO_MANY_VARS
static long	TRY_TO_READ_NONSTRING_DATA
static long	TT2000_0_STRING_LEN
static long	TT2000_1_STRING_LEN

static long	TT2000_2_STRING_LEN
static long	TT2000_3_STRING_LEN
static long	TT2000_4_STRING_LEN
static long	TT2000_CDF_MAYNEEDUPDATE
static long	TT2000_TIME_ERROR
static long	TT2000_USED_OUTDATED_TABLE
static long	UNABLE_TO_PROCESS_CDF
static long	UNKNOWN_COMPRESSION
static long	UNKNOWN_SPARSENESS
static long	UNSUPPORTED_OPERATION
static long	VALIDATE_
static long	VALIDATEFILEoff
static long	VALIDATEFILEon
static long	VAR_ALREADY_CLOSED
static long	VAR_CLOSE_ERROR
static long	VAR_CREATE_ERROR
static long	VAR_DELETE_ERROR
static long	VAR_EXISTS
static long	VAR_NAME_TRUNC
static long	VAR_OPEN_ERROR
static long	VAR_READ_ERROR
static long	VAR_SAVE_ERROR
static long	VAR_WRITE_ERROR
static long	VARIABLE_SCOPE
static long	VARY
static long	VAX_DECODING
static long	VAX_ENCODING
static long	VIRTUAL_RECORD_DATA
static long	zENTRY_
static long	zENTRY_DATA_
static long	zENTRY_DATASPEC_
static long	zENTRY_DATATYPE_
static long	zENTRY_EXISTENCE_
static long	zENTRY_NAME_
static long	zENTRY_NUMELEMS_
static long	zENTRY_NUMSTRINGS_
static long	zENTRY_STRINGSDATA_
static long	ZLIB_COMPRESS_ERROR
static long	ZLIB_UNCOMPRESS_ERROR
static long	zMODEoff
static long	zMODEon1
static long	zMODEon2
static long	zVAR_
static long	zVAR_ALLOCATEBLOCK_

static long	zVAR_ALLOCATEDFROM_
static long	zVAR_ALLOCATEDTO_
static long	zVAR_ALLOCATERECS_
static long	zVAR_BLOCKINGFACTOR_
static long	zVAR_CACHESIZE_
static long	zVAR_COMPRESSION_
static long	zVAR_DATA_
static long	zVAR_DATASPEC_
static long	zVAR_DATATYPE_
static long	zVAR_DIMCOUNTS_
static long	zVAR_DIMINDICES_
static long	zVAR_DIMINTERVALS_
static long	zVAR_DIMSIZES_
static long	zVAR_DIMVARYS_
static long	zVAR_EXISTENCE_
static long	zVAR_HYPERDATA_
static long	zVAR_INITIALRECS_
static long	zVAR_MAXallocREC_
static long	zVAR_MAXREC_
static long	zVAR_NAME_
static long	zVAR_nINDEXENTRIES_
static long	zVAR_nINDEXLEVELS_
static long	zVAR_nINDEXRECORDS_
static long	zVAR_NUMallocRECS_
static long	zVAR_NUMBER_
static long	zVAR_NUMDIMS_
static long	zVAR_NUMELEMS_
static long	zVAR_NUMRECS_
static long	zVAR_PADVALUE_
static long	zVAR_RECCOUNT_
static long	zVAR_RECINTERVAL_
static long	zVAR_RECNUMBER_
static long	zVAR_RECORDS_
static long	zVAR_RECORDS_RENUMBER_
static long	zVAR_RECVARY_
static long	zVAR_RESERVEPERCENT_
static long	zVAR_SEQDATA_
static long	zVAR_SEQPOS_
static long	zVAR_SPARSEARRAYS_
static long	zVAR_SPARSERECORDS_
static long	zVARs_CACHESIZE_
static long	zVARs_MAXREC_
static long	zVARs_RECDATA_
static long	zVARs_RECNUMBER_

Field Detail

CDF_MIN_DIMS

```
static final long CDF_MIN_DIMS
```

See Also:

Constant Field Values

CDF_MAX_DIMS

```
static final long CDF_MAX_DIMS
```

See Also:

Constant Field Values

CDF_VAR_NAME_LEN

```
static final long CDF_VAR_NAME_LEN
```

See Also:

Constant Field Values

CDF_VAR_NAME_LEN256

```
static final long CDF_VAR_NAME_LEN256
```

See Also:

Constant Field Values

CDF_ATTR_NAME_LEN

```
static final long CDF_ATTR_NAME_LEN
```

See Also:

Constant Field Values

CDF_ATTR_NAME_LEN256

```
static final long CDF_ATTR_NAME_LEN256
```

See Also:

Constant Field Values

CDF_COPYRIGHT_LEN

```
static final long CDF_COPYRIGHT_LEN
```

See Also:

Constant Field Values

CDF_STATUSTEXT_LEN

```
static final long CDF_STATUSTEXT_LEN
```

See Also:

Constant Field Values

CDF_PATHNAME_LEN

```
static final long CDF_PATHNAME_LEN
```

See Also:

Constant Field Values

EPOCH_STRING_LEN

```
static final long EPOCH_STRING_LEN
```

See Also:

Constant Field Values

EPOCH1_STRING_LEN

```
static final long EPOCH1_STRING_LEN
```

See Also:

Constant Field Values

EPOCH2_STRING_LEN

```
static final long EPOCH2_STRING_LEN
```

See Also:

Constant Field Values

EPOCH3_STRING_LEN

```
static final long EPOCH3_STRING_LEN
```

See Also:

Constant Field Values

EPOCH4_STRING_LEN

```
static final long EPOCH4_STRING_LEN
```

See Also:

Constant Field Values

EPOCHx_STRING_MAX

```
static final long EPOCHx_STRING_MAX
```

See Also:

Constant Field Values

EPOCHx_FORMAT_MAX

```
static final long EPOCHx_FORMAT_MAX
```

See Also:

Constant Field Values

EPOCH_STRING_LEN_EXTEND

```
static final long EPOCH_STRING_LEN_EXTEND
```

See Also:

Constant Field Values

EPOCH1_STRING_LEN_EXTEND

```
static final long EPOCH1_STRING_LEN_EXTEND
```

See Also:

Constant Field Values

EPOCH2_STRING_LEN_EXTEND

```
static final long EPOCH2_STRING_LEN_EXTEND
```

See Also:

Constant Field Values

EPOCH3_STRING_LEN_EXTEND

static final long EPOCH3_STRING_LEN_EXTEND

See Also:

Constant Field Values

EPOCH4_STRING_LEN_EXTEND

static final long EPOCH4_STRING_LEN_EXTEND

See Also:

Constant Field Values

TT2000_0_STRING_LEN

static final long TT2000_0_STRING_LEN

See Also:

Constant Field Values

TT2000_1_STRING_LEN

static final long TT2000_1_STRING_LEN

See Also:

Constant Field Values

TT2000_2_STRING_LEN

static final long TT2000_2_STRING_LEN

See Also:

Constant Field Values

TT2000_3_STRING_LEN

static final long TT2000_3_STRING_LEN

See Also:

Constant Field Values

TT2000_4_STRING_LEN

static final long TT2000_4_STRING_LEN

See Also:

Constant Field Values

CDF_INT1

```
static final long CDF_INT1
```

See Also:

Constant Field Values

CDF_INT2

```
static final long CDF_INT2
```

See Also:

Constant Field Values

CDF_INT4

```
static final long CDF_INT4
```

See Also:

Constant Field Values

CDF_INT8

```
static final long CDF_INT8
```

See Also:

Constant Field Values

CDF_UINT1

```
static final long CDF_UINT1
```

See Also:

Constant Field Values

CDF_UINT2

```
static final long CDF_UINT2
```

See Also:

Constant Field Values

CDF_UINT4

```
static final long CDF_UINT4
```

See Also:

Constant Field Values

CDF_REAL4

```
static final long CDF_REAL4
```

See Also:

Constant Field Values

CDF_REAL8

```
static final long CDF_REAL8
```

See Also:

Constant Field Values

CDF_EPOCH

```
static final long CDF_EPOCH
```

See Also:

Constant Field Values

CDF_EPOCH16

```
static final long CDF_EPOCH16
```

See Also:

Constant Field Values

CDF_TIME_TT2000

```
static final long CDF_TIME_TT2000
```

See Also:

Constant Field Values

CDF_BYTE

```
static final long CDF_BYTE
```

See Also:

Constant Field Values

CDF_FLOAT

```
static final long CDF_FLOAT
```

See Also:

Constant Field Values

CDF_DOUBLE

```
static final long CDF_DOUBLE
```

See Also:

Constant Field Values

CDF_CHAR

```
static final long CDF_CHAR
```

See Also:

Constant Field Values

CDF_UCHAR

```
static final long CDF_UCHAR
```

See Also:

Constant Field Values

NETWORK_ENCODING

```
static final long NETWORK_ENCODING
```

See Also:

Constant Field Values

SUN_ENCODING

```
static final long SUN_ENCODING
```

See Also:

Constant Field Values

VAX_ENCODING

```
static final long VAX_ENCODING
```

See Also:

Constant Field Values

DECSTATION_ENCODING

```
static final long DECSTATION_ENCODING
```

See Also:

Constant Field Values

SGi_ENCODING

```
static final long SGi_ENCODING
```

See Also:

Constant Field Values

IBMPC_ENCODING

```
static final long IBMPC_ENCODING
```

See Also:

Constant Field Values

IBMRS_ENCODING

```
static final long IBMRS_ENCODING
```

See Also:

Constant Field Values

HOST_ENCODING

```
static final long HOST_ENCODING
```

See Also:

Constant Field Values

PPC_ENCODING

```
static final long PPC_ENCODING
```

See Also:

Constant Field Values

HP_ENCODING

```
static final long HP_ENCODING
```

See Also:

Constant Field Values

NeXT_ENCODING

```
static final long NeXT_ENCODING
```

See Also:

Constant Field Values

ALPHAOSF1_ENCODING

```
static final long ALPHAOSF1_ENCODING
```

See Also:

Constant Field Values

ALPHAVMSd_ENCODING

```
static final long ALPHAVMSd_ENCODING
```

See Also:

Constant Field Values

ALPHAVMSg_ENCODING

```
static final long ALPHAVMSg_ENCODING
```

See Also:

Constant Field Values

ALPHAVMSi_ENCODING

```
static final long ALPHAVMSi_ENCODING
```

See Also:

Constant Field Values

ARM_LITTLE_ENCODING

```
static final long ARM_LITTLE_ENCODING
```

See Also:

Constant Field Values

ARM_BIG_ENCODING

```
static final long ARM_BIG_ENCODING
```

See Also:

Constant Field Values

IA64VMSi_ENCODING

```
static final long IA64VMSi_ENCODING
```

See Also:

Constant Field Values

IA64VMSd_ENCODING

```
static final long IA64VMSd_ENCODING
```

See Also:

Constant Field Values

IA64VMSg_ENCODING

```
static final long IA64VMSg_ENCODING
```

See Also:

Constant Field Values

NETWORK_DECODING

```
static final long NETWORK_DECODING
```

See Also:

Constant Field Values

SUN_DECODING

```
static final long SUN_DECODING
```

See Also:

Constant Field Values

VAX_DECODING

```
static final long VAX_DECODING
```

See Also:

Constant Field Values

DECSTATION_DECODING

```
static final long DECSTATION_DECODING
```

See Also:

Constant Field Values

SGi_DECODING

```
static final long SGi_DECODING
```

See Also:

Constant Field Values

IBMPC_DECODING

```
static final long IBMPC_DECODING
```

See Also:

Constant Field Values

IBMRS_DECODING

```
static final long IBMRS_DECODING
```

See Also:

Constant Field Values

HOST_DECODING

```
static final long HOST_DECODING
```

See Also:

Constant Field Values

PPC_DECODING

```
static final long PPC_DECODING
```

See Also:

Constant Field Values

MAC_ENCODING

```
static final long MAC_ENCODING
```

See Also:

Constant Field Values

MAC_DECODING

```
static final long MAC_DECODING
```

See Also:

Constant Field Values

HP_DECODING

```
static final long HP_DECODING
```

See Also:

Constant Field Values

NeXT_DECODING

```
static final long NeXT_DECODING
```

See Also:

Constant Field Values

ALPHAOSF1_DECODING

```
static final long ALPHAOSF1_DECODING
```

See Also:

Constant Field Values

ALPHAVMSd_DECODING

```
static final long ALPHAVMSd_DECODING
```

See Also:

Constant Field Values

ALPHAVMSg_DECODING

```
static final long ALPHAVMSg_DECODING
```

See Also:

Constant Field Values

ALPHAVMSi_DECODING

static final long ALPHAVMSi_DECODING

See Also:

Constant Field Values

ARM_LITTLE_DECODING

static final long ARM_LITTLE_DECODING

See Also:

Constant Field Values

ARM_BIG_DECODING

static final long ARM_BIG_DECODING

See Also:

Constant Field Values

IA64VMSd_DECODING

static final long IA64VMSd_DECODING

See Also:

Constant Field Values

IA64VMSg_DECODING

static final long IA64VMSg_DECODING

See Also:

Constant Field Values

IA64VMSi_DECODING

static final long IA64VMSi_DECODING

See Also:

Constant Field Values

VARY

static final long VARY

See Also:

Constant Field Values

NOVARY

```
static final long NOVARY
```

See Also:

Constant Field Values

ROW_MAJOR

```
static final long ROW_MAJOR
```

See Also:

Constant Field Values

COLUMN_MAJOR

```
static final long COLUMN_MAJOR
```

See Also:

Constant Field Values

SINGLE_FILE

```
static final long SINGLE_FILE
```

See Also:

Constant Field Values

MULTI_FILE

```
static final long MULTI_FILE
```

See Also:

Constant Field Values

GLOBAL_SCOPE

```
static final long GLOBAL_SCOPE
```

See Also:

Constant Field Values

VARIABLE_SCOPE

```
static final long VARIABLE_SCOPE
```

See Also:

Constant Field Values

READONLYon

```
static final long READONLYon
```

See Also:

Constant Field Values

READONLYoff

```
static final long READONLYoff
```

See Also:

Constant Field Values

zMODEoff

```
static final long zMODEoff
```

See Also:

Constant Field Values

zMODEon1

```
static final long zMODEon1
```

See Also:

Constant Field Values

zMODEon2

```
static final long zMODEon2
```

See Also:

Constant Field Values

NEGtoPOSfp0on

```
static final long NEGtoPOSfp0on
```

See Also:

Constant Field Values

NEGtoPOSfp0off

static final long NEGtoPOSfp0off

See Also:

Constant Field Values

BACKWARDFILEon

static final long BACKWARDFILEon

See Also:

Constant Field Values

BACKWARDFILEoff

static final long BACKWARDFILEoff

See Also:

Constant Field Values

VALIDATEFILEon

static final long VALIDATEFILEon

See Also:

Constant Field Values

VALIDATEFILEoff

static final long VALIDATEFILEoff

See Also:

Constant Field Values

NO_CHECKSUM

static final long NO_CHECKSUM

See Also:

Constant Field Values

NONE_CHECKSUM

static final long NONE_CHECKSUM

See Also:

Constant Field Values

MD5_CHECKSUM

```
static final long MD5_CHECKSUM
```

See Also:

Constant Field Values

OTHER_CHECKSUM

```
static final long OTHER_CHECKSUM
```

See Also:

Constant Field Values

CDF_MAX_PARMS

```
static final long CDF_MAX_PARMS
```

See Also:

Constant Field Values

NO_COMPRESSION

```
static final long NO_COMPRESSION
```

See Also:

Constant Field Values

RLE_COMPRESSION

```
static final long RLE_COMPRESSION
```

See Also:

Constant Field Values

HUFF_COMPRESSION

```
static final long HUFF_COMPRESSION
```

See Also:

Constant Field Values

AHUFF_COMPRESSION

```
static final long AHUFF_COMPRESSION
```

See Also:

Constant Field Values

GZIP_COMPRESSION

```
static final long GZIP_COMPRESSION
```

See Also:

Constant Field Values

RLE_OF_ZEROs

```
static final long RLE_OF_ZEROs
```

See Also:

Constant Field Values

OPTIMAL_ENCODING_TREES

```
static final long OPTIMAL_ENCODING_TREES
```

See Also:

Constant Field Values

NO_SPARSEARRAYS

```
static final long NO_SPARSEARRAYS
```

See Also:

Constant Field Values

NO_SPARSERECORDS

```
static final long NO_SPARSERECORDS
```

See Also:

Constant Field Values

PAD_SPARSERECORDS

```
static final long PAD_SPARSERECORDS
```

See Also:

Constant Field Values

PREV_SPARSERECORDS

static final long PREV_SPARSERECORDS

See Also:

Constant Field Values

DEFAULT_BYTE_PADVALUE

static final byte DEFAULT_BYTE_PADVALUE

See Also:

Constant Field Values

DEFAULT_INT1_PADVALUE

static final byte DEFAULT_INT1_PADVALUE

See Also:

Constant Field Values

DEFAULT_UINT1_PADVALUE

static final short DEFAULT_UINT1_PADVALUE

See Also:

Constant Field Values

DEFAULT_INT2_PADVALUE

static final short DEFAULT_INT2_PADVALUE

See Also:

Constant Field Values

DEFAULT_UINT2_PADVALUE

static final int DEFAULT_UINT2_PADVALUE

See Also:

Constant Field Values

DEFAULT_INT4_PADVALUE

static final int DEFAULT_INT4_PADVALUE

See Also:

Constant Field Values

DEFAULT_UINT4_PADVALUE

```
static final long DEFAULT_UINT4_PADVALUE
```

See Also:

Constant Field Values

DEFAULT_INT8_PADVALUE

```
static final long DEFAULT_INT8_PADVALUE
```

See Also:

Constant Field Values

DEFAULT_REAL4_PADVALUE

```
static final float DEFAULT_REAL4_PADVALUE
```

See Also:

Constant Field Values

DEFAULT_FLOAT_PADVALUE

```
static final float DEFAULT_FLOAT_PADVALUE
```

See Also:

Constant Field Values

DEFAULT_REAL8_PADVALUE

```
static final double DEFAULT_REAL8_PADVALUE
```

See Also:

Constant Field Values

DEFAULT_DOUBLE_PADVALUE

```
static final double DEFAULT_DOUBLE_PADVALUE
```

See Also:

Constant Field Values

DEFAULT_CHAR_PADVALUE

```
static final char DEFAULT_CHAR_PADVALUE
```

See Also:

Constant Field Values

DEFAULT_UCHAR_PADVALUE

```
static final char DEFAULT_UCHAR_PADVALUE
```

See Also:

Constant Field Values

DEFAULT_EPOCH_PADVALUE

```
static final double DEFAULT_EPOCH_PADVALUE
```

See Also:

Constant Field Values

DEFAULT_EPOCH16_PADVALUE

```
static final double DEFAULT_EPOCH16_PADVALUE
```

See Also:

Constant Field Values

DEFAULT_TT2000_PADVALUE

```
static final long DEFAULT_TT2000_PADVALUE
```

See Also:

Constant Field Values

ILLEGAL_EPOCH_VALUE

```
static final double ILLEGAL_EPOCH_VALUE
```

See Also:

Constant Field Values

ILLEGAL_TT2000_VALUE

```
static final long ILLEGAL_TT2000_VALUE
```

See Also:

Constant Field Values

FILLED_TT2000_VALUE

static final long FILLED_TT2000_VALUE

See Also:

Constant Field Values

VIRTUAL_RECORD_DATA

static final long VIRTUAL_RECORD_DATA

See Also:

Constant Field Values

DID_NOT_COMPRESS

static final long DID_NOT_COMPRESS

See Also:

Constant Field Values

VAR_ALREADY_CLOSED

static final long VAR_ALREADY_CLOSED

See Also:

Constant Field Values

SINGLE_FILE_FORMAT

static final long SINGLE_FILE_FORMAT

See Also:

Constant Field Values

NO_PADVALUE_SPECIFIED

static final long NO_PADVALUE_SPECIFIED

See Also:

Constant Field Values

NO_VARS_IN_CDF

static final long NO_VARS_IN_CDF

See Also:

Constant Field Values

MULTI_FILE_FORMAT

```
static final long MULTI_FILE_FORMAT
```

See Also:

Constant Field Values

SOME_ALREADY_ALLOCATED

```
static final long SOME_ALREADY_ALLOCATED
```

See Also:

Constant Field Values

PRECEEDING_RECORDS_ALLOCATED

```
static final long PRECEEDING_RECORDS_ALLOCATED
```

See Also:

Constant Field Values

TT2000_CDF_MAYNEEDUPDATE

```
static final long TT2000_CDF_MAYNEEDUPDATE
```

See Also:

Constant Field Values

BLOCKINGFACTOR_TOO_SMALL

```
static final long BLOCKINGFACTOR_TOO_SMALL
```

See Also:

Constant Field Values

BLOCKINGFACTOR_TOO_SMALL2

```
static final long BLOCKINGFACTOR_TOO_SMALL2
```

See Also:

Constant Field Values

STRING_NOT_UTF8_ENCODING

```
static final long STRING_NOT_UTF8_ENCODING
```

See Also:

Constant Field Values

BLOCKINGFACTOR_TOO_LARGE

```
static final long BLOCKINGFACTOR_TOO_LARGE
```

See Also:

Constant Field Values

CDF_OK

```
static final long CDF_OK
```

See Also:

Constant Field Values

ATTR_NAME_TRUNC

```
static final long ATTR_NAME_TRUNC
```

See Also:

Constant Field Values

CDF_NAME_TRUNC

```
static final long CDF_NAME_TRUNC
```

See Also:

Constant Field Values

VAR_NAME_TRUNC

```
static final long VAR_NAME_TRUNC
```

See Also:

Constant Field Values

NEGATIVE_FP_ZERO

```
static final long NEGATIVE_FP_ZERO
```

See Also:

Constant Field Values

FORCED_PARAMETER

static final long FORCED_PARAMETER

See Also:

Constant Field Values

NA_FOR_VARIABLE

static final long NA_FOR_VARIABLE

See Also:

Constant Field Values

CDF_WARN

static final long CDF_WARN

See Also:

Constant Field Values

ATTR_EXISTS

static final long ATTR_EXISTS

See Also:

Constant Field Values

BAD_CDF_ID

static final long BAD_CDF_ID

See Also:

Constant Field Values

BAD_DATA_TYPE

static final long BAD_DATA_TYPE

See Also:

Constant Field Values

BAD_DIM_SIZE

static final long BAD_DIM_SIZE

See Also:

Constant Field Values

BAD_DIM_INDEX

```
static final long BAD_DIM_INDEX
```

See Also:

Constant Field Values

BAD_ENCODING

```
static final long BAD_ENCODING
```

See Also:

Constant Field Values

BAD_MAJORITY

```
static final long BAD_MAJORITY
```

See Also:

Constant Field Values

BAD_NUM_DIMS

```
static final long BAD_NUM_DIMS
```

See Also:

Constant Field Values

BAD_REC_NUM

```
static final long BAD_REC_NUM
```

See Also:

Constant Field Values

BAD_SCOPE

```
static final long BAD_SCOPE
```

See Also:

Constant Field Values

BAD_NUM_ELEMS

```
static final long BAD_NUM_ELEMS
```

See Also:

Constant Field Values

CDF_OPEN_ERROR

```
static final long CDF_OPEN_ERROR
```

See Also:

Constant Field Values

CDF_EXISTS

```
static final long CDF_EXISTS
```

See Also:

Constant Field Values

BAD_FORMAT

```
static final long BAD_FORMAT
```

See Also:

Constant Field Values

BAD_ALLOCATE_RECS

```
static final long BAD_ALLOCATE_RECS
```

See Also:

Constant Field Values

BAD_CDF_EXTENSION

```
static final long BAD_CDF_EXTENSION
```

See Also:

Constant Field Values

NO_SUCH_ATTR

```
static final long NO_SUCH_ATTR
```

See Also:

Constant Field Values

NO_SUCH_ENTRY

static final long NO_SUCH_ENTRY

See Also:

Constant Field Values

NO_SUCH_VAR

static final long NO_SUCH_VAR

See Also:

Constant Field Values

VAR_READ_ERROR

static final long VAR_READ_ERROR

See Also:

Constant Field Values

VAR_WRITE_ERROR

static final long VAR_WRITE_ERROR

See Also:

Constant Field Values

BAD_ARGUMENT

static final long BAD_ARGUMENT

See Also:

Constant Field Values

IBM_PC_OVERFLOW

static final long IBM_PC_OVERFLOW

See Also:

Constant Field Values

TOO_MANY_VARS

static final long TOO_MANY_VARS

See Also:

Constant Field Values

VAR_EXISTS

```
static final long VAR_EXISTS
```

See Also:

Constant Field Values

BAD_MALLOC

```
static final long BAD_MALLOC
```

See Also:

Constant Field Values

NOT_A_CDF

```
static final long NOT_A_CDF
```

See Also:

Constant Field Values

CORRUPTED_V2_CDF

```
static final long CORRUPTED_V2_CDF
```

See Also:

Constant Field Values

VAR_OPEN_ERROR

```
static final long VAR_OPEN_ERROR
```

See Also:

Constant Field Values

BAD_INITIAL_RECS

```
static final long BAD_INITIAL_RECS
```

See Also:

Constant Field Values

BAD_BLOCKING_FACTOR

```
static final long BAD_BLOCKING_FACTOR
```

See Also:

Constant Field Values

END_OF_VAR

```
static final long END_OF_VAR
```

See Also:

Constant Field Values

BAD_CDFSTATUS

```
static final long BAD_CDFSTATUS
```

See Also:

Constant Field Values

CDF_INTERNAL_ERROR

```
static final long CDF_INTERNAL_ERROR
```

See Also:

Constant Field Values

BAD_NUM_VARS

```
static final long BAD_NUM_VARS
```

See Also:

Constant Field Values

BAD_REC_COUNT

```
static final long BAD_REC_COUNT
```

See Also:

Constant Field Values

BAD_REC_INTERVAL

```
static final long BAD_REC_INTERVAL
```

See Also:

Constant Field Values

BAD_DIM_COUNT

```
static final long BAD_DIM_COUNT
```

See Also:

Constant Field Values

BAD_DIM_INTERVAL

```
static final long BAD_DIM_INTERVAL
```

See Also:

Constant Field Values

BAD_VAR_NUM

```
static final long BAD_VAR_NUM
```

See Also:

Constant Field Values

BAD_ATTR_NUM

```
static final long BAD_ATTR_NUM
```

See Also:

Constant Field Values

BAD_ENTRY_NUM

```
static final long BAD_ENTRY_NUM
```

See Also:

Constant Field Values

BAD_ATTR_NAME

```
static final long BAD_ATTR_NAME
```

See Also:

Constant Field Values

BAD_VAR_NAME

```
static final long BAD_VAR_NAME
```

See Also:

Constant Field Values

NO_ATTR_SELECTED

```
static final long NO_ATTR_SELECTED
```

See Also:

Constant Field Values

NO_ENTRY_SELECTED

```
static final long NO_ENTRY_SELECTED
```

See Also:

Constant Field Values

NO_VAR_SELECTED

```
static final long NO_VAR_SELECTED
```

See Also:

Constant Field Values

BAD_CDF_NAME

```
static final long BAD_CDF_NAME
```

See Also:

Constant Field Values

CANNOT_CHANGE

```
static final long CANNOT_CHANGE
```

See Also:

Constant Field Values

NO_STATUS_SELECTED

```
static final long NO_STATUS_SELECTED
```

See Also:

Constant Field Values

NO_CDF_SELECTED

```
static final long NO_CDF_SELECTED
```

See Also:

Constant Field Values

READ_ONLY_DISTRIBUTION

```
static final long READ_ONLY_DISTRIBUTION
```

See Also:

Constant Field Values

CDF_CLOSE_ERROR

```
static final long CDF_CLOSE_ERROR
```

See Also:

Constant Field Values

VAR_CLOSE_ERROR

```
static final long VAR_CLOSE_ERROR
```

See Also:

Constant Field Values

BAD_FNC_OR_ITEM

```
static final long BAD_FNC_OR_ITEM
```

See Also:

Constant Field Values

ILLEGAL_ON_V1_CDF

```
static final long ILLEGAL_ON_V1_CDF
```

See Also:

Constant Field Values

BAD_CACHE_SIZE

```
static final long BAD_CACHE_SIZE
```

See Also:

Constant Field Values

CDF_CREATE_ERROR

```
static final long CDF_CREATE_ERROR
```

See Also:

Constant Field Values

NO_SUCH_CDF

```
static final long NO_SUCH_CDF
```

See Also:

Constant Field Values

VAR_CREATE_ERROR

```
static final long VAR_CREATE_ERROR
```

See Also:

Constant Field Values

READ_ONLY_MODE

```
static final long READ_ONLY_MODE
```

See Also:

Constant Field Values

ILLEGAL_IN_zMODE

```
static final long ILLEGAL_IN_zMODE
```

See Also:

Constant Field Values

BAD_zMODE

```
static final long BAD_zMODE
```

See Also:

Constant Field Values

BAD_READONLY_MODE

```
static final long BAD_READONLY_MODE
```

See Also:

Constant Field Values

CDF_READ_ERROR

```
static final long CDF_READ_ERROR
```

See Also:

Constant Field Values

CDF_WRITE_ERROR

```
static final long CDF_WRITE_ERROR
```

See Also:

Constant Field Values

ILLEGAL_FOR_SCOPE

```
static final long ILLEGAL_FOR_SCOPE
```

See Also:

Constant Field Values

NO_MORE_ACCESS

```
static final long NO_MORE_ACCESS
```

See Also:

Constant Field Values

BAD_DECODING

```
static final long BAD_DECODING
```

See Also:

Constant Field Values

BAD_NEGtoPOSfp0_MODE

```
static final long BAD_NEGtoPOSfp0_MODE
```

See Also:

Constant Field Values

UNSUPPORTED_OPERATION

static final long UNSUPPORTED_OPERATION

See Also:

Constant Field Values

CDF_SAVE_ERROR

static final long CDF_SAVE_ERROR

See Also:

Constant Field Values

VAR_SAVE_ERROR

static final long VAR_SAVE_ERROR

See Also:

Constant Field Values

NO_WRITE_ACCESS

static final long NO_WRITE_ACCESS

See Also:

Constant Field Values

NO_DELETE_ACCESS

static final long NO_DELETE_ACCESS

See Also:

Constant Field Values

CDF_DELETE_ERROR

static final long CDF_DELETE_ERROR

See Also:

Constant Field Values

VAR_DELETE_ERROR

static final long VAR_DELETE_ERROR

See Also:

Constant Field Values

UNKNOWN_COMPRESSION

```
static final long UNKNOWN_COMPRESSION
```

See Also:

Constant Field Values

CANNOT_COMPRESS

```
static final long CANNOT_COMPRESS
```

See Also:

Constant Field Values

DECOMPRESSION_ERROR

```
static final long DECOMPRESSION_ERROR
```

See Also:

Constant Field Values

COMPRESSION_ERROR

```
static final long COMPRESSION_ERROR
```

See Also:

Constant Field Values

EMPTY_COMPRESSED_CDF

```
static final long EMPTY_COMPRESSED_CDF
```

See Also:

Constant Field Values

BAD_COMPRESSION_PARM

```
static final long BAD_COMPRESSION_PARM
```

See Also:

Constant Field Values

UNKNOWN_SPARSENESS

```
static final long UNKNOWN_SPARSENESS
```

See Also:

Constant Field Values

CANNOT_SPARSERECORDS

```
static final long CANNOT_SPARSERECORDS
```

See Also:

Constant Field Values

CANNOT_SPARSEARRAYS

```
static final long CANNOT_SPARSEARRAYS
```

See Also:

Constant Field Values

TOO_MANY_PARMS

```
static final long TOO_MANY_PARMS
```

See Also:

Constant Field Values

NO_SUCH_RECORD

```
static final long NO_SUCH_RECORD
```

See Also:

Constant Field Values

CANNOT_ALLOCATE_RECORDS

```
static final long CANNOT_ALLOCATE_RECORDS
```

See Also:

Constant Field Values

CANNOT_COPY

```
static final long CANNOT_COPY
```

See Also:

Constant Field Values

SCRATCH_DELETE_ERROR

static final long SCRATCH_DELETE_ERROR

See Also:

Constant Field Values

SCRATCH_CREATE_ERROR

static final long SCRATCH_CREATE_ERROR

See Also:

Constant Field Values

SCRATCH_READ_ERROR

static final long SCRATCH_READ_ERROR

See Also:

Constant Field Values

SCRATCH_WRITE_ERROR

static final long SCRATCH_WRITE_ERROR

See Also:

Constant Field Values

BAD_SPARSEARRAYS_PARM

static final long BAD_SPARSEARRAYS_PARM

See Also:

Constant Field Values

BAD_SCRATCH_DIR

static final long BAD_SCRATCH_DIR

See Also:

Constant Field Values

DATATYPE_MISMATCH

static final long DATATYPE_MISMATCH

See Also:

Constant Field Values

NOT_A_CDF_OR_NOT_SUPPORTED

```
static final long NOT_A_CDF_OR_NOT_SUPPORTED
```

See Also:

Constant Field Values

CORRUPTED_V3_CDF

```
static final long CORRUPTED_V3_CDF
```

See Also:

Constant Field Values

ILLEGAL_EPOCH_FIELD

```
static final long ILLEGAL_EPOCH_FIELD
```

See Also:

Constant Field Values

BAD_CHECKSUM

```
static final long BAD_CHECKSUM
```

See Also:

Constant Field Values

CHECKSUM_ERROR

```
static final long CHECKSUM_ERROR
```

See Also:

Constant Field Values

CHECKSUM_NOT_ALLOWED

```
static final long CHECKSUM_NOT_ALLOWED
```

See Also:

Constant Field Values

IS_A_NETCDF

```
static final long IS_A_NETCDF
```

See Also:

Constant Field Values

TT2000_TIME_ERROR

```
static final long TT2000_TIME_ERROR
```

See Also:

Constant Field Values

UNABLE_TO_PROCESS_CDF

```
static final long UNABLE_TO_PROCESS_CDF
```

See Also:

Constant Field Values

ZLIB_COMPRESS_ERROR

```
static final long ZLIB_COMPRESS_ERROR
```

See Also:

Constant Field Values

ZLIB_UNCOMPRESS_ERROR

```
static final long ZLIB_UNCOMPRESS_ERROR
```

See Also:

Constant Field Values

CANNOT_INSERT_RECORDS

```
static final long CANNOT_INSERT_RECORDS
```

See Also:

Constant Field Values

TT2000_USED_OUTDATED_TABLE

```
static final long TT2000_USED_OUTDATED_TABLE
```

See Also:

Constant Field Values

BADDATE_LEAPSECOND_UPDATED

static final long BADDATE_LEAPSECOND_UPDATED

See Also:

Constant Field Values

FUNCTION_NOT_SUPPORTED

static final long FUNCTION_NOT_SUPPORTED

See Also:

Constant Field Values

TRY_TO_READ_NONSTRING_DATA

static final long TRY_TO_READ_NONSTRING_DATA

See Also:

Constant Field Values

BAD_NUM_STRINGS

static final long BAD_NUM_STRINGS

See Also:

Constant Field Values

CANNOT_CONVERT_WIDECHAR

static final long CANNOT_CONVERT_WIDECHAR

See Also:

Constant Field Values

CREATE_

static final long CREATE_

See Also:

Constant Field Values

OPEN_

static final long OPEN_

See Also:

Constant Field Values

DELETE_

```
static final long DELETE_
```

See Also:

Constant Field Values

CLOSE_

```
static final long CLOSE_
```

See Also:

Constant Field Values

SELECT_

```
static final long SELECT_
```

See Also:

Constant Field Values

CONFIRM_

```
static final long CONFIRM_
```

See Also:

Constant Field Values

GET_

```
static final long GET_
```

See Also:

Constant Field Values

PUT_

```
static final long PUT_
```

See Also:

Constant Field Values

SAVE_

```
static final long SAVE_
```

See Also:

Constant Field Values

BACKWARD_

```
static final long BACKWARD_
```

See Also:

Constant Field Values

GETCDFFILEBACKWARD_

```
static final long GETCDFFILEBACKWARD_
```

See Also:

Constant Field Values

CHECKSUM_

```
static final long CHECKSUM_
```

See Also:

Constant Field Values

GETCDFCHECKSUM_

```
static final long GETCDFCHECKSUM_
```

See Also:

Constant Field Values

VALIDATE_

```
static final long VALIDATE_
```

See Also:

Constant Field Values

GETCDFVALIDATE_

```
static final long GETCDFVALIDATE_
```

See Also:

Constant Field Values

GETLEAPSECONDSENVVAR_

static final long GETLEAPSECONDSENVVAR_

See Also:

Constant Field Values

NULL_

static final long NULL_

See Also:

Constant Field Values

CDF_

static final long CDF_

See Also:

Constant Field Values

CDF_NAME_

static final long CDF_NAME_

See Also:

Constant Field Values

CDF_ENCODING_

static final long CDF_ENCODING_

See Also:

Constant Field Values

CDF_DECODING_

static final long CDF_DECODING_

See Also:

Constant Field Values

CDF_MAJORITY_

static final long CDF_MAJORITY_

See Also:

Constant Field Values

CDF_FORMAT_

```
static final long CDF_FORMAT_
```

See Also:

Constant Field Values

CDF_COPYRIGHT_

```
static final long CDF_COPYRIGHT_
```

See Also:

Constant Field Values

CDF_NUMrVARS_

```
static final long CDF_NUMrVARS_
```

See Also:

Constant Field Values

CDF_NUMzVARS_

```
static final long CDF_NUMzVARS_
```

See Also:

Constant Field Values

CDF_NUMATTRS_

```
static final long CDF_NUMATTRS_
```

See Also:

Constant Field Values

CDF_NUMgATTRS_

```
static final long CDF_NUMgATTRS_
```

See Also:

Constant Field Values

CDF_NUMvATTRS_

```
static final long CDF_NUMvATTRS_
```

See Also:

Constant Field Values

CDF_VERSION_

```
static final long CDF_VERSION_
```

See Also:

Constant Field Values

CDF_RELEASE_

```
static final long CDF_RELEASE_
```

See Also:

Constant Field Values

CDF_INCREMENT_

```
static final long CDF_INCREMENT_
```

See Also:

Constant Field Values

CDF_STATUS_

```
static final long CDF_STATUS_
```

See Also:

Constant Field Values

CDF_READONLY_MODE_

```
static final long CDF_READONLY_MODE_
```

See Also:

Constant Field Values

CDF_zMODE_

```
static final long CDF_zMODE_
```

See Also:

Constant Field Values

CDF_NEGtoPOSfp0_MODE_

static final long CDF_NEGtoPOSfp0_MODE_

See Also:

Constant Field Values

LIB_COPYRIGHT_

static final long LIB_COPYRIGHT_

See Also:

Constant Field Values

LIB_VERSION_

static final long LIB_VERSION_

See Also:

Constant Field Values

LIB_RELEASE_

static final long LIB_RELEASE_

See Also:

Constant Field Values

LIB_INCREMENT_

static final long LIB_INCREMENT_

See Also:

Constant Field Values

LIB_subINCREMENT_

static final long LIB_subINCREMENT_

See Also:

Constant Field Values

rVARs_NUMDIMS_

static final long rVARs_NUMDIMS_

See Also:

Constant Field Values

rVARs_DIMSIZES_

```
static final long rVARs_DIMSIZES_
```

See Also:

Constant Field Values

rVARs_MAXREC_

```
static final long rVARs_MAXREC_
```

See Also:

Constant Field Values

rVARs_RECDATA_

```
static final long rVARs_RECDATA_
```

See Also:

Constant Field Values

rVARs_RECNUMBER_

```
static final long rVARs_RECNUMBER_
```

See Also:

Constant Field Values

rVARs_RECCOUNT_

```
static final long rVARs_RECCOUNT_
```

See Also:

Constant Field Values

rVARs_RECINTERVAL_

```
static final long rVARs_RECINTERVAL_
```

See Also:

Constant Field Values

rVARs_DIMINDICES_

```
static final long rVARs_DIMINDICES_
```

See Also:

Constant Field Values

rVARs_DIMCOUNTS_

```
static final long rVARs_DIMCOUNTS_
```

See Also:

Constant Field Values

rVARs_DIMINTERVALS_

```
static final long rVARs_DIMINTERVALS_
```

See Also:

Constant Field Values

rVAR_

```
static final long rVAR_
```

See Also:

Constant Field Values

rVAR_NAME_

```
static final long rVAR_NAME_
```

See Also:

Constant Field Values

rVAR_DATATYPE_

```
static final long rVAR_DATATYPE_
```

See Also:

Constant Field Values

rVAR_NUMELEMS_

```
static final long rVAR_NUMELEMS_
```

See Also:

Constant Field Values

rVAR_RECVARY_

static final long rVAR_RECVARY_

See Also:

Constant Field Values

rVAR_DIMVARYS_

static final long rVAR_DIMVARYS_

See Also:

Constant Field Values

rVAR_NUMBER_

static final long rVAR_NUMBER_

See Also:

Constant Field Values

rVAR_DATA_

static final long rVAR_DATA_

See Also:

Constant Field Values

rVAR_HYPERDATA_

static final long rVAR_HYPERDATA_

See Also:

Constant Field Values

rVAR_SEQDATA_

static final long rVAR_SEQDATA_

See Also:

Constant Field Values

rVAR_SEQPOS_

static final long rVAR_SEQPOS_

See Also:

Constant Field Values

rVAR_MAXREC_

```
static final long rVAR_MAXREC_
```

See Also:

Constant Field Values

rVAR_MAXallocREC_

```
static final long rVAR_MAXallocREC_
```

See Also:

Constant Field Values

rVAR_DATASPEC_

```
static final long rVAR_DATASPEC_
```

See Also:

Constant Field Values

rVAR_PADVALUE_

```
static final long rVAR_PADVALUE_
```

See Also:

Constant Field Values

rVAR_INITIALRECS_

```
static final long rVAR_INITIALRECS_
```

See Also:

Constant Field Values

rVAR_BLOCKINGFACTOR_

```
static final long rVAR_BLOCKINGFACTOR_
```

See Also:

Constant Field Values

rVAR_nINDEXRECORDS_

```
static final long rVAR_nINDEXRECORDS_
```

See Also:

Constant Field Values

rVAR_nINDEXENTRIES_

```
static final long rVAR_nINDEXENTRIES_
```

See Also:

Constant Field Values

rVAR_EXISTENCE_

```
static final long rVAR_EXISTENCE_
```

See Also:

Constant Field Values

zVARs_MAXREC_

```
static final long zVARs_MAXREC_
```

See Also:

Constant Field Values

zVARs_RECDATA_

```
static final long zVARs_RECDATA_
```

See Also:

Constant Field Values

zVAR_

```
static final long zVAR_
```

See Also:

Constant Field Values

zVAR_NAME_

```
static final long zVAR_NAME_
```

See Also:

Constant Field Values

zVAR_DATATYPE_

static final long zVAR_DATATYPE_

See Also:

Constant Field Values

zVAR_NUMELEMS_

static final long zVAR_NUMELEMS_

See Also:

Constant Field Values

zVAR_NUMDIMS_

static final long zVAR_NUMDIMS_

See Also:

Constant Field Values

zVAR_DIMSIZES_

static final long zVAR_DIMSIZES_

See Also:

Constant Field Values

zVAR_RECVARY_

static final long zVAR_RECVARY_

See Also:

Constant Field Values

zVAR_DIMVARYS_

static final long zVAR_DIMVARYS_

See Also:

Constant Field Values

zVAR_NUMBER_

static final long zVAR_NUMBER_

See Also:

Constant Field Values

zVAR_DATA_

```
static final long zVAR_DATA_
```

See Also:

Constant Field Values

zVAR_HYPERDATA_

```
static final long zVAR_HYPERDATA_
```

See Also:

Constant Field Values

zVAR_SEQDATA_

```
static final long zVAR_SEQDATA_
```

See Also:

Constant Field Values

zVAR_SEQPOS_

```
static final long zVAR_SEQPOS_
```

See Also:

Constant Field Values

zVAR_MAXREC_

```
static final long zVAR_MAXREC_
```

See Also:

Constant Field Values

zVAR_MAXallocREC_

```
static final long zVAR_MAXallocREC_
```

See Also:

Constant Field Values

zVAR_DATASPEC_

```
static final long zVAR_DATASPEC_
```

See Also:

Constant Field Values

zVAR_PADVALUE_

```
static final long zVAR_PADVALUE_
```

See Also:

Constant Field Values

zVAR_INITIALRECS_

```
static final long zVAR_INITIALRECS_
```

See Also:

Constant Field Values

zVAR_BLOCKINGFACTOR_

```
static final long zVAR_BLOCKINGFACTOR_
```

See Also:

Constant Field Values

zVAR_nINDEXRECORDS_

```
static final long zVAR_nINDEXRECORDS_
```

See Also:

Constant Field Values

zVAR_nINDEXENTRIES_

```
static final long zVAR_nINDEXENTRIES_
```

See Also:

Constant Field Values

zVAR_EXISTENCE_

```
static final long zVAR_EXISTENCE_
```

See Also:

Constant Field Values

zVAR_RECNUMBER_

static final long zVAR_RECNUMBER_

See Also:

Constant Field Values

zVAR_RECCOUNT_

static final long zVAR_RECCOUNT_

See Also:

Constant Field Values

zVAR_RECINTERVAL_

static final long zVAR_RECINTERVAL_

See Also:

Constant Field Values

zVAR_DIMINDICES_

static final long zVAR_DIMINDICES_

See Also:

Constant Field Values

zVAR_DIMCOUNTS_

static final long zVAR_DIMCOUNTS_

See Also:

Constant Field Values

zVAR_DIMINTERVALS_

static final long zVAR_DIMINTERVALS_

See Also:

Constant Field Values

ATTR_

static final long ATTR_

See Also:

Constant Field Values

ATTR_SCOPE_

```
static final long ATTR_SCOPE_
```

See Also:

Constant Field Values

ATTR_NAME_

```
static final long ATTR_NAME_
```

See Also:

Constant Field Values

ATTR_NUMBER_

```
static final long ATTR_NUMBER_
```

See Also:

Constant Field Values

ATTR_MAXgENTRY_

```
static final long ATTR_MAXgENTRY_
```

See Also:

Constant Field Values

ATTR_NUMgENTRIES_

```
static final long ATTR_NUMgENTRIES_
```

See Also:

Constant Field Values

ATTR_MAXrENTRY_

```
static final long ATTR_MAXrENTRY_
```

See Also:

Constant Field Values

ATTR_NUMrENTRIES_

```
static final long ATTR_NUMrENTRIES_
```

See Also:

Constant Field Values

ATTR_MAXzENTRY_

```
static final long ATTR_MAXzENTRY_
```

See Also:

Constant Field Values

ATTR_NUMzENTRIES_

```
static final long ATTR_NUMzENTRIES_
```

See Also:

Constant Field Values

ATTR_EXISTENCE_

```
static final long ATTR_EXISTENCE_
```

See Also:

Constant Field Values

gENTRY_

```
static final long gENTRY_
```

See Also:

Constant Field Values

gENTRY_EXISTENCE_

```
static final long gENTRY_EXISTENCE_
```

See Also:

Constant Field Values

gENTRY_DATATYPE_

```
static final long gENTRY_DATATYPE_
```

See Also:

Constant Field Values

gENTRY_NUMELEMS_

static final long gENTRY_NUMELEMS_

See Also:

Constant Field Values

gENTRY_DATASPEC_

static final long gENTRY_DATASPEC_

See Also:

Constant Field Values

gENTRY_DATA_

static final long gENTRY_DATA_

See Also:

Constant Field Values

rENTRY_

static final long rENTRY_

See Also:

Constant Field Values

rENTRY_NAME_

static final long rENTRY_NAME_

See Also:

Constant Field Values

rENTRY_EXISTENCE_

static final long rENTRY_EXISTENCE_

See Also:

Constant Field Values

rENTRY_DATATYPE_

static final long rENTRY_DATATYPE_

See Also:

Constant Field Values

rENTRY_NUMELEMS_

```
static final long rENTRY_NUMELEMS_
```

See Also:

Constant Field Values

rENTRY_DATASPEC_

```
static final long rENTRY_DATASPEC_
```

See Also:

Constant Field Values

rENTRY_DATA_

```
static final long rENTRY_DATA_
```

See Also:

Constant Field Values

zENTRY_

```
static final long zENTRY_
```

See Also:

Constant Field Values

zENTRY_NAME_

```
static final long zENTRY_NAME_
```

See Also:

Constant Field Values

zENTRY_EXISTENCE_

```
static final long zENTRY_EXISTENCE_
```

See Also:

Constant Field Values

zENTRY_DATATYPE_

```
static final long zENTRY_DATATYPE_
```

See Also:

Constant Field Values

zENTRY_NUMELEMS_

```
static final long zENTRY_NUMELEMS_
```

See Also:

Constant Field Values

zENTRY_DATASPEC_

```
static final long zENTRY_DATASPEC_
```

See Also:

Constant Field Values

zENTRY_DATA_

```
static final long zENTRY_DATA_
```

See Also:

Constant Field Values

STATUS_TEXT_

```
static final long STATUS_TEXT_
```

See Also:

Constant Field Values

CDF_CACHESIZE_

```
static final long CDF_CACHESIZE_
```

See Also:

Constant Field Values

rVARs_CACHESIZE_

```
static final long rVARs_CACHESIZE_
```

See Also:

Constant Field Values

zVARs_CACHESIZE_

static final long zVARs_CACHESIZE_

See Also:

Constant Field Values

rVAR_CACHESIZE_

static final long rVAR_CACHESIZE_

See Also:

Constant Field Values

zVAR_CACHESIZE_

static final long zVAR_CACHESIZE_

See Also:

Constant Field Values

zVARs_RECNUMBER_

static final long zVARs_RECNUMBER_

See Also:

Constant Field Values

rVAR_ALLOCATERECS_

static final long rVAR_ALLOCATERECS_

See Also:

Constant Field Values

zVAR_ALLOCATERECS_

static final long zVAR_ALLOCATERECS_

See Also:

Constant Field Values

DATATYPE_SIZE_

static final long DATATYPE_SIZE_

See Also:

Constant Field Values

CURgENTRY_EXISTENCE_

```
static final long CURgENTRY_EXISTENCE_
```

See Also:

Constant Field Values

CURrENTRY_EXISTENCE_

```
static final long CURrENTRY_EXISTENCE_
```

See Also:

Constant Field Values

CURzENTRY_EXISTENCE_

```
static final long CURzENTRY_EXISTENCE_
```

See Also:

Constant Field Values

CDF_INFO_

```
static final long CDF_INFO_
```

See Also:

Constant Field Values

CDF_COMPRESSION_

```
static final long CDF_COMPRESSION_
```

See Also:

Constant Field Values

zVAR_COMPRESSION_

```
static final long zVAR_COMPRESSION_
```

See Also:

Constant Field Values

zVAR_SPARSERECORDS_

```
static final long zVAR_SPARSERECORDS_
```

See Also:

Constant Field Values

zVAR_SPARSEARRAYS_

```
static final long zVAR_SPARSEARRAYS_
```

See Also:

Constant Field Values

zVAR_ALLOCATEBLOCK_

```
static final long zVAR_ALLOCATEBLOCK_
```

See Also:

Constant Field Values

zVAR_NUMRECS_

```
static final long zVAR_NUMRECS_
```

See Also:

Constant Field Values

zVAR_NUMallocRECS_

```
static final long zVAR_NUMallocRECS_
```

See Also:

Constant Field Values

rVAR_COMPRESSION_

```
static final long rVAR_COMPRESSION_
```

See Also:

Constant Field Values

rVAR_SPARSERECORDS_

```
static final long rVAR_SPARSERECORDS_
```

See Also:

Constant Field Values

rVAR_SPARSEARRAYS_

```
static final long rVAR_SPARSEARRAYS_
```

See Also:

Constant Field Values

rVAR_ALLOCATEBLOCK_

```
static final long rVAR_ALLOCATEBLOCK_
```

See Also:

Constant Field Values

rVAR_NUMRECS_

```
static final long rVAR_NUMRECS_
```

See Also:

Constant Field Values

rVAR_NUMallocRECS_

```
static final long rVAR_NUMallocRECS_
```

See Also:

Constant Field Values

rVAR_ALLOCATEDFROM_

```
static final long rVAR_ALLOCATEDFROM_
```

See Also:

Constant Field Values

rVAR_ALLOCATEDTO_

```
static final long rVAR_ALLOCATEDTO_
```

See Also:

Constant Field Values

zVAR_ALLOCATEDFROM_

```
static final long zVAR_ALLOCATEDFROM_
```

See Also:

Constant Field Values

zVAR_ALLOCATEDTO_

```
static final long zVAR_ALLOCATEDTO_
```

See Also:

Constant Field Values

zVAR_nINDEXLEVELS_

```
static final long zVAR_nINDEXLEVELS_
```

See Also:

Constant Field Values

rVAR_nINDEXLEVELS_

```
static final long rVAR_nINDEXLEVELS_
```

See Also:

Constant Field Values

CDF_SCRATCHDIR_

```
static final long CDF_SCRATCHDIR_
```

See Also:

Constant Field Values

rVAR_RESERVEPERCENT_

```
static final long rVAR_RESERVEPERCENT_
```

See Also:

Constant Field Values

zVAR_RESERVEPERCENT_

```
static final long zVAR_RESERVEPERCENT_
```

See Also:

Constant Field Values

rVAR_RECORDS_

static final long rVAR_RECORDS_

See Also:

Constant Field Values

zVAR_RECORDS_

static final long zVAR_RECORDS_

See Also:

Constant Field Values

STAGE_CACHESIZE_

static final long STAGE_CACHESIZE_

See Also:

Constant Field Values

COMPRESS_CACHESIZE_

static final long COMPRESS_CACHESIZE_

See Also:

Constant Field Values

CDF_CHECKSUM_

static final long CDF_CHECKSUM_

See Also:

Constant Field Values

rVAR_RECORDS_RENUMBER_

static final long rVAR_RECORDS_RENUMBER_

See Also:

Constant Field Values

zVAR_RECORDS_RENUMBER_

static final long zVAR_RECORDS_RENUMBER_

See Also:

Constant Field Values

CDF_LEAPSECONDLASTUPDATED_

```
static final long CDF_LEAPSECONDLASTUPDATED_
```

See Also:

Constant Field Values

rENTRY_STRINGSDATA_

```
static final long rENTRY_STRINGSDATA_
```

See Also:

Constant Field Values

zENTRY_STRINGSDATA_

```
static final long zENTRY_STRINGSDATA_
```

See Also:

Constant Field Values

rENTRY_NUMSTRINGS_

```
static final long rENTRY_NUMSTRINGS_
```

See Also:

Constant Field Values

zENTRY_NUMSTRINGS_

```
static final long zENTRY_NUMSTRINGS_
```

See Also:

Constant Field Values

CDFwithSTATS_

```
static final long CDFwithSTATS_
```

See Also:

Constant Field Values

CDF_ACCESS_

```
static final long CDF_ACCESS_
```

See Also:

Constant Field Values

STRINGDELIMITER

```
static final java.lang.String STRINGDELIMITER
```

BeginUnixTimeEPOCH

```
static final double BeginUnixTimeEPOCH
```

See Also:

Constant Field Values

BeginUnixTimeEPOCH16

```
static final double BeginUnixTimeEPOCH16
```

See Also:

Constant Field Values

gsfc.nssdc.cdf

Class CDFData

java.lang.Object
gsfc.nssdc.cdf.CDFData

All Implemented Interfaces:

CDFConstants, CDFObject

```
public class CDFData
extends java.lang.Object
implements CDFObject, CDFConstants
```

This class acts as the glue between the Java code and the Java Native Interface (JNI) code. This class applies only to the Variable object. It handles its data. This class translates a multi-dimensional array data into a 1-dimensional (1D) array prior to sending data to the JNI code for processing. Similarly, data retrieved in 1D array from the JNI code is properly dimensioned for usage or further manipulation.

Version:

1.0, 2.0 03/18/05 Selection of current CDF and variable are done as part of operations passed to JNI. JNI call is synchronized so only one process is allowed in a JVM, due to multi-thread safety. The select method will never be called., 3.0 06/09/09 The number of dimesions returned from the get method depends on the variable dimensions and dimesional elements: e.g., 2-dim (2x1 or 1x2) will have a 2-dim, not 1-dim, object returned, while 2-dim (1x1) returns an object of a single item, not an array.

See Also:

Variable, CDFException

Field Summary

Fields inherited from interface gsfc.nssdc.cdf.CDFConstants

AHUFF_COMPRESSION, ALPHAOSF1_DECODING, ALPHAOSF1_ENCODING, ALPHAVMSd_DECODING, ALPHAVMSd_ENCODING, ALPHAVMSg_DECODING, ALPHAVMSg_ENCODING, ALPHAVMSi_DECODING, ALPHAVMSi_ENCODING, ARM_BIG_DECODING, ARM_BIG_ENCODING, ARM_LITTLE_DECODING, ARM_LITTLE_ENCODING, ATTR_, ATTR_EXISTENCE_, ATTR_EXISTS, ATTR_MAXgENTRY_, ATTR_MAXrENTRY_, ATTR_MAXzENTRY_, ATTR_NAME_, ATTR_NAME_TRUNC, ATTR_NUMBER_, ATTR_NUMgENTRIES_, ATTR_NUMrENTRIES_, ATTR_NUMzENTRIES_, ATTR_SCOPE_, BACKWARD_, BACKWARDFILEoff, BACKWARDFILEon, BAD_ALLOCATE_RECS, BAD_ARGUMENT, BAD_ATTR_NAME, BAD_ATTR_NUM, BAD_BLOCKING_FACTOR, BAD_CACHE_SIZE, BAD_CDF_EXTENSION, BAD_CDF_ID, BAD_CDF_NAME, BAD_CDFSTATUS, BAD_CHECKSUM, BAD_COMPRESSION_PARM, BAD_DATA_TYPE, BAD_DECODING, BAD_DIM_COUNT, BAD_DIM_INDEX, BAD_DIM_INTERVAL, BAD_DIM_SIZE, BAD_ENCODING, BAD_ENTRY_NUM, BAD_FNC_OR_ITEM, BAD_FORMAT, BAD_INITIAL_RECS, BAD_MAJORITY, BAD_MALLOC, BAD_NEGtoPOSfp0_MODE, BAD_NUM_DIMS, BAD_NUM_ELEMS, BAD_NUM_STRINGS, BAD_NUM_VARS, BAD_READONLY_MODE, BAD_REC_COUNT, BAD_REC_INTERVAL, BAD_REC_NUM, BAD_SCOPE, BAD_SCRATCH_DIR, BAD_SPARSEARRAYS_PARM, BAD_VAR_NAME, BAD_VAR_NUM, BAD_zMODE, BADDATE_LEAPSECOND_UPDATED, BeginUnixTimeEPOCH, BeginUnixTimeEPOCH16, BLOCKINGFACTOR_TOO_LARGE, BLOCKINGFACTOR_TOO_SMALL, BLOCKINGFACTOR_TOO_SMALL2, CANNOT_ALLOCATE_RECORDS, CANNOT_CHANGE, CANNOT_COMPRESS, CANNOT_CONVERT_WIDECHAR, CANNOT_COPY, CANNOT_INSERT_RECORDS, CANNOT_SPARSEARRAYS, CANNOT_SPARSERECORDS, CDF_, CDF_ACCESS_, CDF_ATTR_NAME_LEN, CDF_ATTR_NAME_LEN256, CDF_BYTE, CDF_CACHESIZE_, CDF_CHAR, CDF_CHECKSUM_, CDF_CLOSE_ERROR, CDF_COMPRESSION_, CDF_COPYRIGHT_, CDF_COPYRIGHT_LEN, CDF_CREATE_ERROR, CDF_DECODING_, CDF_DELETE_ERROR, CDF_DOUBLE, CDF_ENCODING_, CDF_EPOCH, CDF_EPOCH16, CDF_EXISTS, CDF_FLOAT, CDF_FORMAT_, CDF_INCREMENT_, CDF_INFO_, CDF_INT1, CDF_INT2, CDF_INT4, CDF_INT8, CDF_INTERNAL_ERROR, CDF_LEAPSECONDLASTUPDATED_, CDF_MAJORITY_, CDF_MAX_DIMS, CDF_MAX_PARMS, CDF_MIN_DIMS, CDF_NAME_, CDF_NAME_TRUNC, CDF_NEGtoPOSfp0_MODE_, CDF_NUMATTRS_, CDF_NUMgATTRS_, CDF_NUMrVARS_, CDF_NUMvATTRS_, CDF_NUMzVARS_, CDF_OK, CDF_OPEN_ERROR, CDF_PATHNAME_LEN, CDF_READ_ERROR, CDF_READONLY_MODE_, CDF_REAL4, CDF_REAL8, CDF_RELEASE_, CDF_SAVE_ERROR, CDF_SCRATCHDIR_, CDF_STATUS_, CDF_STATUSTEXT_LEN, CDF_TIME_TT2000, CDF_UCHAR, CDF_UINT1, CDF_UINT2, CDF_UINT4, CDF_VAR_NAME_LEN, CDF_VAR_NAME_LEN256, CDF_VERSION_, CDF_WARN, CDF_WRITE_ERROR, CDF_zMODE_, CDFwithSTATS_, CHECKSUM_, CHECKSUM_ERROR, CHECKSUM_NOT_ALLOWED, CLOSE_, COLUMN_MAJOR, COMPRESS_CACHESIZE_, COMPRESSION_ERROR, CONFIRM_, CORRUPTED_V2_CDF, CORRUPTED_V3_CDF, CREATE_, CURgENTRY_EXISTENCE_, CURrENTRY_EXISTENCE_, CURzENTRY_EXISTENCE_, DATATYPE_MISMATCH, DATATYPE_SIZE_, DECOMPRESSION_ERROR, DECSTATION_DECODING, DECSTATION_ENCODING, DEFAULT_BYTE_PADVALUE, DEFAULT_CHAR_PADVALUE, DEFAULT_DOUBLE_PADVALUE, DEFAULT_EPOCH_PADVALUE, DEFAULT_EPOCH16_PADVALUE, DEFAULT_FLOAT_PADVALUE, DEFAULT_INT1_PADVALUE,

DEFAULT_INT2_PADVALUE, DEFAULT_INT4_PADVALUE, DEFAULT_INT8_PADVALUE, DEFAULT_REAL4_PADVALUE, DEFAULT_REAL8_PADVALUE, DEFAULT_TT2000_PADVALUE, DEFAULT_UCHAR_PADVALUE, DEFAULT_UINT1_PADVALUE, DEFAULT_UINT2_PADVALUE, DEFAULT_UINT4_PADVALUE, DELETE_, DID_NOT_COMPRESS, EMPTY_COMPRESSED_CDF, END_OF_VAR, EPOCH_STRING_LEN, EPOCH_STRING_LEN_EXTEND, EPOCH1_STRING_LEN, EPOCH1_STRING_LEN_EXTEND, EPOCH2_STRING_LEN, EPOCH2_STRING_LEN_EXTEND, EPOCH3_STRING_LEN, EPOCH3_STRING_LEN_EXTEND, EPOCH4_STRING_LEN, EPOCH4_STRING_LEN_EXTEND, EPOCHx_FORMAT_MAX, EPOCHx_STRING_MAX, FILLED_TT2000_VALUE, FORCED_PARAMETER, FUNCTION_NOT_SUPPORTED, gENTRY_, gENTRY_DATA_, gENTRY_DATASPEC_, gENTRY_DATATYPE_, gENTRY_EXISTENCE_, gENTRY_NUMELEMS_, GET_, GETCDFCHECKSUM_, GETCDFFILEBACKWARD_, GETCDFVALIDATE_, GETLEAPSECONDSENVVAR_, GLOBAL_SCOPE, GZIP_COMPRESSION, HOST_DECODING, HOST_ENCODING, HP_DECODING, HP_ENCODING, HUFF_COMPRESSION, IA64VMSd_DECODING, IA64VMSd_ENCODING, IA64VMSg_DECODING, IA64VMSg_ENCODING, IA64VMSi_DECODING, IA64VMSi_ENCODING, IBM_PC_OVERFLOW, IBMPC_DECODING, IBMPC_ENCODING, IBMRS_DECODING, IBMRS_ENCODING, ILLEGAL_EPOCH_FIELD, ILLEGAL_EPOCH_VALUE, ILLEGAL_FOR_SCOPE, ILLEGAL_IN_zMODE, ILLEGAL_ON_V1_CDF, ILLEGAL_TT2000_VALUE, IS_A_NETCDF, LIB_COPYRIGHT_, LIB_INCREMENT_, LIB_RELEASE_, LIB_subINCREMENT_, LIB_VERSION_, MAC_DECODING, MAC_ENCODING, MD5_CHECKSUM, MULTI_FILE, MULTI_FILE_FORMAT, NA_FOR_VARIABLE, NEGATIVE_FP_ZERO, NEGtoPOSfp0off, NEGtoPOSfp0on, NETWORK_DECODING, NETWORK_ENCODING, NeXT_DECODING, NeXT_ENCODING, NO_ATTR_SELECTED, NO_CDF_SELECTED, NO_CHECKSUM, NO_COMPRESSION, NO_DELETE_ACCESS, NO_ENTRY_SELECTED, NO_MORE_ACCESS, NO_PADVALUE_SPECIFIED, NO_SPARSEARRAYS, NO_SPARSERECORDS, NO_STATUS_SELECTED, NO_SUCH_ATTR, NO_SUCH_CDF, NO_SUCH_ENTRY, NO_SUCH_RECORD, NO_SUCH_VAR, NO_VAR_SELECTED, NO_VARS_IN_CDF, NO_WRITE_ACCESS, NONE_CHECKSUM, NOT_A_CDF, NOT_A_CDF_OR_NOT_SUPPORTED, NOVARY, NULL_, OPEN_, OPTIMAL_ENCODING_TREES, OTHER_CHECKSUM, PAD_SPARSERECORDS, PPC_DECODING, PPC_ENCODING, PRECEEDING_RECORDS_ALLOCATED, PREV_SPARSERECORDS, PUT_, READ_ONLY_DISTRIBUTION, READ_ONLY_MODE, READONLYoff, READONLYon, rENTRY_, rENTRY_DATA_, rENTRY_DATASPEC_, rENTRY_DATATYPE_, rENTRY_EXISTENCE_, rENTRY_NAME_, rENTRY_NUMELEMS_, rENTRY_NUMSTRINGS_, rENTRY_STRINGSDATA_, RLE_COMPRESSION, RLE_OF_ZEROs, ROW_MAJOR, rVAR_, rVAR_ALLOCATEBLOCK_, rVAR_ALLOCATEDFROM_, rVAR_ALLOCATEDTO_, rVAR_ALLOCATERECS_, rVAR_BLOCKINGFACTOR_, rVAR_CACHESIZE_, rVAR_COMPRESSION_, rVAR_DATA_, rVAR_DATASPEC_, rVAR_DATATYPE_, rVAR_DIMVARYS_, rVAR_EXISTENCE_, rVAR_HYPERDATA_, rVAR_INITIALRECS_, rVAR_MAXallocREC_, rVAR_MAXREC_, rVAR_NAME_, rVAR_nINDEXENTRIES_, rVAR_nINDEXLEVELS_, rVAR_nINDEXRECORDS_, rVAR_NUMallocRECS_, rVAR_NUMBER_, rVAR_NUMELEMS_, rVAR_NUMRECS_, rVAR_PADVALUE_, rVAR_RECORDS_, rVAR_RECORDS_RENUMBER_, rVAR_RECVARY_, rVAR_RESERVEPERCENT_, rVAR_SEQDATA_, rVAR_SEQPOS_, rVAR_SPARSEARRAYS_, rVAR_SPARSERECORDS_, rVARs_CACHESIZE_, rVARs_DIMCOUNTS_, rVARs_DIMINDICES_, rVARs_DIMINTERVALS_, rVARs_DIMSIZES_, rVARs_MAXREC_, rVARs_NUMDIMS_, rVARs_RECCOUNT_, rVARs_RECDATA_, rVARs_RECINTERVAL_, rVARs_RECNUMBER_, SAVE_, SCRATCH_CREATE_ERROR, SCRATCH_DELETE_ERROR, SCRATCH_READ_ERROR, SCRATCH_WRITE_ERROR, SELECT_, SGI_DECODING, SGi_ENCODING, SINGLE_FILE, SINGLE_FILE_FORMAT, SOME_ALREADY_ALLOCATED, STAGE_CACHESIZE_, STATUS_TEXT_, STRING_NOT_UTF8_ENCODING, STRINGDELIMITER, SUN_DECODING, SUN_ENCODING, TOO_MANY_PARMS, TOO_MANY_VARS, TRY_TO_READ_NONSTRING_DATA, TT2000_0_STRING_LEN, TT2000_1_STRING_LEN, TT2000_2_STRING_LEN, TT2000_3_STRING_LEN, TT2000_4_STRING_LEN, TT2000_CDF_MAYNEEDUPDATE, TT2000_TIME_ERROR, TT2000_USED_OUTDATED_TABLE, UNABLE_TO_PROCESS_CDF, UNKNOWN_COMPRESSION, UNKNOWN_SPARSENESS, UNSUPPORTED_OPERATION, VALIDATE_, VALIDATEFILEoff, VALIDATEFILEon, VAR_ALREADY_CLOSED, VAR_CLOSE_ERROR, VAR_CREATE_ERROR, VAR_DELETE_ERROR, VAR_EXISTS, VAR_NAME_TRUNC, VAR_OPEN_ERROR, VAR_READ_ERROR, VAR_SAVE_ERROR, VAR_WRITE_ERROR, VARIABLE_SCOPE, VARY, VAX_DECODING, VAX_ENCODING, VIRTUAL_RECORD_DATA, zENTRY_, zENTRY_DATA_, zENTRY_DATASPEC_, zENTRY_DATATYPE_, zENTRY_EXISTENCE_, zENTRY_NAME_, zENTRY_NUMELEMS_, zENTRY_NUMSTRINGS_, zENTRY_STRINGSDATA_, ZLIB_COMPRESS_ERROR, ZLIB_UNCOMPRESS_ERROR, zMODEoff, zMODEon1, zMODEon2, zVAR_, zVAR_ALLOCATEBLOCK_, zVAR_ALLOCATEDFROM_, zVAR_ALLOCATEDTO_, zVAR_ALLOCATERECS_, zVAR_BLOCKINGFACTOR_, zVAR_CACHESIZE_, zVAR_COMPRESSION_, zVAR_DATA_, zVAR_DATASPEC_, zVAR_DATATYPE_, zVAR_DIMCOUNTS_, zVAR_DIMINDICES_, zVAR_DIMINTERVALS_, zVAR_DIMSIZES_, zVAR_DIMVARYS_, zVAR_EXISTENCE_, zVAR_HYPERDATA_, zVAR_INITIALRECS_, zVAR_MAXallocREC_, zVAR_MAXREC_, zVAR_NAME_, zVAR_nINDEXENTRIES_, zVAR_nINDEXLEVELS_, zVAR_nINDEXRECORDS_, zVAR_NUMallocRECS_, zVAR_NUMBER_, zVAR_NUMDIMS_, zVAR_NUMELEMS_, zVAR_NUMRECS_, zVAR_PADVALUE_, zVAR_RECCOUNT_, zVAR_RECINTERVAL_, zVAR_RECNUMBER_, zVAR_RECORDS_, zVAR_RECORDS_RENUMBER_, zVAR_RECVARY_, zVAR_RESERVEPERCENT_, zVAR_SEQDATA_, zVAR_SEQPOS_, zVAR_SPARSEARRAYS_, zVAR_SPARSERECORDS_, zVARs_CACHESIZE_, zVARs_MAXREC_, zVARs_RECDATA_, zVARs_RECNUMBER_

Method Summary

Methods

Modifier and Type	Method and Description
void	**delete**() See the description of the getName() method in this class.
void	**dump**() Dump data information and values, one row at a time, to the stdErr.
void	**dumpData**() Dumps variable data, one row at a time per record.
void	**dumpDataWithFormat**(java.lang.String format) Dumps variable data, one row at a time per record, with format.
java.lang.Object	**getData**() Returns an object that is properly dimensioned.
long[]	**getDimCounts**() Gets the value of the dimension counts that represents the number of elements read or write starting at the location <dimension index> for a hyper get/put function.

long[]	**getDimIndices**() Gets the starting dimension index within a record for a hyper get/put function.
long[]	**getDimIntervals**() Gets the value of the dimension intervals that represent the number of elements to skip between reads or writes for a hyper get/put function.
int[]	**getDimSizes**() Gets the dimension sizes of this variable.
java.lang.String	**getName**() CDFData implements CDFObject to enable CDFDelegate calls.
int	**getnDims**() Gets the dimensionality of this variable.
java.lang.Object	**getRawData**() Returns an object of a 1-dimensional array, which presents a sequence of raw data values retrieved and presented by JNI from a CDF file.
long	**getRecCount**() Gets the number of records to read or write for a hyper get/put function.
long	**getRecInterval**() Gets the number of records to skip for a hyper get/put function.
long	**getRecStart**() Gets the record number at which a hyper get/put function starts.
void	**rename**(java.lang.String name) See the description of the getName() method in this class.

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Method Detail

getData

```
public java.lang.Object getData()
```

Returns an object that is properly dimensioned. The returned object can be casted in an application for usage or further manipulation.

The following example retrieves the Temperature data. The user should know how the data was stored before casting the generic object to a variable.

```
    Variable var = cdf.getVariable("Temperature");
    CDFData data = var.getHyperDataObject (recNum,
                                           recCount,
                                           recInterval,
                                           dimIndicies,
                                           dimSizes,
                                           dimCounts);
    long[][] temperature = (long [][]) data.getData();
```

Returns:

a generic Object that is properly dimensioned

getRawData

```
public java.lang.Object getRawData()
```

Returns an object of a 1-dimensional array, which presents a sequence of raw data values retrieved and presented by JNI from a CDF file. The data stream may or may not reflect how the data is stored in the file, depending the file majority. It is up to the calling application to decipher the data stream. Note: for column-major CDF files, the sequence of the returned data stream is reordered so data can be properly assigned into Java's arrays. Normally, getData method should be called so the retrieved data are properly dimensioned.

The following example retrieves the 2-D Temperature data. The user should know how to organize the data, e.g., number of records, row/column major, data type, data value sequence, etc.

```
    Variable var = cdf.getVariable("Temperature");
    CDFData data = var.getHyperDataObject (recNum,
                                           recCount,
                                           recInterval,
                                           dimIndicies,
                                           dimSizes,
                                           dimCounts);
    long[] temperature = (long []) data.getRawData();
```

Returns:

a generic Object that is properly dimensioned

getnDims

```
public int getnDims()
```

Gets the dimensionality of this variable.

```
    Variable var = cdf.getVariable("Temperature");
    CDFData data = var.getHyperDataObject (recNum,
                                           recCount,
                                           recInterval,
                                           dimIndicies,
                                           dimSizes,
                                           dimCounts);
    long[][] temperature = (long [][]) data.getData();
    nDims = data.getnDims();   // Gives the dimenionality of temperature
```

Returns:

the dimensionality of this variable

getDimSizes

```
public int[] getDimSizes()
```

Gets the dimension sizes of this variable. For example, 3 X 10 (3 rows and 10 columns) two-dimentional array is returned as an one-dimensional integer array, containing 3 in the first element and 10 in the second element.

Returns:

the dimension sizes of this variable

getRecStart

```
public long getRecStart()
```

Gets the record number at which a hyper get/put function starts.

Returns:

the starting record number for a hyper get/put function

getRecCount

```
public long getRecCount()
```

Gets the number of records to read or write for a hyper get/put function.

Returns:

the number of records involved for a hyper get/put function involves

getRecInterval

```
public long getRecInterval()
```

Gets the number of records to skip for a hyper get/put function. The record interval of 1 represents every record. The value of 2 represents every other record, the value of 3 represents every third record and so on.

Returns:

the value of record interval

getDimIndices

```
public long[] getDimIndices()
```

Gets the starting dimension index within a record for a hyper get/put function. Dimension index indicates where the data search started from within a record. Let's say a record is comprised of a 2x5 two-dimensional array (2 rows and 5 columns). If the index returned from this method has a value of {1,0}, then the data search was performed starting at the first element of the second row. Similarly, the value of {0,0} represents that the data search search was performed starting at the first element of the first record.

Returns:

the dimension index for this variable

getDimCounts

```
public long[] getDimCounts()
```

Gets the value of the dimension counts that represents the number of elements read or write starting at the location <dimension index> for a hyper get/put function.

Returns:

the dimension counts for this variable

getDimIntervals

```
public long[] getDimIntervals()
```

Gets the value of the dimension intervals that represent the number of elements to skip between reads or writes for a hyper get/put function. The value of 1 represents every element. The value of 2 represents every other element, and the value of 3 represents every third element and so on.

Returns:

the dimension intervals for this variable

dumpData

```
public void dumpData()
```

Dumps variable data, one row at a time per record. This is a generic utility for dumping data to a screen. Data can be scalar or 1-dimensional or multi-dimensional array of any data type. This method dumps the data without using format.

The following example retrieves the first record, comprised of 3x5 (3 rows and 5 columns) array, into a generic object and dumps its contents to screen one row at a time. In this case three rows will be displayed on a screen, each row containing 5 elements.

```
CDFData data;
long[] dimIndices   = {0,0};
long[] dimIntervals = {3,5};
long[] dimSizes     = {1,1};
data = var.getHyperDataObject(0L,          // record start
                              1,           // record counts
                              1,           // record interval
                              dimIndices,
                              dimSizes,
                              dimIntervals);
data.dumpData();
```

dumpDataWithFormat

```
public void dumpDataWithFormat(java.lang.String format)
```

Dumps variable data, one row at a time per record, with format. This is a generic * utility for dumping data to a screen. Data can be scalar or 1-dimensional or multi-dimensional array of any data type. The format is only applicable to variables of numerical data type. the format comes from variable's FORMAT attribute entry. Note: The FORMAT might be in original C or Fortran format form. This format is mainly used for floating point data. The following example retrieves the first record, comprised of 3x5 (3 rows and 5 columns) array, into a generic object and dumps its contents to screen one row at a time. In this case three rows will be displayed on a screen, each row containing 5 elements.

```
CDFData data;
String format = (String) var.getEntryData("FORMAT");
long[] dimIndices   = {0,0};
long[] dimIntervals = {3,5};
long[] dimSizes     = {1,1};
data = var.getHyperDataObject(0L,          // record start
                              1,           // record counts
                              1,           // record interval
                              dimIndices,
                              dimSizes,
                              dimIntervals);
data.dumpDataWithFormat(format);
```

Parameters:

`format` - A string format

dump

```
public void dump()
```

Dump data information and values, one row at a time, to the stdErr. This method is provided for debugging purposes only. The information is printed in the following manner: <VAR_TYPE>/<VAR_#Elements> nDims:[sizes] recStart/recCount/recInterval/dimIndices/dimsSizes/dimIntervals/dataArraySignature <values row-by-row>

getName

```
public java.lang.String getName()
```

CDFData implements CDFObject to enable CDFDelegate calls. CDFObject specifies the following three methods: getName(), rename(String), and delete(). Since CDFData implements CDFObject, it must have the methods defined in CDFObject. That's why this method is here; it doesn't do anything.

Specified by:

`getName` in interface `CDFObject`

Returns:

the name of the current object

rename

```
public void rename(java.lang.String name)
            throws CDFException
```

See the description of the getName() method in this class.

Specified by:

`rename` in interface `CDFObject`

Parameters:

`name` - the new object name

Throws:

`CDFException` - No exception is thrown since this method is a placeholder

delete

```
public void delete()
            throws CDFException
```

See the description of the getName() method in this class.

Specified by:

`delete` in interface `CDFObject`

Throws:

`CDFException` - No exception is thrown since this method is a placeholder

gsfc.nssdc.cdf

Interface CDFDelegate

All Known Implementing Classes:

CDFNativeLibrary

```
public interface CDFDelegate
```

This class defines the method that is responsible for acting as the gateway between the CDF Java code and the CDF library. The CDFNativeLibrary class that implementing this interface will cause the JNI to be loaded. This class is available only to the CDF object that uses the CDFDelegate to make requests to JNI. All CDF's other objects, i.e., Attribute, Entry, Variable (and its CDFData), need to refer to the containing CDF object to make requests.

Version:

1.0

See Also:

CDFNativeLibrary

Method Summary

Methods

Modifier and Type	Method and Description
void	**cdflib**(**CDF** theCDF, **CDFObject** cdfObject, java.util.Vector cmds) Defines the method that is responsible for acting as the gateway between the CDF Java code and the CDF library.

Method Detail

cdflib

```
void cdflib(CDF theCDF,
            CDFObject cdfObject,
            java.util.Vector cmds)
              throws CDFException
```

Defines the method that is responsible for acting as the gateway between the CDF Java code and the CDF library. This method is responsible for sending Java's request to the CDF library and returning the results from the CDF library to the Java side.

Parameters:

`theCDF` - the current CDF to be processed

`cdfObject` - the calling CDF object (e.g. Attribute, variable, etc.)

`cmds` - a Vector that contains the CDF internal interface library commands to be executed

Throws:

`CDFException` - if an error occurs processing the requested commands in JNI

gsfc.nssdc.cdf

Class CDFException

java.lang.Object
  java.lang.Throwable
    java.lang.Exception
      gsfc.nssdc.cdf.CDFException

All Implemented Interfaces:

CDFConstants, java.io.Serializable

```
public class CDFException
extends java.lang.Exception
implements CDFConstants
```

This class defines the informational, warning, and error messages that can arise from CDF operations.

See Also:

Serialized Form

Field Summary

Fields inherited from interface gsfc.nssdc.cdf.CDFConstants

AHUFF_COMPRESSION, ALPHAOSF1_DECODING, ALPHAOSF1_ENCODING, ALPHAVMSd_DECODING, ALPHAVMSd_ENCODING, ALPHAVMSg_DECODING, ALPHAVMSg_ENCODING, ALPHAVMSi_DECODING, ALPHAVMSi_ENCODING, ARM_BIG_DECODING, ARM_BIG_ENCODING, ARM_LITTLE_DECODING, ARM_LITTLE_ENCODING, ATTR_, ATTR_EXISTENCE_, ATTR_EXISTS, ATTR_MAXgENTRY_, ATTR_MAXrENTRY_, ATTR_MAXzENTRY_, ATTR_NAME_, ATTR_NAME_TRUNC, ATTR_NUMBER_, ATTR_NUMgENTRIES_, ATTR_NUMrENTRIES_, ATTR_NUMzENTRIES_, ATTR_SCOPE_, BACKWARD_, BACKWARDFILEoff, BACKWARDFILEon, BAD_ALLOCATE_RECS, BAD_ARGUMENT, BAD_ATTR_NAME, BAD_ATTR_NUM, BAD_BLOCKING_FACTOR, BAD_CACHE_SIZE, BAD_CDF_EXTENSION, BAD_CDF_ID, BAD_CDF_NAME, BAD_CDFSTATUS, BAD_CHECKSUM, BAD_COMPRESSION_PARM, BAD_DATA_TYPE, BAD_DECODING, BAD_DIM_COUNT, BAD_DIM_INDEX, BAD_DIM_INTERVAL, BAD_DIM_SIZE, BAD_ENCODING, BAD_ENTRY_NUM, BAD_FNC_OR_ITEM, BAD_FORMAT, BAD_INITIAL_RECS, BAD_MAJORITY, BAD_MALLOC, BAD_NEGtoPOSfp0_MODE, BAD_NUM_DIMS, BAD_NUM_ELEMS, BAD_NUM_STRINGS, BAD_NUM_VARS, BAD_READONLY_MODE, BAD_REC_COUNT, BAD_REC_INTERVAL, BAD_REC_NUM, BAD_SCOPE, BAD_SCRATCH_DIR, BAD_SPARSEARRAYS_PARM, BAD_VAR_NAME, BAD_VAR_NUM, BAD_zMODE, BADDATE_LEAPSECOND_UPDATED, BeginUnixTimeEPOCH, BeginUnixTimeEPOCH16, BLOCKINGFACTOR_TOO_LARGE, BLOCKINGFACTOR_TOO_SMALL, BLOCKINGFACTOR_TOO_SMALL2, CANNOT_ALLOCATE_RECORDS, CANNOT_CHANGE, CANNOT_COMPRESS, CANNOT_CONVERT_WIDECHAR, CANNOT_COPY, CANNOT_INSERT_RECORDS, CANNOT_SPARSEARRAYS, CANNOT_SPARSERECORDS, CDF_, CDF_ACCESS_, CDF_ATTR_NAME_LEN, CDF_ATTR_NAME_LEN256, CDF_BYTE, CDF_CACHESIZE_, CDF_CHAR, CDF_CHECKSUM_, CDF_CLOSE_ERROR, CDF_COMPRESSION_, CDF_COPYRIGHT_, CDF_COPYRIGHT_LEN, CDF_CREATE_ERROR, CDF_DECODING_, CDF_DELETE_ERROR, CDF_DOUBLE, CDF_ENCODING_, CDF_EPOCH, CDF_EPOCH16, CDF_EXISTS, CDF_FLOAT, CDF_FORMAT_, CDF_INCREMENT_, CDF_INFO_, CDF_INT1, CDF_INT2, CDF_INT4, CDF_INT8, CDF_INTERNAL_ERROR, CDF_LEAPSECONDLASTUPDATED_, CDF_MAJORITY_, CDF_MAX_DIMS, CDF_MAX_PARMS, CDF_MIN_DIMS, CDF_NAME_, CDF_NAME_TRUNC, CDF_NEGtoPOSfp0_MODE_, CDF_NUMATTRS_, CDF_NUMgATTRS_, CDF_NUMrVARS_, CDF_NUMvATTRS_, CDF_NUMzVARS_, CDF_OK, CDF_OPEN_ERROR, CDF_PATHNAME_LEN, CDF_READ_ERROR, CDF_READONLY_MODE_, CDF_REAL4, CDF_REAL8, CDF_RELEASE_, CDF_SAVE_ERROR, CDF_SCRATCHDIR_, CDF_STATUS_, CDF_STATUSTEXT_LEN, CDF_TIME_TT2000, CDF_UCHAR, CDF_UINT1, CDF_UINT2, CDF_UINT4, CDF_VAR_NAME_LEN, CDF_VAR_NAME_LEN256, CDF_VERSION_, CDF_WARN, CDF_WRITE_ERROR, CDF_zMODE_, CDFwithSTATS_, CHECKSUM_, CHECKSUM_ERROR, CHECKSUM_NOT_ALLOWED, CLOSE_, COLUMN_MAJOR, COMPRESS_CACHESIZE_, COMPRESSION_ERROR, CONFIRM_, CORRUPTED_V2_CDF, CORRUPTED_V3_CDF, CREATE_, CURgENTRY_EXISTENCE_, CURrENTRY_EXISTENCE_, CURzENTRY_EXISTENCE_, DATATYPE_MISMATCH, DATATYPE_SIZE_, DECOMPRESSION_ERROR, DECSTATION_DECODING, DECSTATION_ENCODING, DEFAULT_BYTE_PADVALUE, DEFAULT_CHAR_PADVALUE, DEFAULT_DOUBLE_PADVALUE, DEFAULT_EPOCH_PADVALUE, DEFAULT_EPOCH16_PADVALUE, DEFAULT_FLOAT_PADVALUE, DEFAULT_INT1_PADVALUE, DEFAULT_INT2_PADVALUE, DEFAULT_INT4_PADVALUE, DEFAULT_INT8_PADVALUE, DEFAULT_REAL4_PADVALUE, DEFAULT_REAL8_PADVALUE, DEFAULT_TT2000_PADVALUE, DEFAULT_UCHAR_PADVALUE, DEFAULT_UINT1_PADVALUE, DEFAULT_UINT2_PADVALUE, DEFAULT_UINT4_PADVALUE, DELETE_, DID_NOT_COMPRESS, EMPTY_COMPRESSED_CDF, END_OF_VAR, EPOCH_STRING_LEN, EPOCH_STRING_LEN_EXTEND, EPOCH1_STRING_LEN, EPOCH1_STRING_LEN_EXTEND, EPOCH2_STRING_LEN, EPOCH2_STRING_LEN_EXTEND, EPOCH3_STRING_LEN, EPOCH3_STRING_LEN_EXTEND, EPOCH4_STRING_LEN, EPOCH4_STRING_LEN_EXTEND, EPOCHx_FORMAT_MAX, EPOCHx_STRING_MAX, FILLED_TT2000_VALUE, FORCED_PARAMETER, FUNCTION_NOT_SUPPORTED, gENTRY_, gENTRY_DATA_, gENTRY_DATASPEC_, gENTRY_DATATYPE_, gENTRY_EXISTENCE_, gENTRY_NUMELEMS_, GET_, GETCDFCHECKSUM_, GETCDFFILEBACKWARD_, GETCDFVALIDATE_, GETLEAPSECONDSENVVAR_, GLOBAL_SCOPE, GZIP_COMPRESSION, HOST_DECODING, HOST_ENCODING, HP_DECODING, HP_ENCODING, HUFF_COMPRESSION,

IA64VMSd_DECODING, IA64VMSd_ENCODING, IA64VMSg_DECODING, IA64VMSg_ENCODING, IA64VMSi_DECODING, IA64VMSi_ENCODING, IBM_PC_OVERFLOW, IBMPC_DECODING, IBMPC_ENCODING, IBMRS_DECODING, IBMRS_ENCODING, ILLEGAL_EPOCH_FIELD, ILLEGAL_EPOCH_VALUE, ILLEGAL_FOR_SCOPE, ILLEGAL_IN_zMODE, ILLEGAL_ON_V1_CDF, ILLEGAL_TT2000_VALUE, IS_A_NETCDF, LIB_COPYRIGHT_, LIB_INCREMENT_, LIB_RELEASE_, LIB_subINCREMENT_, LIB_VERSION_, MAC_DECODING, MAC_ENCODING, MD5_CHECKSUM, MULTI_FILE, MULTI_FILE_FORMAT, NA_FOR_VARIABLE, NEGATIVE_FP_ZERO, NEGtoPOSfp0off, NEGtoPOSfp0on, NETWORK_DECODING, NETWORK_ENCODING, NeXT_DECODING, NeXT_ENCODING, NO_ATTR_SELECTED, NO_CDF_SELECTED, NO_CHECKSUM, NO_COMPRESSION, NO_DELETE_ACCESS, NO_ENTRY_SELECTED, NO_MORE_ACCESS, NO_PADVALUE_SPECIFIED, NO_SPARSEARRAYS, NO_SPARSERECORDS, NO_STATUS_SELECTED, NO_SUCH_ATTR, NO_SUCH_CDF, NO_SUCH_ENTRY, NO_SUCH_RECORD, NO_SUCH_VAR, NO_VAR_SELECTED, NO_VARS_IN_CDF, NO_WRITE_ACCESS, NONE_CHECKSUM, NOT_A_CDF, NOT_A_CDF_OR_NOT_SUPPORTED, NOVARY, NULL_, OPEN_, OPTIMAL_ENCODING_TREES, OTHER_CHECKSUM, PAD_SPARSERECORDS, PPC_DECODING, PPC_ENCODING, PRECEEDING_RECORDS_ALLOCATED, PREV_SPARSERECORDS, PUT_, READ_ONLY_DISTRIBUTION, READ_ONLY_MODE, READONLYoff, READONLYon, rENTRY_, rENTRY_DATA_, rENTRY_DATASPEC_, rENTRY_DATATYPE_, rENTRY_EXISTENCE_, rENTRY_NAME_, rENTRY_NUMELEMS_, rENTRY_NUMSTRINGS_, rENTRY_STRINGSDATA_, RLE_COMPRESSION, RLE_OF_ZEROs, ROW_MAJOR, rVAR_, rVAR_ALLOCATEBLOCK_, rVAR_ALLOCATEDFROM_, rVAR_ALLOCATEDTO_, rVAR_ALLOCATERECS_, rVAR_BLOCKINGFACTOR_, rVAR_CACHESIZE_, rVAR_COMPRESSION_, rVAR_DATA_, rVAR_DATASPEC_, rVAR_DATATYPE_, rVAR_DIMVARYS_, rVAR_EXISTENCE_, rVAR_HYPERDATA_, rVAR_INITIALRECS_, rVAR_MAXallocREC_, rVAR_MAXREC_, rVAR_NAME_, rVAR_nINDEXENTRIES_, rVAR_nINDEXLEVELS_, rVAR_nINDEXRECORDS_, rVAR_NUMallocRECS_, rVAR_NUMBER_, rVAR_NUMELEMS_, rVAR_NUMRECS_, rVAR_PADVALUE_, rVAR_RECORDS_, rVAR_RECORDS_RENUMBER_, rVAR_RECVARY_, rVAR_RESERVEPERCENT_, rVAR_SEQDATA_, rVAR_SEQPOS_, rVAR_SPARSEARRAYS_, rVAR_SPARSERECORDS_, rVARs_CACHESIZE_, rVARs_DIMCOUNTS_, rVARs_DIMINDICES_, rVARs_DIMINTERVALS_, rVARs_DIMSIZES_, rVARs_MAXREC_, rVARs_NUMDIMS_, rVARs_RECCOUNT_, rVARs_RECDATA_, rVARs_RECINTERVAL_, rVARs_RECNUMBER_, SAVE_, SCRATCH_CREATE_ERROR, SCRATCH_DELETE_ERROR, SCRATCH_READ_ERROR, SCRATCH_WRITE_ERROR, SELECT_, SGI_DECODING, SGi_ENCODING, SINGLE_FILE, SINGLE_FILE_FORMAT, SOME_ALREADY_ALLOCATED, STAGE_CACHESIZE_, STATUS_TEXT_, STRING_NOT_UTF8_ENCODING, STRINGDELIMITER, SUN_DECODING, SUN_ENCODING, TOO_MANY_PARMS, TOO_MANY_VARS, TRY_TO_READ_NONSTRING_DATA, TT2000_0_STRING_LEN, TT2000_1_STRING_LEN, TT2000_2_STRING_LEN, TT2000_3_STRING_LEN, TT2000_4_STRING_LEN, TT2000_CDF_MAYNEEDUPDATE, TT2000_TIME_ERROR, TT2000_USED_OUTDATED_TABLE, UNABLE_TO_PROCESS_CDF, UNKNOWN_COMPRESSION, UNKNOWN_SPARSENESS, UNSUPPORTED_OPERATION, VALIDATE_, VALIDATEFILEoff, VALIDATEFILEon, VAR_ALREADY_CLOSED, VAR_CLOSE_ERROR, VAR_CREATE_ERROR, VAR_DELETE_ERROR, VAR_EXISTS, VAR_NAME_TRUNC, VAR_OPEN_ERROR, VAR_READ_ERROR, VAR_SAVE_ERROR, VAR_WRITE_ERROR, VARIABLE_SCOPE, VARY, VAX_DECODING, VAX_ENCODING, VIRTUAL_RECORD_DATA, zENTRY_, zENTRY_DATA_, zENTRY_DATASPEC_, zENTRY_DATATYPE_, zENTRY_EXISTENCE_, zENTRY_NAME_, zENTRY_NUMELEMS_, zENTRY_NUMSTRINGS_, zENTRY_STRINGSDATA_, ZLIB_COMPRESS_ERROR, ZLIB_UNCOMPRESS_ERROR, zMODEoff, zMODEon1, zMODEon2, zVAR_, zVAR_ALLOCATEBLOCK_, zVAR_ALLOCATEDFROM_, zVAR_ALLOCATEDTO_, zVAR_ALLOCATERECS_, zVAR_BLOCKINGFACTOR_, zVAR_CACHESIZE_, zVAR_COMPRESSION_, zVAR_DATA_, zVAR_DATASPEC_, zVAR_DATATYPE_, zVAR_DIMCOUNTS_, zVAR_DIMINDICES_, zVAR_DIMINTERVALS_, zVAR_DIMSIZES_, zVAR_DIMVARYS_, zVAR_EXISTENCE_, zVAR_HYPERDATA_, zVAR_INITIALRECS_, zVAR_MAXallocREC_, zVAR_MAXREC_, zVAR_NAME_, zVAR_nINDEXENTRIES_, zVAR_nINDEXLEVELS_, zVAR_nINDEXRECORDS_, zVAR_NUMallocRECS_, zVAR_NUMBER_, zVAR_NUMDIMS_, zVAR_NUMELEMS_, zVAR_NUMRECS_, zVAR_PADVALUE_, zVAR_RECCOUNT_, zVAR_RECINTERVAL_, zVAR_RECNUMBER_, zVAR_RECORDS_, zVAR_RECORDS_RENUMBER_, zVAR_RECVARY_, zVAR_RESERVEPERCENT_, zVAR_SEQDATA_, zVAR_SEQPOS_, zVAR_SPARSEARRAYS_, zVAR_SPARSERECORDS_, zVARs_CACHESIZE_, zVARs_MAXREC_, zVARs_RECDATA_, zVARs_RECNUMBER_

Constructor Summary

Constructors

Constructor and Description
CDFException(long statusCode) Takes a status code and throws a CDFException with the message that corresponds to the status code that is passed in.
CDFException(long statusCode, java.lang.String where) Takes a status code and throws a CDFException with the message that corresponds to the status code that is passed in.
CDFException(java.lang.String message) Takes a text message from the calling program and throws a CDFException.

Method Summary

Methods

Modifier and Type	Method and Description
long	**getCurrentStatus**() Gets the status code that caused CDFException.
static java.lang.String	**getStatusMsg**(long statusCode) Get the status text message for the given status code.

Methods inherited from class java.lang.Throwable

addSuppressed, fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, getSuppressed, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

CDFException

```
public CDFException(java.lang.String message)
```

Takes a text message from the calling program and throws a CDFException.

Parameters:

`message` - the message to be thrown with CDFException

CDFException

```
public CDFException(long statusCode)
```

Takes a status code and throws a CDFException with the message that corresponds to the status code that is passed in.

Parameters:

`statusCode` - the CDF statusCode to be thrown

CDFException

```
public CDFException(long statusCode,
            java.lang.String where)
```

Takes a status code and throws a CDFException with the message that corresponds to the status code that is passed in. It also specifies where (which routine) the problem was.

Parameters:

`statusCode` - the CDF statusCode to be thrown

`where` - the place (routine/method) where the problem occurred

Method Detail

getCurrentStatus

```
public long getCurrentStatus()
```

Gets the status code that caused CDFException. This method comes in handy when there are times one may want to examine the cause of the CDFException and determine whether to continue or not.

```
try {
     ...

} catch (CDFException e) {
      if (e.getCurrentStatus() == NO_SUCH_VAR) {
          Variable latitude = Variable.create(cdf, "Latitude",
                                              CDF_INT1,
                                              numElements,
                                              numDims,
                                              dimSizes,
                                              recVary,
                                              dimVary);
          ...
      }
      else {
         System.out.println ("StatusCode = "+e.getCurrentStatus());
         e.printStackTrace();
      }
}
```

Returns:

the status code that caused CDFException

getStatusMsg

```
public static java.lang.String getStatusMsg(long statusCode)
```

Get the status text message for the given status code.

Parameters:

`statusCode` - the status code from which the status text is retrieved

Returns:

the status text message for the given status code

gsfc.nssdc.cdf

Class CDFNativeLibrary

java.lang.Object
gsfc.nssdc.cdf.CDFNativeLibrary

All Implemented Interfaces:

CDFDelegate

```
public class CDFNativeLibrary
extends java.lang.Object
implements CDFDelegate
```

This class implements the method that act as the gateway between the CDF Java APIs and the CDF library.

Version:

Version 1.0

Constructor Summary

Constructors

Constructor and Description
`CDFNativeLibrary()`

Method Summary

Methods

Modifier and Type	Method and Description
void	`cdflib(CDF theCDF, CDFObject cdfObject, java.util.Vector cmds)` Calls the Java Native Interface (JNI) program, cdfNativeLibrary.c.

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

CDFNativeLibrary

```
public CDFNativeLibrary()
```

Method Detail

cdflib

```
public void cdflib(CDF theCDF,
          CDFObject cdfObject,
          java.util.Vector cmds)
            throws CDFException
```

Calls the Java Native Interface (JNI) program, cdfNativeLibrary.c. This method is internal and called by various core CDF Java programs.

End users should never call this method from their applications.

Specified by:

`cdflib` in interface `CDFDelegate`

Parameters:

`theCDF` - the CDF being dealt with

`cdfObject` - the calling program/object (e.g. Variable.java, Attribute.java, etc.)

`cmds` - a vector that contains the CDFlib commands to be executed

Throws:

`CDFException` - if a problem occurs while executing the requested CDFlib commands in cdfNativeLibrary.c.

gsfc.nssdc.cdf

Interface CDFObject

All Known Implementing Classes:

Attribute, CDF, CDFData, Entry, Variable

```
public interface CDFObject
```

CDFObject provides the base interface for all CDF objects. CDF objects mean the CDF, Attribute, Entry and Variable objects. All these objects need to implement this interface.

Version:

1.0

Method Summary

Methods

Modifier and Type	Method and Description
void	**delete**() Deletes the current object.
java.lang.String	**getName**() Returns the name of the current object.
void	**rename**(java.lang.String name) Renames the current object.

Method Detail

getName

```
java.lang.String getName()
```

Returns the name of the current object.

Returns:

the name of the current object

rename

```
void rename(java.lang.String name)
            throws CDFException
```

Renames the current object.

Parameters:

`name` - the new object name

Throws:

`CDFException` - if an error occurs renaming the current object

delete

```
void delete()
            throws CDFException
```

Deletes the current object.

Throws:

`CDFException` - if an error occurs deleting the current object

gsfc.nssdc.cdf

Class CDFTools

java.lang.Object
gsfc.nssdc.cdf.CDFTools

All Implemented Interfaces:

CDFConstants

```
public class CDFTools
extends java.lang.Object
implements CDFConstants
```

CDFTools.java Created: Tue Nov 24 16:14:50 1998

Version:

$Id: CDFTools.java,v 1.1.1.1 2023/08/01 12:28:29 mhliu Exp $

Field Summary

Fields

Modifier and Type	Field and Description
static int	ALL_VALUES
static int	NAMED_VALUES
static int	NO_REPORTS
static int	NO_VALUES
static int	NRV_VALUES
static int	REPORT_ERRORS
static int	REPORT_INFORMATION
static int	REPORT_WARNINGS
static int	RV_VALUES

Fields inherited from interface gsfc.nssdc.cdf.CDFConstants

AHUFF_COMPRESSION, ALPHAOSF1_DECODING, ALPHAOSF1_ENCODING, ALPHAVMSd_DECODING, ALPHAVMSd_ENCODING, ALPHAVMSg_DECODING, ALPHAVMSg_ENCODING, ALPHAVMSi_DECODING, ALPHAVMSi_ENCODING, ARM_BIG_DECODING, ARM_BIG_ENCODING, ARM_LITTLE_DECODING, ARM_LITTLE_ENCODING, ATTR_, ATTR_EXISTENCE_, ATTR_EXISTS, ATTR_MAXgENTRY_, ATTR_MAXrENTRY_, ATTR_MAXzENTRY_, ATTR_NAME_, ATTR_NAME_TRUNC, ATTR_NUMBER_, ATTR_NUMgENTRIES_, ATTR_NUMrENTRIES_, ATTR_NUMzENTRIES_, ATTR_SCOPE_, BACKWARD_, BACKWARDFILEoff, BACKWARDFILEon, BAD_ALLOCATE_RECS, BAD_ARGUMENT, BAD_ATTR_NAME, BAD_ATTR_NUM, BAD_BLOCKING_FACTOR, BAD_CACHE_SIZE, BAD_CDF_EXTENSION, BAD_CDF_ID, BAD_CDF_NAME, BAD_CDFSTATUS, BAD_CHECKSUM, BAD_COMPRESSION_PARM, BAD_DATA_TYPE, BAD_DECODING, BAD_DIM_COUNT, BAD_DIM_INDEX, BAD_DIM_INTERVAL, BAD_DIM_SIZE, BAD_ENCODING, BAD_ENTRY_NUM, BAD_FNC_OR_ITEM, BAD_FORMAT, BAD_INITIAL_RECS, BAD_MAJORITY, BAD_MALLOC, BAD_NEGtoPOSfp0_MODE, BAD_NUM_DIMS, BAD_NUM_ELEMS, BAD_NUM_STRINGS, BAD_NUM_VARS, BAD_READONLY_MODE, BAD_REC_COUNT, BAD_REC_INTERVAL, BAD_REC_NUM, BAD_SCOPE, BAD_SCRATCH_DIR, BAD_SPARSEARRAYS_PARM, BAD_VAR_NAME, BAD_VAR_NUM, BAD_zMODE, BADDATE_LEAPSECOND_UPDATED, BeginUnixTimeEPOCH, BeginUnixTimeEPOCH16, BLOCKINGFACTOR_TOO_LARGE, BLOCKINGFACTOR_TOO_SMALL, BLOCKINGFACTOR_TOO_SMALL2, CANNOT_ALLOCATE_RECORDS, CANNOT_CHANGE, CANNOT_COMPRESS, CANNOT_CONVERT_WIDECHAR, CANNOT_COPY, CANNOT_INSERT_RECORDS, CANNOT_SPARSEARRAYS, CANNOT_SPARSERECORDS, CDF_, CDF_ACCESS_, CDF_ATTR_NAME_LEN, CDF_ATTR_NAME_LEN256, CDF_BYTE, CDF_CACHESIZE_, CDF_CHAR, CDF_CHECKSUM_, CDF_CLOSE_ERROR, CDF_COMPRESSION_, CDF_COPYRIGHT_, CDF_COPYRIGHT_LEN, CDF_CREATE_ERROR, CDF_DECODING_, CDF_DELETE_ERROR, CDF_DOUBLE, CDF_ENCODING_, CDF_EPOCH, CDF_EPOCH16, CDF_EXISTS, CDF_FLOAT, CDF_FORMAT_, CDF_INCREMENT_, CDF_INFO_, CDF_INT1,

CDF_INT2, CDF_INT4, CDF_INT8, CDF_INTERNAL_ERROR, CDF_LEAPSECONDLASTUPDATED_, CDF_MAJORITY_, CDF_MAX_DIMS, CDF_MAX_PARMS, CDF_MIN_DIMS, CDF_NAME_, CDF_NAME_TRUNC, CDF_NEGtoPOSfp0_MODE_, CDF_NUMATTRS_, CDF_NUMgATTRS_, CDF_NUMrVARS_, CDF_NUMvATTRS_, CDF_NUMzVARS_, CDF_OK, CDF_OPEN_ERROR, CDF_PATHNAME_LEN, CDF_READ_ERROR, CDF_READONLY_MODE_, CDF_REAL4, CDF_REAL8, CDF_RELEASE_, CDF_SAVE_ERROR, CDF_SCRATCHDIR_, CDF_STATUS_, CDF_STATUSTEXT_LEN, CDF_TIME_TT2000, CDF_UCHAR, CDF_UINT1, CDF_UINT2, CDF_UINT4, CDF_VAR_NAME_LEN, CDF_VAR_NAME_LEN256, CDF_VERSION_, CDF_WARN, CDF_WRITE_ERROR, CDF_zMODE_, CDFwithSTATS_, CHECKSUM_, CHECKSUM_ERROR, CHECKSUM_NOT_ALLOWED, CLOSE_, COLUMN_MAJOR, COMPRESS_CACHESIZE_, COMPRESSION_ERROR, CONFIRM_, CORRUPTED_V2_CDF, CORRUPTED_V3_CDF, CREATE_, CURgENTRY_EXISTENCE_, CURrENTRY_EXISTENCE_, CURzENTRY_EXISTENCE_, DATATYPE_MISMATCH, DATATYPE_SIZE_, DECOMPRESSION_ERROR, DECSTATION_DECODING, DECSTATION_ENCODING, DEFAULT_BYTE_PADVALUE, DEFAULT_CHAR_PADVALUE, DEFAULT_DOUBLE_PADVALUE, DEFAULT_EPOCH_PADVALUE, DEFAULT_EPOCH16_PADVALUE, DEFAULT_FLOAT_PADVALUE, DEFAULT_INT1_PADVALUE, DEFAULT_INT2_PADVALUE, DEFAULT_INT4_PADVALUE, DEFAULT_INT8_PADVALUE, DEFAULT_REAL4_PADVALUE, DEFAULT_REAL8_PADVALUE, DEFAULT_TT2000_PADVALUE, DEFAULT_UCHAR_PADVALUE, DEFAULT_UINT1_PADVALUE, DEFAULT_UINT2_PADVALUE, DEFAULT_UINT4_PADVALUE, DELETE_, DID_NOT_COMPRESS, EMPTY_COMPRESSED_CDF, END_OF_VAR, EPOCH_STRING_LEN, EPOCH_STRING_LEN_EXTEND, EPOCHI_STRING_LEN, EPOCH1_STRING_LEN_EXTEND, EPOCH2_STRING_LEN, EPOCH2_STRING_LEN_EXTEND, EPOCH3_STRING_LEN, EPOCH3_STRING_LEN_EXTEND, EPOCH4_STRING_LEN, EPOCH4_STRING_LEN_EXTEND, EPOCHx_FORMAT_MAX, EPOCHx_STRING_MAX, FILLED_TT2000_VALUE, FORCED_PARAMETER, FUNCTION_NOT_SUPPORTED, gENTRY_, gENTRY_DATA_, gENTRY_DATASPEC_, gENTRY_DATATYPE_, gENTRY_EXISTENCE_, gENTRY_NUMELEMS_, GET_, GETCDFCHECKSUM_, GETCDFFILEBACKWARD_, GETCDFVALIDATE_, GETLEAPSECONDSENVVAR_, GLOBAL_SCOPE, GZIP_COMPRESSION, HOST_DECODING, HOST_ENCODING, HP_DECODING, HP_ENCODING, HUFF_COMPRESSION, IA64VMSd_DECODING, IA64VMSd_ENCODING, IA64VMSg_DECODING, IA64VMSg_ENCODING, IA64VMSi_DECODING, IA64VMSi_ENCODING, IBM_PC_OVERFLOW, IBMPC_DECODING, IBMPC_ENCODING, IBMRS_DECODING, IBMRS_ENCODING, ILLEGAL_EPOCH_FIELD, ILLEGAL_EPOCH_VALUE, ILLEGAL_FOR_SCOPE, ILLEGAL_IN_zMODE, ILLEGAL_ON_V1_CDF, ILLEGAL_TT2000_VALUE, IS_A_NETCDF, LIB_COPYRIGHT_, LIB_INCREMENT_, LIB_RELEASE_, LIB_subINCREMENT_, LIB_VERSION_, MAC_DECODING, MAC_ENCODING, MD5_CHECKSUM, MULTI_FILE, MULTI_FILE_FORMAT, NA_FOR_VARIABLE, NEGATIVE_FP_ZERO, NEGtoPOSfp0off, NEGtoPOSfp0on, NETWORK_DECODING, NETWORK_ENCODING, NeXT_DECODING, NeXT_ENCODING, NO_ATTR_SELECTED, NO_CDF_SELECTED, NO_CHECKSUM, NO_COMPRESSION, NO_DELETE_ACCESS, NO_ENTRY_SELECTED, NO_MORE_ACCESS, NO_PADVALUE_SPECIFIED, NO_SPARSEARRAYS, NO_SPARSERECORDS, NO_STATUS_SELECTED, NO_SUCH_ATTR, NO_SUCH_CDF, NO_SUCH_ENTRY, NO_SUCH_RECORD, NO_SUCH_VAR, NO_VAR_SELECTED, NO_VARS_IN_CDF, NO_WRITE_ACCESS, NONE_CHECKSUM, NOT_A_CDF, NOT_A_CDF_OR_NOT_SUPPORTED, NOVARY, NULL_, OPEN_, OPTIMAL_ENCODING_TREES, OTHER_CHECKSUM, PAD_SPARSERECORDS, PPC_DECODING, PPC_ENCODING, PRECEEDING_RECORDS_ALLOCATED, PREV_SPARSERECORDS, PUT_, READ_ONLY_DISTRIBUTION, READ_ONLY_MODE, READONLYoff, READONLYon, rENTRY_, rENTRY_DATA_, rENTRY_DATASPEC_, rENTRY_DATATYPE_, rENTRY_EXISTENCE_, rENTRY_NAME_, rENTRY_NUMELEMS_, rENTRY_NUMSTRINGS_, rENTRY_STRINGSDATA_, RLE_COMPRESSION, RLE_OF_ZEROs, ROW_MAJOR, rVAR_, rVAR_ALLOCATEBLOCK_, rVAR_ALLOCATEDFROM_, rVAR_ALLOCATEDTO_, rVAR_ALLOCATERECS_, rVAR_BLOCKINGFACTOR_, rVAR_CACHESIZE_, rVAR_COMPRESSION_, rVAR_DATA_, rVAR_DATASPEC_, rVAR_DATATYPE_, rVAR_DIMVARYS_, rVAR_EXISTENCE_, rVAR_HYPERDATA_, rVAR_INITIALRECS_, rVAR_MAXallocREC_, rVAR_MAXREC_, rVAR_NAME_, rVAR_nINDEXENTRIES_, rVAR_nINDEXLEVELS_, rVAR_nINDEXRECORDS_, rVAR_NUMallocRECS_, rVAR_NUMBER_, rVAR_NUMELEMS_, rVAR_NUMRECS_, rVAR_PADVALUE_, rVAR_RECORDS_, rVAR_RECORDS_RENUMBER_, rVAR_RECVARY_, rVAR_RESERVEPERCENT_, rVAR_SEQDATA_, rVAR_SEQPOS_, rVAR_SPARSEARRAYS_, rVAR_SPARSERECORDS_, rVARs_CACHESIZE_, rVARs_DIMCOUNTS_, rVARs_DIMINDICES_, rVARs_DIMINTERVALS_, rVARs_DIMSIZES_, rVARs_MAXREC_, rVARs_NUMDIMS_, rVARs_RECCOUNT_, rVARs_RECDATA_, rVARs_RECINTERVAL_, rVARs_RECNUMBER_, SAVE_, SCRATCH_CREATE_ERROR, SCRATCH_DELETE_ERROR, SCRATCH_READ_ERROR, SCRATCH_WRITE_ERROR, SELECT_, SGI_DECODING, SGi_ENCODING, SINGLE_FILE, SINGLE_FILE_FORMAT, SOME_ALREADY_ALLOCATED, STAGE_CACHESIZE_, STATUS_TEXT_, STRING_NOT_UTF8_ENCODING, STRINGDELIMITER, SUN_DECODING, SUN_ENCODING, TOO_MANY_PARMS, TOO_MANY_VARS, TRY_TO_READ_NONSTRING_DATA, TT2000_0_STRING_LEN, TT2000_1_STRING_LEN, TT2000_2_STRING_LEN, TT2000_3_STRING_LEN, TT2000_4_STRING_LEN, TT2000_CDF_MAYNEEDUPDATE, TT2000_TIME_ERROR, TT2000_USED_OUTDATED_TABLE, UNABLE_TO_PROCESS_CDF, UNKNOWN_COMPRESSION, UNKNOWN_SPARSENESS, UNSUPPORTED_OPERATION, VALIDATE_, VALIDATEFILEoff, VALIDATEFILEon, VAR_ALREADY_CLOSED, VAR_CLOSE_ERROR, VAR_CREATE_ERROR, VAR_DELETE_ERROR, VAR_EXISTS, VAR_NAME_TRUNC, VAR_OPEN_ERROR, VAR_READ_ERROR, VAR_SAVE_ERROR, VAR_WRITE_ERROR, VARIABLE_SCOPE, VARY, VAX_DECODING, VAX_ENCODING, VIRTUAL_RECORD_DATA, zENTRY_, zENTRY_DATA_, zENTRY_DATASPEC_, zENTRY_DATATYPE_, zENTRY_EXISTENCE_, zENTRY_NAME_, zENTRY_NUMELEMS_, zENTRY_NUMSTRINGS_, zENTRY_STRINGSDATA_, ZLIB_COMPRESS_ERROR, ZLIB_UNCOMPRESS_ERROR, zMODEoff, zMODEon1, zMODEon2, zVAR_, zVAR_ALLOCATEBLOCK_, zVAR_ALLOCATEDFROM_, zVAR_ALLOCATEDTO_, zVAR_ALLOCATERECS_, zVAR_BLOCKINGFACTOR_, zVAR_CACHESIZE_, zVAR_COMPRESSION_, zVAR_DATA_, zVAR_DATASPEC_, zVAR_DATATYPE_, zVAR_DIMCOUNTS_, zVAR_DIMINDICES_, zVAR_DIMINTERVALS_, zVAR_DIMSIZES_, zVAR_DIMVARYS_, zVAR_EXISTENCE_, zVAR_HYPERDATA_, zVAR_INITIALRECS_, zVAR_MAXallocREC_, zVAR_MAXREC_, zVAR_NAME_, zVAR_nINDEXENTRIES_, zVAR_nINDEXLEVELS_, zVAR_nINDEXRECORDS_, zVAR_NUMallocRECS_, zVAR_NUMBER_, zVAR_NUMDIMS_, zVAR_NUMELEMS_, zVAR_NUMRECS_, zVAR_PADVALUE_, zVAR_RECCOUNT_, zVAR_RECINTERVAL_, zVAR_RECNUMBER_, zVAR_RECORDS_, zVAR_RECORDS_RENUMBER_, zVAR_RECVARY_, zVAR_RESERVEPERCENT_, zVAR_SEQDATA_, zVAR_SEQPOS_, zVAR_SPARSEARRAYS_, zVAR_SPARSERECORDS_, zVARs_CACHESIZE_, zVARs_MAXREC_, zVARs_RECDATA_, zVARs_RECNUMBER_

Constructor Summary

Constructors

Constructor and Description
CDFTools()

Method Summary

Methods

Modifier and Type	Method and Description
static void	**skeletonCDF**(java.lang.String skeletonName, java.lang.String cdfName, boolean delete, boolean log, boolean neg2posfp0, boolean statistics, int zMode, int reportType, int cacheSizeD) skeletonCDF produces a CDF file from a skeleton table.
static void	**skeletonCDF**(java.lang.String skeletonName, java.lang.String cdfName, boolean delete, boolean log, boolean neg2posfp0, boolean statistics, int zMode, int reportType, int cacheSizeD, int cacheSizeS) skeletonCDF produces a CDF file from a skeleton table.
static void	**skeletonCDF**(java.lang.String skeletonName, java.lang.String cdfName, boolean delete, boolean log, boolean neg2posfp0, boolean statistics, int zMode, int reportType, int cacheSizeD, int cacheSizeS, int cacheSizeC) skeletonCDF produces a CDF file from a skeleton table.
static void	**skeletonCDF**(java.lang.String skeletonName, java.lang.String cdfName, boolean delete, boolean log, boolean neg2posfp0, boolean statistics, int zMode, int reportType, java.lang.String cacheSize) skeletonCDF produces a CDF file from a skeleton table.
static void	**skeletonTable**(java.lang.String skeletonName, java.lang.String cdfName, boolean log, boolean format, boolean neg2posfp0, boolean statistics, boolean screen, boolean page, int values, java.lang.String[] valueList, int zMode, int reportType, int cacheSize) skeletonTable produces a skeleton table from a CDF.
static void	**skeletonTable**(java.lang.String skeletonName, java.lang.String cdfName, boolean log, boolean format, boolean neg2posfp0, boolean statistics, boolean screen, boolean page, int values, java.lang.String[] valueList, int zMode, int reportType, int cacheSizeD, int cacheSizeS) skeletonTable produces a skeleton table from a CDF.
static void	**skeletonTable**(java.lang.String skeletonName, java.lang.String cdfName, boolean log, boolean format, boolean neg2posfp0, boolean statistics, boolean screen, boolean page, int values, java.lang.String[] valueList, int zMode, int reportType, int cacheSizeD, int cacheSizeS, int cacheSizeC) skeletonTable produces a skeleton table from a CDF.
static void	**skeletonTable**(java.lang.String skeletonName, java.lang.String cdfName, boolean log, boolean format, boolean neg2posfp0, boolean statistics, boolean screen, boolean page, int values, java.lang.String[] valueList, int zMode, int reportType, java.lang.String cacheSize) skeletonTable produces a skeleton table from a CDF.
static void	**skeletonTable**(java.lang.String skeletonName, java.lang.String cdfName, boolean log, boolean format, boolean neg2posfp0, boolean statistics, boolean screen, boolean page, int values, java.lang.String[] valueList, int zMode, int reportType, java.lang.String cacheSize, java.lang.String advFormat) skeletonTable produces a skeleton table from a CDF.

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

NO_VALUES

public static final int NO_VALUES

See Also:

Constant Field Values

NRV_VALUES

```
public static final int NRV_VALUES
```

See Also:

Constant Field Values

RV_VALUES

```
public static final int RV_VALUES
```

See Also:

Constant Field Values

ALL_VALUES

```
public static final int ALL_VALUES
```

See Also:

Constant Field Values

NAMED_VALUES

```
public static final int NAMED_VALUES
```

See Also:

Constant Field Values

NO_REPORTS

```
public static final int NO_REPORTS
```

See Also:

Constant Field Values

REPORT_ERRORS

```
public static final int REPORT_ERRORS
```

See Also:

Constant Field Values

REPORT_WARNINGS

```
public static final int REPORT_WARNINGS
```

See Also:

Constant Field Values

REPORT_INFORMATION

```
public static final int REPORT_INFORMATION
```

See Also:

Constant Field Values

Constructor Detail

CDFTools

```
public CDFTools()
```

Method Detail

skeletonTable

```
public static void skeletonTable(java.lang.String skeletonName,
                 java.lang.String cdfName,
                 boolean log,
                 boolean format,
                 boolean neg2posfp0,
                 boolean statistics,
                 boolean screen,
                 boolean page,
                 int values,
                 java.lang.String[] valueList,
                 int zMode,
                 int reportType,
                 int cacheSize)
                          throws java.io.IOException,
                                 java.lang.InterruptedException
```

skeletonTable produces a skeleton table from a CDF. A skeleton table is a text file which can be read by the SkeletonCDF program to build a skeleton CDF.

Parameters:

`skeletonName` - is the pathname of the skeleton table to be created. (Do not enter an extension because ".skt" is appended automatically). If **null** is specified, the skeleton table is named <cdfName>.skt in the current directory

`cdfName` - The pathname of the CDF from which the skeleton table will be created. Do not enter an extension.

`log` - Specifies whether or not messages are displayed as the program executes.

`format` - Specifies whether or not the FORMAT attribute is used when writing variable values (if the FORMAT attribute exists and an entry exists for the variable).

`neg2posfp0` - Specifies whether or not -0.0 is converted to 0.0 by the CDF library when read from a CDF. -0.0 is an illegal floating point value on VAXes and DEC Alphas running OpenVMS.

`statistics` - Specifies whether or not caching statistics are displayed at the end of each CDF.

`screen` - Specifies whether or not the skeleton table is displayed on the terminal screen (written to the "standard output"). If not, the skeleton table is written to a file.

`page` - If the skeleton table is being displayed on the terminal screen, specifies whether or not the output is displayed one page (screen) at a time.

`values` - Specifies which variable values are to be put in the skeleton table. It may be one of the following...

CDFTools.NO_VALUES

Ignore all NRV data values.

CDFTools.NRV_VALUES

Put NRV data values in the skeleton table.

CDFTools.RV_VALUES

Put RV variable values in the skeleton table.

CDFTools.ALL_VALUES

Put all variable values in the skeleton table.

CDFTools.NAMED_VALUES

Put named variables values in the skeleton table. This requires that valueList be non-null

`valueList` - the named variables to list values.

`zMode` - Specifies which zMode should be used. May be one of the following...

0

Indicates that zMode is disabled.

1

Indicates that zMode/1 should be used (the dimension variances of rVariables will be preserved).

2

Indicates that zMode/2 should be used (the dimensions of rVariables having a variance of NOVARY (false) are hidden.

`reportType` - Specifies the types of return status codes from the CDF library which should be reported/displayed. `report` is a bit mask made up from the following CDFTools.NO_REPORTS, CDFTools.REPORT_ERRORS, CDFTools.REPORT_WARNINGS and CDFTools.REPORT_INFORMATION

`cacheSize` - The number of buffers to be used for the CDF's dotCDF file. Pass 0 to use the default.

Throws:

`java.io.IOException` - An exception is thrown if the file is bad or non-existent

`java.lang.InterruptedException` - An exception is thrown if an interrupt is issued **Note:**
When running an application that use this static method, make sure this **cdf.base** system property is defined (to the cdf installation). For example,
java -Dcdf.base=/home/cdf/cdf34_1-dist MyApp

skeletonTable

```
public static void skeletonTable(java.lang.String skeletonName,
                  java.lang.String cdfName,
                  boolean log,
                  boolean format,
                  boolean neg2posfp0,
                  boolean statistics,
                  boolean screen,
                  boolean page,
                  int values,
                  java.lang.String[] valueList,
                  int zMode,
                  int reportType,
                  int cacheSizeD,
                  int cacheSizeS)
                           throws java.io.IOException,
                                  java.lang.InterruptedException
```

skeletonTable produces a skeleton table from a CDF. A skeleton table is a text file which can be read by the SkeletonCDF program to build a skeleton CDF.

Parameters:

`skeletonName` - is the pathname of the skeleton table to be created. (Do not enter an extension because ".skt" is appended automatically). If **null** is specified, the skeleton table is named <cdfName>.skt in the current directory

`cdfName` - The pathname of the CDF from which the skeleton table will be created. Do not enter an extension.

`log` - Specifies whether or not messages are displayed as the program executes.

`format` - Specifies whether or not the FORMAT attribute is used when writing variable values (if the FORMAT attribute exists and an entry exists for the variable).

`neg2posfp0` - Specifies whether or not -0.0 is converted to 0.0 by the CDF library when read from a CDF. -0.0 is an illegal floating point value on VAXes and DEC Alphas running OpenVMS.

`statistics` - Specifies whether or not caching statistics are displayed at the end of each CDF.

`screen` - Specifies whether or not the skeleton table is displayed on the terminal screen (written to the "standard output"). If not, the skeleton table is written to a file.

`page` - If the skeleton table is being displayed on the terminal screen, specifies whether or not the output is displayed one page (screen) at a time.

`values` - Specifies which variable values are to be put in the skeleton table. It may be one of the following...

CDFTools.NO_VALUES

Ignore all NRV data values.

CDFTools.NRV_VALUES

Put NRV data values in the skeleton table.

CDFTools.RV_VALUES

Put RV variable values in the skeleton table.

CDFTools.ALL_VALUES

Put all variable values in the skeleton table.

CDFTools.NAMED_VALUES

Put named variables values in the skeleton table. This requires that valueList be non-null

`valueList` - the named variables to list values.

`zMode` - Specifies which zMode should be used. May be one of the following...

0

Indicates that zMode is disabled.

1

Indicates that zMode/1 should be used (the dimension variances of rVariables will be preserved).

2

Indicates that zMode/2 should be used (the dimensions of rVariables having a variance of NOVARY (false) are hidden.

`reportType` - Specifies the types of return status codes from the CDF library which should be reported/displayed. `report` is a bit mask made up from the following CDFTools.NO_REPORTS, CDFTools.REPORT_ERRORS, CDFTools.REPORT_WARNINGS and CDFTools.REPORT_INFORMATION

`cacheSizeD` - The number of buffers to be used for the CDF's dotCDF file. Pass 0 to use the default.

`cacheSizeS` - The number of buffers to be used for the CDF's staging file. Pass 0 to use the default.

Throws:

`java.io.IOException` - An exception is thrown if the file is bad or non-existent

`java.lang.InterruptedException` - An exception is thrown if an interrupt is issued **Note:**
When running an application that use this static method, make sure this **cdf.base** system property is defined (to the cdf installation). For example,
java -Dcdf.base=/home/cdf/cdf34_1-dist MyApp

skeletonTable

```
public static void skeletonTable(java.lang.String skeletonName,
                 java.lang.String cdfName,
                 boolean log,
                 boolean format,
                 boolean neg2posfp0,
                 boolean statistics,
```

```
                boolean screen,
                boolean page,
                int values,
                java.lang.String[] valueList,
                int zMode,
                int reportType,
                int cacheSizeD,
                int cacheSizeS,
                int cacheSizeC)
                        throws java.io.IOException,
                               java.lang.InterruptedException
```

skeletonTable produces a skeleton table from a CDF. A skeleton table is a text file which can be read by the SkeletonCDF program to build a skeleton CDF.

Parameters:

`skeletonName` - is the pathname of the skeleton table to be created. (Do not enter an extension because ".skt" is appended automatically). If **null** is specified, the skeleton table is named <cdfName>.skt in the current directory

`cdfName` - The pathname of the CDF from which the skeleton table will be created. Do not enter an extension.

`log` - Specifies whether or not messages are displayed as the program executes.

`format` - Specifies whether or not the FORMAT attribute is used when writing variable values (if the FORMAT attribute exists and an entry exists for the variable).

`neg2posfp0` - Specifies whether or not -0.0 is converted to 0.0 by the CDF library when read from a CDF. -0.0 is an illegal floating point value on VAXes and DEC Alphas running OpenVMS.

`statistics` - Specifies whether or not caching statistics are displayed at the end of each CDF.

`screen` - Specifies whether or not the skeleton table is displayed on the terminal screen (written to the "standard output"). If not, the skeleton table is written to a file.

`page` - If the skeleton table is being displayed on the terminal screen, specifies whether or not the output is displayed one page (screen) at a time.

`values` - Specifies which variable values are to be put in the skeleton table. It may be one of the following...

CDFTools.NO_VALUES

Ignore all NRV data values.

CDFTools.NRV_VALUES

Put NRV data values in the skeleton table.

CDFTools.RV_VALUES

Put RV variable values in the skeleton table.

CDFTools.ALL_VALUES

Put all variable values in the skeleton table.

CDFTools.NAMED_VALUES

Put named variables values in the skeleton table. This requires that valueList be non-null

`valueList` - the named variables to list values.

`zMode` - Specifies which zMode should be used. May be one of the following...

0

Indicates that zMode is disabled.

1

Indicates that zMode/1 should be used (the dimension variances of rVariables will be preserved).

2

Indicates that zMode/2 should be used (the dimensions of rVariables having a variance of NOVARY (false) are hidden.

`reportType` - Specifies the types of return status codes from the CDF library which should be reported/displayed. `report` is a bit mask made up from the following CDFTools.NO_REPORTS, CDFTools.REPORT_ERRORS, CDFTools.REPORT_WARNINGS and CDFTools.REPORT_INFORMATION

`cacheSizeD` - The number of buffers to be used for the CDF's dotCDF file. Pass 0 to use the default.

`cacheSizeS` - The number of buffers to be used for the CDF's staging file. Pass 0 to use the default.

`cacheSizeC` - The number of buffers to be used for the CDF's compression scratch file. Pass 0 to use the default.

Throws:

`java.io.IOException` - An exception is thrown if the file is bad or non-existent

`java.lang.InterruptedException` - An exception is thrown if an interrupt is issued **Note:**
When running an application that use this static method, make sure this **cdf.base** system property is defined (to the cdf installation). For example,
java -Dcdf.base=/home/cdf/cdf34_1-dist MyApp

skeletonTable

```
public static void skeletonTable(java.lang.String skeletonName,
                 java.lang.String cdfName,
                 boolean log,
                 boolean format,
                 boolean neg2posfp0,
                 boolean statistics,
                 boolean screen,
                 boolean page,
                 int values,
                 java.lang.String[] valueList,
                 int zMode,
                 int reportType,
                 java.lang.String cacheSize)
                          throws java.io.IOException,
                                 java.lang.InterruptedException
```

skeletonTable produces a skeleton table from a CDF. A skeleton table is a text file which can be read by the SkeletonCDF program to build a skeleton CDF.

Parameters:

`skeletonName` - is the pathname of the skeleton table to be created. (Do not enter an extension because ".skt" is appended

automatically). If **null** is specified, the skeleton table is named <cdfName>.skt in the current directory

`cdfName` - The pathname of the CDF from which the skeleton table will be created. Do not enter an extension.

`log` - Specifies whether or not messages are displayed as the program executes.

`format` - Specifies whether or not the FORMAT attribute is used when writing variable values (if the FORMAT attribute exists and an entry exists for the variable).

`neg2posfp0` - Specifies whether or not -0.0 is converted to 0.0 by the CDF library when read from a CDF. -0.0 is an illegal floating point value on VAXes and DEC Alphas running OpenVMS.

`statistics` - Specifies whether or not caching statistics are displayed at the end of each CDF.

`screen` - Specifies whether or not the skeleton table is displayed on the terminal screen (written to the "standard output"). If not, the skeleton table is written to a file.

`page` - If the skeleton table is being displayed on the terminal screen, specifies whether or not the output is displayed one page (screen) at a time.

`values` - Specifies which variable values are to be put in the skeleton table. It may be one of the following...

CDFTools.NO_VALUES

Ignore all NRV data values.

CDFTools.NRV_VALUES

Put NRV data values in the skeleton table.

CDFTools.RV_VALUES

Put RV variable values in the skeleton table.

CDFTools.ALL_VALUES

Put all variable values in the skeleton table.

CDFTools.NAMED_VALUES

Put named variables values in the skeleton table. This requires that valueList be non-null

`valueList` - the named variables to list values.

`zMode` - Specifies which zMode should be used. May be one of the following...

0

Indicates that zMode is disabled.

1

Indicates that zMode/1 should be used (the dimension variances of rVariables will be preserved).

2

Indicates that zMode/2 should be used (the dimensions of rVariables having a variance of NOVARY (false) are hidden.

`reportType` - Specifies the types of return status codes from the CDF library which should be reported/displayed. `report` is a bit mask made up from the following CDFTools.NO_REPORTS, CDFTools.REPORT_ERRORS, CDFTools.REPORT_WARNINGS and CDFTools.REPORT_INFORMATION

`cacheSize` - The number of buffers to be used for the CDF's dotCDF file, staging file, and compression scratch file. This field used only in the CDF access process will not change anything to the contents of the skeleton table. Large number(s) will likely reduce physical I/Os if variable data are involved. If this field is null, the default cache sizes chosen by the CDF library are used. The cache sizes are specified with a comma-separated list of <number><type> pairs where <number> is the number of cache buffers and <type> is the type of file. The file <type>'s are as follows: `d' for the dotCDF file, `s' for the staging file, and `c' for the compression scratch file. For example, `200d,100s' specifies 200 cache buffers for the dotCDF file and 100 cache buffers for the staging file. The dotCDF file cache size can also be specified without the `d' <type> for compatibility with older CDF releases (eg. `200,100s'). Note that not all of the file types must be specified. Those not specified will receive a default cache size.

Throws:

`java.io.IOException` - An exception is thrown if the file is bad or non-existent

`java.lang.InterruptedException` - An exception is thrown if an interrupt is issued **Note:**
When running an application that use this static method, make sure this **cdf.base** system property is defined (to the cdf installation). For example,
java -Dcdf.base=/home/cdf/cdf34_1-dist MyApp

skeletonTable

```
public static void skeletonTable(java.lang.String skeletonName,
                 java.lang.String cdfName,
                 boolean log,
                 boolean format,
                 boolean neg2posfp0,
                 boolean statistics,
                 boolean screen,
                 boolean page,
                 int values,
                 java.lang.String[] valueList,
                 int zMode,
                 int reportType,
                 java.lang.String cacheSize,
                 java.lang.String advFormat)
                          throws java.io.IOException,
                                 java.lang.InterruptedException
```

skeletonTable produces a skeleton table from a CDF. A skeleton table is a text file which can be read by the SkeletonCDF program to build a skeleton CDF.

Parameters:

`skeletonName` - is the pathname of the skeleton table to be created. (Do not enter an extension because ".skt" is appended automatically). If **null** is specified, the skeleton table is named <cdfName>.skt in the current directory

`cdfName` - The pathname of the CDF from which the skeleton table will be created. Do not enter an extension.

`log` - Specifies whether or not messages are displayed as the program executes.

`format` - Specifies whether or not the FORMAT attribute is used when writing variable values (if the FORMAT attribute exists and an entry exists for the variable). Metadata always uses noformat form.

`neg2posfp0` - Specifies whether or not -0.0 is converted to 0.0 by the CDF library when read from a CDF. -0.0 is an illegal floating point value on VAXes and DEC Alphas running OpenVMS.

`statistics` - Specifies whether or not caching statistics are displayed at the end of each CDF.

`screen` - Specifies whether or not the skeleton table is displayed on the terminal screen (written to the "standard output"). If not, the skeleton table is written to a file.

`page` - If the skeleton table is being displayed on the terminal screen, specifies whether or not the output is displayed one page (screen) at a time.

`values` - Specifies which variable values are to be put in the skeleton table. It may be one of the following...

CDFTools.NO_VALUES

Ignore all NRV data values.

CDFTools.NRV_VALUES

Put NRV data values in the skeleton table.

CDFTools.RV_VALUES

Put RV variable values in the skeleton table.

CDFTools.ALL_VALUES

Put all variable values in the skeleton table.

CDFTools.NAMED_VALUES

Put named variables values in the skeleton table. This requires that valueList be non-null

`valueList` - the named variables to list values.

`zMode` - Specifies which zMode should be used. May be one of the following...

0

Indicates that zMode is disabled.

1

Indicates that zMode/1 should be used (the dimension variances of rVariables will be preserved).

2

Indicates that zMode/2 should be used (the dimensions of rVariables having a variance of NOVARY (false) are hidden.

`reportType` - Specifies the types of return status codes from the CDF library which should be reported/displayed. `report` is a bit mask made up from the following CDFTools.NO_REPORTS, CDFTools.REPORT_ERRORS, CDFTools.REPORT_WARNINGS and CDFTools.REPORT_INFORMATION

`cacheSize` - The number of buffers to be used for the CDF's dotCDF file, staging file, and compression scratch file. This field used only in the CDF access process will not change anything to the contents of the skeleton table. Large number(s) will likely reduce physical I/Os if variable data are involved. If this field is null, the default cache sizes chosen by the CDF library are used.

The cache sizes are specified with a comma-separated list of <number><type> pairs where <number> is the number of cache buffers and <type> is the type of file. The file <type>'s are as follows: `d' for the dotCDF file, `s' for the staging file, and `c' for the compression scratch file. For example, `200d,100s' specifies 200 cache buffers for the dotCDF file and 100 cache buffers for the staging file. The dotCDF file cache size can also be specified without the `d' <type> for compatibility with older CDF releases (eg. `200,100s'). Note that not all of the file types must be specified. Those not specified will receive a default cache size.

`advFormat` - This new argument supersedes the prior format argument. It accepts one of "best", "no", "metaonly", "dataonly" or "all" options. "best" option all use the best approach to encode floating point data with a higher precision, even larger than the FORMAT attribute for both metadata and data. "no" option will not use and FORMAT attribute, "metaonly" will use FORMAT attribute for metdata only. "dataonly" will use FORMAT attribute for data only. "all" option will use FORMAT attribute for both metadata and data. "best" option is the default. If null is passed in, which is the old method calling this method, it is changed to "best" option that works as C based tool would do.

Throws:

`java.io.IOException` - An exception is thrown if the file is bad or non-existent

`java.lang.InterruptedException` - An exception is thrown if an interrupt is issued **Note:**
When running an application that use this static method, make sure this **cdf.base** system property is defined (to the cdf installation). For example,
java -Dcdf.base=/home/cdf/cdf34_1-dist MyApp

skeletonCDF

```
public static void skeletonCDF(java.lang.String skeletonName,
               java.lang.String cdfName,
               boolean delete,
               boolean log,
               boolean neg2posfp0,
               boolean statistics,
               int zMode,
               int reportType,
               int cacheSizeD)
                        throws java.io.IOException,
                               java.lang.InterruptedException
```

skeletonCDF produces a CDF file from a skeleton table. A skeleton table is a text file which can be made through a text editor, or by SkeletonCDF program from an existng CDF.

Parameters:

`skeletonName` - is the pathname of the skeleton table to be used to create a CDF. (Do not enter an extension because ".skt" is appended automatically). If **null** is specified, the skeleton table is named <cdfName>.skt in the current directory

`cdfName` - The pathname of the created CDF from the skeleton table. Do not enter an extension.

`delete` - specifies whether or not the CDF should be deleted if it already exists.

`log` - Specifies whether or not messages are displayed as the program executes.

`neg2posfp0` - Specifies whether or not -0.0 is converted to 0.0 by the CDF library when read from a CDF. -0.0 is an illegal floating point value on VAXes and DEC Alphas running OpenVMS.

`statistics` - Specifies whether or not caching statistics are displayed at the end of each CDF.

`zMode` - Specifies which zMode should be used. May be one of the following...

0

Indicates that zMode is disabled.

1

Indicates that zMode/1 should be used (the dimension variances of rVariables will be preserved).

2

Indicates that zMode/2 should be used (the dimensions of rVariables having a variance of NOVARY (false) are hidden.

`reportType` - Specifies the types of return status codes from the CDF library which should be reported/displayed. `report` is a bit mask made up from the following CDFTools.NO_REPORTS, CDFTools.REPORT_ERRORS, CDFTools.REPORT_WARNINGS and CDFTools.REPORT_INFORMATION

`cacheSizeD` - The number of buffers to be used for the CDF's dotCDF file. Pass 0 to use the default.

Throws:

`java.io.IOException` - An exception is thrown if the file is bad or non-existent

`java.lang.InterruptedException` - An exception is thrown if an interrupt is issued **Note:**
When running an application that use this static method, make sure this **cdf.base** system property is defined (to the cdf installation). For example,
java -Dcdf.base=/home/cdf/cdf34_1-dist MyApp

skeletonCDF

```
public static void skeletonCDF(java.lang.String skeletonName,
               java.lang.String cdfName,
               boolean delete,
               boolean log,
               boolean neg2posfp0,
               boolean statistics,
               int zMode,
               int reportType,
               int cacheSizeD,
               int cacheSizeS)
                        throws java.io.IOException,
                               java.lang.InterruptedException
```

skeletonCDF produces a CDF file from a skeleton table. A skeleton table is a text file which can be made through a text editor, or by SkeletonCDF program from an existng CDF.

Parameters:

`skeletonName` - is the pathname of the skeleton table to be used to create a CDF. (Do not enter an extension because ".skt" is appended automatically). If **null** is specified, the skeleton table is named <cdfName>.skt in the current directory

`cdfName` - The pathname of the created CDF from the skeleton table. Do not enter an extension.

`delete` - specifies whether or not the CDF should be deleted if it already exists.

`log` - Specifies whether or not messages are displayed as the program executes.

`neg2posfp0` - Specifies whether or not -0.0 is converted to 0.0 by the CDF library when read from a CDF. -0.0 is an illegal floating point value on VAXes and DEC Alphas running OpenVMS.

`statistics` - Specifies whether or not caching statistics are displayed at the end of each CDF.

`zMode` - Specifies which zMode should be used. May be one of the following...

0

Indicates that zMode is disabled.

1

Indicates that zMode/1 should be used (the dimension variances of rVariables will be preserved).

2

Indicates that zMode/2 should be used (the dimensions of rVariables having a variance of NOVARY (false) are hidden.

`reportType` - Specifies the types of return status codes from the CDF library which should be reported/displayed. `report` is a bit mask made up from the following CDFTools.NO_REPORTS, CDFTools.REPORT_ERRORS, CDFTools.REPORT_WARNINGS and CDFTools.REPORT_INFORMATION

`cacheSizeD` - The number of buffers to be used for the CDF's dotCDF file. Pass 0 to use the default.

`cacheSizeS` - The number of buffers to be used for the CDF's staging file. Pass 0 to use the default.

Throws:

`java.io.IOException` - An exception is thrown if the file is bad or non-existent

`java.lang.InterruptedException` - An exception is thrown if an interrupt is issued **Note:**
When running an application that use this static method, make sure this **cdf.base** system property is defined (to the cdf installation). For example,
java -Dcdf.base=/home/cdf/cdf34_1-dist MyApp

skeletonCDF

```
public static void skeletonCDF(java.lang.String skeletonName,
               java.lang.String cdfName,
               boolean delete,
               boolean log,
               boolean neg2posfp0,
               boolean statistics,
               int zMode,
               int reportType,
               int cacheSizeD,
               int cacheSizeS,
               int cacheSizeC)
                        throws java.io.IOException,
                               java.lang.InterruptedException
```

skeletonCDF produces a CDF file from a skeleton table. A skeleton table is a text file which can be made through a text editor, or by SkeletonCDF program from an existng CDF.

Parameters:

`skeletonName` - is the pathname of the skeleton table to be used to create a CDF. (Do not enter an extension because ".skt" is appended automatically). If **null** is specified, the skeleton table is named <cdfName>.skt in the current directory

`cdfName` - The pathname of the created CDF from the skeleton table. Do not enter an extension.

`delete` - specifies whether or not the CDF should be deleted if it already exists.

`log` - Specifies whether or not messages are displayed as the program executes.

`neg2posfp0` - Specifies whether or not -0.0 is converted to 0.0 by the CDF library when read from a CDF. -0.0 is an illegal floating point value on VAXes and DEC Alphas running OpenVMS.

`statistics` - Specifies whether or not caching statistics are displayed at the end of each CDF.

`zMode` - Specifies which zMode should be used. May be one of the following...

0

Indicates that zMode is disabled.

1

Indicates that zMode/1 should be used (the dimension variances of rVariables will be preserved).

2

Indicates that zMode/2 should be used (the dimensions of rVariables having a variance of NOVARY (false) are hidden.

`reportType` - Specifies the types of return status codes from the CDF library which should be reported/displayed. `report` is a bit mask made up from the following CDFTools.NO_REPORTS, CDFTools.REPORT_ERRORS, CDFTools.REPORT_WARNINGS and CDFTools.REPORT_INFORMATION

`cacheSizeD` - The number of buffers to be used for the CDF's dotCDF file. Pass 0 to use the default.

`cacheSizeS` - The number of buffers to be used for the CDF's staging file. Pass 0 to use the default.

`cacheSizeC` - The number of buffers to be used for the CDF's compression scratch file. Pass 0 to use the default.

Throws:

`java.io.IOException` - An exception is thrown if the file is bad or non-existent

`java.lang.InterruptedException` - An exception is thrown if an interrupt is issued **Note:**
When running an application that use this static method, make sure this **cdf.base** system property is defined (to the cdf installation). For example,
java -Dcdf.base=/home/cdf/cdf34_1-dist MyApp

skeletonCDF

```
public static void skeletonCDF(java.lang.String skeletonName,
               java.lang.String cdfName,
               boolean delete,
               boolean log,
               boolean neg2posfp0,
               boolean statistics,
               int zMode,
               int reportType,
               java.lang.String cacheSize)
                        throws java.io.IOException,
                               java.lang.InterruptedException
```

skeletonCDF produces a CDF file from a skeleton table. A skeleton table is a text file which can be made through a text editor, or by SkeletonCDF program from an existng CDF.

Parameters:

`skeletonName` - is the pathname of the skeleton table to be used to create a CDF. (Do not enter an extension because ".skt" is appended automatically). If **null** is specified, the skeleton table is named <cdfName>.skt in the current directory

`cdfName` - The pathname of the created CDF from the skeleton table. Do not enter an extension.

`delete` - specifies whether or not the CDF should be deleted if it already exists.

`log` - Specifies whether or not messages are displayed as the program executes.

`neg2posfp0` - Specifies whether or not -0.0 is converted to 0.0 by the CDF library when read from a CDF. -0.0 is an illegal floating point value on VAXes and DEC Alphas running OpenVMS.

`statistics` - Specifies whether or not caching statistics are displayed at the end of each CDF.

`zMode` - Specifies which zMode should be used. May be one of the following...

0

Indicates that zMode is disabled.

1

Indicates that zMode/1 should be used (the dimension variances of rVariables will be preserved).

2

Indicates that zMode/2 should be used (the dimensions of rVariables having a variance of NOVARY (false) are hidden.

`reportType` - Specifies the types of return status codes from the CDF library which should be reported/displayed. `report` is a bit mask made up from the following CDFTools.NO_REPORTS, CDFTools.REPORT_ERRORS, CDFTools.REPORT_WARNINGS and CDFTools.REPORT_INFORMATION

`cacheSize` - The number of buffers to be used for the CDF's dotCDF file, staging file, and compression scratch file. This field used only in the CDF access process will not change the contents of the CDF. Large number(s) will likely reduce physical I/Os if variable data are involved. If this field is null, default cache sizes chosen by the CDF library are used. The cache sizes are specified with a comma-separated list of <number><type> pairs where <number> is the number of cache buffers and <type> is the type of file. The file <type>'s are as follows: `d' for the dotCDF file, `s' for the staging file, and `c' for the compression scratch file. For example, `200d,100s' specifies 200 cache buffers for the dotCDF file and 100 cache buffers for the staging file. The dotCDF file cache size can also be specified without the `d' <type> for compatibility with older CDF releases (eg. `200,100s'). Note that not all of the file types must be specified. Those not specified will receive a default cache size.

Throws:

`java.io.IOException` - An exception is thrown if the file is bad or non-existent

`java.lang.InterruptedException` - An exception is thrown if an interrupt is issued **Note:**
When running an application that use this static method, make sure this **cdf.base** system property is defined (to the cdf installation). For example,
java -Dcdf.base=/home/cdf/cdf34_1-dist MyApp

gsfc.nssdc.cdf.util

Class CDFTT2000

java.lang.Object
gsfc.nssdc.cdf.util.CDFTT2000

All Implemented Interfaces:

CDFConstants

```
public class CDFTT2000
extends java.lang.Object
implements CDFConstants
```

This class contains the handy utility routines (methods) called by the CDF applications to handle epoch data of CDF's CDF_TIME_TT2000 data type.

Version:

1.0, 2.0 01/05/15 Add a new leap second for 2015/07/01., 3.0 10/20/16 Add a new leap second for 2016/01/01.

Field Summary

Fields inherited from interface gsfc.nssdc.cdf.CDFConstants

AHUFF_COMPRESSION, ALPHAOSF1_DECODING, ALPHAOSF1_ENCODING, ALPHAVMSd_DECODING, ALPHAVMSd_ENCODING, ALPHAVMSg_DECODING, ALPHAVMSg_ENCODING, ALPHAVMSi_DECODING, ALPHAVMSi_ENCODING, ARM_BIG_DECODING, ARM_BIG_ENCODING, ARM_LITTLE_DECODING, ARM_LITTLE_ENCODING, ATTR_, ATTR_EXISTENCE_, ATTR_EXISTS, ATTR_MAXgENTRY_, ATTR_MAXrENTRY_, ATTR_MAXzENTRY_, ATTR_NAME_, ATTR_NAME_TRUNC, ATTR_NUMBER_, ATTR_NUMgENTRIES_, ATTR_NUMrENTRIES_, ATTR_NUMzENTRIES_, ATTR_SCOPE_, BACKWARD_, BACKWARDFILEoff, BACKWARDFILEon, BAD_ALLOCATE_RECS, BAD_ARGUMENT, BAD_ATTR_NAME, BAD_ATTR_NUM, BAD_BLOCKING_FACTOR, BAD_CACHE_SIZE, BAD_CDF_EXTENSION, BAD_CDF_ID, BAD_CDF_NAME, BAD_CDFSTATUS, BAD_CHECKSUM, BAD_COMPRESSION_PARM, BAD_DATA_TYPE, BAD_DECODING, BAD_DIM_COUNT, BAD_DIM_INDEX, BAD_DIM_INTERVAL, BAD_DIM_SIZE, BAD_ENCODING, BAD_ENTRY_NUM, BAD_FNC_OR_ITEM, BAD_FORMAT, BAD_INITIAL_RECS, BAD_MAJORITY, BAD_MALLOC, BAD_NEGtoPOSfp0_MODE, BAD_NUM_DIMS, BAD_NUM_ELEMS, BAD_NUM_STRINGS, BAD_NUM_VARS, BAD_READONLY_MODE, BAD_REC_COUNT, BAD_REC_INTERVAL, BAD_REC_NUM, BAD_SCOPE, BAD_SCRATCH_DIR, BAD_SPARSEARRAYS_PARM, BAD_VAR_NAME, BAD_VAR_NUM, BAD_zMODE, BADDATE_LEAPSECOND_UPDATED, BeginUnixTimeEPOCH, BeginUnixTimeEPOCH16, BLOCKINGFACTOR_TOO_LARGE, BLOCKINGFACTOR_TOO_SMALL, BLOCKINGFACTOR_TOO_SMALL2, CANNOT_ALLOCATE_RECORDS, CANNOT_CHANGE, CANNOT_COMPRESS, CANNOT_CONVERT_WIDECHAR, CANNOT_COPY, CANNOT_INSERT_RECORDS, CANNOT_SPARSEARRAYS, CANNOT_SPARSERECORDS, CDF_, CDF_ACCESS_, CDF_ATTR_NAME_LEN, CDF_ATTR_NAME_LEN256, CDF_BYTE, CDF_CACHESIZE_, CDF_CHAR, CDF_CHECKSUM_, CDF_CLOSE_ERROR, CDF_COMPRESSION_, CDF_COPYRIGHT_, CDF_COPYRIGHT_LEN, CDF_CREATE_ERROR, CDF_DECODING_, CDF_DELETE_ERROR, CDF_DOUBLE, CDF_ENCODING_, CDF_EPOCH, CDF_EPOCH16, CDF_EXISTS, CDF_FLOAT, CDF_FORMAT_, CDF_INCREMENT_, CDF_INFO_, CDF_INT1, CDF_INT2, CDF_INT4, CDF_INT8, CDF_INTERNAL_ERROR, CDF_LEAPSECONDLASTUPDATED_, CDF_MAJORITY_, CDF_MAX_DIMS, CDF_MAX_PARMS, CDF_MIN_DIMS, CDF_NAME_, CDF_NAME_TRUNC, CDF_NEGtoPOSfp0_MODE_, CDF_NUMATTRS_, CDF_NUMgATTRS_, CDF_NUMrVARS_, CDF_NUMvATTRS_, CDF_NUMzVARS_, CDF_OK, CDF_OPEN_ERROR, CDF_PATHNAME_LEN, CDF_READ_ERROR, CDF_READONLY_MODE_, CDF_REAL4, CDF_REAL8, CDF_RELEASE_, CDF_SAVE_ERROR, CDF_SCRATCHDIR_, CDF_STATUS_, CDF_STATUSTEXT_LEN, CDF_TIME_TT2000, CDF_UCHAR, CDF_UINT1, CDF_UINT2, CDF_UINT4, CDF_VAR_NAME_LEN, CDF_VAR_NAME_LEN256, CDF_VERSION_, CDF_WARN, CDF_WRITE_ERROR, CDF_zMODE_, CDFwithSTATS_, CHECKSUM_, CHECKSUM_ERROR, CHECKSUM_NOT_ALLOWED, CLOSE_, COLUMN_MAJOR, COMPRESS_CACHESIZE_, COMPRESSION_ERROR, CONFIRM_, CORRUPTED_V2_CDF, CORRUPTED_V3_CDF, CREATE_, CURgENTRY_EXISTENCE_, CURrENTRY_EXISTENCE_, CURzENTRY_EXISTENCE_, DATATYPE_MISMATCH, DATATYPE_SIZE_, DECOMPRESSION_ERROR, DECSTATION_DECODING, DECSTATION_ENCODING, DEFAULT_BYTE_PADVALUE, DEFAULT_CHAR_PADVALUE, DEFAULT_DOUBLE_PADVALUE, DEFAULT_EPOCH_PADVALUE, DEFAULT_EPOCH16_PADVALUE, DEFAULT_FLOAT_PADVALUE, DEFAULT_INT1_PADVALUE, DEFAULT_INT2_PADVALUE, DEFAULT_INT4_PADVALUE, DEFAULT_INT8_PADVALUE, DEFAULT_REAL4_PADVALUE, DEFAULT_REAL8_PADVALUE, DEFAULT_TT2000_PADVALUE, DEFAULT_UCHAR_PADVALUE, DEFAULT_UINT1_PADVALUE, DEFAULT_UINT2_PADVALUE, DEFAULT_UINT4_PADVALUE, DELETE_, DID_NOT_COMPRESS, EMPTY_COMPRESSED_CDF, END_OF_VAR, EPOCH_STRING_LEN, EPOCH_STRING_LEN_EXTEND, EPOCH1_STRING_LEN, EPOCH1_STRING_LEN_EXTEND, EPOCH2_STRING_LEN, EPOCH2_STRING_LEN_EXTEND, EPOCH3_STRING_LEN, EPOCH3_STRING_LEN_EXTEND, EPOCH4_STRING_LEN, EPOCH4_STRING_LEN_EXTEND, EPOCHx_FORMAT_MAX, EPOCHx_STRING_MAX, FILLED_TT2000_VALUE, FORCED_PARAMETER, FUNCTION_NOT_SUPPORTED, gENTRY_, gENTRY_DATA_, gENTRY_DATASPEC_, gENTRY_DATATYPE_, gENTRY_EXISTENCE_, gENTRY_NUMELEMS_, GET_, GETCDFCHECKSUM_, GETCDFFILEBACKWARD_, GETCDFVALIDATE_, GETLEAPSECONDSENVVAR_, GLOBAL_SCOPE, GZIP_COMPRESSION, HOST_DECODING, HOST_ENCODING, HP_DECODING, HP_ENCODING, HUFF_COMPRESSION, IA64VMSd_DECODING, IA64VMSd_ENCODING, IA64VMSg_DECODING, IA64VMSg_ENCODING, IA64VMSi_DECODING, IA64VMSi_ENCODING, IBM_PC_OVERFLOW, IBMPC_DECODING, IBMPC_ENCODING, IBMRS_DECODING,

IBMRS_ENCODING, ILLEGAL_EPOCH_FIELD, ILLEGAL_EPOCH_VALUE, ILLEGAL_FOR_SCOPE, ILLEGAL_IN_zMODE, ILLEGAL_ON_V1_CDF, ILLEGAL_TT2000_VALUE, IS_A_NETCDF, LIB_COPYRIGHT, LIB_INCREMENT, LIB_RELEASE, LIB_subINCREMENT, LIB_VERSION, MAC_DECODING, MAC_ENCODING, MD5_CHECKSUM, MULTI_FILE, MULTI_FILE_FORMAT, NA_FOR_VARIABLE, NEGATIVE_FP_ZERO, NEGtoPOSfp0off, NEGtoPOSfp0on, NETWORK_DECODING, NETWORK_ENCODING, NeXT_DECODING, NeXT_ENCODING, NO_ATTR_SELECTED, NO_CDF_SELECTED, NO_CHECKSUM, NO_COMPRESSION, NO_DELETE_ACCESS, NO_ENTRY_SELECTED, NO_MORE_ACCESS, NO_PADVALUE_SPECIFIED, NO_SPARSEARRAYS, NO_SPARSERECORDS, NO_STATUS_SELECTED, NO_SUCH_ATTR, NO_SUCH_CDF, NO_SUCH_ENTRY, NO_SUCH_RECORD, NO_SUCH_VAR, NO_VAR_SELECTED, NO_VARS_IN_CDF, NO_WRITE_ACCESS, NONE_CHECKSUM, NOT_A_CDF, NOT_A_CDF_OR_NOT_SUPPORTED, NOVARY, NULL_, OPEN_, OPTIMAL_ENCODING_TREES, OTHER_CHECKSUM, PAD_SPARSERECORDS, PPC_DECODING, PPC_ENCODING, PRECEEDING_RECORDS_ALLOCATED, PREV_SPARSERECORDS, PUT_, READ_ONLY_DISTRIBUTION, READ_ONLY_MODE, READONLYoff, READONLYon, rENTRY_, rENTRY_DATA_, rENTRY_DATASPEC_, rENTRY_DATATYPE_, rENTRY_EXISTENCE_, rENTRY_NAME_, rENTRY_NUMELEMS_, rENTRY_NUMSTRINGS_, rENTRY_STRINGSDATA_, RLE_COMPRESSION, RLE_OF_ZEROs, ROW_MAJOR, rVAR_, rVAR_ALLOCATEBLOCK_, rVAR_ALLOCATEDFROM_, rVAR_ALLOCATEDTO_, rVAR_ALLOCATERECS_, rVAR_BLOCKINGFACTOR_, rVAR_CACHESIZE_, rVAR_COMPRESSION_, rVAR_DATA_, rVAR_DATASPEC_, rVAR_DATATYPE_, rVAR_DIMVARYS_, rVAR_EXISTENCE_, rVAR_HYPERDATA_, rVAR_INITIALRECS_, rVAR_MAXallocREC_, rVAR_MAXREC_, rVAR_NAME_, rVAR_nINDEXENTRIES_, rVAR_nINDEXLEVELS_, rVAR_nINDEXRECORDS_, rVAR_NUMallocRECS_, rVAR_NUMBER_, rVAR_NUMELEMS_, rVAR_NUMRECS_, rVAR_PADVALUE_, rVAR_RECORDS_, rVAR_RECORDS_RENUMBER_, rVAR_RECVARY_, rVAR_RESERVEPERCENT_, rVAR_SEQDATA_, rVAR_SEQPOS_, rVAR_SPARSEARRAYS_, rVAR_SPARSERECORDS_, rVARs_CACHESIZE_, rVARs_DIMCOUNTS_, rVARs_DIMINDICES_, rVARs_DIMINTERVALS_, rVARs_DIMSIZES_, rVARs_MAXREC_, rVARs_NUMDIMS_, rVARs_RECCOUNT_, rVARs_RECDATA_, rVARs_RECINTERVAL_, rVARs_RECNUMBER_, SAVE_, SCRATCH_CREATE_ERROR, SCRATCH_DELETE_ERROR, SCRATCH_READ_ERROR, SCRATCH_WRITE_ERROR, SELECT_, SGI_DECODING, SGi_ENCODING, SINGLE_FILE, SINGLE_FILE_FORMAT, SOME_ALREADY_ALLOCATED, STAGE_CACHESIZE_, STATUS_TEXT_, STRING_NOT_UTF8_ENCODING, STRINGDELIMITER, SUN_DECODING, SUN_ENCODING, TOO_MANY_PARMS, TOO_MANY_VARS, TRY_TO_READ_NONSTRING_DATA, TT2000_0_STRING_LEN, TT2000_1_STRING_LEN, TT2000_2_STRING_LEN, TT2000_3_STRING_LEN, TT2000_4_STRING_LEN, TT2000_CDF_MAYNEEDUPDATE, TT2000_TIME_ERROR, TT2000_USED_OUTDATED_TABLE, UNABLE_TO_PROCESS_CDF, UNKNOWN_COMPRESSION, UNKNOWN_SPARSENESS, UNSUPPORTED_OPERATION, VALIDATE_, VALIDATEFILEoff, VALIDATEFILEon, VAR_ALREADY_CLOSED, VAR_CLOSE_ERROR, VAR_CREATE_ERROR, VAR_DELETE_ERROR, VAR_EXISTS, VAR_NAME_TRUNC, VAR_OPEN_ERROR, VAR_READ_ERROR, VAR_SAVE_ERROR, VAR_WRITE_ERROR, VARIABLE_SCOPE, VARY, VAX_DECODING, VAX_ENCODING, VIRTUAL_RECORD_DATA, zENTRY_, zENTRY_DATA_, zENTRY_DATASPEC_, zENTRY_DATATYPE_, zENTRY_EXISTENCE_, zENTRY_NAME_, zENTRY_NUMELEMS_, zENTRY_NUMSTRINGS_, zENTRY_STRINGSDATA_, ZLIB_COMPRESS_ERROR, ZLIB_UNCOMPRESS_ERROR, zMODEoff, zMODEon1, zMODEon2, zVAR_, zVAR_ALLOCATEBLOCK_, zVAR_ALLOCATEDFROM_, zVAR_ALLOCATEDTO_, zVAR_ALLOCATERECS_, zVAR_BLOCKINGFACTOR_, zVAR_CACHESIZE_, zVAR_COMPRESSION_, zVAR_DATA_, zVAR_DATASPEC_, zVAR_DATATYPE_, zVAR_DIMCOUNTS_, zVAR_DIMINDICES_, zVAR_DIMINTERVALS_, zVAR_DIMSIZES_, zVAR_DIMVARYS_, zVAR_EXISTENCE_, zVAR_HYPERDATA_, zVAR_INITIALRECS_, zVAR_MAXallocREC_, zVAR_MAXREC_, zVAR_NAME_, zVAR_nINDEXENTRIES_, zVAR_nINDEXLEVELS_, zVAR_nINDEXRECORDS_, zVAR_NUMallocRECS_, zVAR_NUMBER_, zVAR_NUMDIMS_, zVAR_NUMELEMS_, zVAR_NUMRECS_, zVAR_PADVALUE_, zVAR_RECCOUNT_, zVAR_RECINTERVAL_, zVAR_RECNUMBER_, zVAR_RECORDS_, zVAR_RECORDS_RENUMBER_, zVAR_RECVARY_, zVAR_RESERVEPERCENT_, zVAR_SEQDATA_, zVAR_SEQPOS_, zVAR_SPARSEARRAYS_, zVAR_SPARSERECORDS_, zVARs_CACHESIZE_, zVARs_MAXREC_, zVARs_RECDATA_, zVARs_RECNUMBER_

Constructor Summary

Constructors

Constructor and Description
CDFTT2000()

Method Summary

Methods

Modifier and Type	Method and Description
static long[]	**breakdown**(long nanoSecSinceJ2000) Breaks a TT2000 epoch value down into its full, UTC-based date/time component parts.
static long[][]	**breakdown**(long[] nanoSecSinceJ2000) Breaks an array of TT2000 epoch values down into two dimensional array, the first dimension being the count of the epoch values, and the second dimension being their full, UTC-based date/time component parts for each value.
static long[]	**breakdownTT2000**(long nanoSecSinceJ2000) Breaks a TT2000 epoch value down into its full, UTC-based date/time component parts - new name as toUTCparts.
static long[]	**breakdownTT2000withBasedLeapDay**(long tt2000, int yymmdd)

	breakdownTT2000withBasedLeapDay.
static long[]	**CDFgetLastDateinLeapSecondsTable**() /** This method returns the last UTC date that a leap second was added in the leap second table used in the class.
static double[][]	**CDFgetLeapSecondsTable**() This method returns the leap seconds table.
static int	**CDFgetLeapSecondsTableStatus**() This method returns the status code reflecting whether the leap seconds are from a external file, defined by an environment variable, or the leap seconds are based on the hard-coded table in the class.
static int	**CDFgetRowsinLeapSecondsTable**() This method returns the number of entries in the leap seconds table.
static long[]	**compute**(long[] year, long[] month, long[] day) Computes an array of TT2000 epochs, nanoseconds since J2000, based on its UTC-based date/time component parts.
static long[]	**compute**(long[] year, long[] month, long[] day, long[] hour) Computes an array of TT2000 epochs, nanoseconds since J2000, based on its UTC-based date/time component parts.
static long[]	**compute**(long[] year, long[] month, long[] day, long[] hour, long[] minute) Computes an array of TT2000 epochs, nanoseconds since J2000, based on its * UTC-based date/time component parts.
static long[]	**compute**(long[] year, long[] month, long[] day, long[] hour, long[] minute, long[] second) Computes an array of TT2000 epochs, nanoseconds since J2000, based on its * UTC-based date/time component parts.
static long[]	**compute**(long[] year, long[] month, long[] day, long[] hour, long[] minute, long[] second, long[] msec) Computes an array of TT2000 epochs, nanoseconds since J2000, based on its * UTC-based date/time component parts.
static long[]	**compute**(long[] year, long[] month, long[] day, long[] hour, long[] minute, long[] second, long[] msec, long[] usec) Computes an array of TT2000 epochs, nanoseconds since J2000, based on its * UTC-based date/time component parts.
static long[]	**compute**(long[] year, long[] month, long[] day, long[] hour, long[] minute, long[] second, long[] msec, long[] usec, long[] nsec) Computes an array of TT2000 epochs, nanoseconds since J2000, based on its * UTC-based date/time component parts.
static long	**compute**(long year, long month, long day, long hour, long minute, long second, long msec, long usec, long nsec) Computes a TT2000 epoch, nanoseconds since J2000, based on its UTC-based date/time component parts.
static long	**computeTT2000**(double year, double month, double day) Computes a TT2000 epoch, nanoseconds since J2000, based on its UTC-based date/time component parts.
static long	**computeTT2000**(double year, double month, double day, double hour) Computes a TT2000 epoch, nanoseconds since J2000, based on its UTC-based date/time component parts.
static long	**computeTT2000**(double year, double month, double day, double hour, double minute) Computes a TT2000 epoch, nanoseconds since J2000, based on its UTC-based date/time component parts.
static long	**computeTT2000**(double year, double month, double day, double hour, double minute, double second) Computes a TT2000 epoch, nanoseconds since J2000, based on its UTC-based date/time component parts.

static long	**computeTT2000**(double year, double month, double day, double hour, double minute, double second, double milsec) Computes a TT2000 epoch, nanoseconds since J2000, based on its UTC-based date/time component parts.
static long	**computeTT2000**(double year, double month, double day, double hour, double minute, double second, double milsec, double micsec) Computes a TT2000 epoch, nanoseconds since J2000, based on its UTC-based date/time component parts.
static long	**computeTT2000**(double year, double month, double day, double hour, double minute, double second, double msec, double usec, double nsec) Computes a TT2000 epoch, nanoseconds since J2000, based on its UTC-based date/time component parts - a new name from fromUTCparts.
static long	**computeTT2000withBasedLeapDay**(double yy, double mm, double dd, double hh, double mn, double ss, double ms, double us, double ns, int yymmdd) This method returns the TT2000 time in nanoseconds based on the last leap second day.
static long	**computeTT2000withBasedLeapDay**(long yy, long mm, long dd, long hh, long mn, long ss, long ms, long us, long ns, int yymmdd) This method returns the TT2000 time in nanoseconds based on the last leap second day.
static java.lang.String	**encode**(long nanoSecSinceJ2000) Converts an epoch value in TT2000 type into a readable, UTC-based date/time string in ISO 8601 style.
static java.lang.String[]	**encode**(long[] nanoSecSinceJ2000) Converts an array of epoch values in TT2000 type into an array of readable, UTC-based date/time strings in ISO 8601 style.
static java.lang.String[]	**encode**(long[] nanoSecSinceJ2000, int style) Converts an array of epoch values in TT2000 type into readable, UTC-based date/time strings of chosen style.
static java.lang.String	**encode**(long nanoSecSinceJ2000, int style) Converts an epoch value in TT2000 type into a readable, UTC-based date/time string of chosen style.
static java.lang.String	**encodeTT2000withBasedLeapDay**(long tt2000, int yymmdd) encodeTT2000withBasedLeapDay.
static long	**fromGregorianTime**(java.util.GregorianCalendar gc) This method converts the UTC-based date/time in a GregorianCalendar class object to TT2000 time.
static long	**fromUTCEPOCH**(double epoch) Convert an epoch value in CDF_EPOCH type to TT2000 type.
static long	**fromUTCEPOCH16**(double[] epoch) Convert an epoch data in CDF_EPOCH16 type to TT2000 type.
static long[]	**fromUTCISO8601string**(java.lang.String string) This method parses an input, UTC-based, date/time string in ISO 8601 and returns their date/time components.
static long	**fromUTCparts**(double year, double month, double day) Computes a TT2000 epoch, nanoseconds since J2000, based on its UTC-based date/time component parts.
static long	**fromUTCparts**(double year, double month, double day, double hour) Computes a TT2000 epoch, nanoseconds since J2000, based on its UTC-based date/time component parts.
static long	**fromUTCparts**(double year, double month, double day, double hour, double minute) Computes a TT2000 epoch, nanoseconds since J2000, based on its UTC-based

	date/time component parts.
static long	**fromUTCparts**(double year, double month, double day, double hour, double minute, double second) Computes a TT2000 epoch, nanoseconds since J2000, based on its UTC-based date/time component parts.
static long	**fromUTCparts**(double year, double month, double day, double hour, double minute, double second, double milsec) Computes a TT2000 epoch, nanoseconds since J2000, based on its UTC-based date/time component parts.
static long	**fromUTCparts**(double year, double month, double day, double hour, double minute, double second, double milsec, double micsec) Computes a TT2000 epoch, nanoseconds since J2000, based on its UTC-based date/time component parts.
static long	**fromUTCparts**(double year, double month, double day, double hour, double minute, double second, double msec, double usec, double nsec) Computes a TT2000 epoch, nanoseconds since J2000, based on its UTC-based date/time component parts.
static long	**fromUTCstring**(java.lang.String string) This method parses an input, UTC-based, date/time string and returns a TT2000 epoch value, nanoseconds since J2000.
static java.lang.String	**getLeapSecondsTableEnvVar**() Find the environment variable "CDF_LEAPSECONDSTABLE" that is defined for the leap seconds table for CDF.
static double	**LeapSecondsfromYMD**(long year, long month, long day) Find the leap seconds from a given, UTC-based year/month/day.
static long	**parse**(java.lang.String string) This method parses an input date/time string and returns a TT2000 epoch value, nanoseconds since J2000.
static long[]	**parse**(java.lang.String[] strings) This method parses an array of input date/time strings and returns an array of TT2000 epoch values, nanoseconds since J2000.
static long	**parseTT2000**(java.lang.String string) This method parses an input, UTC-based, date/time string and returns a TT2000 epoch value, nanoseconds since J2000.
static long[]	**parseTT2000**(java.lang.String[] strings) This method parses an input, UTC-based, date/time strings and returns a TT2000 epoch value array, nanoseconds since J2000.
static long	**parseTT2000withBasedLeapDay**(java.lang.String tt2000, int yymmdd) parseTT2000withBasedLeapDay.
static java.lang.String	**toEncode**(long nanoSecSinceJ2000) Converts an epoch value in TT2000 type into a readable, UTC-based date/time string of ISO 8601 forme.
static java.lang.String[]	**toEncode**(long[] nanoSecSinceJ2000) Converts an epoch value in TT2000 type into a readable, UTC-based date/time string of ISO8601 style (style 3).
static java.lang.String[]	**toEncode**(long[] nanoSecSinceJ2000, int style) Converts an array of epoch values in TT2000 type into readable, UTC-based date/time strings of chosen style.
static java.lang.String	**toEncode**(long nanoSecSinceJ2000, int style) Converts an epoch value in TT2000 type into a readable, UTC-based date/time string of chosen style.
static java.lang.String	**toEncode**(java.lang.Long nanoSecSinceJ2000, int style) Converts an epoch value in TT2000 type into a readable, UTC-based date/time string of chosen style.

static java.lang.Object	**toEncode**(java.lang.Object epoch) Converts an epoch object in TT2000 type into a scalar or array(s) of readable, UTC-based date/time string of ISO8601 style (style 3).
static java.util.GregorianCalendar	**toGregorianTime**(long tt2000) This method converts the UTC-based date/time in TT2000 type to a GregorianCalendar class object in default, local time zone.
static java.util.GregorianCalendar	**toGregorianTime**(long tt2000, java.util.TimeZone tz) This method converts the UTC-based date/time in TT2000 type to a GregorianCalendar class object in specified time zone.
static long	**toParse**(java.lang.String string) This method parses an input, UTC-based, date/time string and returns a TT2000 epoch value, nanoseconds since J2000.
static long[]	**toParse**(java.lang.String[] strings) This method parses an input, UTC-based, date/time strings and returns a TT2000 epoch value array, nanoseconds since J2000.
static double	**toUTCEPOCH**(long nanoSecSinceJ2000) Convert an epoch value in TT2000 type to CDF_EPOCH type.
static double	**toUTCEPOCH16**(long nanoSecSinceJ2000, double[] epoch) Convert an epoch value in TT2000 type to CDF_EPOCH16 type.
static long[]	**toUTCparts**(long nanoSecSinceJ2000) Breaks a TT2000 epoch value down into its full, UTC-based date/time component parts.
static java.lang.String	**toUTCstring**(long nanoSecSinceJ2000) Converts an epoch value in TT2000 type into a readable, UTC-based date/time string of ISO 8601 style.
static java.lang.String	**toUTCstring**(java.lang.Long nanoSecSinceJ2000) Converts an epoch value in TT2000 type into a readable, UTC-based date/time string of ISO 8601 style.
static java.lang.String[]	**toUTCstring**(long[] nanoSecSinceJ2000) Converts an iarray of epoch values in TT2000 type into readable, UTC-based date/time strings of ISO 8601 style.
static java.lang.String[]	**toUTCstring**(long[] nanoSecSinceJ2000, int style) Converts an array of epoch values in TT2000 type into readable, UTC-based date/time strings of chosen style.
static java.lang.String	**toUTCstring**(long nanoSecSinceJ2000, int style) Converts an epoch value in TT2000 type into a readable, UTC-based date/time string of chosen style.
static java.lang.String	**toUTCstring**(java.lang.Long nanoSecSinceJ2000, int style) Converts an epoch value in TT2000 type into a readable, UTC-based date/time string of chosen style.
static double	**TT2000toUnixTime**(long epoch) Converts a TT2000 value to a unix time of a double value (seconds from 1970-01-01 00:00:00 UTC).
static double[]	**TT2000toUnixTime**(long[] epoch) Converts an array of TT2000 values to an array of unix time of a double value (seconds from 1970-01-01 00:00:00 UTC).
static long	**UnixTimetoTT2000**(double unixTime) Converts a Unix time of a double value (seconds from 1970-01-01 00:00:00 UTC) to a TT2000 time.
static long[]	**UnixTimetoTT2000**(double[] unixTimes) Converts an array of Unix time of double values (seconds from 1970-01-01 00:00:00 UTC) to an array of TT2000 times.

Methods inherited from class java.lang.Object

`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

CDFTT2000

```
public CDFTT2000()
```

Method Detail

getLeapSecondsTableEnvVar

```
public static java.lang.String getLeapSecondsTableEnvVar()
```

Find the environment variable "CDF_LEAPSECONDSTABLE" that is defined for the leap seconds table for CDF.

Returns:

the leap seconds environment variable if defined null if not.

LeapSecondsfromYMD

```
public static double LeapSecondsfromYMD(long year,
                        long month,
                        long day)
```

Find the leap seconds from a given, UTC-based year/month/day.

Parameters:

`year` - the year

`month` - the month

`day` - the day

Returns:

the leap second value (TAI-UTC)

breakdownTT2000

```
public static long[] breakdownTT2000(long nanoSecSinceJ2000)
```

Breaks a TT2000 epoch value down into its full, UTC-based date/time component parts - new name as toUTCparts.

Parameters:

`nanoSecSinceJ2000` - the epoch value, in nanoseconds since J2000, to break down

Returns:

an array of long containing the epoch parts:

Index	Part
0	year
1	month
2	day
3	hour
4	minute
5	second
6	millisecond
7	microsecond
8	nanosecond

toUTCparts

```
public static long[] toUTCparts(long nanoSecSinceJ2000)
```

Breaks a TT2000 epoch value down into its full, UTC-based date/time component parts.

Parameters:

`nanoSecSinceJ2000` - the epoch value, in nanoseconds since J2000, to break down

Returns:

an array of long containing the epoch parts:

Index	Part
0	year
1	month
2	day
3	hour
4	minute
5	second
6	millisecond
7	microsecond
8	nanosecond

breakdown

```
public static long[] breakdown(long nanoSecSinceJ2000)
```

Breaks a TT2000 epoch value down into its full, UTC-based date/time component parts.

Parameters:

`nanoSecSinceJ2000` - the epoch value, in nanoseconds since J2000, to break down

Returns:

an array of long containing the epoch parts:

Index	Part

0	year
1	month
2	day
3	hour
4	minute
5	second
6	millisecond
7	microsecond
8	nanosecond

breakdown

```
public static long[][] breakdown(long[] nanoSecSinceJ2000)
```

Breaks an array of TT2000 epoch values down into two dimensional array, the first dimension being the count of the epoch values, and the second dimension being their full, UTC-based date/time component parts for each value.

Parameters:

`nanoSecSinceJ2000` - an array of epoch values, in nanoseconds since J2000, to break down

Returns:

a 2-dim array of long, the first dimension being the value count while the second dimension containing the epoch parts, as follows:

Index	Part
0	year
1	month
2	day
3	hour
4	minute
5	second
6	millisecond
7	microsecond
8	nanosecond

or null if null or an empty input array is detected

computeTT2000

```
public static long computeTT2000(double year,
                 double month,
                 double day)
                          throws CDFException
```

Computes a TT2000 epoch, nanoseconds since J2000, based on its UTC-based date/time component parts. The day field can contain a fraction of a day - a new name from fromUTCparts.

Parameters:

`year` - the year, a full year in double

`month` - the month, a full month in double

`day` - the day, either the day of the month or day of the year (DOY). For DOY, month has to be one (1), otherwise an exception is thrown. It is in double

Returns:

the TT2000 epoch value in long

Throws:

`CDFException` - an TT2000_TIME_ERROR if an illegal component value is detected.

fromUTCparts

```
public static long fromUTCparts(double year,
                double month,
                double day)
                         throws CDFException
```

Computes a TT2000 epoch, nanoseconds since J2000, based on its UTC-based date/time component parts. The day field can contain a fraction of a day.

Parameters:

`year` - the year, a full year in double

`month` - the month, a full month in double

`day` - the day, either the day of the month or day of the year (DOY). For DOY, month has to be one (1), otherwise an exception is thrown. It is in double

Returns:

the TT2000 epoch value in long

Throws:

`CDFException` - an TT2000_TIME_ERROR if an illegal component value is detected.

computeTT2000

```
public static long computeTT2000(double year,
                 double month,
                 double day,
                 double hour)
                          throws CDFException
```

Computes a TT2000 epoch, nanoseconds since J2000, based on its UTC-based date/time component parts. The hour field can contain a fraction of a hour - a new name from fromUTCparts.

Parameters:

`year` - the year, a full year in double

`month` - the month, a full month in double

`day` - the day, either the day of the month or day of the year (DOY). For DOY, month has to be one (1), otherwise an exception is thrown. A full day in double

`hour` - the hour in double **Note:** Avoid passing in the time component that extends into the next day as that could present a potential problem.

Returns:

the TT2000 epoch value in long

Throws:

`CDFException` - an TT2000_TIME_ERROR if an illegal component value is detected.

fromUTCparts

```
public static long fromUTCparts(double year,
                double month,
                double day,
                double hour)
                         throws CDFException
```

Computes a TT2000 epoch, nanoseconds since J2000, based on its UTC-based date/time component parts. The hour field can contain a fraction of a hour.

Parameters:

`year` - the year, a full year in double

`month` - the month, a full month in double

`day` - the day, either the day of the month or day of the year (DOY). For DOY, month has to be one (1), otherwise an exception is thrown. A full day in double

`hour` - the hour in double **Note:** Avoid passing in the time component that extends into the next day as that could present a potential problem.

Returns:

the TT2000 epoch value in long

Throws:

`CDFException` - an TT2000_TIME_ERROR if an illegal component value is detected.

computeTT2000

```
public static long computeTT2000(double year,
                double month,
                double day,
                double hour,
                double minute)
                          throws CDFException
```

Computes a TT2000 epoch, nanoseconds since J2000, based on its UTC-based date/time component parts. The minute field can contain a fraction of a minute - a new name from fromUTCparts.

Parameters:

`year` - the year, a full year in double

`month` - the month, a full month in double

`day` - the day, either the day of the month or day of the year (DOY). For DOY, month has to be one (1), otherwise an exception is thrown. A full day in double

`hour` - the hour, a full hour in double

`minute` - the minute in double **Note:** Avoid passing in the time components that extend into the next day as that could present a potential problem.

Returns:

the TT2000 epoch value in long

Throws:

CDFException - an TT2000_TIME_ERROR if an illegal component value is detected.

fromUTCparts

```
public static long fromUTCparts(double year,
                double month,
                double day,
                double hour,
                double minute)
                         throws CDFException
```

Computes a TT2000 epoch, nanoseconds since J2000, based on its UTC-based date/time component parts. The minute field can contain a fraction of a minute.

Parameters:

year - the year, a full year in double

month - the month, a full month in double

day - the day, either the day of the month or day of the year (DOY). For DOY, month has to be one (1), otherwise an exception is thrown. A full day in double

hour - the hour, a full hour in double

minute - the minute in double **Note:** Avoid passing in the time components that extend into the next day as that could present a potential problem.

Returns:

the TT2000 epoch value in long

Throws:

CDFException - an TT2000_TIME_ERROR if an illegal component value is detected.

computeTT2000

```
public static long computeTT2000(double year,
                double month,
                double day,
                double hour,
                double minute,
                double second)
                          throws CDFException
```

Computes a TT2000 epoch, nanoseconds since J2000, based on its UTC-based date/time component parts. The second field can contain a fraction of a second - a new name from fromUTCparts.

Parameters:

year - the year, a full year in double

month - the month, a full month in double

day - the day, either the day of the month or day of the year (DOY). For DOY, month has to be one (1), otherwise an exception is thrown. A full day in double

hour - the hour, a full hour in double

minute - the minute, a full minute in double

second - the second in double **Note:** Avoid passing in the time components that extend into the next day as that could present a potential problem.

Returns:

the TT2000 epoch value in long

Throws:

`CDFException` - an TT2000_TIME_ERROR if an illegal component value is detected.

fromUTCparts

```
public static long fromUTCparts(double year,
                 double month,
                 double day,
                 double hour,
                 double minute,
                 double second)
                          throws CDFException
```

Computes a TT2000 epoch, nanoseconds since J2000, based on its UTC-based date/time component parts. The second field can contain a fraction of a second.

Parameters:

`year` - the year, a full year in double

`month` - the month, a full month in double

`day` - the day, either the day of the month or day of the year (DOY). For DOY, month has to be one (1), otherwise an exception is thrown. A full day in double

`hour` - the hour, a full hour in double

`minute` - the minute, a full minute in double

`second` - the second in double **Note:** Avoid passing in the time components that extend into the next day as that could present a potential problem.

Returns:

the TT2000 epoch value in long

Throws:

`CDFException` - an TT2000_TIME_ERROR if an illegal component value is detected.

computeTT2000

```
public static long computeTT2000(double year,
                 double month,
                 double day,
                 double hour,
                 double minute,
                 double second,
                 double milsec)
                          throws CDFException
```

Computes a TT2000 epoch, nanoseconds since J2000, based on its UTC-based date/time component parts. The millisecond field can contain a fraction of a millisecond - a new name from fromUTCparts.

Parameters:

`year` - the year, a full year in double

`month` - the month, a full month in double

`day` - the day, either the day of the month or day of the year (DOY). For DOY, month has to be one (1), otherwise an exception is thrown. A full day in double

`hour` - the hour, a full hour in double

`minute` - the minute, a full minute in double

`second` - the second, a full second in double

`milsec` - the millisecond in double **Note:** Avoid passing in the time components that extend into the next day as that could present a potential problem.

Returns:

the TT2000 epoch value in long

Throws:

`CDFException` - an TT2000_TIME_ERROR if an illegal component value is detected.

fromUTCparts

```
public static long fromUTCparts(double year,
                 double month,
                 double day,
                 double hour,
                 double minute,
                 double second,
                 double milsec)
                          throws CDFException
```

Computes a TT2000 epoch, nanoseconds since J2000, based on its UTC-based date/time component parts. The millisecond field can contain a fraction of a millisecond.

Parameters:

`year` - the year, a full year in double

`month` - the month, a full month in double

`day` - the day, either the day of the month or day of the year (DOY). For DOY, month has to be one (1), otherwise an exception is thrown. A full day in double

`hour` - the hour, a full hour in double

`minute` - the minute, a full minute in double

`second` - the second, a full second in double

`milsec` - the millisecond in double **Note:** Avoid passing in the time components that extend into the next day as that could present a potential problem.

Returns:

the TT2000 epoch value in long

Throws:

`CDFException` - an TT2000_TIME_ERROR if an illegal component value is detected.

computeTT2000

```
public static long computeTT2000(double year,
                  double month,
                  double day,
                  double hour,
                  double minute,
                  double second,
                  double milsec,
                  double micsec)
                           throws CDFException
```

Computes a TT2000 epoch, nanoseconds since J2000, based on its UTC-based date/time component parts. The microsecond field

can contain a fraction, which usually indicates the end of the parameter list - a new name from fromUTCparts.

Parameters:

`year` - the year, a full year in double

`month` - the month, a full month in double

`day` - the day, either the day of the month or day of the year (DOY). For DOY, month has to be one (1), otherwise an exception is thrown. A full day in double

`hour` - the hour, a full hour in double

`minute` - the minute, a full minute in double

`second` - the second, a full second in double

`milsec` - the millisecond, a full millisecond in double

`micsec` - the microsecond in double **Note:** Avoid passing in the time components that extend into the next day as that could present a potential problem.

Returns:

the TT2000 epoch value in long

Throws:

`CDFException` - an TT2000_TIME_ERROR if an illegal component value is detected.

fromUTCparts

```
public static long fromUTCparts(double year,
                double month,
                double day,
                double hour,
                double minute,
                double second,
                double milsec,
                double micsec)
                         throws CDFException
```

Computes a TT2000 epoch, nanoseconds since J2000, based on its UTC-based date/time component parts. The microsecond field can contain a fraction, which usually indicates the end of the parameter list.

Parameters:

`year` - the year, a full year in double

`month` - the month, a full month in double

`day` - the day, either the day of the month or day of the year (DOY). For DOY, month has to be one (1), otherwise an exception is thrown. A full day in double

`hour` - the hour, a full hour in double

`minute` - the minute, a full minute in double

`second` - the second, a full second in double

`milsec` - the millisecond, a full millisecond in double

`micsec` - the microsecond in double **Note:** Avoid passing in the time components that extend into the next day as that could present a potential problem.

Returns:

the TT2000 epoch value in long

Throws:

CDFException - an TT2000_TIME_ERROR if an illegal component value is detected.

computeTT2000

```
public static long computeTT2000(double year,
                 double month,
                 double day,
                 double hour,
                 double minute,
                 double second,
                 double msec,
                 double usec,
                 double nsec)
                          throws CDFException
```

Computes a TT2000 epoch, nanoseconds since J2000, based on its UTC-based date/time component parts - a new name from fromUTCparts.

Parameters:

year - the year, a full year in double

month - the month, a full month in double

day - the day, either the day of the month or day of the year (DOY). For DOY, month has to be one (1), otherwise an exception is thrown. A full day in double

hour - the hour, a full hour in double

minute - the minute, a full minute in double

second - the second, a full second in double

msec - the millisecond, a full millisecond in double

usec - the microsecond, a full microsecond in double

nsec - the nanosecond, a full nanosecond in double **Note:** Avoid passing in the time components that extend into the next day as that could present a potential problem.

Returns:

the TT2000 epoch value in long

Throws:

CDFException - an TT2000_TIME_ERROR if an illegal component value is detected.

fromUTCparts

```
public static long fromUTCparts(double year,
                double month,
                double day,
                double hour,
                double minute,
                double second,
                double msec,
                double usec,
                double nsec)
                          throws CDFException
```

Computes a TT2000 epoch, nanoseconds since J2000, based on its UTC-based date/time component parts.

Parameters:

year - the year, a full year in double

month - the month, a full month in double

`day` - the day, either the day of the month or day of the year (DOY). For DOY, month has to be one (1). A full day in double

`hour` - the hour, a full hour in double

`minute` - the minute, a full minute in double

`second` - the second, a full second in double

`msec` - the millisecond, a full millisecond in double

`usec` - the microsecond, a full microsecond in double

`nsec` - the nanosecond, a full nanosecond in double **Note:** Avoid passing in the time components that extend into the next day as that could present a potential problem.

Returns:

the TT2000 epoch value in long

Throws:

`CDFException` - an TT2000_TIME_ERROR if an illegal component value is detected.

compute

```
public static long compute(long year,
           long month,
           long day,
           long hour,
           long minute,
           long second,
           long msec,
           long usec,
           long nsec)
                    throws CDFException
```

Computes a TT2000 epoch, nanoseconds since J2000, based on its UTC-based date/time component parts.

Parameters:

`year` - the year, a full year in long

`month` - the month, a full month in long

`day` - the day, a full day in long

`hour` - the hour, a full hour in long

`minute` - the minute, a full minute in long

`second` - the second, a full second in long

`msec` - the millisecond, a full millisecond in long

`usec` - the microsecond, a full microsecond in long

`nsec` - the nanosecond, a full nanosecond in long **Note:** Avoid passing in the time components that extend into the next day as that could present a potential problem.

Returns:

the TT2000 epoch value in long

Throws:

`CDFException` - an TT2000_TIME_ERROR if an illegal component value is detected.

compute

```
public static long[] compute(long[] year,
             long[] month,
             long[] day)
                      throws CDFException
```

Computes an array of TT2000 epochs, nanoseconds since J2000, based on its UTC-based date/time component parts.

Parameters:

`year` - an array of years, a full year in long

`month` - an array of months, a full month in long

`day` - an array of days, a full day in long

Returns:

the TT2000 epoch values in long or null if an invalid input array(s) is detected

Throws:

`CDFException` - an TT2000_TIME_ERROR if an illegal component value is detected.

compute

```
public static long[] compute(long[] year,
             long[] month,
             long[] day,
             long[] hour)
                      throws CDFException
```

Computes an array of TT2000 epochs, nanoseconds since J2000, based on its UTC-based date/time component parts.

Parameters:

`year` - an array of years, a full year in long

`month` - an array of months, a full month in long

`day` - an array of days, a full day in long

`hour` - an array of hours, a full hour in long **Note:** Avoid passing in the time components that extend into the next day as that could present a potential problem.

Returns:

the TT2000 epoch value in long or null if an invalid input array(s) is detected

Throws:

`CDFException` - an TT2000_TIME_ERROR if an illegal component value is detected.

compute

```
public static long[] compute(long[] year,
             long[] month,
             long[] day,
             long[] hour,
             long[] minute)
                      throws CDFException
```

Computes an array of TT2000 epochs, nanoseconds since J2000, based on its * UTC-based date/time component parts.

Parameters:

`year` - an array of years, a full year in long

`month` - an array of months, a full month in long

`day` - an array of days, a full day in long

`hour` - an array of hours, a full hour in long

`minute` - an array of minutes, a full minute in long **Note:** Avoid passing in the time components that extend into the next day as that could present a potential problem.

Returns:

the TT2000 epoch values in long or null if an invalid input array(s) is detected

Throws:

`CDFException` - an TT2000_TIME_ERROR if an illegal component value is detected.

compute

```
public static long[] compute(long[] year,
             long[] month,
             long[] day,
             long[] hour,
             long[] minute,
             long[] second)
                      throws CDFException
```

Computes an array of TT2000 epochs, nanoseconds since J2000, based on its * UTC-based date/time component parts.

Parameters:

`year` - an array of years, a full year in long

`month` - an array of months, a full month in long

`day` - an array of days, a full day in long

`hour` - an array of hours, a full hour in long

`minute` - an array of minutes, a full minute in long

`second` - an array of seconds, a full second in long **Note:** Avoid passing in the time components that extend into the next day as that could present a potential problem.

Returns:

the TT2000 epoch value in long or null if an invalid input array(s) is detected

Throws:

`CDFException` - an TT2000_TIME_ERROR if an illegal component value is detected.

compute

```
public static long[] compute(long[] year,
             long[] month,
             long[] day,
             long[] hour,
             long[] minute,
             long[] second,
             long[] msec)
                      throws CDFException
```

Computes an array of TT2000 epochs, nanoseconds since J2000, based on its * UTC-based date/time component parts.

Parameters:

`year` - an array of years, a full year in long

`month` - an array of months, a full month in long

`day` - an array of days, a full day in long

`hour` - an array of hours, a full hour in long

`minute` - an array of minutes, a full minute in long

`second` - an array of seconds, a full second in long

`msec` - an array of milliseconds, a full millisecond in long **Note:** Avoid passing in the time components that extend into the next day as that could present a potential problem.

Returns:

the TT2000 epoch values in long or null if an invalid input array(s) is detected

Throws:

`CDFException` - an TT2000_TIME_ERROR if an illegal component value is detected.

compute

```
public static long[] compute(long[] year,
             long[] month,
             long[] day,
             long[] hour,
             long[] minute,
             long[] second,
             long[] msec,
             long[] usec)
                      throws CDFException
```

Computes an array of TT2000 epochs, nanoseconds since J2000, based on its * UTC-based date/time component parts.

Parameters:

`year` - an array of years, a full year in long

`month` - an array of months, a full month in long

`day` - an array of days, a full day in long

`hour` - an array of hours, a full hour in long

`minute` - an array of minutes, a full minute in long

`second` - an array of seconds, a full second in long

`msec` - an array of milliseconds, a full millisecond in long

`usec` - an array of microseconds, a full microsecond in long **Note:** Avoid passing in the time components that extend into the next day as that could present a potential problem.

Returns:

the TT2000 epoch value in long or null if an invalid input array(s) is detected

Throws:

`CDFException` - an TT2000_TIME_ERROR if an illegal component value is detected.

compute

```
public static long[] compute(long[] year,
             long[] month,
             long[] day,
             long[] hour,
```

```
                long[] minute,
                long[] second,
                long[] msec,
                long[] usec,
                long[] nsec)
                         throws CDFException
```

Computes an array of TT2000 epochs, nanoseconds since J2000, based on its * UTC-based date/time component parts.

Parameters:

`year` - an array of years, a full year in long

`month` - an array of months, a full month in long

`day` - an array of days, a full day in long

`hour` - an array of hours, a full hour in long

`minute` - an array of minutes, a full minute in long

`second` - an array of seconds, a full second in long

`msec` - an array of milliseconds, a full millisecond in long

`usec` - an array of microseconds, a full microsecond in long

`nsec` - an array of nanoseconds, a full nanosecond in long **Note:** Avoid passing in the time components that extend into the next day as that could present a potential problem.

Returns:

the TT2000 epoch values in long or null if an invalid input array(s) is detected

Throws:

`CDFException` - an TT2000_TIME_ERROR if an illegal component value is detected.

toUTCEPOCH

```
public static double toUTCEPOCH(long nanoSecSinceJ2000)
                          throws CDFException
```

Convert an epoch value in TT2000 type to CDF_EPOCH type.

Parameters:

`nanoSecSinceJ2000` - the nanoseconds since J2000

Returns:

the epoch value

Throws:

`CDFException` - an ILLEGAL_EPOCH_FIELD if an illegal component value is detected.

fromUTCEPOCH

```
public static long fromUTCEPOCH(double epoch)
                          throws CDFException
```

Convert an epoch value in CDF_EPOCH type to TT2000 type. Reset the predefined CDF_EPOCH value of -1.0E31, -1.0E-31, or the default 0.0 or -0.0 (all are invalid for TT2000) to 0.

Parameters:

`epoch`

- the CDF_EPOCH value

Returns:

the TT2000 epoch in nanoseconds since J2000

Throws:

`CDFException` - an TT2000_TIME_ERROR if date is out of the valid range for TT2000.

toUTCEPOCH16

```
public static double toUTCEPOCH16(long nanoSecSinceJ2000,
                  double[] epoch)
                           throws CDFException
```

Convert an epoch value in TT2000 type to CDF_EPOCH16 type.

Parameters:

`nanoSecSinceJ2000` - the nanoseconds since J2000

`epoch` - the returned CDF_EPOCH16 value, a double[2] object

Returns:

the status

Throws:

`CDFException` - an ILLEGAL_EPOCH_FIELD if an illegal component value is detected.

fromUTCEPOCH16

```
public static long fromUTCEPOCH16(double[] epoch)
                           throws CDFException
```

Convert an epoch data in CDF_EPOCH16 type to TT2000 type. Reset the predefined CDF_EPOCH value of -1.0E31, -1.0E-31, or the default 0.0 or -0.0 (all are invalid for TT2000) to 0.

Parameters:

`epoch` - the CDF_EPOCH16 value, a double[2] object

Returns:

the TT2000 epoch in nanoseconds since J2000

Throws:

`CDFException` - an TT2000_TIME_ERROR if date is out of the valid range for TT2000.

toUTCstring

```
public static java.lang.String toUTCstring(java.lang.Long nanoSecSinceJ2000)
```

Converts an epoch value in TT2000 type into a readable, UTC-based date/time string of ISO 8601 style.

```
            yyyy-mm-ddThh:mm:ss.ccccccccc
 Examples:  1990-04-01T03:05:02.000000000
            1993-10-10T23:45:49.777888999
```

Parameters:

`nanoSecSinceJ2000` - the TT2000 epoch value, nanoseconds since J2000, in Long object

Returns:

A string representation of the epoch in ISO 8601

toUTCstring

```
public static java.lang.String toUTCstring(long nanoSecSinceJ2000)
```

Converts an epoch value in TT2000 type into a readable, UTC-based date/time string of ISO 8601 style.

```
            yyyy-mm-ddThh:mm:ss.ccccccccc
 Examples:  1990-04-01T03:05:02.000000000
            1993-10-10T23:45:49.777888999
```

Parameters:

`nanoSecSinceJ2000` - the TT2000 epoch value, nanoseconds since J2000, in long

Returns:

A string representation of the epoch

toUTCstring

```
public static java.lang.String[] toUTCstring(long[] nanoSecSinceJ2000)
```

Converts an iarray of epoch values in TT2000 type into readable, UTC-based date/time strings of ISO 8601 style.

```
            yyyy-mm-ddThh:mm:ss.ccccccccc
 Examples:  1990-04-01T03:05:02.000000000
            1993-10-10T23:45:49.777888999
```

Parameters:

`nanoSecSinceJ2000` - the TT2000 epoch values, nanoseconds since J2000, in long

Returns:

A string array representation of the epochs

toUTCstring

```
public static java.lang.String toUTCstring(java.lang.Long nanoSecSinceJ2000,
                           int style)
```

Converts an epoch value in TT2000 type into a readable, UTC-based date/time string of chosen style.

```
 Style:0    dd-mmm-yyyy hh:mm:ss.ccccccccc
 Examples:  01-Apr-1990 03:05:02.000000000
            10-Oct-1993 23:45:49.777888999

 Style:1    yyyymmdd.ccccccccc
 Examples:  19900401.1234567890
            19931010.9998887776

 Style:2    yyyymmddhhmmss
 Examples:  19900401030502
            19931010234549

 Style:3    yyyy-mm-ddThh:mm:ss.ccccccccc
 Examples:  1990-04-01T03:05:02.000000000
```

```
            1993-10-10T23:45:49.777888999
 This is the default, ISO 8601, output.

 Style:4    yyyy-mm-ddThh:mm:ss.ccccccccZ
 Examples:  1990-04-01T03:05:02.000000000Z
            1993-10-10T23:45:49.777888999Z
```

These styles are the same as those expected by fromUTCstring.

Parameters:

`nanoSecSinceJ2000` - the TT2000 epoch value, nanoseconds since J2000, in Long object

`style` - the style (from 0 to 4, 3 being the default), an optional

Returns:

A string representation of the epoch

toEncode

```
public static java.lang.String toEncode(java.lang.Long nanoSecSinceJ2000,
                        int style)
```

Converts an epoch value in TT2000 type into a readable, UTC-based date/time string of chosen style.

```
 Style:0    dd-mmm-yyyy hh:mm:ss.ccccccccc
 Examples:  01-Apr-1990 03:05:02.000000000
            10-Oct-1993 23:45:49.777888999

 Style:1    yyyymmdd.ccccccccc
 Examples:  19900401.1234567890
            19931010.9998887776

 Style:2    yyyymmddhhmmss
 Examples:  19900401030502
            19931010234549

 Style:3    yyyy-mm-ddThh:mm:ss.ccccccccc
 Examples:  1990-04-01T03:05:02.000000000
            1993-10-10T23:45:49.777888999
 This is the default, ISO 8601, output.

 Style:4    yyyy-mm-ddThh:mm:ss.ccccccccZ
 Examples:  1990-04-01T03:05:02.000000000Z
            1993-10-10T23:45:49.777888999Z
```

These styles are the same as those expected by fromUTCstring.

Parameters:

`nanoSecSinceJ2000` - the TT2000 epoch value, nanoseconds since J2000, in Long object

`style` - the style (from 0 to 4, 3 being the default), an optional

Returns:

A string representation of the epoch

toEncode

```
public static java.lang.String toEncode(long nanoSecSinceJ2000)
```

Converts an epoch value in TT2000 type into a readable, UTC-based date/time string of ISO 8601 forme.

```
 Examples:  1990-04-01T03:05:02.000000000
            1993-10-10T23:45:49.777888999
 This is the default, ISO 8601, output.
```

These styles are the same as those expected by fromUTCstring.

Parameters:

`nanoSecSinceJ2000` - the TT2000 epoch value, nanoseconds since J2000, in long

Returns:

A string representation of the epoch

toEncode

```
public static java.lang.String toEncode(long nanoSecSinceJ2000,
                        int style)
```

Converts an epoch value in TT2000 type into a readable, UTC-based date/time string of chosen style.

```
 Style:0    dd-mmm-yyyy hh:mm:ss.ccccccccc
 Examples:  01-Apr-1990 03:05:02.000000000
            10-Oct-1993 23:45:49.777888999

 Style:1    yyyymmdd.ccccccccc
 Examples:  19900401.1234567890
            19931010.9998887776

 Style:2    yyyymmddhhmmss
 Examples:  19900401030502
            19931010234549

 Style:3    yyyy-mm-ddThh:mm:ss.ccccccccc
 Examples:  1990-04-01T03:05:02.000000000
            1993-10-10T23:45:49.777888999
 This is the default, ISO 8601, output.

 Style:4    yyyy-mm-ddThh:mm:ss.cccccccccZ
 Examples:  1990-04-01T03:05:02.000000000Z
            1993-10-10T23:45:49.777888999Z
```

These styles are the same as those expected by fromUTCstring.

Parameters:

`nanoSecSinceJ2000` - the TT2000 epoch value, nanoseconds since J2000, in long

`style` - the style (from 0 to 4, 3 being the default), an optional

Returns:

A string representation of the epoch

toEncode

```
public static java.lang.Object toEncode(java.lang.Object epoch)
```

Converts an epoch object in TT2000 type into a scalar or array(s) of readable, UTC-based date/time string of ISO8601 style (style 3). Each string is presented in the following form:

```
 Examples:  1990-04-01T03:05:02.000000000
            1993-10-10T23:45:49.777888999
```

Parameters:

`epoch` - the TT2000 epoch object, nanoseconds since J2000, in long

Returns:

A string or array(s) of strings representation of the epoch

toEncode

```
public static java.lang.String[] toEncode(long[] nanoSecSinceJ2000)
```

Converts an epoch value in TT2000 type into a readable, UTC-based date/time string of ISO8601 style (style 3).

```
 Examples:  1990-04-01T03:05:02.000000000
            1993-10-10T23:45:49.777888999
```

/** Converts an array of epoch values in TT2000 type into readable, UTC-based date/time strings of ISO8601 style.

```
 Examples:  1990-04-01T03:05:02.000000000
            1993-10-10T23:45:49.777888999
```

Parameters:

`nanoSecSinceJ2000` - the TT2000 epoch values, nanoseconds since J2000, in long

Returns:

A string array representation of the epochs

toEncode

```
public static java.lang.String[] toEncode(long[] nanoSecSinceJ2000,
                          int style)
```

Converts an array of epoch values in TT2000 type into readable, UTC-based date/time strings of chosen style.

```
 Style:0    dd-mmm-yyyy hh:mm:ss.ccccccccc
 Examples:  01-Apr-1990 03:05:02.000000000
            10-Oct-1993 23:45:49.777888999

 Style:1    yyyymmdd.ccccccccc
 Examples:  19900401.1234567890
            19931010.9998887776

 Style:2    yyyymmddhhmmss
 Examples:  19900401030502
            19931010234549

 Style:3    yyyy-mm-ddThh:mm:ss.ccccccccc
 Examples:  1990-04-01T03:05:02.000000000
            1993-10-10T23:45:49.777888999
 This is the default, ISO 8601, output.

 Style:4    yyyy-mm-ddThh:mm:ss.cccccccccZ
 Examples:  1990-04-01T03:05:02.000000000Z
            1993-10-10T23:45:49.777888999Z
```

These styles are the same as those expected by fromUTCstring.

Parameters:

`nanoSecSinceJ2000` - the TT2000 epoch values, nanoseconds since J2000, in long

`style` - the style (from 0 to 4, 3 being the default), an optional

Returns:

A string array representation of the epoch

toUTCstring

```
public static java.lang.String toUTCstring(long nanoSecSinceJ2000,
                           int style)
```

Converts an epoch value in TT2000 type into a readable, UTC-based date/time string of chosen style.

```
 Style:0    dd-mmm-yyyy hh:mm:ss.ccccccccc
 Examples:  01-Apr-1990 03:05:02.000000000
            10-Oct-1993 23:45:49.777888999

 Style:1    yyyymmdd.ccccccccc
 Examples:  19900401.1234567890
            19931010.9998887776

 Style:2    yyyymmddhhmmss
 Examples:  19900401030502
            19931010234549

 Style:3    yyyy-mm-ddThh:mm:ss.ccccccccc
 Examples:  1990-04-01T03:05:02.000000000
            1993-10-10T23:45:49.777888999
 This is the default, ISO 8601, output.

 Style:4    yyyy-mm-ddThh:mm:ss.cccccccccZ
 Examples:  1990-04-01T03:05:02.000000000Z
            1993-10-10T23:45:49.777888999Z
```

These styles are the same as those expected by fromUTCstring.

Parameters:

`nanoSecSinceJ2000` - the TT2000 epoch value, nanoseconds since J2000, in long

`style` - the style (from 0 to 4, 3 being the default), an optional

Returns:

A string representation of the epoch

toUTCstring

```
public static java.lang.String[] toUTCstring(long[] nanoSecSinceJ2000,
                             int style)
```

Converts an array of epoch values in TT2000 type into readable, UTC-based date/time strings of chosen style.

```
 Style:0    dd-mmm-yyyy hh:mm:ss.ccccccccc
 Examples:  01-Apr-1990 03:05:02.000000000
            10-Oct-1993 23:45:49.777888999

 Style:1    yyyymmdd.ccccccccc
 Examples:  19900401.1234567890
            19931010.9998887776

 Style:2    yyyymmddhhmmss
 Examples:  19900401030502
            19931010234549

 Style:3    yyyy-mm-ddThh:mm:ss.ccccccccc
 Examples:  1990-04-01T03:05:02.000000000
            1993-10-10T23:45:49.777888999
 This is the default, ISO 8601, output.

 Style:4    yyyy-mm-ddThh:mm:ss.cccccccccZ
 Examples:  1990-04-01T03:05:02.000000000Z
            1993-10-10T23:45:49.777888999Z
```

These styles are the same as those expected by fromUTCstring.

Parameters:

`nanoSecSinceJ2000` - the TT2000 epoch values, nanoseconds since J2000, in long

`style` - the style (from 0 to 3, as the default), an optional

Returns:

A string array representation of the epochs

encode

```
public static java.lang.String[] encode(long[] nanoSecSinceJ2000,
                        int style)
```

Converts an array of epoch values in TT2000 type into readable, UTC-based date/time strings of chosen style.

```
 Style:0    dd-mmm-yyyy hh:mm:ss.ccccccccc
 Examples:  01-Apr-1990 03:05:02.000000000
            10-Oct-1993 23:45:49.777888999

 Style:1    yyyymmdd.cccccccccc
 Examples:  19900401.1234567890
            19931010.9998887776

 Style:2    yyyymmddhhmmss
 Examples:  19900401030502
            19931010234549

 Style:3    yyyy-mm-ddThh:mm:ss.ccccccccc
 Examples:  1990-04-01T03:05:02.000000000
            1993-10-10T23:45:49.777888999
 This is the default, ISO 8601, output.

 Style:4    yyyy-mm-ddThh:mm:ss.cccccccccZ
 Examples:  1990-04-01T03:05:02.000000000Z
            1993-10-10T23:45:49.777888999Z
```

These styles are the same as those expected by fromUTCstring.

Parameters:

`nanoSecSinceJ2000` - the TT2000 epoch values, nanoseconds since J2000, in long

`style` - the style (from 0 to 3, as the default), an optional

Returns:

A string array representation of the epochs

encode

```
public static java.lang.String encode(long nanoSecSinceJ2000)
```

Converts an epoch value in TT2000 type into a readable, UTC-based date/time string in ISO 8601 style.

```
            yyyy-mm-ddThh:mm:ss.ccccccccc
 Examples:  1990-04-01T03:05:02.000000000
            1993-10-10T23:45:49.777888999
```

Parameters:

`nanoSecSinceJ2000` - the TT2000 epoch value, nanoseconds since J2000, in long

Returns:

A string representation of the epoch

encode

```
public static java.lang.String[] encode(long[] nanoSecSinceJ2000)
```

Converts an array of epoch values in TT2000 type into an array of readable, UTC-based date/time strings in ISO 8601 style.

```
            yyyy-mm-ddThh:mm:ss.ccccccccc
 Examples:  1990-04-01T03:05:02.000000000
            1993-10-10T23:45:49.777888999
```

Parameters:

`nanoSecSinceJ2000` - the TT2000 epoch value, nanoseconds since J2000, in long

Returns:

strings representing the epochs or null if null or an empty input array is detected

encode

```
public static java.lang.String encode(long nanoSecSinceJ2000,
                      int style)
```

Converts an epoch value in TT2000 type into a readable, UTC-based date/time string of chosen style.

```
 Style:0    dd-mmm-yyyy hh:mm:ss.ccccccccc
 Examples:  01-Apr-1990 03:05:02.000000000
            10-Oct-1993 23:45:49.777888999

 Style:1    yyyymmdd.ccccccccc
 Examples:  19900401.1234567890
            19931010.9998887776

 Style:2    yyyymmddhhmmss
 Examples:  19900401030502
            19931010234549

 Style:3    yyyy-mm-ddThh:mm:ss.ccccccccc
 Examples:  1990-04-01T03:05:02.000000000
            1993-10-10T23:45:49.777888999
 This is the default, ISO 8601, output.

 Style:4    yyyy-mm-ddThh:mm:ss.cccccccccZ
 Examples:  1990-04-01T03:05:02.000000000Z
            1993-10-10T23:45:49.777888999Z
```

These styles are the same as those expected by fromUTCstring.

Parameters:

`nanoSecSinceJ2000` - the TT2000 epoch value, nanoseconds since J2000, in long

`style` - the style (from 0 to 3, as the default), an optional

Returns:

A string representation of the epoch

parseTT2000

```
public static long parseTT2000(java.lang.String string)
                        throws CDFException
```

This method parses an input, UTC-based, date/time string and returns a TT2000 epoch value, nanoseconds since J2000. The string must be in one of the styles as shown below. Month abbreviations may be in any case and are always the first three letters of the month - a new name from fromUTCstring.

```
 Style:0    dd-mmm-yyyy hh:mm:ss.ccccccccc
 Examples:  01-Apr-1990 03:05:02.000000000
            10-Oct-1993 23:45:49.777888999

 Style:1    yyyymmdd.ccccccccc
 Examples:  19900401.1234567890
            19931010.9998887776

 Style:2    yyyymmddhhmmss
 Examples:  19900401030502
            19931010234549

 Style:3    yyyy-mm-ddThh:mm:ss.ccccccccc
 Examples:  1990-04-01T03:05:02.000000000
            1993-10-10T23:45:49.777888999

 Style:4    yyyy-mm-ddThh:mm:ss.cccccccccZ
 Examples:  1990-04-01T03:05:02.000000000Z
            1993-10-10T23:45:49.777888999Z
```

These styles are the same as those created by toUTCstring.

Parameters:

`string` - the epoch in string representation

Returns:

A TT2000 epoch the epoch value in nanoseconds since J2000

Throws:

`CDFException` - an TT2000_TIME_ERROR if an illegal component value is detected.

toParse

```
public static long toParse(java.lang.String string)
                  throws CDFException
```

This method parses an input, UTC-based, date/time string and returns a TT2000 epoch value, nanoseconds since J2000. The string must be in one of the styles as shown below. Month abbreviations may be in any case and are always the first three letters of the month - a new name from fromUTCstring.

```
 Style:0    dd-mmm-yyyy hh:mm:ss.ccccccccc
 Examples:  01-Apr-1990 03:05:02.000000000
            10-Oct-1993 23:45:49.777888999

 Style:1    yyyymmdd.ccccccccc
 Examples:  19900401.1234567890
            19931010.9998887776

 Style:2    yyyymmddhhmmss
 Examples:  19900401030502
            19931010234549

 Style:3    yyyy-mm-ddThh:mm:ss.ccccccccc
 Examples:  1990-04-01T03:05:02.000000000
            1993-10-10T23:45:49.777888999

 Style:4    yyyy-mm-ddThh:mm:ss.cccccccccZ
 Examples:  1990-04-01T03:05:02.000000000Z
            1993-10-10T23:45:49.777888999Z
```

These styles are the same as those created by toUTCstring.

Parameters:

`string` - the epoch in string representation

Returns:

A TT2000 epoch the epoch value in nanoseconds since J2000

Throws:

`CDFException` - an TT2000_TIME_ERROR if an illegal component value is detected.

parseTT2000

```
public static long[] parseTT2000(java.lang.String[] strings)
                          throws CDFException
```

This method parses an input, UTC-based, date/time strings and returns a TT2000 epoch value array, nanoseconds since J2000. The string must be in one of the styles as shown below. Month abbreviations may be in any case and are always the first three letters of the month - a new name from fromUTCstring.

```
 Style:0    dd-mmm-yyyy hh:mm:ss.ccccccccc
 Examples:  01-Apr-1990 03:05:02.000000000
            10-Oct-1993 23:45:49.777888999

 Style:1    yyyymmdd.ccccccccc
 Examples:  19900401.1234567890
            19931010.9998887776

 Style:2    yyyymmddhhmmss
 Examples:  19900401030502
            19931010234549

 Style:3    yyyy-mm-ddThh:mm:ss.ccccccccc
 Examples:  1990-04-01T03:05:02.000000000
            1993-10-10T23:45:49.777888999

 Style:4    yyyy-mm-ddThh:mm:ss.cccccccccZ
 Examples:  1990-04-01T03:05:02.000000000Z
            1993-10-10T23:45:49.777888999Z
```

These styles are the same as those created by toUTCstring.

Parameters:

`strings` - the epoch array in string representation

Returns:

A TT2000 epoch the epoch values in nanoseconds since J2000

Throws:

`CDFException` - an TT2000_TIME_ERROR if an illegal component value is detected.

toParse

```
public static long[] toParse(java.lang.String[] strings)
                      throws CDFException
```

This method parses an input, UTC-based, date/time strings and returns a TT2000 epoch value array, nanoseconds since J2000. The string must be in one of the styles as shown below. Month abbreviations may be in any case and are always the first three letters of the month - a new name from fromUTCstring.

```
 Style:0    dd-mmm-yyyy hh:mm:ss.ccccccccc
 Examples:  01-Apr-1990 03:05:02.000000000
            10-Oct-1993 23:45:49.777888999

 Style:1    yyyymmdd.ccccccccc
 Examples:  19900401.1234567890
            19931010.9998887776

 Style:2    yyyymmddhhmmss
 Examples:  19900401030502
```

```
            19931010234549

 Style:3    yyyy-mm-ddThh:mm:ss.ccccccccc
 Examples:  1990-04-01T03:05:02.000000000
            1993-10-10T23:45:49.777888999

 Style:4    yyyy-mm-ddThh:mm:ss.cccccccccZ
 Examples:  1990-04-01T03:05:02.000000000Z
            1993-10-10T23:45:49.777888999Z
```

These styles are the same as those created by toUTCstring.

Parameters:

`strings` - the epoch array in string representation

Returns:

A TT2000 epoch the epoch values in nanoseconds since J2000

Throws:

`CDFException` - an TT2000_TIME_ERROR if an illegal component value is detected.

fromUTCstring

```
public static long fromUTCstring(java.lang.String string)
                          throws CDFException
```

This method parses an input, UTC-based, date/time string and returns a TT2000 epoch value, nanoseconds since J2000. The string must be in one of the styles as shown below. Month abbreviations may be in any case and are always the first three letters of the month.

```
 Style:0    dd-mmm-yyyy hh:mm:ss.ccccccccc
 Examples:  01-Apr-1990 03:05:02.000000000
            10-Oct-1993 23:45:49.777888999

 Style:1    yyyymmdd.ccccccccc
 Examples:  19900401.1234567890
            19931010.9998887776

 Style:2    yyyymmddhhmmss
 Examples:  19900401030502
            19931010234549

 Style:3    yyyy-mm-ddThh:mm:ss.ccccccccc
 Examples:  1990-04-01T03:05:02.000000000
            1993-10-10T23:45:49.777888999

 Style:4    yyyy-mm-ddThh:mm:ss.cccccccccZ
 Examples:  1990-04-01T03:05:02.000000000Z
            1993-10-10T23:45:49.777888999Z
```

These styles are the same as those created by toUTCstring.

Parameters:

`string` - the epoch in string representation

Returns:

A TT2000 epoch the epoch value in nanoseconds since J2000

Throws:

`CDFException` - an TT2000_TIME_ERROR if an illegal component value is detected.

fromUTCISO8601string

```
public static long[] fromUTCISO8601string(java.lang.String string)
                                    throws CDFException
```

This method parses an input, UTC-based, date/time string in ISO 8601 and returns their date/time components. String has to be in ISO 8601 form: yyyy-mm-ddThh:mm:ss.ccccccccc 1990-04-01T03:05:02.000000000 1993-10-10T23:45:49.777888999

Parameters:

`string` - the epoch in string representation

Returns:

UTC date/time components

Throws:

`CDFException` - an TT2000_TIME_ERROR if an illegal component value is detected.

parse

```
public static long parse(java.lang.String string)
                  throws CDFException
```

This method parses an input date/time string and returns a TT2000 epoch value, nanoseconds since J2000. The string must be in one of the styles as shown below. Month abbreviations may be in any case and are always the first three letters of the month.

```
 Style:0    dd-mmm-yyyy hh:mm:ss.ccccccccc
 Examples:  01-Apr-1990 03:05:02.000000000
            10-Oct-1993 23:45:49.777888999

 Style:1    yyyymmdd.ccccccccc
 Examples:  19900401.1234567890
            19931010.9998887776

 Style:2    yyyymmddhhmmss
 Examples:  19900401030502
            19931010234549

 Style:3    yyyy-mm-ddThh:mm:ss.ccccccccc
 Examples:  1990-04-01T03:05:02.000000000
            1993-10-10T23:45:49.777888999

 Style:4    yyyy-mm-ddThh:mm:ss.cccccccccZ
 Examples:  1990-04-01T03:05:02.000000000Z
            1993-10-10T23:45:49.777888999Z
```

These styles are the same as those created by toUTCstring.

Parameters:

`string` - the epoch in string representation

Returns:

A TT2000 epoch the epoch value in nanoseconds since J2000

Throws:

`CDFException` - an TT2000_TIME_ERROR if an illegal component value is detected.

parse

```
public static long[] parse(java.lang.String[] strings)
                    throws CDFException
```

This method parses an array of input date/time strings and returns an array of TT2000 epoch values, nanoseconds since J2000. The

string must be in one of the styles as shown below. Month abbreviations may be in any case and are always the first three letters of the month.

```
 Style:0    dd-mmm-yyyy hh:mm:ss.ccccccccc
 Examples:  01-Apr-1990 03:05:02.000000000
            10-Oct-1993 23:45:49.777888999

 Style:1    yyyymmdd.ccccccccc
 Examples:  19900401.1234567890
            19931010.9998887776

 Style:2    yyyymmddhhmmss
 Examples:  19900401030502
            19931010234549

 Style:3    yyyy-mm-ddThh:mm:ss.ccccccccc
 Examples:  1990-04-01T03:05:02.000000000
            1993-10-10T23:45:49.777888999

 Style:4    yyyy-mm-ddThh:mm:ss.cccccccccZ
 Examples:  1990-04-01T03:05:02.000000000Z
            1993-10-10T23:45:49.777888999Z
```

These styles are the same as those created by toUTCstring.

Parameters:

`strings` - an array of epochs in string representation

Returns:

TT2000 epoch values in nanoseconds since J2000 or null if null or an empty input array is detected

Throws:

`CDFException` - an TT2000_TIME_ERROR if an illegal component value is detected.

CDFgetLastDateinLeapSecondsTable

```
public static long[] CDFgetLastDateinLeapSecondsTable()
```

/** This method returns the last UTC date that a leap second was added in the leap second table used in the class. This can be used to check whether the table is up-to-date.

Returns:

the date (year, month, day) in a long array

fromGregorianTime

```
public static long fromGregorianTime(java.util.GregorianCalendar gc)
```

This method converts the UTC-based date/time in a GregorianCalendar class object to TT2000 time.

Parameters:

`gc` - the Gregorian calendar object

Returns:

the TT2000 time

toGregorianTime

```
public static java.util.GregorianCalendar toGregorianTime(long tt2000)
```

This method converts the UTC-based date/time in TT2000 type to a GregorianCalendar class object in default, local time zone.

Parameters:

`tt2000` - the TT2000 data value

Returns:

a GregorianCalendar object

toGregorianTime

```
public static java.util.GregorianCalendar toGregorianTime(long tt2000,
                                           java.util.TimeZone tz)
```

This method converts the UTC-based date/time in TT2000 type to a GregorianCalendar class object in specified time zone.

Parameters:

`tt2000` - the TT2000 data value

`tz` - the time zone

Returns:

a GregorianCalendar object

CDFgetLeapSecondsTableStatus

```
public static int CDFgetLeapSecondsTableStatus()
```

This method returns the status code reflecting whether the leap seconds are from a external file, defined by an environment variable, or the leap seconds are based on the hard-coded table in the class.

Returns:

status 1 if form a file, 0 hard-coded

CDFgetLeapSecondsTable

```
public static double[][] CDFgetLeapSecondsTable()
```

This method returns the leap seconds table.

Returns:

table The table contents of the leap seconds

CDFgetRowsinLeapSecondsTable

```
public static int CDFgetRowsinLeapSecondsTable()
```

This method returns the number of entries in the leap seconds table.

Returns:

enrtyCnt The entry count in the leap seconds table

computeTT2000withBasedLeapDay

```
public static long computeTT2000withBasedLeapDay(long yy,
                                  long mm,
                                  long dd,
                                  long hh,
                                  long mn,
                                  long ss,
                                  long ms,
                                  long us,
                                  long ns,
                                  int yymmdd)
```

This method returns the TT2000 time in nanoseconds based on the last leap second day.

Parameters:

`yy` - the year, a full year in long

`mm` - the month, a full month in long

`dd` - the day, a full day in long

`hh` - the hour, a full hour in long

`mn` - the minute, a full minute in long

`ss` - the second, a full second in long

`ms` - the millisecond, a full millisecond in long

`us` - the microsecond, a full microsecond in long

`ns` - the nanosecond, a full nanosecond in long

`yymmdd` - the last leap second day (in YYYYMMDD) the UTC date is based upon, an int value

Returns:

TT2000 time in nanoseconds

computeTT2000withBasedLeapDay

```
public static long computeTT2000withBasedLeapDay(double yy,
                                  double mm,
                                  double dd,
                                  double hh,
                                  double mn,
                                  double ss,
                                  double ms,
                                  double us,
                                  double ns,
                                  int yymmdd)
```

This method returns the TT2000 time in nanoseconds based on the last leap second day. This function works similarly to fromUTCparts, with an adjustment if necessary. It returns the TT2000 from UTC date/time based on the passed last leap second day, while fromUTCparts function returns based on the day from the last entry in the current leap second table (LST). If the based leap second day is not a zero (0) or if the UTC date/time is extended over the next leap second day in the LST, one second(s) will be adjusted (deduction), due to the newly added leap second(s) in the LST. Otherwise, it returns the same value as fromUTCparts.

Parameters:

`yy` - the year, a full year in double

`mm` - the month, a full month in double

`dd` - the day, a full day in double

`hh` - the hour, a full hour in double

`mn` - the minute, a full minute in double

`ss` - the second, a full second in double

`ms` - the millisecond, a full millisecond in double

`us` - the microsecond, a full microsecond in double

`ns` - the nanosecond, a full nanosecond in double

`yymmdd` - the last leap second day (in YYYYMMDD) the UTC date is based upon, an int value

Returns:

TT2000 time in nanoseconds

breakdownTT2000withBasedLeapDay

```
public static long[] breakdownTT2000withBasedLeapDay(long tt2000,
                                     int yymmdd)
```

breakdownTT2000withBasedLeapDay. This method is similar to toUTCparts (aka breakdown), with an adjustment if necessary. It returns UTC date/time from TT2000 value based on the passed last leap second day, while toUTCparts method is based on the last leap second updated day in the current leap second table (LST). If the based leap second day is zero (0), this method functions just like toUTCparts. If the TT2000 value was computed based on the LST that was not up-to-date, one second(s) adjustment (addition) might be necessay if its value crosses over the next new leap second day in the LST.

Parameters:

`tt2000` - the nanoseconds since J2000

`yymmdd` - the last leap second day (in YYYYMMDD) the UTC date is based upon, an int value

Returns:

TT2000 time in nanoseconds

encodeTT2000withBasedLeapDay

```
public static java.lang.String encodeTT2000withBasedLeapDay(long tt2000,
                                            int yymmdd)
```

encodeTT2000withBasedLeapDay. This method is similar to toUTCstring (aka encodeTT2000), with an adjustment if necessary. It encodes TT2000 to UTC string in ISO 8601 form based on the passed last leap second day, while toUTC_string method is based on the last leap second updated day in the current leap second table (LST). If the based leap second day is zero (0), this method functions just like toUTCstring. If the TT2000 value was computed based on the LST that was not up-to-date, one second(s) adjustment (addition) might be necessay if its value crosses over the next new leap second day in the LST.

Parameters:

`tt2000` - the nanoseconds since J2000 with leap seconds second day

`yymmdd` - the last leap second day (in YYYYMMDD) the UTC date is based upon, an int value

Returns:

TT2000 an ISO 8601 UTC string

parseTT2000withBasedLeapDay

```
public static long parseTT2000withBasedLeapDay(java.lang.String tt2000,
                               int yymmdd)
```

parseTT2000withBasedLeapDay. This method is similar to fromUTCstring (aka parseTT2000), with an adjustment if necessary. It returns a TT2000 value from the UTC string in ISO 8601 form based on the passed last leap second day, while fromUTCstring method is based on the last leap second updated day in the current leap second table (LST). If the based leap second day is zero (0), this method functions just like fromUTCstring. If the UTC string was encoded based on the LST that was not up-to-date, one second(s) adjustment (addition) might be necessary if its day crosses over the next new leap second day in the LST.

Parameters:

`tt2000` - the UTC string (in ISO 8601 form) based on the last leap second day

`yymmdd` - the last leap second day (in YYYYMMDD) the UTC date is based upon, an int value

Returns:

TT2000 time in nanoseconds

TT2000toUnixTime

```
public static double TT2000toUnixTime(long epoch)
```

Converts a TT2000 value to a unix time of a double value (seconds from 1970-01-01 00:00:00 UTC). The unix time will have a time resolution of microseconds at its fraction part. Note: As unix time does not have leap seconds, the converted time might not be properly presented. Also, TT2000 have a higher time resolution than the unix time. The conversion might not precisely preserve the microseconds due to the floating point presentation.

Parameters:

`epoch` - the epoch value

Returns:

A unix time value

TT2000toUnixTime

```
public static double[] TT2000toUnixTime(long[] epoch)
```

Converts an array of TT2000 values to an array of unix time of a double value (seconds from 1970-01-01 00:00:00 UTC). The unix time will have a time resolution of microseconds at its fraction part. Note: As unix time does not have leap seconds, the converted time might not be properly presented. Also, TT2000 have a higher time resolution than the unix time. The conversion might not precisely preserve the microseconds due to the floating point presentation.

Parameters:

`epoch` - the epoch values

Returns:

Array of unix time values

UnixTimetoTT2000

```
public static long UnixTimetoTT2000(double unixTime)
```

Converts a Unix time of a double value (seconds from 1970-01-01 00:00:00 UTC) to a TT2000 time. The unix time can have a time resolution of milliseconds or microseconds at its fraction part. Note: As unix time does not have leap seconds, the converted time might not be properly presented. Also, TT2000 have a higher time resolution than the unix time. The conversion might not precisely preserve the microseconds due to the floating point presentation.

Parameters:

`unixTime` - the unix time value

Returns:

An TT2000 value

UnixTimetoTT2000

```
public static long[] UnixTimetoTT2000(double[] unixTimes)
```

Converts an array of Unix time of double values (seconds from 1970-01-01 00:00:00 UTC) to an array of TT2000 times. The unix time can have a time resolution of milliseconds or microseconds at its fraction part. Note: As unix time does not have leap seconds, the converted time might not be properly presented. Also, TT2000 may have a higher time resolution than the unix time. The conversion might not precisely preserve the microseconds due to the floating point presentation.

Parameters:

`unixTimes` - the unix time values

Returns:

An array of TT2000 values

Overview Package Class Tree Deprecated Index Help

Prev Class **Next Class** Frames No Frames All Classes

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

gsfc.nssdc.cdf.util

Class CDFUtils

java.lang.Object
gsfc.nssdc.cdf.util.CDFUtils

All Implemented Interfaces:

CDFConstants

```
public class CDFUtils
extends java.lang.Object
implements CDFConstants
```

This class contains the handy utility routines (methods) called by the core CDF Java APIs.

Version:

1.0

Field Summary

Fields inherited from interface gsfc.nssdc.cdf.CDFConstants

AHUFF_COMPRESSION, ALPHAOSF1_DECODING, ALPHAOSF1_ENCODING, ALPHAVMSd_DECODING, ALPHAVMSd_ENCODING, ALPHAVMSg_DECODING, ALPHAVMSg_ENCODING, ALPHAVMSi_DECODING, ALPHAVMSi_ENCODING, ARM_BIG_DECODING, ARM_BIG_ENCODING, ARM_LITTLE_DECODING, ARM_LITTLE_ENCODING, ATTR_, ATTR_EXISTENCE_, ATTR_EXISTS, ATTR_MAXgENTRY_, ATTR_MAXrENTRY_, ATTR_MAXzENTRY_, ATTR_NAME_, ATTR_NAME_TRUNC, ATTR_NUMBER_, ATTR_NUMgENTRIES_, ATTR_NUMrENTRIES_, ATTR_NUMzENTRIES_, ATTR_SCOPE_, BACKWARD_, BACKWARDFILEoff, BACKWARDFILEon, BAD_ALLOCATE_RECS, BAD_ARGUMENT, BAD_ATTR_NAME, BAD_ATTR_NUM, BAD_BLOCKING_FACTOR, BAD_CACHE_SIZE, BAD_CDF_EXTENSION, BAD_CDF_ID, BAD_CDF_NAME, BAD_CDFSTATUS, BAD_CHECKSUM, BAD_COMPRESSION_PARM, BAD_DATA_TYPE, BAD_DECODING, BAD_DIM_COUNT, BAD_DIM_INDEX, BAD_DIM_INTERVAL, BAD_DIM_SIZE, BAD_ENCODING, BAD_ENTRY_NUM, BAD_FNC_OR_ITEM, BAD_FORMAT, BAD_INITIAL_RECS, BAD_MAJORITY, BAD_MALLOC, BAD_NEGtoPOSfp0_MODE, BAD_NUM_DIMS, BAD_NUM_ELEMS, BAD_NUM_STRINGS, BAD_NUM_VARS, BAD_READONLY_MODE, BAD_REC_COUNT, BAD_REC_INTERVAL, BAD_REC_NUM, BAD_SCOPE, BAD_SCRATCH_DIR, BAD_SPARSEARRAYS_PARM, BAD_VAR_NAME, BAD_VAR_NUM, BAD_zMODE, BADDATE_LEAPSECOND_UPDATED, BeginUnixTimeEPOCH, BeginUnixTimeEPOCH16, BLOCKINGFACTOR_TOO_LARGE, BLOCKINGFACTOR_TOO_SMALL, BLOCKINGFACTOR_TOO_SMALL2, CANNOT_ALLOCATE_RECORDS, CANNOT_CHANGE, CANNOT_COMPRESS, CANNOT_CONVERT_WIDECHAR, CANNOT_COPY, CANNOT_INSERT_RECORDS, CANNOT_SPARSEARRAYS, CANNOT_SPARSERECORDS, CDF_, CDF_ACCESS_, CDF_ATTR_NAME_LEN, CDF_ATTR_NAME_LEN256, CDF_BYTE, CDF_CACHESIZE_, CDF_CHAR, CDF_CHECKSUM_, CDF_CLOSE_ERROR, CDF_COMPRESSION_, CDF_COPYRIGHT_, CDF_COPYRIGHT_LEN, CDF_CREATE_ERROR, CDF_DECODING_, CDF_DELETE_ERROR, CDF_DOUBLE, CDF_ENCODING_, CDF_EPOCH, CDF_EPOCH16, CDF_EXISTS, CDF_FLOAT, CDF_FORMAT_, CDF_INCREMENT_, CDF_INFO_, CDF_INTI, CDF_INT2, CDF_INT4, CDF_INT8, CDF_INTERNAL_ERROR, CDF_LEAPSECONDLASTUPDATED_, CDF_MAJORITY_, CDF_MAX_DIMS, CDF_MAX_PARMS, CDF_MIN_DIMS, CDF_NAME_, CDF_NAME_TRUNC, CDF_NEGtoPOSfp0_MODE_, CDF_NUMATTRS_, CDF_NUMgATTRS_, CDF_NUMrVARS_, CDF_NUMvATTRS_, CDF_NUMzVARS_, CDF_OK, CDF_OPEN_ERROR, CDF_PATHNAME_LEN, CDF_READ_ERROR, CDF_READONLY_MODE_, CDF_REAL4, CDF_REAL8, CDF_RELEASE_, CDF_SAVE_ERROR, CDF_SCRATCHDIR_, CDF_STATUS_, CDF_STATUSTEXT_LEN, CDF_TIME_TT2000, CDF_UCHAR, CDF_UINT1, CDF_UINT2, CDF_UINT4, CDF_VAR_NAME_LEN, CDF_VAR_NAME_LEN256, CDF_VERSION_, CDF_WARN, CDF_WRITE_ERROR, CDF_zMODE_, CDFwithSTATS_, CHECKSUM_, CHECKSUM_ERROR, CHECKSUM_NOT_ALLOWED, CLOSE_, COLUMN_MAJOR, COMPRESS_CACHESIZE_, COMPRESSION_ERROR, CONFIRM_, CORRUPTED_V2_CDF, CORRUPTED_V3_CDF, CREATE_, CURgENTRY_EXISTENCE_, CURrENTRY_EXISTENCE_, CURzENTRY_EXISTENCE_, DATATYPE_MISMATCH, DATATYPE_SIZE_, DECOMPRESSION_ERROR, DECSTATION_DECODING, DECSTATION_ENCODING, DEFAULT_BYTE_PADVALUE, DEFAULT_CHAR_PADVALUE, DEFAULT_DOUBLE_PADVALUE, DEFAULT_EPOCH_PADVALUE, DEFAULT_EPOCH16_PADVALUE, DEFAULT_FLOAT_PADVALUE, DEFAULT_INT1_PADVALUE, DEFAULT_INT2_PADVALUE, DEFAULT_INT4_PADVALUE, DEFAULT_INT8_PADVALUE, DEFAULT_REAL4_PADVALUE, DEFAULT_REAL8_PADVALUE, DEFAULT_TT2000_PADVALUE, DEFAULT_UCHAR_PADVALUE, DEFAULT_UINT1_PADVALUE, DEFAULT_UINT2_PADVALUE, DEFAULT_UINT4_PADVALUE, DELETE_, DID_NOT_COMPRESS, EMPTY_COMPRESSED_CDF, END_OF_VAR, EPOCH_STRING_LEN, EPOCH_STRING_LEN_EXTEND, EPOCH1_STRING_LEN, EPOCH1_STRING_LEN_EXTEND, EPOCH2_STRING_LEN, EPOCH2_STRING_LEN_EXTEND, EPOCH3_STRING_LEN, EPOCH3_STRING_LEN_EXTEND, EPOCH4_STRING_LEN, EPOCH4_STRING_LEN_EXTEND, EPOCHx_FORMAT_MAX, EPOCHx_STRING_MAX, FILLED_TT2000_VALUE, FORCED_PARAMETER, FUNCTION_NOT_SUPPORTED, gENTRY_, gENTRY_DATA_, gENTRY_DATASPEC_, gENTRY_DATATYPE_, gENTRY_EXISTENCE_, gENTRY_NUMELEMS_, GET_, GETCDFCHECKSUM_, GETCDFFILEBACKWARD_, GETCDFVALIDATE_, GETLEAPSECONDSENVVAR_, GLOBAL_SCOPE, GZIP_COMPRESSION, HOST_DECODING, HOST_ENCODING, HP_DECODING, HP_ENCODING, HUFF_COMPRESSION, IA64VMSd_DECODING, IA64VMSd_ENCODING, IA64VMSg_DECODING, IA64VMSg_ENCODING, IA64VMSi_DECODING, IA64VMSi_ENCODING, IBM_PC_OVERFLOW, IBMPC_DECODING, IBMPC_ENCODING, IBMRS_DECODING, IBMRS_ENCODING, ILLEGAL_EPOCH_FIELD, ILLEGAL_EPOCH_VALUE, ILLEGAL_FOR_SCOPE, ILLEGAL_IN_zMODE, ILLEGAL_ON_V1_CDF, ILLEGAL_TT2000_VALUE, IS_A_NETCDF, LIB_COPYRIGHT_, LIB_INCREMENT_,

LIB_RELEASE_, LIB_subINCREMENT_, LIB_VERSION_, MAC_DECODING, MAC_ENCODING, MD5_CHECKSUM, MULTI_FILE, MULTI_FILE_FORMAT, NA_FOR_VARIABLE, NEGATIVE_FP_ZERO, NEGtoPOSfp0off, NEGtoPOSfp0on, NETWORK_DECODING, NETWORK_ENCODING, NeXT_DECODING, NeXT_ENCODING, NO_ATTR_SELECTED, NO_CDF_SELECTED, NO_CHECKSUM, NO_COMPRESSION, NO_DELETE_ACCESS, NO_ENTRY_SELECTED, NO_MORE_ACCESS, NO_PADVALUE_SPECIFIED, NO_SPARSEARRAYS, NO_SPARSERECORDS, NO_STATUS_SELECTED, NO_SUCH_ATTR, NO_SUCH_CDF, NO_SUCH_ENTRY, NO_SUCH_RECORD, NO_SUCH_VAR, NO_VAR_SELECTED, NO_VARS_IN_CDF, NO_WRITE_ACCESS, NONE_CHECKSUM, NOT_A_CDF, NOT_A_CDF_OR_NOT_SUPPORTED, NOVARY, NULL_, OPEN_, OPTIMAL_ENCODING_TREES, OTHER_CHECKSUM, PAD_SPARSERECORDS, PPC_DECODING, PPC_ENCODING, PRECEEDING_RECORDS_ALLOCATED, PREV_SPARSERECORDS, PUT_, READ_ONLY_DISTRIBUTION, READ_ONLY_MODE, READONLYoff, READONLYon, rENTRY_, rENTRY_DATA_, rENTRY_DATASPEC_, rENTRY_DATATYPE_, rENTRY_EXISTENCE_, rENTRY_NAME_, rENTRY_NUMELEMS_, rENTRY_NUMSTRINGS_, rENTRY_STRINGSDATA_, RLE_COMPRESSION, RLE_OF_ZEROs, ROW_MAJOR, rVAR_, rVAR_ALLOCATEBLOCK_, rVAR_ALLOCATEDFROM_, rVAR_ALLOCATEDTO_, rVAR_ALLOCATERECS_, rVAR_BLOCKINGFACTOR_, rVAR_CACHESIZE_, rVAR_COMPRESSION_, rVAR_DATA_, rVAR_DATASPEC_, rVAR_DATATYPE_, rVAR_DIMVARYS_, rVAR_EXISTENCE_, rVAR_HYPERDATA_, rVAR_INITIALRECS_, rVAR_MAXallocREC_, rVAR_MAXREC_, rVAR_NAME_, rVAR_nINDEXENTRIES_, rVAR_nINDEXLEVELS_, rVAR_nINDEXRECORDS_, rVAR_NUMallocRECS_, rVAR_NUMBER_, rVAR_NUMELEMS_, rVAR_NUMRECS_, rVAR_PADVALUE_, rVAR_RECORDS_, rVAR_RECORDS_RENUMBER_, rVAR_RECVARY_, rVAR_RESERVEPERCENT_, rVAR_SEQDATA_, rVAR_SEQPOS_, rVAR_SPARSEARRAYS_, rVAR_SPARSERECORDS_, rVARs_CACHESIZE_, rVARs_DIMCOUNTS_, rVARs_DIMINDICES_, rVARs_DIMINTERVALS_, rVARs_DIMSIZES_, rVARs_MAXREC_, rVARs_NUMDIMS_, rVARs_RECCOUNT_, rVARs_RECDATA_, rVARs_RECINTERVAL_, rVARs_RECNUMBER_, SAVE_, SCRATCH_CREATE_ERROR, SCRATCH_DELETE_ERROR, SCRATCH_READ_ERROR, SCRATCH_WRITE_ERROR, SELECT_, SGI_DECODING, SGi_ENCODING, SINGLE_FILE, SINGLE_FILE_FORMAT, SOME_ALREADY_ALLOCATED, STAGE_CACHESIZE_, STATUS_TEXT_, STRING_NOT_UTF8_ENCODING, STRINGDELIMITER, SUN_DECODING, SUN_ENCODING, TOO_MANY_PARMS, TOO_MANY_VARS, TRY_TO_READ_NONSTRING_DATA, TT2000_0_STRING_LEN, TT2000_1_STRING_LEN, TT2000_2_STRING_LEN, TT2000_3_STRING_LEN, TT2000_4_STRING_LEN, TT2000_CDF_MAYNEEDUPDATE, TT2000_TIME_ERROR, TT2000_USED_OUTDATED_TABLE, UNABLE_TO_PROCESS_CDF, UNKNOWN_COMPRESSION, UNKNOWN_SPARSENESS, UNSUPPORTED_OPERATION, VALIDATE_, VALIDATEFILEoff, VALIDATEFILEon, VAR_ALREADY_CLOSED, VAR_CLOSE_ERROR, VAR_CREATE_ERROR, VAR_DELETE_ERROR, VAR_EXISTS, VAR_NAME_TRUNC, VAR_OPEN_ERROR, VAR_READ_ERROR, VAR_SAVE_ERROR, VAR_WRITE_ERROR, VARIABLE_SCOPE, VARY, VAX_DECODING, VAX_ENCODING, VIRTUAL_RECORD_DATA, zENTRY_, zENTRY_DATA_, zENTRY_DATASPEC_, zENTRY_DATATYPE_, zENTRY_EXISTENCE_, zENTRY_NAME_, zENTRY_NUMELEMS_, zENTRY_NUMSTRINGS_, zENTRY_STRINGSDATA_, ZLIB_COMPRESS_ERROR, ZLIB_UNCOMPRESS_ERROR, zMODEoff, zMODEon1, zMODEon2, zVAR_, zVAR_ALLOCATEBLOCK_, zVAR_ALLOCATEDFROM_, zVAR_ALLOCATEDTO_, zVAR_ALLOCATERECS_, zVAR_BLOCKINGFACTOR_, zVAR_CACHESIZE_, zVAR_COMPRESSION_, zVAR_DATA_, zVAR_DATASPEC_, zVAR_DATATYPE_, zVAR_DIMCOUNTS_, zVAR_DIMINDICES_, zVAR_DIMINTERVALS_, zVAR_DIMSIZES_, zVAR_DIMVARYS_, zVAR_EXISTENCE_, zVAR_HYPERDATA_, zVAR_INITIALRECS_, zVAR_MAXallocREC_, zVAR_MAXREC_, zVAR_NAME_, zVAR_nINDEXENTRIES_, zVAR_nINDEXLEVELS_, zVAR_nINDEXRECORDS_, zVAR_NUMallocRECS_, zVAR_NUMBER_, zVAR_NUMDIMS_, zVAR_NUMELEMS_, zVAR_NUMRECS_, zVAR_PADVALUE_, zVAR_RECCOUNT_, zVAR_RECINTERVAL_, zVAR_RECNUMBER_, zVAR_RECORDS_, zVAR_RECORDS_RENUMBER_, zVAR_RECVARY_, zVAR_RESERVEPERCENT_, zVAR_SEQDATA_, zVAR_SEQPOS_, zVAR_SPARSEARRAYS_, zVAR_SPARSERECORDS_, zVARs_CACHESIZE_, zVARs_MAXREC_, zVARs_RECDATA_, zVARs_RECNUMBER_

Constructor Summary

Constructors

Constructor and Description
CDFUtils()

Method Summary

Methods

Modifier and Type	Method and Description
static java.lang.String[]	**breakIntoStrings**(java.lang.String data) Break down a merged string into their original array of strings.
static boolean	**cdfFileExists**(java.lang.String fileName) Checks the existence of the given CDF file name.
static java.lang.String	**CFtoJformat**(java.lang.String cfFormat) Convert a C/Fortran-based print format to a corresponding Java style format for numerical data.
static byte[]	**Cp1252toUTF8**(java.lang.Object cp1252)
static java.lang.String	**CtoJformat**(java.lang.String cFormat) Convert a C-based print format to a corresponding Java style format for numerical data.
static java.lang.String	**FtoJformat**(java.lang.String fFormat) Convert a Fortran-based print format to a corresponding Java style format for numerical data.

static long	**getDataTypeValue**(java.lang.String cdfDataType) Gets the long value of the given CDF data type in string.
static long[]	**getDimensionSizes**(java.lang.Object obj)
static long	**getLeapSecondLastUpdated**(**CDF** cdf) Gets the date of the given CDF's last updated for the leap second.
static long	**getLongChecksum**(java.lang.String checksum) Gets the long value of the given CDF's checksum in string.
static long	**getLongCompressionType**(java.lang.String compressionType) Gets the long representation of the given CDF compression type in string.
static long	**getLongEncoding**(java.lang.String encodingType) Gets the long value of the given CDF encoding type in string.
static long	**getLongFormat**(java.lang.String formatType) Gets the long value of the given CDF file format in string.
static long	**getLongMajority**(java.lang.String majorityType) Gets the long value of the given CDF majority.
static long	**getLongSparseRecord**(java.lang.String sparseRecordType) Gets the long value of the given sparse record type in string.
static int	**getNumDimensions**(java.lang.Object obj)
static long	**getNumElements**(long dataType, java.lang.Object data) Gets the number of elements contained in the given data object.
static long	**getNumStrings**(java.lang.Object data) Gets the number of strings contained in the given data object.
static java.lang.String	**getSignature**(java.lang.Object obj) Gets the java signature of the given object.
static java.lang.String	**getStringChecksum**(**CDF** cdf) Gets the string value of the given CDF's checksum.
static java.lang.String	**getStringChecksum**(long checksumType) Gets the string value of the given CDF's checksum.
static java.lang.String	**getStringCompressionType**(**CDF** cdf) Gets the string representation of the given CDF file's compression type.
static java.lang.String	**getStringCompressionType**(long compressionType) Gets the string representation of the given CDF compression type.
static java.lang.String	**getStringCompressionType**(**Variable** var) Gets the string representation of the given variable's compression type.
static java.lang.String	**getStringData**(java.lang.Object data) Returns the string value of the given data.
static java.lang.String	**getStringData**(java.lang.Object data, int epochType) Returns the string value of the given data.
static java.lang.String	**getStringData**(java.lang.Object data, java.lang.String separator) Returns the string of the value of the given data.
static java.lang.String	**getStringData**(java.lang.Object data, java.lang.String separator, int epochType) Returns the string of the value of the given data.
static java.lang.String	**getStringData**(java.lang.Object data, java.lang.String separator, int epochType, java.lang.String addZ) Returns the string of the value of the given data.
static java.lang.String	**getStringDataType**(**Entry** entry) Gets the string value of the CDF data type for the given entry.
static java.lang.String	**getStringDataType**(long cdfDataType) Gets the string representation of the given CDF data type.
static java.lang.String	**getStringDataType**(**Variable** var) Gets the string value of the CDF data type for the given variable.

static java.lang.String	**getStringDecoding**(**CDF** cdf) Gets the string value of the given CDF file's decoding type.
static java.lang.String	**getStringDecoding**(long decodingType) Gets the string value of the given CDF decoding type .
static java.lang.String	**getStringEncoding**(**CDF** cdf) Get the string value of the given CDF's encoding type.
static java.lang.String	**getStringEncoding**(long encodingType) Gets the string value of the given CDF encoding type.
static java.lang.String	**getStringFloatingData**(java.lang.Object data, java.lang.String format, java.lang.String separator) Returns the string of the value of the given data.
static java.lang.String	**getStringFormat**(**CDF** cdf) Gets the string value of the given CDF's file format.
static java.lang.String	**getStringFormat**(long formatType) Gets the string value of the given CDF's file format.
static java.lang.String	**getStringMajority**(**CDF** cdf) Gets the string value of the given CDF file's majority.
static java.lang.String	**getStringMajority**(long majorityType) Gets the string value of the given CDF majority.
static java.lang.String	**getStringSparseRecord**(long sparseRecordType) Gets the string value of the given sparse record type.
static java.lang.String	**getStringSparseRecord**(**Variable** var) Gets the string value of the given variable's sparse record type.
static boolean	**isEpochDataType**(long dataType) Returns whether a CDF data type is an epoch related type.
static int	**isFloatingPadValue**(double value) Check if a double data is a pad value.
static int	**isFloatingPadValue**(float value) Check if a float data is a pad value.
static int	**isFloatingPadValue**(long dataType, double value) Check if a data is a floating point and its value is a pad value.
static int	**isFloatingPadValue**(long dataType, float value) Check if a data is a floating point and its value is a pad value.
static int	**isFloatingPadValue**(long dataType, java.lang.Object value) Check if a data is a floating point and its value is a pad value.
static int	**isFloatingPadValue**(java.lang.Object value) Check if a data is a floating point and its value is a pad value.
static boolean	**isStringDataType**(long dataType) Returns whether a CDF data type is a string type.
static java.lang.String	**mergeFromStrings**(java.lang.Object data) Merge an array of strings into a single string with "\ N ", a 3-char separator.
static void	**printData**(java.lang.Object data) Prints the value of the given data on the screen.
static void	**printData**(java.lang.Object data, int which) Prints the value of the given data on the screen.
static void	**printData**(java.lang.Object data, int which, boolean iso8601) Prints the value of the given data on the screen.
static void	**printData**(java.lang.Object data, int which, boolean iso8601, java.lang.String addZ) Prints the value of the given data on the screen.
static void	**printData**(java.lang.Object data, long dataType) Prints the value of the given data on the screen.

static void	**printData**(java.lang.Object data, java.io.PrintWriter outWriter) Prints the value of the given data to the place designated by PrintWriter that can be a file, Sysem.out, System.err, and etc.
static void	**printData**(java.lang.Object data, java.io.PrintWriter outWriter, int which) Prints the value of the given data to the place designated by PrintWriter that can be a file, Sysem.out, System.err, and etc.
static void	**printData**(java.lang.Object data, java.io.PrintWriter outWriter, int which, boolean iso8601) Prints the value of the given data to the place designated by PrintWriter that can be a file, Sysem.out, System.err, and etc.
static void	**printData**(java.lang.Object data, java.io.PrintWriter outWriter, int which, boolean iso8601, java.lang.String addZ) Prints the value of the given data to the place designated by PrintWriter that can be a file, Sysem.out, System.err, and etc.
static void	**printDataWithFormat**(java.lang.Object data, java.lang.String format) Prints the value of the given data on the screen.
static void	**printDataWithFormat**(java.lang.Object data, java.lang.String format, int which) Prints the value of the given data on the screen.
static void	**printDataWithFormat**(java.lang.Object data, java.lang.String format, int which, boolean iso8601) Prints the value of the given data on the screen.
static void	**printDataWithFormat**(java.lang.Object data, java.lang.String format, int which, boolean iso8601, java.lang.String addZ) Prints the value of the given data on the screen.
static void	**printDataWithFormat**(java.lang.Object data, java.lang.String format, java.io.PrintWriter outWriter) Prints the value of the given data to the place designated by PrintWriter that can be a file, Sysem.out, System.err, and etc.
static void	**printDataWithFormat**(java.lang.Object data, java.lang.String format, java.io.PrintWriter outWriter, int which) Prints the value of the given data to the place designated by PrintWriter that can be a file, Sysem.out, System.err, and etc.
static void	**printDataWithFormat**(java.lang.Object data, java.lang.String format, java.io.PrintWriter outWriter, int which, boolean iso8601) Prints the value of the given data to the place designated by PrintWriter that can be a file, Sysem.out, System.err, and etc.
static void	**printDataWithFormat**(java.lang.Object data, java.lang.String format, java.io.PrintWriter outWriter, int which, boolean iso8601, java.lang.String addZ) Prints the value of the given data to the place designated by PrintWriter that can be a file, Sysem.out, System.err, and etc.
static byte[]	**UTF16LEtoUTF8**(java.lang.Object utf16)
static byte[]	**UTF8toCp1252**(java.lang.Object utf8)
static byte[]	**UTF8toUTF16LE**(java.lang.Object utf8)

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

CDFUtils

```
public CDFUtils()
```

Method Detail

getSignature

```
public static java.lang.String getSignature(java.lang.Object obj)
```

Gets the java signature of the given object.

NOTE: Java primitive data types (e.g. int, long, byte, etc.) are not Objects. Thus they must be passed-in as an Object by using a wrapper (e.g. Integer(23)).

```
  Signature                 Java Programming Language Type
  ---------                 ------------------------------
  [Z                        array of boolean
  [B                        array of byte
  [C                        array of char
  [S                        array of short
  [I                        array of int
  [J                        array of long
  [F                        array of float
  [D                        array of double

  L fully-qualified-class   fully-qualified class
  L fully-qualified-class;  array of fully-qualified class
  java.lang.Boolean         Boolean
  Ljava.lang.Boolean;       array of Boolean
  java.lang.Byte            Byte
  Ljava.lang.Byte;          array of Byte
  java.lang.Short           Short
  Ljava.lang.Short;         array of Short
  java.lang.Integer         Integer
  Ljava.lang.Integer;       array of Integer
  java.lang.Long            Long
  Ljava.lang.Long;          array of Long
  java.lang.Float           Float
  Ljava.lang.Float;         array of Float
  java.lang.Double          Double
  Ljava.lang.Double;        array of Double
  java.lang.String          String
  Ljava.lang.String;        array of String
```

Parameters:

`obj` - the object from which Java signature is retrieved

Returns:

Java signature of the given object

getNumElements

```
public static long getNumElements(long dataType,
                  java.lang.Object data)
                           throws CDFException
```

Gets the number of elements contained in the given data object.

Parameters:

`dataType` - the CDF data type of the object to be examined

`data` - the data object to be examined

Returns:

If the data is a string: number of characters in the string
If the data is an array: number of elements in the array
Otherwise: 1

Throws:

`CDFException` - if a problem occurs getting the number of elements

getNumStrings

```
public static long getNumStrings(java.lang.Object data)
```

Gets the number of strings contained in the given data object.

Parameters:

`data` - the data object to be examined

Returns:

The number of strings if the object is of string type and the data has the predefined STRINGDELIMITER (as "\\N ") delimiter in it. Or, 0 if non-string datat type.

printData

```
public static void printData(java.lang.Object data)
```

Prints the value of the given data on the screen. Data can be a java primitive data type, Java Object (non-array), or 1-dimensional array of primitive Java data type.

Parameters:

`data` - the data to be printed

printData

```
public static void printData(java.lang.Object data,
             int which,
             boolean iso8601)
```

Prints the value of the given data on the screen. Data can be a java primitive data type, Java Object (non-array), or 1-dimensional array of primitive Java data type.

Parameters:

`data` - the data to be printed

`which` - the Epoch data type data indicator 1 if CDF_EPOCH, 2 if CDF_EPOCH16 or 3 if CDF_TIME_TT2000 0 Otherwise

`iso8601` - the ISO 8601 indicator for EPOCH data

printData

```
public static void printData(java.lang.Object data,
             int which,
             boolean iso8601,
```

```
            java.lang.String addZ)
```

Prints the value of the given data on the screen. Data can be a java primitive data type, Java Object (non-array), or one or multi-dimensional array of primitive Java data type.

Parameters:

`data` - the data to be printed

`which` - the Epoch data type data indicator 1 if CDF_EPOCH, 2 if CDF_EPOCH16 or 3 if CDF_TIME_TT2000 0 Otherwise

`iso8601` - the ISO 8601 indicator for EPOCH data

`addZ` - Added the passed string to the end, (likely a "Z", to all CDF epoch data types, not just CDF_TIME_TT2000)

printData

```
public static void printData(java.lang.Object data,
            int which)
```

Prints the value of the given data on the screen. Data can be a java primitive data type, Java Object (non-array), or 1-dimensional array of primitive Java data type.

Parameters:

`data` - the data to be printed

`which` - the Epoch data type data indicator 1 if CDF_EPOCH, 2 if CDF_EPOCH16 or 3 if CDF_TIME_TT2000 0 Otherwise

printData

```
public static void printData(java.lang.Object data,
            long dataType)
```

Prints the value of the given data on the screen. Data can be a java primitive data type, Java Object (non-array), or 1-dimensional array of primitive Java data type.

Parameters:

`data` - the data to be printed

`dataType` - the CDF data type of the data

printData

```
public static void printData(java.lang.Object data,
            java.io.PrintWriter outWriter)
```

Prints the value of the given data to the place designated by PrintWriter that can be a file, Sysem.out, System.err, and etc. Data can be a java primitive data type, Java Object (non-array), or 1-dimensional array of primitive Java data type.

The following example will send the contents of the given data to "myoutput.dat".

```
OutputStreamWriter outWriter = null;
PrintWriter out = null;
try {
     outWriter = new OutputStreamWriter("myoutput.dat", "UTF-8");
     out = new PrintWriter(outWriter, true);
} catch (Exception e) {
    System.out.println ("Exception occurred: "+e);
}
CDFUtils.printData (data, out);
```

Parameters:

`data` - the data to be printed

`outWriter` - the print writer to which formatted representations of the object/data is printed as a text-output stream

printData

```
public static void printData(java.lang.Object data,
             java.io.PrintWriter outWriter,
             int which)
```

Prints the value of the given data to the place designated by PrintWriter that can be a file, Sysem.out, System.err, and etc. Data can be a java primitive data type, Java Object (non-array), or 1-dimensional array of primitive Java data type.

Parameters:

`data` - the data to be printed

`outWriter` - the print writer to which formatted representations of the object/data is printed as a text-output stream

`which` - the Epoch data type data indicator 1 if CDF_EPOCH, 2 if CDF_EPOCH16 or 3 if CDF_TIME_TT2000 0 Otherwise

printData

```
public static void printData(java.lang.Object data,
             java.io.PrintWriter outWriter,
             int which,
             boolean iso8601)
```

Prints the value of the given data to the place designated by PrintWriter that can be a file, Sysem.out, System.err, and etc. Data can be a java primitive data type, Java Object (non-array), or 1-dimensional array of primitive Java data type.

Parameters:

`data` - the data to be printed

`outWriter` - the print writer to which formatted representations of the object/data is printed as a text-output stream

`which` - the Epoch data type data indicator 1 if CDF_EPOCH, 2 if CDF_EPOCH16 or 3 if CDF_TIME_TT2000 0 Otherwise

`iso8601` - the ISO 8601 indicator for EPOCH data

printData

```
public static void printData(java.lang.Object data,
             java.io.PrintWriter outWriter,
             int which,
             boolean iso8601,
             java.lang.String addZ)
```

Prints the value of the given data to the place designated by PrintWriter that can be a file, Sysem.out, System.err, and etc. Data can be a java primitive data type, Java Object (non-array), or 1-dimensional array of primitive Java data type.

Parameters:

`data` - the data to be printed

`outWriter` - the print writer to which formatted representations of the object/data is printed as a text-output stream

`which` - the Epoch data type data indicator 1 if CDF_EPOCH, 2 if CDF_EPOCH16 or 3 if CDF_TIME_TT2000

`iso8601` - the ISO 8601 indicator for EPOCH data

`addZ` - Added the passed string to the end of an ISO8601 format string, (likely a "Z", to all CDF epoch data types, not just CDF_TIME_TT2000).

printDataWithFormat

```
public static void printDataWithFormat(java.lang.Object data,
                        java.lang.String format)
```

Prints the value of the given data on the screen. Data can be a java primitive data type, Java Object (non-array), or 1-dimensional array of primitive Java data type. The format is mainly applicable to floating point typed data.

Parameters:

`data` - the data to be printed

`format` - the format specifically for the floating point data

printDataWithFormat

```
public static void printDataWithFormat(java.lang.Object data,
                        java.lang.String format,
                        int which,
                        boolean iso8601)
```

Prints the value of the given data on the screen. Data can be a java primitive data type, Java Object (non-array), or 1-dimensional array of primitive Java data type. Note: The format is mainly used for floating point data.

Parameters:

`data` - the data to be printed

`format` - the format specifically for the floating point data

`which` - the Epoch data type data indicator 1 if CDF_EPOCH, 2 if CDF_EPOCH16 or 3 if CDF_TIME_TT2000 0 otherwise

`iso8601` - the ISO 8601 indicator for EPOCH data

printDataWithFormat

```
public static void printDataWithFormat(java.lang.Object data,
                        java.lang.String format,
                        int which,
                        boolean iso8601,
                        java.lang.String addZ)
```

Prints the value of the given data on the screen. Data can be a java primitive data type, Java Object (non-array), or 1-dimensional array of primitive Java data type. Note: The format is mainly used for floating point data.

Parameters:

`data` - the data to be printed

`format` - the format specifically for the floating point data

`which` - the Epoch data type data indicator 1 if CDF_EPOCH, 2 if CDF_EPOCH16 or 3 if CDF_TIME_TT2000

`iso8601` - the ISO 8601 indicator for EPOCH data

`addZ` - Added the passed string to the end, (likely a "Z", to all CDF epoch data types, not just CDF_TIME_TT2000)

printDataWithFormat

```
public static void printDataWithFormat(java.lang.Object data,
                        java.lang.String format,
                        int which)
```

Prints the value of the given data on the screen. Data can be a java primitive data type, Java Object (non-array), or 1-dimensional array of primitive Java data type.

Parameters:

`data` - the data to be printed

`format` - the format specifically for the floating point data

`which` - the Epoch data type data indicator 1 if CDF_EPOCH, 2 if CDF_EPOCH16 or 3 if CDF_TIME_TT2000 0 otherwise

printDataWithFormat

```
public static void printDataWithFormat(java.lang.Object data,
                        java.lang.String format,
                        java.io.PrintWriter outWriter)
```

Prints the value of the given data to the place designated by PrintWriter that can be a file, Sysem.out, System.err, and etc. Data can be a java primitive data type, Java Object (non-array), or 1-dimensional array of primitive Java data type.

The following example will send the contents of the given data to "myoutput.dat".

```
     OutputStreamWriter outWriter = null;
     PrintWriter out = null;
     try {
          outWriter = new OutputStreamWriter("myoutput.dat", "UTF-8");
          out = new PrintWriter(outWriter, true);
     } catch (Exception e) {
         System.out.println ("Exception occurred: "+e);
     }
     CDFUtils.printData (data, out);
```

Parameters:

`data` - the data to be printed

`format` - the format specifically for the floating point data

`outWriter` - the print writer to which formatted representations of the object/data is printed as a text-output stream

printDataWithFormat

```
public static void printDataWithFormat(java.lang.Object data,
                        java.lang.String format,
                        java.io.PrintWriter outWriter,
                        int which)
```

Prints the value of the given data to the place designated by PrintWriter that can be a file, Sysem.out, System.err, and etc. Data can be a java primitive data type, Java Object (non-array), or 1-dimensional array of primitive Java data type.

Parameters:

`data` - the data to be printed

`format` - the format specifically for the floating point data

`outWriter` - the print writer to which formatted representations of the object/data is printed as a text-output stream

`which` - the Epoch data type data indicator 1 if CDF_EPOCH, 2 if CDF_EPOCH16 or 3 if CDF_TIME_TT2000

printDataWithFormat

```
public static void printDataWithFormat(java.lang.Object data,
                        java.lang.String format,
                        java.io.PrintWriter outWriter,
                        int which,
                        boolean iso8601)
```

Prints the value of the given data to the place designated by PrintWriter that can be a file, Sysem.out, System.err, and etc. Data can be a java primitive data type, Java Object (non-array), or 1-dimensional array of primitive Java data type.

Parameters:

`data` - the data to be printed

`format` - the format specifically for the floating point data

`outWriter` - the print writer to which formatted representations of the object/data is printed as a text-output stream

`which` - the Epoch data type data indicator 1 if CDF_EPOCH, 2 if CDF_EPOCH16 or 3 if CDF_TIME_TT2000

`iso8601` - the ISO 8601 indicator for EPOCH data

printDataWithFormat

```
public static void printDataWithFormat(java.lang.Object data,
                        java.lang.String format,
                        java.io.PrintWriter outWriter,
                        int which,
                        boolean iso8601,
                        java.lang.String addZ)
```

Prints the value of the given data to the place designated by PrintWriter that can be a file, Sysem.out, System.err, and etc. Data can be a java primitive data type, Java Object (non-array), or 1-dimensional array of primitive Java data type.

Parameters:

`data` - the data to be printed

`format` - the format specifically for the floating point data

`outWriter` - the print writer to which formatted representations of the object/data is printed as a text-output stream

`which` - the Epoch data type data indicator 1 if CDF_EPOCH, 2 if CDF_EPOCH16 or 3 if CDF_TIME_TT2000

`iso8601` - the ISO 8601 indicator for EPOCH data

`addZ` - Added the passed string to the end of an ISO8601 format string, (likely a "Z", to all CDF epoch data types, not just CDF_TIME_TT2000).

getStringData

```
public static java.lang.String getStringData(java.lang.Object data)
```

Returns the string value of the given data. Data can be a java primitive data type, Java Object (non-array), or 1-dimensional array of primitive Java data type.

Parameters:

`data` - the data to be parsed

Returns:

The string value of the given data/object.
If the data is an array, its elements are delimited by a space.

mergeFromStrings

```
public static java.lang.String mergeFromStrings(java.lang.Object data)
```

Merge an array of strings into a single string with "\ N ", a 3-char separator.

Parameters:

`data` - the strings to be merged.

Returns:

The merged string.
If the input is any single string, then it returns as is.

breakIntoStrings

```
public static java.lang.String[] breakIntoStrings(java.lang.String data)
```

Break down a merged string into their original array of strings. This is a reversed method of MergeStrings. The mergeed string uses "\ N ", a 3-char, as the separator.

Parameters:

`data` - the merged string to be broken down.

Returns:

An array of strings.
If the input string has no separator, it is a single string. The returned array will have only one element.

getStringData

```
public static java.lang.String getStringData(java.lang.Object data,
                              int epochType)
```

Returns the string value of the given data. Data can be a java primitive data type, Java Object (non-array), or 1-dimensional array of primitive Java data type.

Parameters:

`data` - the data to be encoded.

`epochType` - epoch type indicator (==1 CDF_EPOCH, ==2 CDF_EPOCH16, ==0 others)

Returns:

The string value of the given data/object.
If the data is an array, its elements are delimited by a space.

getStringData

```
public static java.lang.String getStringData(java.lang.Object data,
                              java.lang.String separator)
```

Returns the string of the value of the given data. Data can be a java primitive data type, Java Object (non-array), or 1-dimensional array of primitive Java data type.

Parameters:

`data` - the data to be encoded.

`separator` - the delimiter for array elements.

Returns:

The string value of the given data/object.
If the data is an array, its elements are delimited by the user defined separator.

getStringData

```
public static java.lang.String getStringData(java.lang.Object data,
                              java.lang.String separator,
                              int epochType)
```

Returns the string of the value of the given data. Data can be a java primitive data type, Java Object (non-array), or 1-dimensional array of primitive Java data type.

Parameters:

`data` - the data to be encoded.

`separator` - the delimiter for array elements.

`epochType` - Epoch or Epoch16 data type indicator
== 1 for EPOCH, == 2 for EPOCH16, == 3 for TT2000, == 0 for other data types

Returns:

The string value of the given data/object.
If the data is an array, its elements are delimited by the user defined separator.

getStringData

```
public static java.lang.String getStringData(java.lang.Object data,
                              java.lang.String separator,
                              int epochType,
                              java.lang.String addZ)
```

Returns the string of the value of the given data. Data can be a java primitive data type, Java Object (non-array), or 1-dimensional array of primitive Java data type.

Parameters:

`data` - the data to be encoded.

`separator` - the delimiter for array elements.

`epochType` - Epoch or Epoch16 data type indicator
== 1 for EPOCH, == 2 for EPOCH16, == 3 for TT2000, == 0 for other data types

`addZ` - Added the passed string to the end, likely a "Z", to CDF_TIME_TT2000 data type

Returns:

The string value of the given data/object.
If the data is an array, its elements are delimited by the user defined separator.

getStringFloatingData

```
public static java.lang.String getStringFloatingData(java.lang.Object data,
                                      java.lang.String format,
```

java.lang.String separator)

Returns the string of the value of the given data. Data can be a java primitive data type, Java Object (non-array), or 1-dimensional array of primitive or Java data type. The data shall be one of CDF numerical data types. The value will be formatted in format form if it is presented.

Parameters:

`data` - the data to be encoded.

`format` - the data format to be used.

`separator` - the delimiter for array elements.

Returns:

The string value of the given data/object.
If the data is an array, its elements are delimited by the user defined separator.

getStringDataType

`public static java.lang.String getStringDataType(Variable var)`

Gets the string value of the CDF data type for the given variable.

Parameters:

`var` - the CDF variable to be examined

Returns:

See getStringDataType (long cdfDataType) for possible return values.

getStringDataType

`public static java.lang.String getStringDataType(Entry entry)`

Gets the string value of the CDF data type for the given entry.

Parameters:

`entry` - the entry to be examined

Returns:

String representation of the entry's CDF data type. See getStringDataType (long cdfDataType) for possible return values.

isStringDataType

`public static boolean isStringDataType(long dataType)`

Returns whether a CDF data type is a string type.

Parameters:

`dataType` - the data type to be examined

Returns:

true if it is (CDF_CHAR or CDF_UCHAR) false otherwise

isEpochDataType

```
public static boolean isEpochDataType(long dataType)
```

Returns whether a CDF data type is an epoch related type.

Parameters:

`dataType` - the data type to be examined

Returns:

true if it is (CDF_EPOCH, CDF_EPOCH16 or CDF_TIME_TT2000) false otherwise

getStringDataType

```
public static java.lang.String getStringDataType(long cdfDataType)
```

Gets the string representation of the given CDF data type.

Parameters:

`cdfDataType` - the CDF data type to be examined and translated

It should be one of the following:

- CDF_BYTE
- CDF_CHAR
- CDF_UCHAR
- CDF_INT1
- CDF_UINT1
- CDF_INT2
- CDF_UINT2
- CDF_INT4
- CDF_UINT4
- CDF_INT8
- CDF_REAL4
- CDF_FLOAT
- CDF_REAL8
- CDF_DOUBLE
- CDF_EPOCH
- CDF_EPOCH16
- CDF_TIME_TT2000

Returns:

String representation of cdfDataType. The returned value is one of the valid values describe above for cdfDataType. "UNKNOWN" is returned if invalid cdfDataType is given.

getDataTypeValue

```
public static long getDataTypeValue(java.lang.String cdfDataType)
```

Gets the long value of the given CDF data type in string. This is a reverse function from getStringDataType.

Parameters:

`cdfDataType` - the string CDF data type to be examined and translated. It should be one of the following values:

- CDF_BYTE
- CDF_CHAR
- CDF_UCHAR

- CDF_INT1
- CDF_UINT1
- CDF_INT2
- CDF_UINT2
- CDF_INT4
- CDF_UINT4
- CDF_INT8
- CDF_REAL4
- CDF_FLOAT
- CDF_REAL8
- CDF_DOUBLE
- CDF_EPOCH
- CDF_EPOCH16
- CDF_TIME_TT2000

Returns:

long representation of cdfDataType. The returned value is one of the valid values described above for cdfDataType. -1 is returned if invalid cdfDataType is given.

getStringCompressionType

```
public static java.lang.String getStringCompressionType(long compressionType)
```

Gets the string representation of the given CDF compression type.

Parameters:

`compressionType` - the CDF compression type to be translated. it should be one of the following:

- NO_COMPRESSION
- RLE_COMPRESSION
- HUFF_COMPRESSION
- AHUFF_COMPRESSION
- GZIP_COMPRESSION

Returns:

String representation of compressionType. The returned value is one of the following:

- NONE
- RLE
- Huffman
- Adaptive Huffman
- GZIP
- UNKNOWN (for unknown compressionType)

getLongCompressionType

```
public static long getLongCompressionType(java.lang.String compressionType)
```

Gets the long representation of the given CDF compression type in string.

Parameters:

`compressionType` - the CDF compression type to be translated. It should be one of the following:

- NONE
- RLE
- Huffman
- Adaptive Huffman
- GZIP

Returns:

long representation of compressionType. The returned value is one of the following:

- NO_COMPRESSION
- RLE_COMPRESSION
- HUFF_COMPRESSION
- AHUFF_COMPRESSION
- GZIP_COMPRESSION
- -1 (for unknown compressionType)

getStringCompressionType

```
public static java.lang.String getStringCompressionType(Variable var)
```

Gets the string representation of the given variable's compression type.

Parameters:

`var` - the variable to be examined

Returns:

string representation of the given variable's compression type. See getStringCompressionType(long compressionType) for possible return values.

getStringCompressionType

```
public static java.lang.String getStringCompressionType(CDF cdf)
```

Gets the string representation of the given CDF file's compression type.

Parameters:

`cdf` - the CDF to be examined

Returns:

string representation of the given CDF file's compression type. See getStringCompressionType(long compressionType) for possible return values.

getStringEncoding

```
public static java.lang.String getStringEncoding(long encodingType)
```

Gets the string value of the given CDF encoding type.

Parameters:

`encodingType` - the CDF encoding type to be examined. It should be one of the following:

- NETWORK_ENCODING
- SUN_ENCODING
- DECSTATION_ENCODING
- SGi_ENCODING
- IBMPC_ENCODING
- IBMRS_ENCODING
- HOST_ENCODING
- PPC_ENCODING
- HP_ENCODING
- NeXT_ENCODING

ALPHAOSF1_ENCODING
- ALPHAVMSd_ENCODING
- ALPHAVMSg_ENCODING
- ALPHAVMSi_ENCODING
- ARM_LITTLE_ENCODING
- ARM_BIG_ENCODING
- IA64VMSi_ENCODING
- IA64VMSd_ENCODING
- IA64VMSg_ENCODING

Returns:

string representation of encodingType. The returned value is one of the following:
- NETWORK
- SUN
- DECSTATION
- SGi
- IBMPC
- IBMRS
- HOST
- PPC
- HP
- NeXT
- ALPHAOSF1
- ALPHAVMSd
- ALPHAVMSg
- ALPHAVMSi
- ARM_LITTLE
- ARM_BIG
- IA64VMSi
- IA64VMSd
- IA64VMSg
- UNKNOWN (for unknown encodingType)

getLongEncoding

```
public static long getLongEncoding(java.lang.String encodingType)
```

Gets the long value of the given CDF encoding type in string.

Parameters:

`encodingType` - the CDF encoding type to be examined. It should be one of the following:
- NETWORK
- SUN
- DECSTATION
- SGi
- IBMPC
- IBMRS
- HOST
- PPC
- HP
- NeXT
- ALPHAOSF1
- ALPHAVMSd
- ALPHAVMSg
- ALPHAVMSi
- ARM_LITTLE
- ARM_BIG
- IA64VMSi

- IA64VMSd
- IA64VMSg

Returns:

long representation of encodingType. The returned value is one of the following:

- NETWORK_ENCODING
- SUN_ENCODING
- DECSTATION_ENCODING
- SGi_ENCODING
- IBMPC_ENCODING
- IBMRS_ENCODING
- HOST_ENCODING
- PPC_ENCODING
- HP_ENCODING
- NeXT_ENCODING
- ALPHAOSF1_ENCODING
- ALPHAVMSd_ENCODING
- ALPHAVMSg_ENCODING
- ALPHAVMSi_ENCODING
- ARM_LITTLE_ENCODING
- ARM_BIG_ENCODING
- IA64VMSi_ENCODING
- IA64VMSd_ENCODING
- IA64VMSg_ENCODING
- -1 (for unknown encodingType)

getStringEncoding

```
public static java.lang.String getStringEncoding(CDF cdf)
```

Get the string value of the given CDF's encoding type.

Parameters:

`cdf` - the CDF to be examined

Returns:

string representation of the given CDF's encoding type. See getStringEncoding(long encodingType) for possible return values.

getStringDecoding

```
public static java.lang.String getStringDecoding(long decodingType)
                                          throws CDFException
```

Gets the string value of the given CDF decoding type

.

Parameters:

`decodingType` - the CDF decoding type to be examined. It should be one of the following:

- NETWORK_DECODING
- SUN_DECODING
- DECSTATION_DECODING
- SGi_DECODING
- IBMPC_DECODING
- IBMRS_DECODING
- HOST_DECODING

PPC_DECODING
- HP_DECODING
- NeXT_DECODING
- ALPHAOSF1_DECODING
- ALPHAVMSd_DECODING
- ALPHAVMSg_DECODING
- ALPHAVMSi_DECODING
- ARM_LITTLE_DECODING
- ARM_BIG_DECODING
- IA64VMSi_DECODING
- IA64VMSd_DECODING
- IA64VMSg_DECODING
- -1 (for unknown encodingType)

Returns:

string representation of decodingType. See getStringEncoding (long encodingType) for possible return values.

Throws:

`CDFException` - if a problem occurs getting the string value of the given decoding type

getStringDecoding

```
public static java.lang.String getStringDecoding(CDF cdf)
                                          throws CDFException
```

Gets the string value of the given CDF file's decoding type.

Parameters:

`cdf` - the CDF to be examined

Returns:

string representation of the given CDF file's decoding type. See getStringEncoding (long encodingType) for possible return values.

Throws:

`CDFException` - if a problem occurs getting the value of the decoding type defined for the given CDF

getStringMajority

```
public static java.lang.String getStringMajority(long majorityType)
```

Gets the string value of the given CDF majority.

Parameters:

`majorityType` - the CDF majority to be translated

Returns:

string representation of majorityType. The returned value is one of the following:
- ROW
- COLUMN
- UNKNOWN (for unknown majorityType)

getLongMajority

```
public static long getLongMajority(java.lang.String majorityType)
```

Gets the long value of the given CDF majority.

Parameters:

`majorityType` - the CDF majority to be translated. It should be either ROW or COLUMN

Returns:

long representation of majorityType. The returned value is one of the following:

- ROW_MAJOR
- COLUMN_MAJOR
- -1 (for unknown majorityType)

getStringMajority

```
public static java.lang.String getStringMajority(CDF cdf)
```

Gets the string value of the given CDF file's majority.

Parameters:

`cdf` - the CDF to be examined

Returns:

string representation of the given CDF file's majority. The returned value is one of the following:

- ROW
- COLUMN

getStringFormat

```
public static java.lang.String getStringFormat(long formatType)
```

Gets the string value of the given CDF's file format.

Parameters:

`formatType` - the CDF file format to be translated. It should be either SINGLE or MULTI

Returns:

string representation of formatType. The returned value is either SINGLE, MULTI, or UNKNOWN.

getLongFormat

```
public static long getLongFormat(java.lang.String formatType)
```

Gets the long value of the given CDF file format in string.

Parameters:

`formatType` - the CDF file format to be translated. It should be either SINGLE or MULTI.

Returns:

long representation of formatType. The returned value is one of the following:

- SINGLE_FILE
- MULTI_FILE
- -1 (for unknown format type)

getStringFormat

```
public static java.lang.String getStringFormat(CDF cdf)
```

Gets the string value of the given CDF's file format.

Parameters:

`cdf` - the CDF to be examined

Returns:

string representation of given CDF's file format. The returned value is either SINGLE, MULTI, or UNKNOWN.

getStringSparseRecord

```
public static java.lang.String getStringSparseRecord(long sparseRecordType)
```

Gets the string value of the given sparse record type.

Parameters:

`sparseRecordType` - the sparse record type to be translated. It should be one of the following:

- NO_SPARSERECORDS
- PAD_SPARSERECORDS
- PREV_SPARSERECORDS

Returns:

string representation of sparseRecordType. The returned value is one of the following:

- None
- PAD
- PREV
- UNKNOWN

getLongChecksum

```
public static long getLongChecksum(java.lang.String checksum)
```

Gets the long value of the given CDF's checksum in string.

Parameters:

`checksum` - the checksum string of which to be translated.

Returns:

long value of checksum type. The returned value is either NONE_CHECKSUM, MD5_CHECKSUM, or OTHER_CHECKSUM.

getLeapSecondLastUpdated

```
public static long getLeapSecondLastUpdated(CDF cdf)
                                     throws CDFException
```

Gets the date of the given CDF's last updated for the leap second.

Parameters:

`cdf` - the CDF with which its leap second last updated.

Returns:

value (in YYYYMMDD) representation of leap second last updated, if set. -1 if the value is not set (prior to CDF V3.5.1).

Throws:

`CDFException` - An exception is thrown if the operation fails

getStringChecksum

```
public static java.lang.String getStringChecksum(CDF cdf)
```

Gets the string value of the given CDF's checksum.

Parameters:

`cdf` - the CDF with which its checksum to be translated.

Returns:

string representation of checksum type. The returned value is either NONE, MD5, or OTHER.

getStringChecksum

```
public static java.lang.String getStringChecksum(long checksumType)
```

Gets the string value of the given CDF's checksum.

Parameters:

`checksumType` - the CDF checksum to be translated. It should be either NO_CHECKSUM (or NONE_CHECKSUM) or MD5_CHECKSUM

Returns:

string representation of checksumType. The returned value is either NONE, MD5, or OTHER.

getLongSparseRecord

```
public static long getLongSparseRecord(java.lang.String sparseRecordType)
```

Gets the long value of the given sparse record type in string.

Parameters:

`sparseRecordType` - the sparse record type to be translated. It should be one of the following:

- None
- PAD or sRecords.PAD
- PREV or sRecords.PREV

Returns:

long representation of sparseRecordType. The returned value is one of the following:

- NO_SPARSERECORDS
- PAD_SPARSERECORDS
- PREV_SPARSERECORDS
- -1 (for unknown sparse record type)

getStringSparseRecord

```
public static java.lang.String getStringSparseRecord(Variable var)
```

Gets the string value of the given variable's sparse record type.

Parameters:

`var` - the variable to be examined

Returns:

string representation of the given variable's sparse record type. The returned value is one of the following:

- None
- PAD
- PREV
- UNKNOWN

cdfFileExists

```
public static boolean cdfFileExists(java.lang.String fileName)
```

Checks the existence of the given CDF file name. If the file name doesn't have ".cdf" file extension, it adds ".cdf" suffix at the end of the file name before checking the existence of the file. If the file exists in the current directory, it returns TRUE. Otherwise, FALSE is returned.

Parameters:

`fileName` - the name of the CDF file to be checked for existence

Returns:

true - if fileName exists in the current directory
false - if fileName doesn't exist in the current directory

CFtoJformat

```
public static java.lang.String CFtoJformat(java.lang.String cfFormat)
```

Convert a C/Fortran-based print format to a corresponding Java style format for numerical data.

Parameters:

`cfFormat` - C/Fortran-based format string

Returns:

Java style format or null if not a valid or a non-handled C/Fortran format

FtoJformat

```
public static java.lang.String FtoJformat(java.lang.String fFormat)
```

Convert a Fortran-based print format to a corresponding Java style format for numerical data. Limited Fortran format with I, F, E and G forms.

Parameters:

`fFormat` - Fortran-based format string

Returns:

jFormat Java style format or null if not a valid or a non-handled Fortran format

CtoJformat

```
public static java.lang.String CtoJformat(java.lang.String cFormat)
```

Convert a C-based print format to a corresponding Java style format for numerical data. Limited C format with I, F, E and G forms.

Parameters:

`cFormat` - C-based format string

Returns:

jFormat Java style format or null if not a valid or a non-handled Fortran format

isFloatingPadValue

```
public static int isFloatingPadValue(long dataType,
                     java.lang.Object value)
```

Check if a data is a floating point and its value is a pad value.

Parameters:

`dataType` - The data type of the data

`value` - A single data object

Returns:

1 if the data is of floating point and it is a pad value 0 otherwise

isFloatingPadValue

```
public static int isFloatingPadValue(java.lang.Object value)
```

Check if a data is a floating point and its value is a pad value.

Parameters:

`value` - A data object

Returns:

1 if the data is of floating point and it is a pad value 0 otherwise

isFloatingPadValue

```
public static int isFloatingPadValue(long dataType,
                     float value)
```

Check if a data is a floating point and its value is a pad value.

Parameters:

`dataType` - The data type of the data

`value` - The float data

Returns:

1 if the data is of floating point and it is a pad value 0 otherwise

isFloatingPadValue

```
public static int isFloatingPadValue(long dataType,
                      double value)
```

Check if a data is a floating point and its value is a pad value.

Parameters:

`dataType` - The data type of the data

`value` - The double data

Returns:

1 if the data is of floating point and it is a pad value 0 otherwise

isFloatingPadValue

```
public static int isFloatingPadValue(float value)
```

Check if a float data is a pad value.

Parameters:

`value` - a float data

Returns:

1 if the data is a pad value 0 otherwise

isFloatingPadValue

```
public static int isFloatingPadValue(double value)
```

Check if a double data is a pad value.

Parameters:

`value` - a double data

Returns:

1 if the data is a pad value 0 otherwise

getNumDimensions

```
public static int getNumDimensions(java.lang.Object obj)
```

getDimensionSizes

```
public static long[] getDimensionSizes(java.lang.Object obj)
```

UTF8toUTF16LE

```
public static byte[] UTF8toUTF16LE(java.lang.Object utf8)
```

UTF16LEtoUTF8

```
public static byte[] UTF16LEtoUTF8(java.lang.Object utf16)
```

UTF8toCp1252

```
public static byte[] UTF8toCp1252(java.lang.Object utf8)
```

Cp1252toUTF8

```
public static byte[] Cp1252toUTF8(java.lang.Object cp1252)
```

gsfc.nssdc.cdf

Class Entry

java.lang.Object
gsfc.nssdc.cdf.Entry

All Implemented Interfaces:

CDFConstants, CDFObject

```
public class Entry
extends java.lang.Object
implements CDFObject, CDFConstants
```

This class describes a CDF global or variable attribute entry.

Note: In the Java CDF API there is no concept of an rEntry since r variables are not supported. Only z variables are supported since it is far superior and efficient than r variables.

Version:

1.0, 2.0 03/18/05 Selection of current CDF, attribute and entry are done as part of operations passed to JNI. JNI call is synchronized so only one process is allowed in a JVM, due to multi-thread safety. The select method will never be called.

See Also:

Attribute

Field Summary

Fields inherited from interface gsfc.nssdc.cdf.CDFConstants

AHUFF_COMPRESSION, ALPHAOSF1_DECODING, ALPHAOSF1_ENCODING, ALPHAVMSd_DECODING, ALPHAVMSd_ENCODING, ALPHAVMSg_DECODING, ALPHAVMSg_ENCODING, ALPHAVMSi_DECODING, ALPHAVMSi_ENCODING, ARM_BIG_DECODING, ARM_BIG_ENCODING, ARM_LITTLE_DECODING, ARM_LITTLE_ENCODING, ATTR_, ATTR_EXISTENCE_, ATTR_EXISTS, ATTR_MAXgENTRY_, ATTR_MAXrENTRY_, ATTR_MAXzENTRY_, ATTR_NAME_, ATTR_NAME_TRUNC, ATTR_NUMBER_, ATTR_NUMgENTRIES_, ATTR_NUMrENTRIES_, ATTR_NUMzENTRIES_, ATTR_SCOPE_, BACKWARD_, BACKWARDFILEoff, BACKWARDFILEon, BAD_ALLOCATE_RECS, BAD_ARGUMENT, BAD_ATTR_NAME, BAD_ATTR_NUM, BAD_BLOCKING_FACTOR, BAD_CACHE_SIZE, BAD_CDF_EXTENSION, BAD_CDF_ID, BAD_CDF_NAME, BAD_CDFSTATUS, BAD_CHECKSUM, BAD_COMPRESSION_PARM, BAD_DATA_TYPE, BAD_DECODING, BAD_DIM_COUNT, BAD_DIM_INDEX, BAD_DIM_INTERVAL, BAD_DIM_SIZE, BAD_ENCODING, BAD_ENTRY_NUM, BAD_FNC_OR_ITEM, BAD_FORMAT, BAD_INITIAL_RECS, BAD_MAJORITY, BAD_MALLOC, BAD_NEGtoPOSfp0_MODE, BAD_NUM_DIMS, BAD_NUM_ELEMS, BAD_NUM_STRINGS, BAD_NUM_VARS, BAD_READONLY_MODE, BAD_REC_COUNT, BAD_REC_INTERVAL, BAD_REC_NUM, BAD_SCOPE, BAD_SCRATCH_DIR, BAD_SPARSEARRAYS_PARM, BAD_VAR_NAME, BAD_VAR_NUM, BAD_zMODE, BADDATE_LEAPSECOND_UPDATED, BeginUnixTimeEPOCH, BeginUnixTimeEPOCH16, BLOCKINGFACTOR_TOO_LARGE, BLOCKINGFACTOR_TOO_SMALL, BLOCKINGFACTOR_TOO_SMALL2, CANNOT_ALLOCATE_RECORDS, CANNOT_CHANGE, CANNOT_COMPRESS, CANNOT_CONVERT_WIDECHAR, CANNOT_COPY, CANNOT_INSERT_RECORDS, CANNOT_SPARSEARRAYS, CANNOT_SPARSERECORDS, CDF_, CDF_ACCESS_, CDF_ATTR_NAME_LEN, CDF_ATTR_NAME_LEN256, CDF_BYTE, CDF_CACHESIZE_, CDF_CHAR, CDF_CHECKSUM_, CDF_CLOSE_ERROR, CDF_COMPRESSION_, CDF_COPYRIGHT_, CDF_COPYRIGHT_LEN, CDF_CREATE_ERROR, CDF_DECODING_, CDF_DELETE_ERROR, CDF_DOUBLE, CDF_ENCODING_, CDF_EPOCH, CDF_EPOCH16, CDF_EXISTS, CDF_FLOAT, CDF_FORMAT_, CDF_INCREMENT_, CDF_INFO_, CDF_INT1, CDF_INT2, CDF_INT4, CDF_INT8, CDF_INTERNAL_ERROR, CDF_LEAPSECONDLASTUPDATED_, CDF_MAJORITY_, CDF_MAX_DIMS, CDF_MAX_PARMS, CDF_MIN_DIMS, CDF_NAME_, CDF_NAME_TRUNC, CDF_NEGtoPOSfp0_MODE_, CDF_NUMATTRS_, CDF_NUMgATTRS_, CDF_NUMrVARS_, CDF_NUMvATTRS_, CDF_NUMzVARS_, CDF_OK, CDF_OPEN_ERROR, CDF_PATHNAME_LEN, CDF_READ_ERROR, CDF_READONLY_MODE_, CDF_REAL4, CDF_REAL8, CDF_RELEASE_, CDF_SAVE_ERROR, CDF_SCRATCHDIR_, CDF_STATUS_, CDF_STATUSTEXT_LEN, CDF_TIME_TT2000, CDF_UCHAR, CDF_UINT1, CDF_UINT2, CDF_UINT4, CDF_VAR_NAME_LEN, CDF_VAR_NAME_LEN256, CDF_VERSION_, CDF_WARN, CDF_WRITE_ERROR, CDF_zMODE_, CDFwithSTATS_, CHECKSUM_, CHECKSUM_ERROR, CHECKSUM_NOT_ALLOWED, CLOSE_, COLUMN_MAJOR, COMPRESS_CACHESIZE_, COMPRESSION_ERROR, CONFIRM_, CORRUPTED_V2_CDF, CORRUPTED_V3_CDF, CREATE_, CURgENTRY_EXISTENCE_, CURrENTRY_EXISTENCE_, CURzENTRY_EXISTENCE_, DATATYPE_MISMATCH, DATATYPE_SIZE_, DECOMPRESSION_ERROR, DECSTATION_DECODING, DECSTATION_ENCODING, DEFAULT_BYTE_PADVALUE, DEFAULT_CHAR_PADVALUE, DEFAULT_DOUBLE_PADVALUE, DEFAULT_EPOCH_PADVALUE, DEFAULT_EPOCH16_PADVALUE, DEFAULT_FLOAT_PADVALUE, DEFAULT_INT1_PADVALUE, DEFAULT_INT2_PADVALUE, DEFAULT_INT4_PADVALUE, DEFAULT_INT8_PADVALUE, DEFAULT_REAL4_PADVALUE, DEFAULT_REAL8_PADVALUE, DEFAULT_TT2000_PADVALUE, DEFAULT_UCHAR_PADVALUE, DEFAULT_UINT1_PADVALUE, DEFAULT_UINT2_PADVALUE, DEFAULT_UINT4_PADVALUE, DELETE_, DID_NOT_COMPRESS, EMPTY_COMPRESSED_CDF,

END_OF_VAR, EPOCH_STRING_LEN, EPOCH_STRING_LEN_EXTEND, EPOCH1_STRING_LEN, EPOCH1_STRING_LEN_EXTEND, EPOCH2_STRING_LEN, EPOCH2_STRING_LEN_EXTEND, EPOCH3_STRING_LEN, EPOCH3_STRING_LEN_EXTEND, EPOCH4_STRING_LEN, EPOCH4_STRING_LEN_EXTEND, EPOCHx_FORMAT_MAX, EPOCHx_STRING_MAX, FILLED_TT2000_VALUE, FORCED_PARAMETER, FUNCTION_NOT_SUPPORTED, gENTRY_, gENTRY_DATA_, gENTRY_DATASPEC_, gENTRY_DATATYPE_, gENTRY_EXISTENCE_, gENTRY_NUMELEMS_, GET_, GETCDFCHECKSUM_, GETCDFFILEBACKWARD_, GETCDFVALIDATE_, GETLEAPSECONDSENVVAR_, GLOBAL_SCOPE, GZIP_COMPRESSION, HOST_DECODING, HOST_ENCODING, HP_DECODING, HP_ENCODING, HUFF_COMPRESSION, IA64VMSd_DECODING, IA64VMSd_ENCODING, IA64VMSg_DECODING, IA64VMSg_ENCODING, IA64VMSi_DECODING, IA64VMSi_ENCODING, IBM_PC_OVERFLOW, IBMPC_DECODING, IBMPC_ENCODING, IBMRS_DECODING, IBMRS_ENCODING, ILLEGAL_EPOCH_FIELD, ILLEGAL_EPOCH_VALUE, ILLEGAL_FOR_SCOPE, ILLEGAL_IN_zMODE, ILLEGAL_ON_V1_CDF, ILLEGAL_TT2000_VALUE, IS_A_NETCDF, LIB_COPYRIGHT_, LIB_INCREMENT_, LIB_RELEASE_, LIB_subINCREMENT_, LIB_VERSION_, MAC_DECODING, MAC_ENCODING, MD5_CHECKSUM, MULTI_FILE, MULTI_FILE_FORMAT, NA_FOR_VARIABLE, NEGATIVE_FP_ZERO, NEGtoPOSfp0off, NEGtoPOSfp0on, NETWORK_DECODING, NETWORK_ENCODING, NeXT_DECODING, NeXT_ENCODING, NO_ATTR_SELECTED, NO_CDF_SELECTED, NO_CHECKSUM, NO_COMPRESSION, NO_DELETE_ACCESS, NO_ENTRY_SELECTED, NO_MORE_ACCESS, NO_PADVALUE_SPECIFIED, NO_SPARSEARRAYS, NO_SPARSERECORDS, NO_STATUS_SELECTED, NO_SUCH_ATTR, NO_SUCH_CDF, NO_SUCH_ENTRY, NO_SUCH_RECORD, NO_SUCH_VAR, NO_VAR_SELECTED, NO_VARS_IN_CDF, NO_WRITE_ACCESS, NONE_CHECKSUM, NOT_A_CDF, NOT_A_CDF_OR_NOT_SUPPORTED, NOVARY, NULL_, OPEN_, OPTIMAL_ENCODING_TREES, OTHER_CHECKSUM, PAD_SPARSERECORDS, PPC_DECODING, PPC_ENCODING, PRECEEDING_RECORDS_ALLOCATED, PREV_SPARSERECORDS, PUT_, READ_ONLY_DISTRIBUTION, READ_ONLY_MODE, READONLYoff, READONLYon, rENTRY_, rENTRY_DATA_, rENTRY_DATASPEC_, rENTRY_DATATYPE_, rENTRY_EXISTENCE_, rENTRY_NAME_, rENTRY_NUMELEMS_, rENTRY_NUMSTRINGS_, rENTRY_STRINGSDATA_, RLE_COMPRESSION, RLE_OF_ZEROs, ROW_MAJOR, rVAR_, rVAR_ALLOCATEBLOCK_, rVAR_ALLOCATEDFROM_, rVAR_ALLOCATEDTO_, rVAR_ALLOCATERECS_, rVAR_BLOCKINGFACTOR_, rVAR_CACHESIZE_, rVAR_COMPRESSION_, rVAR_DATA_, rVAR_DATASPEC_, rVAR_DATATYPE_, rVAR_DIMVARYS_, rVAR_EXISTENCE_, rVAR_HYPERDATA_, rVAR_INITIALRECS_, rVAR_MAXallocREC_, rVAR_MAXREC_, rVAR_NAME_, rVAR_nINDEXENTRIES_, rVAR_nINDEXLEVELS_, rVAR_nINDEXRECORDS_, rVAR_NUMallocRECS_, rVAR_NUMBER_, rVAR_NUMELEMS_, rVAR_NUMRECS_, rVAR_PADVALUE_, rVAR_RECORDS_, rVAR_RECORDS_RENUMBER_, rVAR_RECVARY_, rVAR_RESERVEPERCENT_, rVAR_SEQDATA_, rVAR_SEQPOS_, rVAR_SPARSEARRAYS_, rVAR_SPARSERECORDS_, rVARs_CACHESIZE_, rVARs_DIMCOUNTS_, rVARs_DIMINDICES_, rVARs_DIMINTERVALS_, rVARs_DIMSIZES_, rVARs_MAXREC_, rVARs_NUMDIMS_, rVARs_RECCOUNT_, rVARs_RECDATA_, rVARs_RECINTERVAL_, rVARs_RECNUMBER_, SAVE_, SCRATCH_CREATE_ERROR, SCRATCH_DELETE_ERROR, SCRATCH_READ_ERROR, SCRATCH_WRITE_ERROR, SELECT_, SGI_DECODING, SGi_ENCODING, SINGLE_FILE, SINGLE_FILE_FORMAT, SOME_ALREADY_ALLOCATED, STAGE_CACHESIZE_, STATUS_TEXT_, STRING_NOT_UTF8_ENCODING, STRINGDELIMITER, SUN_DECODING, SUN_ENCODING, TOO_MANY_PARMS, TOO_MANY_VARS, TRY_TO_READ_NONSTRING_DATA, TT2000_0_STRING_LEN, TT2000_1_STRING_LEN, TT2000_2_STRING_LEN, TT2000_3_STRING_LEN, TT2000_4_STRING_LEN, TT2000_CDF_MAYNEEDUPDATE, TT2000_TIME_ERROR, TT2000_USED_OUTDATED_TABLE, UNABLE_TO_PROCESS_CDF, UNKNOWN_COMPRESSION, UNKNOWN_SPARSENESS, UNSUPPORTED_OPERATION, VALIDATE_, VALIDATEFILEoff, VALIDATEFILEon, VAR_ALREADY_CLOSED, VAR_CLOSE_ERROR, VAR_CREATE_ERROR, VAR_DELETE_ERROR, VAR_EXISTS, VAR_NAME_TRUNC, VAR_OPEN_ERROR, VAR_READ_ERROR, VAR_SAVE_ERROR, VAR_WRITE_ERROR, VARIABLE_SCOPE, VARY, VAX_DECODING, VAX_ENCODING, VIRTUAL_RECORD_DATA, zENTRY_, zENTRY_DATA_, zENTRY_DATASPEC_, zENTRY_DATATYPE_, zENTRY_EXISTENCE_, zENTRY_NAME_, zENTRY_NUMELEMS_, zENTRY_NUMSTRINGS_, zENTRY_STRINGSDATA_, ZLIB_COMPRESS_ERROR, ZLIB_UNCOMPRESS_ERROR, zMODEoff, zMODEon1, zMODEon2, zVAR_, zVAR_ALLOCATEBLOCK_, zVAR_ALLOCATEDFROM_, zVAR_ALLOCATEDTO_, zVAR_ALLOCATERECS_, zVAR_BLOCKINGFACTOR_, zVAR_CACHESIZE_, zVAR_COMPRESSION_, zVAR_DATA_, zVAR_DATASPEC_, zVAR_DATATYPE_, zVAR_DIMCOUNTS_, zVAR_DIMINDICES_, zVAR_DIMINTERVALS_, zVAR_DIMSIZES_, zVAR_DIMVARYS_, zVAR_EXISTENCE_, zVAR_HYPERDATA_, zVAR_INITIALRECS_, zVAR_MAXallocREC_, zVAR_MAXREC_, zVAR_NAME_, zVAR_nINDEXENTRIES_, zVAR_nINDEXLEVELS_, zVAR_nINDEXRECORDS_, zVAR_NUMallocRECS_, zVAR_NUMBER_, zVAR_NUMDIMS_, zVAR_NUMELEMS_, zVAR_NUMRECS_, zVAR_PADVALUE_, zVAR_RECCOUNT_, zVAR_RECINTERVAL_, zVAR_RECNUMBER_, zVAR_RECORDS_, zVAR_RECORDS_RENUMBER_, zVAR_RECVARY_, zVAR_RESERVEPERCENT_, zVAR_SEQDATA_, zVAR_SEQPOS_, zVAR_SPARSEARRAYS_, zVAR_SPARSERECORDS_, zVARs_CACHESIZE_, zVARs_MAXREC_, zVARs_RECDATA_, zVARs_RECNUMBER_

Method Summary

Methods

Modifier and Type	Method and Description
static **Entry**	**create**(**Attribute** myAttribute, long id, long dataType, java.lang.Object data) Creates a new global or variable attribute entry.
void	**delete**() Deletes this entry.
java.lang.Object	**getData**() Gets the data for this entry.
long	**getDataType**() Gets the CDF data type of this entry.
long	**getID**() Gets the ID of this entry.
java.lang.String	**getName**() Gets the name of this entry.
long	**getNumElements**() Gets the number of elements in this entry.

long	**getNumStrings**() Gets the number of strings in the data.
long	**getScope**() Gets the scope of the attribute this entry in in.
void	**putData**(long dataType, java.lang.Object data) Put the entry data into the CDF.
void	**rename**(java.lang.String name) This method is here as a placeholder since the Entry class implements the CDFObject interface that includes "rename".
void	**updateDataSpec**(long dataType, long numElements) Update the data specification (data type and number of elements) of the entry.

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Method Detail

create

```
public static Entry create(Attribute myAttribute,
           long id,
           long dataType,
           java.lang.Object data)
                    throws CDFException
```

Creates a new global or variable attribute entry. One can create as many global and variable entries as needed. The following example creates four entries for the global attribute "Project":

```
     Attribute project  = Attribute.create(cdf, "Project", GLOBAL_SCOPE);
     Entry.create(project, 0, CDF_CHAR, "Project name: IMAGE");
     Entry.create(project, 1, CDF_CHAR, "Description 1");
     Entry.create(project, 2, CDF_CHAR, "Description 2");
```

The following example creates a variable attribute entry for the variable "Longitude" associated with the attribute "VALIDMIN":

```
     Variable longitude = cdf.getVariable("Longitude");
     Attribute validMin = Attribute.create(cdf, "VALIDMIN",
                                           VARIABLE_SCOPE);
     Entry.create(validMin, longitude.getID(), CDF_INT2,
                  new Short((short)10));
             OR
     longitude.putEntry(validMin, CDF_INT2, new Short((short)180));
```

Parameters:

`myAttribute` - the attribute to which this entry belongs

`id` - the entry id

`dataType` - the CDF data type for this entry that should be one of the following:

- 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_UNIT2 - 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 - 2*8-byte, floating point
- CDF_TIME_TT2000 - 8-byte, signed integer

`data` - the entry data to be added

Note: From CDF V3.7.0, the string typed data of variable entry can be an array of strings. Data from previous versions can only be a single string.

Returns:

newly created attribute entry

Throws:

`CDFException` - CDFException if there is a problem creating an entry

delete

```
public void delete()
            throws CDFException
```

Deletes this entry.

Specified by:

`delete` in interface `CDFObject`

Throws:

`CDFException` - CDFException if there is a problem deleting this entry

getDataType

```
public long getDataType()
```

Gets the CDF data type of this entry. See the description of the create method for the CDF data types supported by the CDF library.

Returns:

the CDF data type of this entry

getScope

```
public long getScope()
```

Gets the scope of the attribute this entry in in.

Returns:

the scope of this entry

getNumElements

```
public long getNumElements()
```

Gets the number of elements in this entry. For CDF_CHAR, it returns the number of characters stored.

```
     Entry data         Number of elements
     ----------         ------------------
     10                 1
     20.8               1
     10 20 30           3
     20.8 20.9          2
     "Upper Limits"     12
```

Returns:

the number of elements stored in this entry

getNumStrings

```
public long getNumStrings()
```

Gets the number of strings in the data. If the data is not string type, 0 is returned. For global attribute entry, it always returns 1.

Returns:

the string number for this entry

getData

```
public java.lang.Object getData()
                         throws CDFException
```

Gets the data for this entry. Build a new data object so a duplicated variable will have its own copy of the entry data.

Returns:

the data for this entry Note: From CDF V3.7.0, the string typed data of variable entry can return an array of strings. Data from previous versions only returns a single string.

Throws:

`CDFException` - An exception is thrown if the operation fails

getID

```
public long getID()
```

Gets the ID of this entry.

Returns:

the ID/number of this entry

getName

```
public java.lang.String getName()
```

Gets the name of this entry. Since an entry doesn't have its own name, the string representation of this entry ID is returned.

This method overrides the getName() method defined in the Java Object class. If this method is called explicitly or implicitly (i.e. just

the entry name by itself), it returns the string representation of the entry ID.

Specified by:

`getName` in interface `CDFObject`

Returns:

string representation of this attribute entry ID

rename

```
public void rename(java.lang.String name)
            throws CDFException
```

This method is here as a placeholder since the Entry class implements the CDFObject interface that includes "rename".

Specified by:

`rename` in interface `CDFObject`

Parameters:

`name` - - not applicable

Throws:

`CDFException` - CDFException - not applicable

updateDataSpec

```
public void updateDataSpec(long dataType,
                  long numElements)
                    throws CDFException
```

Update the data specification (data type and number of elements) of the entry.

Parameters:

`dataType` - the data type of the entry data

`numElements` - the number of elements of the entry data

Throws:

`CDFException` - An exception is thrown if the operation fails

putData

```
public void putData(long dataType,
            java.lang.Object data)
              throws CDFException
```

Put the entry data into the CDF.

Parameters:

`dataType` - the data type of the entry data

`data` - the entry data to write Note: From CDF V3.7.0, the string typed data of variable entry can be an array of strings. Data from previous versions can only be a single string.

Throws:

`CDFException` - An exception is thrown if the operation fails

gsfc.nssdc.cdf.util

Class Epoch

java.lang.Object
gsfc.nssdc.cdf.util.Epoch

All Implemented Interfaces:

CDFConstants

```
public class Epoch
extends java.lang.Object
implements CDFConstants
```

This class contains the handy utility routines (methods) called by the CDF applications to handle epoch data of CDF's CDF_EPOCH data type.

Version:

1.0 **Example:**

```
// Get the milliseconds to Aug 5, 1990 at 5:00
double ep = Epoch.compute(1990, 8, 5, 5, 0, 0, 0);
//Get the year, month, day, hour, minutes, seconds, milliseconds for ep
long[] times = Epoch.breakdown(ep);
for (int i=0;i<times.length;i++)
    System.out.print(times[i]+" ");
System.out.println();
// Printout the epoch in various formats
System.out.println(Epoch.encode(ep));
System.out.println(Epoch.encode1(ep));
System.out.println(Epoch.encode2(ep));
System.out.println(Epoch.encode3(ep));
System.out.println(Epoch.encode4(ep));
// Print out the date using format
String format = "<month> <dom.02>, <year> at <hour>:<min>";
System.out.println(Epoch.encodex(ep,format));
```

Field Summary

Fields inherited from interface gsfc.nssdc.cdf.CDFConstants

AHUFF_COMPRESSION, ALPHAOSF1_DECODING, ALPHAOSF1_ENCODING, ALPHAVMSd_DECODING, ALPHAVMSd_ENCODING, ALPHAVMSg_DECODING, ALPHAVMSg_ENCODING, ALPHAVMSi_DECODING, ALPHAVMSi_ENCODING, ARM_BIG_DECODING, ARM_BIG_ENCODING, ARM_LITTLE_DECODING, ARM_LITTLE_ENCODING, ATTR_, ATTR_EXISTENCE_, ATTR_EXISTS, ATTR_MAXgENTRY_, ATTR_MAXrENTRY_, ATTR_MAXzENTRY_, ATTR_NAME_, ATTR_NAME_TRUNC, ATTR_NUMBER_, ATTR_NUMgENTRIES_, ATTR_NUMrENTRIES_, ATTR_NUMzENTRIES_, ATTR_SCOPE_, BACKWARD_, BACKWARDFILEoff, BACKWARDFILEon, BAD_ALLOCATE_RECS, BAD_ARGUMENT, BAD_ATTR_NAME, BAD_ATTR_NUM, BAD_BLOCKING_FACTOR, BAD_CACHE_SIZE, BAD_CDF_EXTENSION, BAD_CDF_ID, BAD_CDF_NAME, BAD_CDFSTATUS, BAD_CHECKSUM, BAD_COMPRESSION_PARM, BAD_DATA_TYPE, BAD_DECODING, BAD_DIM_COUNT, BAD_DIM_INDEX, BAD_DIM_INTERVAL, BAD_DIM_SIZE, BAD_ENCODING, BAD_ENTRY_NUM, BAD_FNC_OR_ITEM, BAD_FORMAT, BAD_INITIAL_RECS, BAD_MAJORITY, BAD_MALLOC, BAD_NEGtoPOSfp0_MODE, BAD_NUM_DIMS, BAD_NUM_ELEMS, BAD_NUM_STRINGS, BAD_NUM_VARS, BAD_READONLY_MODE, BAD_REC_COUNT, BAD_REC_INTERVAL, BAD_REC_NUM, BAD_SCOPE, BAD_SCRATCH_DIR, BAD_SPARSEARRAYS_PARM, BAD_VAR_NAME, BAD_VAR_NUM, BAD_zMODE, BADDATE_LEAPSECOND_UPDATED, BeginUnixTimeEPOCH, BeginUnixTimeEPOCH16, BLOCKINGFACTOR_TOO_LARGE, BLOCKINGFACTOR_TOO_SMALL, BLOCKINGFACTOR_TOO_SMALL2, CANNOT_ALLOCATE_RECORDS, CANNOT_CHANGE, CANNOT_COMPRESS, CANNOT_CONVERT_WIDECHAR, CANNOT_COPY, CANNOT_INSERT_RECORDS, CANNOT_SPARSEARRAYS, CANNOT_SPARSERECORDS, CDF_, CDF_ACCESS_, CDF_ATTR_NAME_LEN, CDF_ATTR_NAME_LEN256, CDF_BYTE, CDF_CACHESIZE_, CDF_CHAR, CDF_CHECKSUM_, CDF_CLOSE_ERROR, CDF_COMPRESSION_, CDF_COPYRIGHT_, CDF_COPYRIGHT_LEN, CDF_CREATE_ERROR, CDF_DECODING_, CDF_DELETE_ERROR, CDF_DOUBLE, CDF_ENCODING_, CDF_EPOCH, CDF_EPOCH16, CDF_EXISTS, CDF_FLOAT, CDF_FORMAT_, CDF_INCREMENT_, CDF_INFO_, CDF_INT1, CDF_INT2, CDF_INT4, CDF_INT8, CDF_INTERNAL_ERROR, CDF_LEAPSECONDLASTUPDATED_, CDF_MAJORITY_, CDF_MAX_DIMS, CDF_MAX_PARMS, CDF_MIN_DIMS, CDF_NAME_, CDF_NAME_TRUNC, CDF_NEGtoPOSfp0_MODE_,

CDF_NUMATTRS_, CDF_NUMgATTRS_, CDF_NUMrVARS_, CDF_NUMvATTRS_, CDF_NUMzVARS_, CDF_OK,
CDF_OPEN_ERROR, CDF_PATHNAME_LEN, CDF_READ_ERROR, CDF_READONLY_MODE_, CDF_REAL4, CDF_REAL8,
CDF_RELEASE_, CDF_SAVE_ERROR, CDF_SCRATCHDIR_, CDF_STATUS_, CDF_STATUSTEXT_LEN, CDF_TIME_TT2000,
CDF_UCHAR, CDF_UINT1, CDF_UINT2, CDF_UINT4, CDF_VAR_NAME_LEN, CDF_VAR_NAME_LEN256, CDF_VERSION_,
CDF_WARN, CDF_WRITE_ERROR, CDF_zMODE_, CDFwithSTATS_, CHECKSUM_, CHECKSUM_ERROR,
CHECKSUM_NOT_ALLOWED, CLOSE_, COLUMN_MAJOR, COMPRESS_CACHESIZE_, COMPRESSION_ERROR, CONFIRM_,
CORRUPTED_V2_CDF, CORRUPTED_V3_CDF, CREATE_, CURgENTRY_EXISTENCE_, CURrENTRY_EXISTENCE_,
CURzENTRY_EXISTENCE_, DATATYPE_MISMATCH, DATATYPE_SIZE_, DECOMPRESSION_ERROR, DECSTATION_DECODING,
DECSTATION_ENCODING, DEFAULT_BYTE_PADVALUE, DEFAULT_CHAR_PADVALUE, DEFAULT_DOUBLE_PADVALUE,
DEFAULT_EPOCH_PADVALUE, DEFAULT_EPOCH16_PADVALUE, DEFAULT_FLOAT_PADVALUE, DEFAULT_INT1_PADVALUE,
DEFAULT_INT2_PADVALUE, DEFAULT_INT4_PADVALUE, DEFAULT_INT8_PADVALUE, DEFAULT_REAL4_PADVALUE,
DEFAULT_REAL8_PADVALUE, DEFAULT_TT2000_PADVALUE, DEFAULT_UCHAR_PADVALUE, DEFAULT_UINT1_PADVALUE,
DEFAULT_UINT2_PADVALUE, DEFAULT_UINT4_PADVALUE, DELETE_, DID_NOT_COMPRESS, EMPTY_COMPRESSED_CDF,
END_OF_VAR, EPOCH_STRING_LEN, EPOCH_STRING_LEN_EXTEND, EPOCHI_STRING_LEN,
EPOCH1_STRING_LEN_EXTEND, EPOCH2_STRING_LEN, EPOCH2_STRING_LEN_EXTEND, EPOCH3_STRING_LEN,
EPOCH3_STRING_LEN_EXTEND, EPOCH4_STRING_LEN, EPOCH4_STRING_LEN_EXTEND, EPOCHx_FORMAT_MAX,
EPOCHx_STRING_MAX, FILLED_TT2000_VALUE, FORCED_PARAMETER, FUNCTION_NOT_SUPPORTED, gENTRY_,
gENTRY_DATA_, gENTRY_DATASPEC_, gENTRY_DATATYPE_, gENTRY_EXISTENCE_, gENTRY_NUMELEMS_, GET_,
GETCDFCHECKSUM_, GETCDFFILEBACKWARD_, GETCDFVALIDATE_, GETLEAPSECONDSENVVAR_, GLOBAL_SCOPE,
GZIP_COMPRESSION, HOST_DECODING, HOST_ENCODING, HP_DECODING, HP_ENCODING, HUFF_COMPRESSION,
IA64VMSd_DECODING, IA64VMSd_ENCODING, IA64VMSg_DECODING, IA64VMSg_ENCODING, IA64VMSi_DECODING,
IA64VMSi_ENCODING, IBM_PC_OVERFLOW, IBMPC_DECODING, IBMPC_ENCODING, IBMRS_DECODING,
IBMRS_ENCODING, ILLEGAL_EPOCH_FIELD, ILLEGAL_EPOCH_VALUE, ILLEGAL_FOR_SCOPE, ILLEGAL_IN_zMODE,
ILLEGAL_ON_V1_CDF, ILLEGAL_TT2000_VALUE, IS_A_NETCDF, LIB_COPYRIGHT_, LIB_INCREMENT_,
LIB_RELEASE_, LIB_subINCREMENT_, LIB_VERSION_, MAC_DECODING, MAC_ENCODING, MD5_CHECKSUM,
MULTI_FILE, MULTI_FILE_FORMAT, NA_FOR_VARIABLE, NEGATIVE_FP_ZERO, NEGtoPOSfp0off, NEGtoPOSfp0on,
NETWORK_DECODING, NETWORK_ENCODING, NeXT_DECODING, NeXT_ENCODING, NO_ATTR_SELECTED,
NO_CDF_SELECTED, NO_CHECKSUM, NO_COMPRESSION, NO_DELETE_ACCESS, NO_ENTRY_SELECTED, NO_MORE_ACCESS,
NO_PADVALUE_SPECIFIED, NO_SPARSEARRAYS, NO_SPARSERECORDS, NO_STATUS_SELECTED, NO_SUCH_ATTR,
NO_SUCH_CDF, NO_SUCH_ENTRY, NO_SUCH_RECORD, NO_SUCH_VAR, NO_VAR_SELECTED, NO_VARS_IN_CDF,
NO_WRITE_ACCESS, NONE_CHECKSUM, NOT_A_CDF, NOT_A_CDF_OR_NOT_SUPPORTED, NOVARY, NULL_, OPEN_,
OPTIMAL_ENCODING_TREES, OTHER_CHECKSUM, PAD_SPARSERECORDS, PPC_DECODING, PPC_ENCODING,
PRECEEDING_RECORDS_ALLOCATED, PREV_SPARSERECORDS, PUT_, READ_ONLY_DISTRIBUTION, READ_ONLY_MODE,
READONLYoff, READONLYon, rENTRY_, rENTRY_DATA_, rENTRY_DATASPEC_, rENTRY_DATATYPE_,
rENTRY_EXISTENCE_, rENTRY_NAME_, rENTRY_NUMELEMS_, rENTRY_NUMSTRINGS_, rENTRY_STRINGSDATA_,
RLE_COMPRESSION, RLE_OF_ZEROs, ROW_MAJOR, rVAR_, rVAR_ALLOCATEBLOCK_, rVAR_ALLOCATEDFROM_,
rVAR_ALLOCATEDTO_, rVAR_ALLOCATERECS_, rVAR_BLOCKINGFACTOR_, rVAR_CACHESIZE_, rVAR_COMPRESSION_,
rVAR_DATA_, rVAR_DATASPEC_, rVAR_DATATYPE_, rVAR_DIMVARYS_, rVAR_EXISTENCE_, rVAR_HYPERDATA_,
rVAR_INITIALRECS_, rVAR_MAXallocREC_, rVAR_MAXREC_, rVAR_NAME_, rVAR_nINDEXENTRIES_,
rVAR_nINDEXLEVELS_, rVAR_nINDEXRECORDS_, rVAR_NUMallocRECS_, rVAR_NUMBER_, rVAR_NUMELEMS_,
rVAR_NUMRECS_, rVAR_PADVALUE_, rVAR_RECORDS_, rVAR_RECORDS_RENUMBER_, rVAR_RECVARY_,
rVAR_RESERVEPERCENT_, rVAR_SEQDATA_, rVAR_SEQPOS_, rVAR_SPARSEARRAYS_, rVAR_SPARSERECORDS_,
rVARs_CACHESIZE_, rVARs_DIMCOUNTS_, rVARs_DIMINDICES_, rVARs_DIMINTERVALS_, rVARs_DIMSIZES_,
rVARs_MAXREC_, rVARs_NUMDIMS_, rVARs_RECCOUNT_, rVARs_RECDATA_, rVARs_RECINTERVAL_,
rVARs_RECNUMBER_, SAVE_, SCRATCH_CREATE_ERROR, SCRATCH_DELETE_ERROR, SCRATCH_READ_ERROR,
SCRATCH_WRITE_ERROR, SELECT_, SGI_DECODING, SGi_ENCODING, SINGLE_FILE, SINGLE_FILE_FORMAT,
SOME_ALREADY_ALLOCATED, STAGE_CACHESIZE_, STATUS_TEXT_, STRING_NOT_UTF8_ENCODING, STRINGDELIMITER,
SUN_DECODING, SUN_ENCODING, TOO_MANY_PARMS, TOO_MANY_VARS, TRY_TO_READ_NONSTRING_DATA,
TT2000_0_STRING_LEN, TT2000_1_STRING_LEN, TT2000_2_STRING_LEN, TT2000_3_STRING_LEN,
TT2000_4_STRING_LEN, TT2000_CDF_MAYNEEDUPDATE, TT2000_TIME_ERROR, TT2000_USED_OUTDATED_TABLE,
UNABLE_TO_PROCESS_CDF, UNKNOWN_COMPRESSION, UNKNOWN_SPARSENESS, UNSUPPORTED_OPERATION, VALIDATE_,
VALIDATEFILEoff, VALIDATEFILEon, VAR_ALREADY_CLOSED, VAR_CLOSE_ERROR, VAR_CREATE_ERROR,
VAR_DELETE_ERROR, VAR_EXISTS, VAR_NAME_TRUNC, VAR_OPEN_ERROR, VAR_READ_ERROR, VAR_SAVE_ERROR,
VAR_WRITE_ERROR, VARIABLE_SCOPE, VARY, VAX_DECODING, VAX_ENCODING, VIRTUAL_RECORD_DATA, zENTRY_,
zENTRY_DATA_, zENTRY_DATASPEC_, zENTRY_DATATYPE_, zENTRY_EXISTENCE_, zENTRY_NAME_,
zENTRY_NUMELEMS_, zENTRY_NUMSTRINGS_, zENTRY_STRINGSDATA_, ZLIB_COMPRESS_ERROR,
ZLIB_UNCOMPRESS_ERROR, zMODEoff, zMODEon1, zMODEon2, zVAR_, zVAR_ALLOCATEBLOCK_,
zVAR_ALLOCATEDFROM_, zVAR_ALLOCATEDTO_, zVAR_ALLOCATERECS_, zVAR_BLOCKINGFACTOR_, zVAR_CACHESIZE_,
zVAR_COMPRESSION_, zVAR_DATA_, zVAR_DATASPEC_, zVAR_DATATYPE_, zVAR_DIMCOUNTS_, zVAR_DIMINDICES_,
zVAR_DIMINTERVALS_, zVAR_DIMSIZES_, zVAR_DIMVARYS_, zVAR_EXISTENCE_, zVAR_HYPERDATA_,
zVAR_INITIALRECS_, zVAR_MAXallocREC_, zVAR_MAXREC_, zVAR_NAME_, zVAR_nINDEXENTRIES_,
zVAR_nINDEXLEVELS_, zVAR_nINDEXRECORDS_, zVAR_NUMallocRECS_, zVAR_NUMBER_, zVAR_NUMDIMS_,
zVAR_NUMELEMS_, zVAR_NUMRECS_, zVAR_PADVALUE_, zVAR_RECCOUNT_, zVAR_RECINTERVAL_, zVAR_RECNUMBER_,
zVAR_RECORDS_, zVAR_RECORDS_RENUMBER_, zVAR_RECVARY_, zVAR_RESERVEPERCENT_, zVAR_SEQDATA_,
zVAR_SEQPOS_, zVAR_SPARSEARRAYS_, zVAR_SPARSERECORDS_, zVARs_CACHESIZE_, zVARs_MAXREC_,
zVARs_RECDATA_, zVARs_RECNUMBER_

Constructor Summary

Constructors

Constructor and Description
Epoch ()

Method Summary

Methods

Modifier and Type	Method and Description
static long[]	**breakdown**(double epoch) Breaks an EPOCH value down into its component parts.
static long[][]	**breakdown**(double[] epochs) Breaks an array of EPOCH values down into its component parts.
static double[]	**compute**(long[] year, long[] month, long[] day) Computes an array of EPOCH values based on their component parts.
static double[]	**compute**(long[] year, long[] month, long[] day, long[] hour) Computes an array of EPOCH values based on their component parts.
static double[]	**compute**(long[] year, long[] month, long[] day, long[] hour, long[] minute) Computes an array of EPOCH values based on their component parts.
static double[]	**compute**(long[] year, long[] month, long[] day, long[] hour, long[] minute, long[] second) Computes an array of EPOCH values based on their component parts.
static double[]	**compute**(long[] year, long[] month, long[] day, long[] hour, long[] minute, long[] second, long[] msec) Computes an array of EPOCH values based on their component parts.
static double	**compute**(long year, long month, long day, long hour, long minute, long second, long msec) Computes an EPOCH value based on its component parts.
static java.lang.String	**encode**(double epoch) Converts an EPOCH value into a readable date/time string.
static java.lang.String[]	**encode**(double[] epochs) Converts an array of EPOCH values into an array of readable date/time strings.
static java.lang.String	**encode1**(double epoch) Converts an EPOCH value into a readable date/time string.
static java.lang.String[]	**encode1**(double[] epochs) Converts an array EPOCH values into readable date/time strings.
static java.lang.String	**encode2**(double epoch) Converts an EPOCH value into a readable date/time string.
static java.lang.String[]	**encode2**(double[] epochs) Converts an array of EPOCH values into readable date/time strings.
static java.lang.String	**encode3**(double epoch) Converts an EPOCH value into a readable date/time string.
static java.lang.String[]	**encode3**(double[] epochs) Converts an array of EPOCH values into readable date/time strings.
static java.lang.String	**encode4**(double epoch) Converts an EPOCH value into a readable date/time, ISO8601 string.
static java.lang.String[]	**encode4**(double[] epochs) Converts an array of EPOCH values into readable date/time, ISO8601 strings.
static java.lang.String	**encodex**(double epoch, java.lang.String formatString) Converts an EPOCH value into a readable date/time string using the specified format.
static double	**EpochtoUnixTime**(double epoch) Converts an EPOCH value to a unix time of a double value (seconds from 1970-01-01 00:00:00 UTC).
static double[]	**EpochtoUnixTime**(double[] epoch) Converts an array of EPOCH values to an array of unix time of a double value (seconds from 1970-01-01 00:00:00 UTC).
static double	**parse**(java.lang.String inString) This function parses an input date/time string and returns an EPOCH value.
static double[]	**parse**(java.lang.String[] inStrings) This function parses an array of date/time strings and returns an array of EPOCH values.

static double	**parse1**(java.lang.String inString) This function parses an input date/time string and returns an EPOCH value.
static double[]	**parse1**(java.lang.String[] inStrings) This function parses an array of date/time strings and returns an EPOCH value array.
static double	**parse2**(java.lang.String inString) This function parses an input date/time string and returns an EPOCH value.
static double[]	**parse2**(java.lang.String[] inStrings) This function parses an array of date/time strings and returns an EPOCH value array.
static double	**parse3**(java.lang.String inString) This function parses an input date/time string and returns an EPOCH value.
static double[]	**parse3**(java.lang.String[] inStrings) This function parses an array of date/time strings and returns an EPOCH value array.
static double	**parse4**(java.lang.String inString) This function parses an input date/time string and returns an EPOCH value.
static double[]	**parse4**(java.lang.String[] inStrings) This function parses an array of date/time strings and returns an EPOCH value array.
static java.lang.String	**toEncode**(double epoch) Converts an EPOCH value into a readable date/time string.
static java.lang.String[]	**toEncode**(double[] epochs) Converts an array of EPOCH values into readable date/time strings.
static java.lang.String[]	**toEncode**(double[] epochs, int style) Converts an array of EPOCH values into readable date/time strings.
static java.lang.String	**toEncode**(double epoch, int style) Converts an EPOCH value into a readable date/time string.
static java.lang.Object	**toEncode**(java.lang.Object epoch) Converts an EPOCH object into a scalar or array(s) of readable date/time string(s).
static double	**toParse**(java.lang.String inString) This function parses a date/time string and returns an EPOCH value.
static double[]	**toParse**(java.lang.String[] inStrings) This function parses an array of date/time strings and returns an array of EPOCH values.
static double	**UnixTimetoEpoch**(double unixTime) Converts a Unix time value (seconds from 1970-01-01 00:00:00 UTC) to an EPOCH time.
static double[]	**UnixTimetoEpoch**(double[] unixTimes) Converts an array of Unix time values (seconds from 1970-01-01 00:00:00 UTC) to an array of EPOCH times.

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Epoch

```
public Epoch()
```

Method Detail

toParse

```
public static double[] toParse(java.lang.String[] inStrings)
                        throws CDFException
```

This function parses an array of date/time strings and returns an array of EPOCH values. The string format must be exactly as one of the followings (which are made from the corresponding encoding methods).

```
 Style 0 -
 Format:   dd-mmm-yyyy hh:mm:ss.mmm
 Examples:  1-Apr-1990 03:05:02.000
           10-Oct-1993 23:45:49.999

 Style 1 -
 Format:   yyyymmdd.ttttttt
 Examples: 19900401.3658893
           19931010.0000000

 Style 2 -
 Format:   yyyymmddhhmmss
 Examples: 19900401030502
           19931010234549

 Style 3 -
 Format:   yyyy-mm-ddThh:mm:ss.mmmZ
 Examples: 1990-04-01T03:05:02.000Z
           1993-10-10T23:45:49.999Z

 Style 4 -
 Format:   yyyy-mm-ddThh:mm:ss.mmm
 Examples: 1990-04-01T03:05:02.000
           1993-10-10T23:45:49.999
```

Parameters:

`inStrings` - the epochs in string representation

Returns:

the values of the epoch represented by inStrings or null if null or an empty input array is detected

Throws:

`CDFException` - if a bad epoch value is passed in inStrings

toParse

```
public static double toParse(java.lang.String inString)
                      throws CDFException
```

This function parses a date/time string and returns an EPOCH value. The string format must be exactly as one of the followings (which are made from the corresponding encoding methods).

```
 Style 0 -
 Format:   dd-mmm-yyyy hh:mm:ss.mmm
 Examples:  1-Apr-1990 03:05:02.000
           10-Oct-1993 23:45:49.999

 Style 1 -
 Format:   yyyymmdd.ttttttt
 Examples: 19900401.3658893
           19931010.0000000

 Style 2 -
 Format:   yyyymmddhhmmss
 Examples: 19900401030502
           19931010234549
```

```
 Style 3 -
 Format:   yyyy-mm-ddThh:mm:ss.mmmZ
 Examples: 1990-04-01T03:05:02.000Z
           1993-10-10T23:45:49.999Z

 Style 4 -
 Format:   yyyy-mm-ddThh:mm:ss.mmm
 Examples: 1990-04-01T03:05:02.000
           1993-10-10T23:45:49.999
```

Parameters:

`inString` - the epoch in string representation

Returns:

the value of the epoch represented by inStrings or null if null or an empty input array is detected

Throws:

`CDFException` - if a bad epoch value is passed in inStrings

parse

```
public static double parse(java.lang.String inString)
                    throws CDFException
```

This function parses an input date/time string and returns an EPOCH value. The string format must be exactly as shown below. Month abbreviations may be in any case and are always the first three letters of the month.

```
 Format:   dd-mmm-yyyy hh:mm:ss.mmm
 Examples:  1-Apr-1990 03:05:02.000
           10-Oct-1993 23:45:49.999
```

The expected format is the same as that produced by encode.

Parameters:

`inString` - the epoch in string representation

Returns:

the value of the epoch represented by inString

Throws:

`CDFException` - if a bad epoch value is passed in inString

parse

```
public static double[] parse(java.lang.String[] inStrings)
                      throws CDFException
```

This function parses an array of date/time strings and returns an array of EPOCH values. The string format must be exactly as shown below. Month abbreviations may be in any case and are always the first three letters of the month.

```
 Format:   dd-mmm-yyyy hh:mm:ss.mmm
 Examples:  1-Apr-1990 03:05:02.000
           10-Oct-1993 23:45:49.999
```

The expected format is the same as that produced by encode.

Parameters:

`inStrings`

- the epochs in string representation

Returns:

the values of the epoch represented by inStrings or null if null or an empty input array is detected

Throws:

`CDFException` - if a bad epoch value is passed in inStrings

parse1

```
public static double parse1(java.lang.String inString)
                     throws CDFException
```

This function parses an input date/time string and returns an EPOCH value. The format must be exactly as shown below. Note that if there are less than 7 digits after the decimal point, zeros (0's) are assumed for the missing digits.

```
 Format:    yyyymmdd.ttttttt
 Examples:  19950508.0000000
            19671231.58      (== 19671213.5800000)
```

The expected format is the same as that produced by encode1.

Parameters:

`inString` - the epoch in string representation

Returns:

the value of the epoch represented by inString

Throws:

`CDFException` - if a bad epoch value is passed in inString

parse1

```
public static double[] parse1(java.lang.String[] inStrings)
                       throws CDFException
```

This function parses an array of date/time strings and returns an EPOCH value array. The format must be exactly as shown below. Note that if there are less than 7 digits after the decimal point, zeros (0's) are assumed for the missing digits.

```
 Format:    yyyymmdd.ttttttt
 Examples:  19950508.0000000
            19671231.58      (== 19671213.5800000)
```

The expected format is the same as that produced by encode1.

Parameters:

`inStrings` - the epoch array in string representation

Returns:

the value array of the epoch represented by inString

Throws:

`CDFException` - if a bad epoch value is passed in inString

parse2

```
public static double parse2(java.lang.String inString)
                    throws CDFException
```

This function parses an input date/time string and returns an EPOCH value. The format must be exactly as shown below.

```
 Format:   yyyymmddhhmmss
 Examples: 19950508000000
           19671231235959
```

The expected format is the same as that produced by encode2.

Parameters:

`inString` - the epoch in string representation

Returns:

the value of the epoch represented by inString

Throws:

`CDFException` - if a bad epoch value is passed in inString

parse2

```
public static double[] parse2(java.lang.String[] inStrings)
                       throws CDFException
```

This function parses an array of date/time strings and returns an EPOCH value array. The format must be exactly as shown below.

```
 Format:   yyyymmddhhmmss
 Examples: 19950508000000
           19671231235959
```

The expected format is the same as that produced by encode2.

Parameters:

`inStrings` - the epoch array in string representation

Returns:

the value array of the epoch represented by inString

Throws:

`CDFException` - if a bad epoch value is passed in inString

parse3

```
public static double parse3(java.lang.String inString)
                    throws CDFException
```

This function parses an input date/time string and returns an EPOCH value. The format must be exactly as shown below.

```
 Format:    yyyy-mm-ddThh:mm:ss.cccZ
 Examples:  1990-04-01T03:05:02.000Z
            1993-10-10T23:45:49.999Z
```

The expected format is the same as that produced by encode3.

Parameters:

`inString` - the epoch in string representation

Returns:

the value of the epoch represented by inString

Throws:

`CDFException` - if a bad epoch value is passed in inString

parse3

```
public static double[] parse3(java.lang.String[] inStrings)
                       throws CDFException
```

This function parses an array of date/time strings and returns an EPOCH value array. The format must be exactly as shown below.

```
 Format:    yyyy-mm-ddThh:mm:ss.cccZ
 Examples:  1990-04-01T03:05:02.000Z
            1993-10-10T23:45:49.999Z
```

The expected format is the same as that produced by encode3.

Parameters:

`inStrings` - the epoch array in string representation

Returns:

the value array of the epoch represented by inString

Throws:

`CDFException` - if a bad epoch value is passed in inString

parse4

```
public static double parse4(java.lang.String inString)
                     throws CDFException
```

This function parses an input date/time string and returns an EPOCH value. The format must be an ISO8601 and is exactly as shown below.

```
 Format:    yyyy-mm-ddThh:mm:ss.ccc
 Examples:  1990-04-01T03:05:02.000
            1993-10-10T23:45:49.999
```

The expected format is the same as that produced by encode3.

Parameters:

`inString` - the epoch in string representation

Returns:

the value of the epoch represented by inString

Throws:

`CDFException` - if a bad epoch value is passed in inString

parse4

```
public static double[] parse4(java.lang.String[] inStrings)
                       throws CDFException
```

This function parses an array of date/time strings and returns an EPOCH value array. The format must be an ISO8601 and is exactly as shown below.

```
 Format:    yyyy-mm-ddThh:mm:ss.ccc
 Examples:  1990-04-01T03:05:02.000
            1993-10-10T23:45:49.999
```

The expected format is the same as that produced by encode3.

Parameters:

`inStrings` - the epoch array in string representation

Returns:

the value array of the epoch represented by inString

Throws:

`CDFException` - if a bad epoch value is passed in inString

toEncode

```
public static java.lang.String toEncode(double epoch,
                        int style)
```

Converts an EPOCH value into a readable date/time string. Depending on the encoding style, it can present the string to one of the following forms:

```
 For style 0:
 Format:    dd-mmm-yyyy hh:mm:ss.ccc
 Examples:  01-Apr-1990 03:05:02.000
            10-Oct-1993 23:45:49.999

 For style 1:
 Format:    yyyymmdd.ttttttt
 Examples:  19950508.0000000
            19671231.58      (== 19671213.5800000)

 For style 2:
 Format:    yyyymmddhhmmss
 Examples:  19900401030502
            19931010234549

 For style 3:
 Format:    yyyymmddThh:mm:ss.cccZ
 Examples:  19900401T03:05:02.000Z
            19931010T23:45:49.999Z

 For style 4:
 Format:    yyyymmddThh:mm:ss.ccc
 Examples:  19900401T03:05:02.000
            19931010T23:45:49.999
 The default style if the style is invalid.
```

Parameters:

`epoch` - the epoch value

`style` - the encoding style

Returns:

A string representation of the epoch

toEncode

```
public static java.lang.String[] toEncode(double[] epochs,
                                          int style)
```

Converts an array of EPOCH values into readable date/time strings. Depending on the encoding style, it can present the string to one of the following forms:

```
 For style 0:
 Format:    dd-mmm-yyyy hh:mm:ss.ccc
 Examples:  01-Apr-1990 03:05:02.000
            10-Oct-1993 23:45:49.999

 For style 1:
 Format:    yyyymmdd.ttttttt
 Examples:  19950508.0000000
            19671231.58      (== 19671213.5800000)

 For style 2:
 Format:    yyyymmddhhmmss
 Examples:  19900401030502
            19931010234549

 For style 3:
 Format:    yyyymmddThh:mm:ss.cccZ
 Examples:  19900401T03:05:02.000Z
            19931010T23:45:49.999Z

 For style 4:
 Format:    yyyymmddThh:mm:ss.ccc
 Examples:  19900401T03:05:02.000
            19931010T23:45:49.999
```

Parameters:

`epochs` - the epoch values

`style` - the encoding style

Returns:

An array of strings representation of the epochs If the style is invalid, it will return the string "99991231T23:59:59.999"

toEncode

```
public static java.lang.Object toEncode(java.lang.Object epoch)
```

Converts an EPOCH object into a scalar or array(s) of readable date/time string(s). Each string is presented in the following ISO 8601 form:

```
 Format:    yyyymmddThh:mm:ss.ccc
 Examples:  19900401T03:05:02.000
            19931010T23:45:49.999
```

Parameters:

`epoch` - the epoch object

Returns:

A string or array(s) of strings representation of the epoch

toEncode

```
public static java.lang.String toEncode(double epoch)
```

Converts an EPOCH value into a readable date/time string. It present the string to the following ISO 8601 form:

```
Format:    yyyymmddThh:mm:ss.ccc
Examples:  19900401T03:05:02.000
           19931010T23:45:49.999
```

Parameters:

`epoch` - the epoch value

Returns:

A string representation of the epoch

toEncode

```
public static java.lang.String[] toEncode(double[] epochs)
```

Converts an array of EPOCH values into readable date/time strings. The strings are of the following ISO 8601 form:

```
Format:    yyyymmddThh:mm:ss.ccc
Examples:  19900401T03:05:02.000
           19931010T23:45:49.999
```

Parameters:

`epochs` - the epoch values

Returns:

A array of strings representation of the epochs

EpochtoUnixTime

```
public static double EpochtoUnixTime(double epoch)
```

Converts an EPOCH value to a unix time of a double value (seconds from 1970-01-01 00:00:00 UTC). Resolution to milliseconds will be made in unix time at its fraction part.

Parameters:

`epoch` - the epoch value

Returns:

A unix time value

EpochtoUnixTime

```
public static double[] EpochtoUnixTime(double[] epoch)
```

Converts an array of EPOCH values to an array of unix time of a double value (seconds from 1970-01-01 00:00:00 UTC). Resolution to milliseconds will be made in unix time at its fraction part.

Parameters:

`epoch` - the epoch values

Returns:

Array of unix time values

UnixTimetoEpoch

```
public static double UnixTimetoEpoch(double unixTime)
```

Converts a Unix time value (seconds from 1970-01-01 00:00:00 UTC) to an EPOCH time. The Unix time can have a time resoultion of milliseconds (0-999) or microseconds (0-999999) at its fraction part. Only resolution to milliseconds in unix time will be used.

Parameters:

`unixTime` - the unix time value

Returns:

An epoch value

UnixTimetoEpoch

```
public static double[] UnixTimetoEpoch(double[] unixTimes)
```

Converts an array of Unix time values (seconds from 1970-01-01 00:00:00 UTC) to an array of EPOCH times. The Unix time can have a time resoultion of milliseconds (0-999) or microseconds (0-999999) at its fraction part. Only resolution to milliseconds in unix time will be used.

Parameters:

`unixTimes` - the unix time values

Returns:

An array of epoch values

encode

```
public static java.lang.String encode(double epoch)
```

Converts an EPOCH value into a readable date/time string.

```
 Format:    dd-mmm-yyyy hh:mm:ss.ccc
 Examples:  01-Apr-1990 03:05:02.000
            10-Oct-1993 23:45:49.999
```

This format is the same as that expected by parse.

Parameters:

`epoch` - the epoch value

Returns:

A string representation of the epoch

encode

```
public static java.lang.String[] encode(double[] epochs)
```

Converts an array of EPOCH values into an array of readable date/time strings.

```
 Format:    dd-mmm-yyyy hh:mm:ss.ccc
 Examples:  01-Apr-1990 03:05:02.000
            10-Oct-1993 23:45:49.999
```

This format is the same as that expected by parse.

Parameters:

`epochs` - the epoch values

Returns:

Strings representing the epochs or null if null or an empty input array is detected

encode1

```
public static java.lang.String encode1(double epoch)
```

Converts an EPOCH value into a readable date/time string.

```
 Format:    yyyymmdd.ttttttt
 Examples:  19900401.3658893
            19611231.0000000
```

This format is the same as that expected by parse1.

Parameters:

`epoch` - the epoch value

Returns:

A string representation of the epoch

encode1

```
public static java.lang.String[] encode1(double[] epochs)
```

Converts an array EPOCH values into readable date/time strings.

```
 Format:    yyyymmdd.ttttttt
 Examples:  19900401.3658893
            19611231.0000000
```

This format is the same as that expected by parse1.

Parameters:

`epochs` - the epoch values

Returns:

Array of strings representation of the epochs

encode2

```
public static java.lang.String encode2(double epoch)
```

Converts an EPOCH value into a readable date/time string.

```
 Format:     yyyymmddhhmmss
 Examples:   19900401235959
```

```
             19611231000000
```

This format is the same as that expected by parse2.

Parameters:

`epoch` - the epoch value

Returns:

A string representation of the epoch

encode2

```
public static java.lang.String[] encode2(double[] epochs)
```

Converts an array of EPOCH values into readable date/time strings.

```
 Format:     yyyymmddhhmmss
 Examples:   19900401235959
             19611231000000
```

This format is the same as that expected by parse2.

Parameters:

`epochs` - the epoch values

Returns:

A array of strings representation of the epochs

encode3

```
public static java.lang.String encode3(double epoch)
```

Converts an EPOCH value into a readable date/time string.

```
 Format:    yyyy-mm-ddThh:mm:ss.cccZ
 Examples:  1990-04-01T03:05:02.000Z
            1993-10-10T23:45:49.999Z
```

This format is the same as that expected by parse3.

Parameters:

`epoch` - the epoch value

Returns:

A string representation of the epoch

encode3

```
public static java.lang.String[] encode3(double[] epochs)
```

Converts an array of EPOCH values into readable date/time strings.

```
 Format:    yyyy-mm-ddThh:mm:ss.cccZ
 Examples:  1990-04-01T03:05:02.000Z
            1993-10-10T23:45:49.999Z
```

This format is the same as that expected by parse3.

Parameters:

`epochs` - the epoch values

Returns:

A array of strings representation of the epochs

encode4

```
public static java.lang.String encode4(double epoch)
```

Converts an EPOCH value into a readable date/time, ISO8601 string.

```
 Format:    yyyy-mm-ddThh:mm:ss.ccc
 Examples:  1990-04-01T03:05:02.000
            1993-10-10T23:45:49.999
```

This format is the same as that expected by parse3.

Parameters:

`epoch` - the epoch value

Returns:

A string representation of the epoch

encode4

```
public static java.lang.String[] encode4(double[] epochs)
```

Converts an array of EPOCH values into readable date/time, ISO8601 strings.

```
 Format:    yyyy-mm-ddThh:mm:ss.ccc
 Examples:  1990-04-01T03:05:02.000
            1993-10-10T23:45:49.999
```

This format is the same as that expected by parse3.

Parameters:

`epochs` - the epoch values

Returns:

A array of strings representation of the epochs

encodex

```
public static java.lang.String encodex(double epoch,
                        java.lang.String formatString)
```

Converts an EPOCH value into a readable date/time string using the specified format. See the C Reference Manual section 8.7 for details

Parameters:

`epoch` - the epoch value

`formatString` - a string representing the desired format of the epoch

Returns:

A string representation of the epoch according to formatString

compute

```
public static double compute(long year,
             long month,
             long day,
             long hour,
             long minute,
             long second,
             long msec)
                      throws CDFException
```

Computes an EPOCH value based on its component parts.

Parameters:

`year` - the year

`month` - the month

`day` - the day

`hour` - the hour

`minute` - the minute

`second` - the second

`msec` - the millisecond

Returns:

the epoch value

Throws:

`CDFException` - an ILLEGAL_EPOCH_FIELD if an illegal component value is detected.

compute

```
public static double[] compute(long[] year,
               long[] month,
               long[] day)
                        throws CDFException
```

Computes an array of EPOCH values based on their component parts.

Parameters:

`year` - the year array

`month` - the month array

`day` - the day array

Returns:

an array of epoch values or null if an invalid input array(s) is detected.

Throws:

`CDFException` - an ILLEGAL_EPOCH_FIELD if an illegal component value is detected.

compute

```
public static double[] compute(long[] year,
               long[] month,
               long[] day,
               long[] hour)
                        throws CDFException
```

Computes an array of EPOCH values based on their component parts.

Parameters:

`year` - the year array

`month` - the month array

`day` - the day array

`hour` - the hour array

Returns:

an array of epoch values or null if an invalid input array(s) is detected

Throws:

`CDFException` - an ILLEGAL_EPOCH_FIELD if an illegal component value is detected.

compute

```
public static double[] compute(long[] year,
               long[] month,
               long[] day,
               long[] hour,
               long[] minute)
                        throws CDFException
```

Computes an array of EPOCH values based on their component parts.

Parameters:

`year` - the year array

`month` - the month array

`day` - the day array

`hour` - the hour array

`minute` - the minute array

Returns:

an array of epoch values or null if an invalid input array(s) is detected

Throws:

`CDFException` - an ILLEGAL_EPOCH_FIELD if an illegal component value is detected.

compute

```
public static double[] compute(long[] year,
               long[] month,
               long[] day,
               long[] hour,
               long[] minute,
```

```
                long[] second)
                         throws CDFException
```

Computes an array of EPOCH values based on their component parts.

Parameters:

`year` - the year array

`month` - the month array

`day` - the day array

`hour` - the hour array

`minute` - the minute array

`second` - the second array

Returns:

an array of epoch values or null if an invalid input array(s) is detected

Throws:

`CDFException` - an ILLEGAL_EPOCH_FIELD if an illegal component value is detected.

compute

```
public static double[] compute(long[] year,
                long[] month,
                long[] day,
                long[] hour,
                long[] minute,
                long[] second,
                long[] msec)
                         throws CDFException
```

Computes an array of EPOCH values based on their component parts.

Parameters:

`year` - the year array

`month` - the month array

`day` - the day array

`hour` - the hour array

`minute` - the minute array

`second` - the second array

`msec` - the millisecond array

Returns:

an array of epoch values or null if an invalid input array(s) is detected

Throws:

`CDFException` - an ILLEGAL_EPOCH_FIELD if an illegal component value is detected.

breakdown

```
public static long[] breakdown(double epoch)
```

Breaks an EPOCH value down into its component parts.

Parameters:

`epoch` - the epoch value to break down

Returns:

an array containing the epoch parts:

Index	Part
0	year
1	month
2	day
3	hour
4	minute
5	second
6	msec

breakdown

```
public static long[][] breakdown(double[] epochs)
```

Breaks an array of EPOCH values down into its component parts.

Parameters:

`epochs` - the array of epoch values to break down

Returns:

a 2-dimensional array, the first element being the row, while the second element containing the epoch parts, as follwos:

Index	Part
0	year
1	month
2	day
3	hour
4	minute
5	second
6	msec

or null if null or an empty input array is detected

gsfc.nssdc.cdf.util

Class Epoch16

java.lang.Object
gsfc.nssdc.cdf.util.Epoch16

All Implemented Interfaces:

CDFConstants

```
public class Epoch16
extends java.lang.Object
implements CDFConstants
```

This class contains the handy utility routines (methods) called by the CDF applications to handle epoch data of CDF's CDF_EPOCH16 data type.

Version:

1.0 **Example:**

```
// Get the time, down to picoseconds, for Aug 5, 1990 at 5:0:0.0.0.0
double[] epoch16 = new double[2];
double ep = Epoch16.compute(1990, 8, 5, 5, 0, 0, 0, 0, 0, 0, epoch16);
//Get the year, month, day, hour, minutes, seconds, milliseconds,
//    microseconds, nanaseconds and picoseconds for epoch16
long[] times = Epoch16.breakdown(epoch16);
for (int i=0;i<times.length;i++)
    System.out.print(times[i]+" ");
System.out.println();
// Printout the epoch in various formats
System.out.println(Epoch16.encode(epoch16));
System.out.println(Epoch16.encode1(epoch16));
System.out.println(Epoch16.encode2(epoch16));
System.out.println(Epoch16.encode3(epoch16));
System.out.println(Epoch16.encode4(epoch16));
// Print out the date using format
String format = "<month> <dom.02>, <year> at <hour>:<min>";
System.out.println(Epoch16.encodex(epoch16,format));
```

Field Summary

Fields inherited from interface gsfc.nssdc.cdf.CDFConstants

AHUFF_COMPRESSION, ALPHAOSF1_DECODING, ALPHAOSF1_ENCODING, ALPHAVMSd_DECODING, ALPHAVMSd_ENCODING, ALPHAVMSg_DECODING, ALPHAVMSg_ENCODING, ALPHAVMSi_DECODING, ALPHAVMSi_ENCODING, ARM_BIG_DECODING, ARM_BIG_ENCODING, ARM_LITTLE_DECODING, ARM_LITTLE_ENCODING, ATTR, ATTR_EXISTENCE, ATTR_EXISTS, ATTR_MAXgENTRY, ATTR_MAXrENTRY, ATTR_MAXzENTRY, ATTR_NAME, ATTR_NAME_TRUNC, ATTR_NUMBER, ATTR_NUMgENTRIES, ATTR_NUMrENTRIES, ATTR_NUMzENTRIES, ATTR_SCOPE, BACKWARD, BACKWARDFILEoff, BACKWARDFILEon, BAD_ALLOCATE_RECS, BAD_ARGUMENT, BAD_ATTR_NAME, BAD_ATTR_NUM, BAD_BLOCKING_FACTOR, BAD_CACHE_SIZE, BAD_CDF_EXTENSION, BAD_CDF_ID, BAD_CDF_NAME, BAD_CDFSTATUS, BAD_CHECKSUM, BAD_COMPRESSION_PARM, BAD_DATA_TYPE, BAD_DECODING, BAD_DIM_COUNT, BAD_DIM_INDEX, BAD_DIM_INTERVAL, BAD_DIM_SIZE, BAD_ENCODING, BAD_ENTRY_NUM, BAD_FNC_OR_ITEM, BAD_FORMAT, BAD_INITIAL_RECS, BAD_MAJORITY, BAD_MALLOC, BAD_NEGtoPOSfp0_MODE, BAD_NUM_DIMS, BAD_NUM_ELEMS, BAD_NUM_STRINGS, BAD_NUM_VARS, BAD_READONLY_MODE, BAD_REC_COUNT, BAD_REC_INTERVAL, BAD_REC_NUM, BAD_SCOPE, BAD_SCRATCH_DIR, BAD_SPARSEARRAYS_PARM, BAD_VAR_NAME, BAD_VAR_NUM, BAD_zMODE, BADDATE_LEAPSECOND_UPDATED, BeginUnixTimeEPOCH, BeginUnixTimeEPOCH16, BLOCKINGFACTOR_TOO_LARGE, BLOCKINGFACTOR_TOO_SMALL, BLOCKINGFACTOR_TOO_SMALL2, CANNOT_ALLOCATE_RECORDS, CANNOT_CHANGE, CANNOT_COMPRESS, CANNOT_CONVERT_WIDECHAR, CANNOT_COPY, CANNOT_INSERT_RECORDS, CANNOT_SPARSEARRAYS, CANNOT_SPARSERECORDS, CDF, CDF_ACCESS, CDF_ATTR_NAME_LEN, CDF_ATTR_NAME_LEN256, CDF_BYTE, CDF_CACHESIZE, CDF_CHAR, CDF_CHECKSUM, CDF_CLOSE_ERROR, CDF_COMPRESSION, CDF_COPYRIGHT, CDF_COPYRIGHT_LEN, CDF_CREATE_ERROR, CDF_DECODING, CDF_DELETE_ERROR, CDF_DOUBLE, CDF_ENCODING, CDF_EPOCH, CDF_EPOCH16, CDF_EXISTS, CDF_FLOAT, CDF_FORMAT, CDF_INCREMENT, CDF_INFO, CDF_INT1,

CDF_INT2, CDF_INT4, CDF_INT8, CDF_INTERNAL_ERROR, CDF_LEAPSECONDLASTUPDATED , CDF_MAJORITY_,
CDF_MAX_DIMS, CDF_MAX_PARMS, CDF_MIN_DIMS, CDF_NAME_, CDF_NAME_TRUNC, CDF_NEGtoPOSfp0_MODE_,
CDF_NUMATTRS , CDF_NUMgATTRS , CDF_NUMrVARS , CDF_NUMvATTRS , CDF_NUMzVARS , CDF_OK,
CDF_OPEN_ERROR, CDF_PATHNAME_LEN, CDF_READ_ERROR, CDF_READONLY_MODE , CDF_REAL4, CDF_REAL8,
CDF_RELEASE , CDF_SAVE_ERROR, CDF_SCRATCHDIR , CDF_STATUS , CDF_STATUSTEXT_LEN, CDF_TIME_TT2000,
CDF_UCHAR, CDF_UINT1, CDF_UINT2, CDF_UINT4, CDF_VAR_NAME_LEN, CDF_VAR_NAME_LEN256, CDF_VERSION_,
CDF_WARN, CDF_WRITE_ERROR, CDF_zMODE_, CDFwithSTATS_, CHECKSUM_, CHECKSUM_ERROR,
CHECKSUM_NOT_ALLOWED, CLOSE_, COLUMN_MAJOR, COMPRESS_CACHESIZE_, COMPRESSION_ERROR, CONFIRM_,
CORRUPTED_V2_CDF, CORRUPTED_V3_CDF, CREATE_, CURgENTRY_EXISTENCE_, CURrENTRY_EXISTENCE_,
CURzENTRY_EXISTENCE_, DATATYPE_MISMATCH, DATATYPE_SIZE_, DECOMPRESSION_ERROR, DECSTATION_DECODING,
DECSTATION_ENCODING, DEFAULT_BYTE_PADVALUE, DEFAULT_CHAR_PADVALUE, DEFAULT_DOUBLE_PADVALUE,
DEFAULT_EPOCH_PADVALUE, DEFAULT_EPOCH16_PADVALUE, DEFAULT_FLOAT_PADVALUE, DEFAULT_INT1_PADVALUE,
DEFAULT_INT2_PADVALUE, DEFAULT_INT4_PADVALUE, DEFAULT_INT8_PADVALUE, DEFAULT_REAL4_PADVALUE,
DEFAULT_REAL8_PADVALUE, DEFAULT_TT2000_PADVALUE, DEFAULT_UCHAR_PADVALUE, DEFAULT_UINT1_PADVALUE,
DEFAULT_UINT2_PADVALUE, DEFAULT_UINT4_PADVALUE, DELETE_, DID_NOT_COMPRESS, EMPTY_COMPRESSED_CDF,
END_OF_VAR, EPOCH_STRING_LEN, EPOCH_STRING_LEN_EXTEND, EPOCHI_STRING_LEN,
EPOCH1_STRING_LEN_EXTEND, EPOCH2_STRING_LEN, EPOCH2_STRING_LEN_EXTEND, EPOCH3_STRING_LEN,
EPOCH3_STRING_LEN_EXTEND, EPOCH4_STRING_LEN, EPOCH4_STRING_LEN_EXTEND, EPOCHx_FORMAT_MAX,
EPOCHx_STRING_MAX, FILLED_TT2000_VALUE, FORCED_PARAMETER, FUNCTION_NOT_SUPPORTED, gENTRY_,
gENTRY_DATA_, gENTRY_DATASPEC_, gENTRY_DATATYPE_, gENTRY_EXISTENCE_, gENTRY_NUMELEMS_, GET_,
GETCDFCHECKSUM_, GETCDFFILEBACKWARD_, GETCDFVALIDATE_, GETLEAPSECONDSENVVAR_, GLOBAL_SCOPE,
GZIP_COMPRESSION, HOST_DECODING, HOST_ENCODING, HP_DECODING, HP_ENCODING, HUFF_COMPRESSION,
IA64VMSd_DECODING, IA64VMSd_ENCODING, IA64VMSg_DECODING, IA64VMSg_ENCODING, IA64VMSi_DECODING,
IA64VMSi_ENCODING, IBM_PC_OVERFLOW, IBMPC_DECODING, IBMPC_ENCODING, IBMRS_DECODING,
IBMRS_ENCODING, ILLEGAL_EPOCH_FIELD, ILLEGAL_EPOCH_VALUE, ILLEGAL_FOR_SCOPE, ILLEGAL_IN_zMODE,
ILLEGAL_ON_V1_CDF, ILLEGAL_TT2000_VALUE, IS_A_NETCDF, LIB_COPYRIGHT_, LIB_INCREMENT_,
LIB_RELEASE_, LIB_subINCREMENT_, LIB_VERSION_, MAC_DECODING, MAC_ENCODING, MD5_CHECKSUM,
MULTI_FILE, MULTI_FILE_FORMAT, NA_FOR_VARIABLE, NEGATIVE_FP_ZERO, NEGtoPOSfp0off, NEGtoPOSfp0on,
NETWORK_DECODING, NETWORK_ENCODING, NeXT_DECODING, NeXT_ENCODING, NO_ATTR_SELECTED,
NO_CDF_SELECTED, NO_CHECKSUM, NO_COMPRESSION, NO_DELETE_ACCESS, NO_ENTRY_SELECTED, NO_MORE_ACCESS,
NO_PADVALUE_SPECIFIED, NO_SPARSEARRAYS, NO_SPARSERECORDS, NO_STATUS_SELECTED, NO_SUCH_ATTR,
NO_SUCH_CDF, NO_SUCH_ENTRY, NO_SUCH_RECORD, NO_SUCH_VAR, NO_VAR_SELECTED, NO_VARS_IN_CDF,
NO_WRITE_ACCESS, NONE_CHECKSUM, NOT_A_CDF, NOT_A_CDF_OR_NOT_SUPPORTED, NOVARY, NULL_, OPEN_,
OPTIMAL_ENCODING_TREES, OTHER_CHECKSUM, PAD_SPARSERECORDS, PPC_DECODING, PPC_ENCODING,
PRECEEDING_RECORDS_ALLOCATED, PREV_SPARSERECORDS, PUT_, READ_ONLY_DISTRIBUTION, READ_ONLY_MODE,
READONLYoff, READONLYon, rENTRY_, rENTRY_DATA_, rENTRY_DATASPEC_, rENTRY_DATATYPE_,
rENTRY_EXISTENCE_, rENTRY_NAME_, rENTRY_NUMELEMS_, rENTRY_NUMSTRINGS_, rENTRY_STRINGSDATA_,
RLE_COMPRESSION, RLE_OF_ZEROs, ROW_MAJOR, rVAR_, rVAR_ALLOCATEBLOCK_, rVAR_ALLOCATEDFROM_,
rVAR_ALLOCATEDTO_, rVAR_ALLOCATERECS_, rVAR_BLOCKINGFACTOR_, rVAR_CACHESIZE_, rVAR_COMPRESSION_,
rVAR_DATA_, rVAR_DATASPEC_, rVAR_DATATYPE_, rVAR_DIMVARYS_, rVAR_EXISTENCE_, rVAR_HYPERDATA_,
rVAR_INITIALRECS_, rVAR_MAXallocREC_, rVAR_MAXREC_, rVAR_NAME_, rVAR_nINDEXENTRIES_,
rVAR_nINDEXLEVELS_, rVAR_nINDEXRECORDS_, rVAR_NUMallocRECS_, rVAR_NUMBER_, rVAR_NUMELEMS_,
rVAR_NUMRECS_, rVAR_PADVALUE_, rVAR_RECORDS_, rVAR_RECORDS_RENUMBER_, rVAR_RECVARY_,
rVAR_RESERVEPERCENT_, rVAR_SEQDATA_, rVAR_SEQPOS_, rVAR_SPARSEARRAYS_, rVAR_SPARSERECORDS_,
rVARs_CACHESIZE_, rVARs_DIMCOUNTS_, rVARs_DIMINDICES_, rVARs_DIMINTERVALS_, rVARs_DIMSIZES_,
rVARs_MAXREC_, rVARs_NUMDIMS_, rVARs_RECCOUNT_, rVARs_RECDATA_, rVARs_RECINTERVAL_,
rVARs_RECNUMBER_, SAVE_, SCRATCH_CREATE_ERROR, SCRATCH_DELETE_ERROR, SCRATCH_READ_ERROR,
SCRATCH_WRITE_ERROR, SELECT_, SGI_DECODING, SGi_ENCODING, SINGLE_FILE, SINGLE_FILE_FORMAT,
SOME_ALREADY_ALLOCATED, STAGE_CACHESIZE_, STATUS_TEXT_, STRING_NOT_UTF8_ENCODING, STRINGDELIMITER,
SUN_DECODING, SUN_ENCODING, TOO_MANY_PARMS, TOO_MANY_VARS, TRY_TO_READ_NONSTRING_DATA,
TT2000_0_STRING_LEN, TT2000_1_STRING_LEN, TT2000_2_STRING_LEN, TT2000_3_STRING_LEN,
TT2000_4_STRING_LEN, TT2000_CDF_MAYNEEDUPDATE, TT2000_TIME_ERROR, TT2000_USED_OUTDATED_TABLE,
UNABLE_TO_PROCESS_CDF, UNKNOWN_COMPRESSION, UNKNOWN_SPARSENESS, UNSUPPORTED_OPERATION, VALIDATE_,
VALIDATEFILEoff, VALIDATEFILEon, VAR_ALREADY_CLOSED, VAR_CLOSE_ERROR, VAR_CREATE_ERROR,
VAR_DELETE_ERROR, VAR_EXISTS, VAR_NAME_TRUNC, VAR_OPEN_ERROR, VAR_READ_ERROR, VAR_SAVE_ERROR,
VAR_WRITE_ERROR, VARIABLE_SCOPE, VARY, VAX_DECODING, VAX_ENCODING, VIRTUAL_RECORD_DATA, zENTRY_,
zENTRY_DATA_, zENTRY_DATASPEC_, zENTRY_DATATYPE_, zENTRY_EXISTENCE_, zENTRY_NAME_,
zENTRY_NUMELEMS_, zENTRY_NUMSTRINGS_, zENTRY_STRINGSDATA_, ZLIB_COMPRESS_ERROR,
ZLIB_UNCOMPRESS_ERROR, zMODEoff, zMODEon1, zMODEon2, zVAR_, zVAR_ALLOCATEBLOCK_,
zVAR_ALLOCATEDFROM_, zVAR_ALLOCATEDTO_, zVAR_ALLOCATERECS_, zVAR_BLOCKINGFACTOR_, zVAR_CACHESIZE_,
zVAR_COMPRESSION_, zVAR_DATA_, zVAR_DATASPEC_, zVAR_DATATYPE_, zVAR_DIMCOUNTS_, zVAR_DIMINDICES_,
zVAR_DIMINTERVALS_, zVAR_DIMSIZES_, zVAR_DIMVARYS_, zVAR_EXISTENCE_, zVAR_HYPERDATA_,
zVAR_INITIALRECS_, zVAR_MAXallocREC_, zVAR_MAXREC_, zVAR_NAME_, zVAR_nINDEXENTRIES_,
zVAR_nINDEXLEVELS_, zVAR_nINDEXRECORDS_, zVAR_NUMallocRECS_, zVAR_NUMBER_, zVAR_NUMDIMS_,
zVAR_NUMELEMS_, zVAR_NUMRECS_, zVAR_PADVALUE_, zVAR_RECCOUNT_, zVAR_RECINTERVAL_, zVAR_RECNUMBER_,
zVAR_RECORDS_, zVAR_RECORDS_RENUMBER_, zVAR_RECVARY_, zVAR_RESERVEPERCENT_, zVAR_SEQDATA_,
zVAR_SEQPOS_, zVAR_SPARSEARRAYS_, zVAR_SPARSERECORDS_, zVARs_CACHESIZE_, zVARs_MAXREC_,
zVARs_RECDATA_, zVARs_RECNUMBER_

Constructor Summary

Constructors

Constructor and Description
Epoch16()

Method Summary

Methods

Modifier and Type	Method and Description
static long[]	**breakdown**(java.lang.Object epoch) Breaks an EPOCH16 value down into its component parts.
static double	**compute**(long year, long month, long day, long hour, long minute, long second, long msec, long usec, long nsec, long psec, java.lang.Object epoch) Computes an EPOCH16 value based on its component parts.
static java.lang.String	**encode**(java.lang.Object epoch) Converts an EPOCH16 value into a readable date/time string.
static java.lang.String	**encode1**(java.lang.Object epoch) Converts an EPOCH16 value into a readable date/time string.
static java.lang.String	**encode2**(java.lang.Object epoch) Converts an EPOCH16 value into a readable date/time string.
static java.lang.String	**encode3**(java.lang.Object epoch) Converts an EPOCH16 value into a readable date/time string.
static java.lang.String	**encode4**(java.lang.Object epoch) Converts an EPOCH16 value into a readable date/time, ISO8601 string.
static java.lang.String	**encodex**(java.lang.Object epoch, java.lang.String formatString) Converts an EPOCH16 value into a readable date/time string using the specified format.
static double	**Epoch16toUnixTime**(double[] epoch) Converts an EPOCH16 value to a unix time of a double value (seconds from 1970-01-01 00:00:00 UTC).
static java.lang.Object	**parse**(java.lang.String inString) This function parses an input date/time string and returns an EPOCH16 value.
static java.lang.Object	**parse1**(java.lang.String inString) This function parses an input date/time string and returns an EPOCH16 value.
static java.lang.Object	**parse2**(java.lang.String inString) This function parses an input date/time string and returns an EPOCH16 value.
static java.lang.Object	**parse3**(java.lang.String inString) This function parses an input date/time string and returns an EPOCH16 value.
static java.lang.Object	**parse4**(java.lang.String inString) This function parses an input date/time, ISO8601 string and returns an EPOCH16 value.
static java.lang.String	**toEncode**(double[] epoch) Converts an EPOCH object into a scalar or array of readable date/time string.
static java.lang.String	**toEncode**(double[] epoch, int style) Converts an EPOCH16 value into a readable date/time string.
static java.lang.Object	**toEncode**(java.lang.Object epoch) Converts an EPOCH object into a scalar or array of readable date/time string.
static java.lang.Object	**toEncode**(java.lang.Object epoch, long size) Converts an EPOCH object into a scalar or array(s) of readable date/time string.
static double[]	**toParse**(java.lang.String inString) This function parses a date/time string and returns an EPOCH16 value.
static double[]	**UnixTimetoEpoch16**(double unixTime) Converts a unix time of a double value (seconds from 1970-01-01 00:00:00 UTC) to an EPOCH16 value.

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Epoch16

```
public Epoch16()
```

Method Detail

toParse

```
public static double[] toParse(java.lang.String inString)
                        throws CDFException
```

This function parses a date/time string and returns an EPOCH16 value. The string format must be exactly as one of the followings (which are made from the corresponding encoding methods).

```
 Style 0 -
 Format:   dd-mmm-yyyy hh:mm:ss.mmm.uuu.nnn.ppp
 Examples:  1-Apr-1990 03:05:02.000.000.000.000
           10-Oct-1993 23:45:49.999.999.999.999

 Style 1 -
 Format:   yyyymmdd.tttttttttttttt
 Examples: 19900401.365889312345678
           19931010.000000000000000

 Style 2 -
 Format:   yyyymmddhhmmss
 Examples: 19900401030502
           19931010234549

 Style 3 -
 Format:   yyyy-mm-ddThh:mm:ss.mmm.uuu.nnn.pppZ
 Examples: 1990-04-01T03:05:02.000.000.000.000Z
           1993-10-10T23:45:49.999.999.999.999Z

 Style 4 -
 Format:   yyyy-mm-ddThh:mm:ss.mmmuuunnnppp
 Examples: 1990-04-01T03:05:02.000000000000
           1993-10-10T23:45:49.999999999999
```

Parameters:

`inString` - the epoch in string representation

Returns:

the value of the epoch represented by inString or null if null or an empty string is detected

Throws:

`CDFException` - if a bad epoch value is passed in inStrings

parse

```
public static java.lang.Object parse(java.lang.String inString)
                              throws CDFException
```

This function parses an input date/time string and returns an EPOCH16 value. The format must be exactly as shown below. Month abbreviations may be in any case and are always the first three letters of the month.

```
 Format:          dd-mmm-yyyy hh:mm:ss.ccc.mmm.nnn.ppp
 Examples:         1-Apr-1990 03:05:02.000.000.000.000
                  10-Oct-1993 23:45:49.999.999.999.999
```

The expected format is the same as that produced by encode.

Parameters:

`inString` - the epoch in string representation

Returns:

the value of the epoch represented by inString

Throws:

`CDFException` - if a bad epoch value is passed in inString

parse1

```
public static java.lang.Object parse1(java.lang.String inString)
                               throws CDFException
```

This function parses an input date/time string and returns an EPOCH16 value. The format must be exactly as shown below. Note that if there are less than 15 digits after the decimal point, zeros (0's) are assumed for the missing digits.

```
 Format:            yyyymmdd.ttttttttttttttt
 Examples:          19950508.000000000000000
                    19671231.58      (== 19671213.580000000000000)
```

The expected format is the same as that produced by encode1.

Parameters:

`inString` - the epoch in string representation

Returns:

the value of the epoch represented by inString

Throws:

`CDFException` - if a bad epoch value is passed in inString

parse2

```
public static java.lang.Object parse2(java.lang.String inString)
                               throws CDFException
```

This function parses an input date/time string and returns an EPOCH16 value. The format must be exactly as shown below.

```
 Format:           yyyymmddhhmmss
 Examples:         19950508000000
                   19671231235959
```

The expected format is the same as that produced by encode2.

Parameters:

`inString` - the epoch in string representation

Returns:

the value of the epoch represented by inString

Throws:

CDFException - if a bad epoch value is passed in inString

parse3

```
public static java.lang.Object parse3(java.lang.String inString)
                               throws CDFException
```

This function parses an input date/time string and returns an EPOCH16 value. The format must be exactly as shown below.

```
 Format:             yyyy-mm-ddThh:mm:ss.ccc.mmm.nnn.pppZ
 Examples:           1990-04-01T03:05:02.000.000.000.000Z
                     1993-10-10T23:45:49.999.999.999.999Z
```

The expected format is the same as that produced by encode3.

Parameters:

inString - the epoch in string representation

Returns:

the value of the epoch represented by inString

Throws:

CDFException - if a bad epoch value is passed in inString

parse4

```
public static java.lang.Object parse4(java.lang.String inString)
                               throws CDFException
```

This function parses an input date/time, ISO8601 string and returns an EPOCH16 value. The format must be exactly as shown below.

```
 Format:             yyyy-mm-ddThh:mm:ss.cccmmmnnnppp
 Examples:           1990-04-01T03:05:02.000000000000
                     1993-10-10T23:45:49.999999999999
```

The expected format is the same as that produced by encode4.

Parameters:

inString - the epoch in string representation

Returns:

the value of the epoch represented by inString

Throws:

CDFException - if a bad epoch value is passed in inString

toEncode

```
public static java.lang.String toEncode(double[] epoch,
                         int style)
```

Converts an EPOCH16 value into a readable date/time string. Depending on the encoding style, it can present the string to one of the following forms:

```
 For style 0:
 Format:    dd-mmm-yyyy hh:mm:ss.ccc.uuu.nnn.ppp
 Examples:  01-Apr-1990 03:05:02.000.111.222.333
            10-Oct-1993 23:45:49.999.999.999.999

 For style 1:
 Format:    yyyymmdd.ttttttt
 Examples:  19950508.0000000
            19671231.58      (== 19671213.5800000)

 For style 2:
 Format:    yyyymmddhhmmss
 Examples:  19900401030502
            19931010234549

 For style 3:
 Format:    yyyymmddThh:mm:ss.ccc.uuu.nnn.pppZ
 Examples:  19900401T03:05:02.000.111.222.333Z
            19931010T23:45:49.999.999.999.999Z

 For style 4:
 Format:    yyyymmddThh:mm:ss.cccuuunnnppp
 Examples:  19900401T03:05:02.000111222333
            19931010T23:45:49.999999999999
 The default style, if the style is invalid.
```

Parameters:

`epoch` - the epoch value

`style` - the encoding style

Returns:

A string representation of the epoch

toEncode

```
public static java.lang.Object toEncode(java.lang.Object epoch)
```

Converts an EPOCH object into a scalar or array of readable date/time string. Each string is presented in the following ISO 8601 form:

```
 Format:    yyyymmddThh:mm:ss.cccuuunnnppp
 Examples:  19900401T03:05:02.000111222333
            19931010T23:45:49.999999999999
```

Parameters:

`epoch` - the epoch object

Returns:

A string or array of strings representation of the epoch

toEncode

```
public static java.lang.String toEncode(double[] epoch)
```

Converts an EPOCH object into a scalar or array of readable date/time string. Each string is presented in the following ISO 8601 form:

```
 Format:    yyyymmddThh:mm:ss.cccuuunnnppp
 Examples:  19900401T03:05:02.000111222333
```

```
            19931010T23:45:49.999999999999
```

Parameters:

`epoch` - the epoch object

Returns:

A string or array of strings representation of the epoch

toEncode

```
public static java.lang.Object toEncode(java.lang.Object epoch,
                        long size)
```

Converts an EPOCH object into a scalar or array(s) of readable date/time string. Each string is presented in the following ISO 8601 form:

```
 Format:    yyyymmddThh:mm:ss.cccuuunnnppp
 Examples:  19900401T03:05:02.000111222333
            19931010T23:45:49.999999999999
```

Parameters:

`epoch` - the epoch object

`size` - the number of CDF_EPOCH16 data

Returns:

A string or array(s) of strings representation of the epoch

Epoch16toUnixTime

```
public static double Epoch16toUnixTime(double[] epoch)
```

Converts an EPOCH16 value to a unix time of a double value (seconds from 1970-01-01 00:00:00 UTC). The unix time will have a time resolution of microseconds at its fraction part. Note: As EPOCH16 have a higher time resolution than the unix time. The conversion might not precisely preserve the microseconds due to the floating point presentation.

Parameters:

`epoch` - the epoch16 value (2-double)

Returns:

A unix time value

UnixTimetoEpoch16

```
public static double[] UnixTimetoEpoch16(double unixTime)
```

Converts a unix time of a double value (seconds from 1970-01-01 00:00:00 UTC) to an EPOCH16 value. The unix time can have a time resolution in microseconds at its fraction part. Note: As EPOCH16 have a higher time resolution than the unix time. The conversion might not precisely preserve the microseconds due to the floating point presentation.

Parameters:

`unixTime` - the unix time value

Returns:

An EPOCH16 value (2-double)

encode

```
public static java.lang.String encode(java.lang.Object epoch)
```

Converts an EPOCH16 value into a readable date/time string.

```
 Format:            dd-mmm-yyyy hh:mm:ss.ccc.mmm.nnn.ppp
 Examples:          01-Apr-1990 03:05:02.000.000.000.000
                    10-Oct-1993 23:45:49.999.999.999.999
```

This format is the same as that expected by parse.

Parameters:

`epoch` - the epoch value

Returns:

A string representation of the epoch

encode1

```
public static java.lang.String encode1(java.lang.Object epoch)
```

Converts an EPOCH16 value into a readable date/time string.

```
 Format:            yyyymmdd.ttttttttttttttt
 Examples:          19900401.365889312341234
                    19611231.000000000000000
```

This format is the same as that expected by parse1.

Parameters:

`epoch` - the epoch value

Returns:

A string representation of the epoch

encode2

```
public static java.lang.String encode2(java.lang.Object epoch)
```

Converts an EPOCH16 value into a readable date/time string.

```
 Format:          yyyymmddhhmmss
 Examples:        19900401235959
                  19611231000000
```

This format is the same as that expected by parse2.

Parameters:

`epoch` - the epoch value

Returns:

A string representation of the epoch

encode3

```
public static java.lang.String encode3(java.lang.Object epoch)
```

Converts an EPOCH16 value into a readable date/time string.

```
 Format:             yyyy-mm-ddThh:mm:ss.ccc.mmm.nnn.pppZ
 Examples:           1990-04-01T03:05:02.000.000.000.000Z
                     1993-10-10T23:45:49.999.999.999.999Z
```

This format is the same as that expected by parse3.

Parameters:

`epoch` - the epoch value

Returns:

A string representation of the epoch

encode4

```
public static java.lang.String encode4(java.lang.Object epoch)
```

Converts an EPOCH16 value into a readable date/time, ISO8601 string.

```
 Format:             yyyy-mm-ddThh:mm:ss.cccmmmnnnppp
 Examples:           1990-04-01T03:05:02.000000000000
                     1993-10-10T23:45:49.999999999999
```

This format is the same as that expected by parse4.

Parameters:

`epoch` - the epoch value

Returns:

A string representation of the epoch

encodex

```
public static java.lang.String encodex(java.lang.Object epoch,
                       java.lang.String formatString)
```

Converts an EPOCH16 value into a readable date/time string using the specified format. See the C Reference Manual section 8.7 for details

Parameters:

`epoch` - the epoch value

`formatString` - a string representing the desired format of the epoch

Returns:

A string representation of the epoch according to formatString

compute

```
public static double compute(long year,
             long month,
             long day,
             long hour,
             long minute,
             long second,
             long msec,
             long usec,
             long nsec,
             long psec,
             java.lang.Object epoch)
                      throws CDFException
```

Computes an EPOCH16 value based on its component parts.

Parameters:

`year` - the year

`month` - the month

`day` - the day

`hour` - the hour

`minute` - the minute

`second` - the second

`msec` - the milliseconds

`usec` - the microseconds

`nsec` - the nanoseconds

`psec` - the picoseconds

`epoch` - returned epoch value

Returns:

a double value which indicates whether the operation is successful

Throws:

`CDFException` - an ILLEGAL_EPOCH_FIELD if an illegal component value is detected.

breakdown

```
public static long[] breakdown(java.lang.Object epoch)
```

Breaks an EPOCH16 value down into its component parts.

Parameters:

`epoch` - the epoch value to break down

Returns:

an array containing the epoch parts:

Index	Part
0	year
1	month
2	day
3	hour

4	minute
5	second
6	msec
7	usec
8	nsec
9	psec

gsfc.nssdc.cdf.util

Class EpochNative

java.lang.Object
    gsfc.nssdc.cdf.util.EpochNative

```
public class EpochNative
extends java.lang.Object
```

The Epoch class is a Java wrapper to the CDF epoch handling routines. See Chapter 8 of the CDF C Reference Manual Version 2.6 for details **Example:**

```
// Get the milliseconds to Aug 5, 1990 at 5:00
double ep = Epoch.compute(1990, 8, 5, 5, 0, 0, 0);
//Get the year, month, day, hour, minutes, seconds, milliseconds for ep
long times[] = Epoch.breakdown(ep);
for (int i=0;i<times.length;i++)
    System.out.print(times[i]+" ");
System.out.println();
// Printout the epoch in various formats
System.out.println(Epoch.encode(ep));
System.out.println(Epoch.encode1(ep));
System.out.println(Epoch.encode2(ep));
System.out.println(Epoch.encode3(ep));
// Print out the date using format
String format = "<month> <dom.02>, <year> at <hour>:<min>";
System.out.println(Epoch.encodex(ep,format));
```

Constructor Summary

Constructors

Constructor and Description
EpochNative()

Method Summary

Methods

Modifier and Type	Method and Description
static long[]	**breakdown**(double epoch) Mirrors EPOCHbreakdown from the CDF library.
static double	**compute**(long year, long month, long day, long hour, long minute, long second, long msec) Mirrors computeEPOCH from the CDF library.
static java.lang.String	**encode**(double epoch) Mirrors encodeEPOCH from the CDF library.
static java.lang.String	**encode1**(double epoch) Mirrors encodeEPOCH1 from the CDF library.

static java.lang.String	**encode2**(double epoch) Mirrors encodeEPOCH2 from the CDF library.
static java.lang.String	**encode3**(double epoch) Mirrors encodeEPOCH3 from the CDF library.
static java.lang.String	**encode4**(double epoch) Mirrors encodeEPOCH4 from the CDF library.
static java.lang.String	**encodex**(double epoch, java.lang.String format) Mirrors encodeEPOCHx from the CDF library.
static double	**parse**(java.lang.String sEpoch) Mirrors parseEPOCH from CDF library.
static double	**parse1**(java.lang.String sEpoch) Mirrors parseEPOCH from CDF library.
static double	**parse2**(java.lang.String sEpoch) Mirrors parseEPOCH from CDF library.
static double	**parse3**(java.lang.String sEpoch) Mirrors parseEPOCH from CDF library.
static double	**parse4**(java.lang.String sEpoch) Mirrors parseEPOCH from CDF library.

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

EpochNative

```
public EpochNative()
```

Method Detail

compute

```
public static double compute(long year,
             long month,
             long day,
             long hour,
             long minute,
             long second,
             long msec)
```

Mirrors computeEPOCH from the CDF library. See Section 8.1 of the CDF C Reference Manual Version 2.6 for details

Parameters:

`year` - the year componment

`month` - the month componment

`day` - the day componment

`hour` - the hour componment

`minute` - the minute componment

`second` - the second componment

`msec` - the millisecond componment

Returns:

a epoch value

breakdown

```
public static long[] breakdown(double epoch)
```

Mirrors EPOCHbreakdown from the CDF library. See Section 8.2 of the CDF C Reference Manual Version 2.6 for details

Parameters:

`epoch` - the epoch, milliseconds since 0AD

Returns:

the day-time componments

encode

```
public static java.lang.String encode(double epoch)
```

Mirrors encodeEPOCH from the CDF library. See Section 8.3 of the CDF C Reference Manual Version 2.6 for details

Parameters:

`epoch` - the epoch, milliseconds since 0AD

Returns:

the epoch in UTC string

encode1

```
public static java.lang.String encode1(double epoch)
```

Mirrors encodeEPOCH1 from the CDF library. See Section 8.4 of the CDF C Reference Manual Version 2.6 for details

Parameters:

`epoch` - the epoch, milliseconds since 0AD

Returns:

the epoch in UTC string

encode2

```
public static java.lang.String encode2(double epoch)
```

Mirrors encodeEPOCH2 from the CDF library. See Section 8.5 of the CDF C Reference Manual Version 2.6 for details

Parameters:

`epoch`

- the epoch, milliseconds since 0AD

Returns:

the epoch in UTC string

encode3

```
public static java.lang.String encode3(double epoch)
```

Mirrors encodeEPOCH3 from the CDF library. See Section 8.6 of the CDF C Reference Manual Version 2.6 for details

Parameters:

`epoch` - the epoch, milliseconds since 0AD

Returns:

the epoch in UTC string

encode4

```
public static java.lang.String encode4(double epoch)
```

Mirrors encodeEPOCH4 from the CDF library. See Section 8.6 of the CDF C Reference Manual Version 2.6 for details

Parameters:

`epoch` - the epoch, milliseconds since 0AD

Returns:

the epoch in UTC string

encodex

```
public static java.lang.String encodex(double epoch,
                       java.lang.String format)
```

Mirrors encodeEPOCHx from the CDF library. See Section 8.7 of the CDF C Reference Manual Version 2.6 for details

Parameters:

`epoch` - the epoch, milliseconds since 0AD

`format` - the format to encode the string

Returns:

the epoch in UTC string

parse

```
public static double parse(java.lang.String sEpoch)
```

Mirrors parseEPOCH from CDF library. See Section 8.8 of the CDF C Reference Manual Version 2.6 for details

Parameters:

`sEpoch` - the epoch string

Returns:

the epoch

parse1

```
public static double parse1(java.lang.String sEpoch)
```

Mirrors parseEPOCH from CDF library. See Section 8.9 of the CDF C Reference Manual Version 2.6 for details

Parameters:

`sEpoch` - the epoch string

Returns:

the epoch

parse2

```
public static double parse2(java.lang.String sEpoch)
```

Mirrors parseEPOCH from CDF library. See Section 8.10 of the CDF C Reference Manual Version 2.6 for details

Parameters:

`sEpoch` - the epoch string

Returns:

the epoch

parse3

```
public static double parse3(java.lang.String sEpoch)
```

Mirrors parseEPOCH from CDF library. See Section 8.11 of the CDF C Reference Manual Version 2.6 for details

Parameters:

`sEpoch` - the epoch string

Returns:

the epoch

parse4

```
public static double parse4(java.lang.String sEpoch)
```

Mirrors parseEPOCH from CDF library. See Section 8.11 of the CDF C Reference Manual Version 2.6 for details

Parameters:

`sEpoch` - the epoch string

Returns:

the epoch

gsfc.nssdc.cdf

Class Variable

java.lang.Object
gsfc.nssdc.cdf.Variable

All Implemented Interfaces:

CDFConstants, CDFObject

```
public class Variable
extends java.lang.Object
implements CDFObject, CDFConstants
```

The **Variable** class defines a CDF variable.

Notes: Since the CDF JavaAPI always uses `zMODE = 2`, all variables are by default, zVariables.

Version:

1.0, 2.0 03/18/05 Selection of current CDF and variable are done as part of operations passed to JNI. JNI call is synchronized so only one process is allowed in a JVM, due to multi-thread safety. The select method will never be called.

See Also:

Attribute, Entry

Field Summary

Fields inherited from interface gsfc.nssdc.cdf.CDFConstants

AHUFF_COMPRESSION, ALPHAOSF1_DECODING, ALPHAOSF1_ENCODING, ALPHAVMSd_DECODING, ALPHAVMSd_ENCODING, ALPHAVMSg_DECODING, ALPHAVMSg_ENCODING, ALPHAVMSi_DECODING, ALPHAVMSi_ENCODING, ARM_BIG_DECODING, ARM_BIG_ENCODING, ARM_LITTLE_DECODING, ARM_LITTLE_ENCODING, ATTR_, ATTR_EXISTENCE_, ATTR_EXISTS, ATTR_MAXgENTRY_, ATTR_MAXrENTRY_, ATTR_MAXzENTRY_, ATTR_NAME_, ATTR_NAME_TRUNC, ATTR_NUMBER_, ATTR_NUMgENTRIES_, ATTR_NUMrENTRIES_, ATTR_NUMzENTRIES_, ATTR_SCOPE_, BACKWARD_, BACKWARDFILEoff, BACKWARDFILEon, BAD_ALLOCATE_RECS, BAD_ARGUMENT, BAD_ATTR_NAME, BAD_ATTR_NUM, BAD_BLOCKING_FACTOR, BAD_CACHE_SIZE, BAD_CDF_EXTENSION, BAD_CDF_ID, BAD_CDF_NAME, BAD_CDFSTATUS, BAD_CHECKSUM, BAD_COMPRESSION_PARM, BAD_DATA_TYPE, BAD_DECODING, BAD_DIM_COUNT, BAD_DIM_INDEX, BAD_DIM_INTERVAL, BAD_DIM_SIZE, BAD_ENCODING, BAD_ENTRY_NUM, BAD_FNC_OR_ITEM, BAD_FORMAT, BAD_INITIAL_RECS, BAD_MAJORITY, BAD_MALLOC, BAD_NEGtoPOSfp0_MODE, BAD_NUM_DIMS, BAD_NUM_ELEMS, BAD_NUM_STRINGS, BAD_NUM_VARS, BAD_READONLY_MODE, BAD_REC_COUNT, BAD_REC_INTERVAL, BAD_REC_NUM, BAD_SCOPE, BAD_SCRATCH_DIR, BAD_SPARSEARRAYS_PARM, BAD_VAR_NAME, BAD_VAR_NUM, BAD_zMODE, BADDATE_LEAPSECOND_UPDATED, BeginUnixTimeEPOCH, BeginUnixTimeEPOCH16, BLOCKINGFACTOR_TOO_LARGE, BLOCKINGFACTOR_TOO_SMALL, BLOCKINGFACTOR_TOO_SMALL2, CANNOT_ALLOCATE_RECORDS, CANNOT_CHANGE, CANNOT_COMPRESS, CANNOT_CONVERT_WIDECHAR, CANNOT_COPY, CANNOT_INSERT_RECORDS, CANNOT_SPARSEARRAYS, CANNOT_SPARSERECORDS, CDF_, CDF_ACCESS_, CDF_ATTR_NAME_LEN, CDF_ATTR_NAME_LEN256, CDF_BYTE, CDF_CACHESIZE_, CDF_CHAR, CDF_CHECKSUM_, CDF_CLOSE_ERROR, CDF_COMPRESSION_, CDF_COPYRIGHT_, CDF_COPYRIGHT_LEN, CDF_CREATE_ERROR, CDF_DECODING_, CDF_DELETE_ERROR, CDF_DOUBLE, CDF_ENCODING_, CDF_EPOCH, CDF_EPOCH16, CDF_EXISTS, CDF_FLOAT, CDF_FORMAT_, CDF_INCREMENT_, CDF_INFO_, CDF_INT1, CDF_INT2, CDF_INT4, CDF_INT8, CDF_INTERNAL_ERROR, CDF_LEAPSECONDLASTUPDATED_, CDF_MAJORITY_, CDF_MAX_DIMS, CDF_MAX_PARMS, CDF_MIN_DIMS, CDF_NAME_, CDF_NAME_TRUNC, CDF_NEGtoPOSfp0_MODE_, CDF_NUMATTRS_, CDF_NUMgATTRS_, CDF_NUMrVARS_, CDF_NUMvATTRS_, CDF_NUMzVARS_, CDF_OK, CDF_OPEN_ERROR, CDF_PATHNAME_LEN, CDF_READ_ERROR, CDF_READONLY_MODE_, CDF_REAL4, CDF_REAL8, CDF_RELEASE_, CDF_SAVE_ERROR, CDF_SCRATCHDIR_, CDF_STATUS_, CDF_STATUSTEXT_LEN, CDF_TIME_TT2000, CDF_UCHAR, CDF_UINT1, CDF_UINT2, CDF_UINT4, CDF_VAR_NAME_LEN, CDF_VAR_NAME_LEN256, CDF_VERSION_, CDF_WARN, CDF_WRITE_ERROR, CDF_zMODE_, CDFwithSTATS_, CHECKSUM_, CHECKSUM_ERROR, CHECKSUM_NOT_ALLOWED, CLOSE_, COLUMN_MAJOR, COMPRESS_CACHESIZE_, COMPRESSION_ERROR, CONFIRM_, CORRUPTED_V2_CDF, CORRUPTED_V3_CDF, CREATE_, CURgENTRY_EXISTENCE_, CURrENTRY_EXISTENCE_, CURzENTRY_EXISTENCE_, DATATYPE_MISMATCH, DATATYPE_SIZE_, DECOMPRESSION_ERROR, DECSTATION_DECODING, DECSTATION_ENCODING, DEFAULT_BYTE_PADVALUE, DEFAULT_CHAR_PADVALUE, DEFAULT_DOUBLE_PADVALUE, DEFAULT_EPOCH_PADVALUE, DEFAULT_EPOCH16_PADVALUE, DEFAULT_FLOAT_PADVALUE, DEFAULT_INT1_PADVALUE, DEFAULT_INT2_PADVALUE, DEFAULT_INT4_PADVALUE, DEFAULT_INT8_PADVALUE, DEFAULT_REAL4_PADVALUE, DEFAULT_REAL8_PADVALUE, DEFAULT_TT2000_PADVALUE, DEFAULT_UCHAR_PADVALUE, DEFAULT_UINT1_PADVALUE, DEFAULT_UINT2_PADVALUE, DEFAULT_UINT4_PADVALUE, DELETE_, DID_NOT_COMPRESS, EMPTY_COMPRESSED_CDF, END_OF_VAR, EPOCH_STRING_LEN, EPOCH_STRING_LEN_EXTEND, EPOCHI_STRING_LEN,

EPOCH1_STRING_LEN_EXTEND, EPOCH2_STRING_LEN, EPOCH2_STRING_LEN_EXTEND, EPOCH3_STRING_LEN, EPOCH3_STRING_LEN_EXTEND, EPOCH4_STRING_LEN, EPOCH4_STRING_LEN_EXTEND, EPOCHx_FORMAT_MAX, EPOCHx_STRING_MAX, FILLED_TT2000_VALUE, FORCED_PARAMETER, FUNCTION_NOT_SUPPORTED, gENTRY_, gENTRY_DATA_, gENTRY_DATASPEC_, gENTRY_DATATYPE_, gENTRY_EXISTENCE_, gENTRY_NUMELEMS_, GET_, GETCDFCHECKSUM_, GETCDFFILEBACKWARD_, GETCDFVALIDATE_, GETLEAPSECONDSENVVAR_, GLOBAL_SCOPE, GZIP_COMPRESSION, HOST_DECODING, HOST_ENCODING, HP_DECODING, HP_ENCODING, HUFF_COMPRESSION, IA64VMSd_DECODING, IA64VMSd_ENCODING, IA64VMSg_DECODING, IA64VMSg_ENCODING, IA64VMSi_DECODING, IA64VMSi_ENCODING, IBM_PC_OVERFLOW, IBMPC_DECODING, IBMPC_ENCODING, IBMRS_DECODING, IBMRS_ENCODING, ILLEGAL_EPOCH_FIELD, ILLEGAL_EPOCH_VALUE, ILLEGAL_FOR_SCOPE, ILLEGAL_IN_zMODE, ILLEGAL_ON_V1_CDF, ILLEGAL_TT2000_VALUE, IS_A_NETCDF, LIB_COPYRIGHT_, LIB_INCREMENT_, LIB_RELEASE_, LIB_subINCREMENT_, LIB_VERSION_, MAC_DECODING, MAC_ENCODING, MD5_CHECKSUM, MULTI_FILE, MULTI_FILE_FORMAT, NA_FOR_VARIABLE, NEGATIVE_FP_ZERO, NEGtoPOSfp0off, NEGtoPOSfp0on, NETWORK_DECODING, NETWORK_ENCODING, NeXT_DECODING, NeXT_ENCODING, NO_ATTR_SELECTED, NO_CDF_SELECTED, NO_CHECKSUM, NO_COMPRESSION, NO_DELETE_ACCESS, NO_ENTRY_SELECTED, NO_MORE_ACCESS, NO_PADVALUE_SPECIFIED, NO_SPARSEARRAYS, NO_SPARSERECORDS, NO_STATUS_SELECTED, NO_SUCH_ATTR, NO_SUCH_CDF, NO_SUCH_ENTRY, NO_SUCH_RECORD, NO_SUCH_VAR, NO_VAR_SELECTED, NO_VARS_IN_CDF, NO_WRITE_ACCESS, NONE_CHECKSUM, NOT_A_CDF, NOT_A_CDF_OR_NOT_SUPPORTED, NOVARY, NULL_, OPEN_, OPTIMAL_ENCODING_TREES, OTHER_CHECKSUM, PAD_SPARSERECORDS, PPC_DECODING, PPC_ENCODING, PRECEEDING_RECORDS_ALLOCATED, PREV_SPARSERECORDS, PUT_, READ_ONLY_DISTRIBUTION, READ_ONLY_MODE, READONLYoff, READONLYon, rENTRY_, rENTRY_DATA_, rENTRY_DATASPEC_, rENTRY_DATATYPE_, rENTRY_EXISTENCE_, rENTRY_NAME_, rENTRY_NUMELEMS_, rENTRY_NUMSTRINGS_, rENTRY_STRINGSDATA_, RLE_COMPRESSION, RLE_OF_ZEROs, ROW_MAJOR, rVAR_, rVAR_ALLOCATEBLOCK_, rVAR_ALLOCATEDFROM_, rVAR_ALLOCATEDTO_, rVAR_ALLOCATERECS_, rVAR_BLOCKINGFACTOR_, rVAR_CACHESIZE_, rVAR_COMPRESSION_, rVAR_DATA_, rVAR_DATASPEC_, rVAR_DATATYPE_, rVAR_DIMVARYS_, rVAR_EXISTENCE_, rVAR_HYPERDATA_, rVAR_INITIALRECS_, rVAR_MAXallocREC_, rVAR_MAXREC_, rVAR_NAME_, rVAR_nINDEXENTRIES_, rVAR_nINDEXLEVELS_, rVAR_nINDEXRECORDS_, rVAR_NUMallocRECS_, rVAR_NUMBER_, rVAR_NUMELEMS_, rVAR_NUMRECS_, rVAR_PADVALUE_, rVAR_RECORDS_, rVAR_RECORDS_RENUMBER_, rVAR_RECVARY_, rVAR_RESERVEPERCENT_, rVAR_SEQDATA_, rVAR_SEQPOS_, rVAR_SPARSEARRAYS_, rVAR_SPARSERECORDS_, rVARs_CACHESIZE_, rVARs_DIMCOUNTS_, rVARs_DIMINDICES_, rVARs_DIMINTERVALS_, rVARs_DIMSIZES_, rVARs_MAXREC_, rVARs_NUMDIMS_, rVARs_RECCOUNT_, rVARs_RECDATA_, rVARs_RECINTERVAL_, rVARs_RECNUMBER_, SAVE_, SCRATCH_CREATE_ERROR, SCRATCH_DELETE_ERROR, SCRATCH_READ_ERROR, SCRATCH_WRITE_ERROR, SELECT_, SGI_DECODING, SGi_ENCODING, SINGLE_FILE, SINGLE_FILE_FORMAT, SOME_ALREADY_ALLOCATED, STAGE_CACHESIZE_, STATUS_TEXT_, STRING_NOT_UTF8_ENCODING, STRINGDELIMITER, SUN_DECODING, SUN_ENCODING, TOO_MANY_PARMS, TOO_MANY_VARS, TRY_TO_READ_NONSTRING_DATA, TT2000_0_STRING_LEN, TT2000_1_STRING_LEN, TT2000_2_STRING_LEN, TT2000_3_STRING_LEN, TT2000_4_STRING_LEN, TT2000_CDF_MAYNEEDUPDATE, TT2000_TIME_ERROR, TT2000_USED_OUTDATED_TABLE, UNABLE_TO_PROCESS_CDF, UNKNOWN_COMPRESSION, UNKNOWN_SPARSENESS, UNSUPPORTED_OPERATION, VALIDATE_, VALIDATEFILEoff, VALIDATEFILEon, VAR_ALREADY_CLOSED, VAR_CLOSE_ERROR, VAR_CREATE_ERROR, VAR_DELETE_ERROR, VAR_EXISTS, VAR_NAME_TRUNC, VAR_OPEN_ERROR, VAR_READ_ERROR, VAR_SAVE_ERROR, VAR_WRITE_ERROR, VARIABLE_SCOPE, VARY, VAX_DECODING, VAX_ENCODING, VIRTUAL_RECORD_DATA, zENTRY_, zENTRY_DATA_, zENTRY_DATASPEC_, zENTRY_DATATYPE_, zENTRY_EXISTENCE_, zENTRY_NAME_, zENTRY_NUMELEMS_, zENTRY_NUMSTRINGS_, zENTRY_STRINGSDATA_, ZLIB_COMPRESS_ERROR, ZLIB_UNCOMPRESS_ERROR, zMODEoff, zMODEon1, zMODEon2, zVAR_, zVAR_ALLOCATEBLOCK_, zVAR_ALLOCATEDFROM_, zVAR_ALLOCATEDTO_, zVAR_ALLOCATERECS_, zVAR_BLOCKINGFACTOR_, zVAR_CACHESIZE_, zVAR_COMPRESSION_, zVAR_DATA_, zVAR_DATASPEC_, zVAR_DATATYPE_, zVAR_DIMCOUNTS_, zVAR_DIMINDICES_, zVAR_DIMINTERVALS_, zVAR_DIMSIZES_, zVAR_DIMVARYS_, zVAR_EXISTENCE_, zVAR_HYPERDATA_, zVAR_INITIALRECS_, zVAR_MAXallocREC_, zVAR_MAXREC_, zVAR_NAME_, zVAR_nINDEXENTRIES_, zVAR_nINDEXLEVELS_, zVAR_nINDEXRECORDS_, zVAR_NUMallocRECS_, zVAR_NUMBER_, zVAR_NUMDIMS_, zVAR_NUMELEMS_, zVAR_NUMRECS_, zVAR_PADVALUE_, zVAR_RECCOUNT_, zVAR_RECINTERVAL_, zVAR_RECNUMBER_, zVAR_RECORDS_, zVAR_RECORDS_RENUMBER_, zVAR_RECVARY_, zVAR_RESERVEPERCENT_, zVAR_SEQDATA_, zVAR_SEQPOS_, zVAR_SPARSEARRAYS_, zVAR_SPARSERECORDS_, zVARs_CACHESIZE_, zVARs_MAXREC_, zVARs_RECDATA_, zVARs_RECNUMBER_

Method Summary

Methods

Modifier and Type	Method and Description
void	**allocateBlock**(long firstRec, long lastRec) Allocates a range of records for this variable.
void	**allocateRecords**(long num0toRecords) Allocates a number of records, starting from record number 0.
boolean	**checkPadValueExistence**() Checks if the pad value has been defined for this variable.
void	**concatenateDataRecords**(**Variable** destVar) Concatenates this variable's data records to the destination variable.
long	**confirmCacheSize**() Gets the number of 512-byte cache buffers defined for this variable.
long	**confirmPadValue**() Checks the existence of an explicitly specified pad value for the current z variable.
long	**confirmReservePercent**() Gets the reserve percentage set for this variable.
Variable	**copy**(**CDF** destCDF, java.lang.String varName)

	Copies this variable into a new variable and puts it into the designated CDF file.
Variable	**copy**(java.lang.String varName) Copies this variable to a new variable.
void	**copyDataRecords**(**Variable** destVar) Copies this variable's data to the destination variable.
static **Variable**	**create**(**CDF** myCDF, java.lang.String varName, long dataType, long numElements, long numDims, long[] dimSizes, long recVary, long[] dimVarys) Creates a variable.
void	**delete**() Deletes this variable.
void	**deleteRecords**(long firstRec, long lastRec) Deletes a range of records from this variable.
void	**deleteRecordsRenumber**(long firstRec, long lastRec) Deletes a range of records from this variable.
void	**dumpDataWithFormat**(**CDFData** data) Dump the data, either the variable value or metadata, with the format.
void	**dumpDataWithFormat**(java.lang.Object data) Dump the data, either the variable value or metadata, with or without the format.
void	**dumpDataWithFormat**(java.lang.Object data, java.io.PrintWriter outWriter) Dump the data, either the variable value or metadata, with or without the format.
Variable	**duplicate**(**CDF** destCDF, java.lang.String varName) Duplicates this variable and put it into the designated CDF file.
Variable	**duplicate**(java.lang.String varName) Duplicates this variable to a new variable.
long	**getAllocatedFrom**(long recNum) Inquires the next allocated record at or after a given record for this variable.
long	**getAllocatedTo**(long firstRec) Inquires the last allocated record (before the next unallocated record) at or after a given record for this variable.
java.lang.Object	**getAllRecords**() Get all written records, from record number 0 till the last written record number, from this variable.
CDFData	**getAllRecordsObject**() Get all written records, from record number 0 till the last written record number, from this variable.
java.util.Vector	**getAttributes**() Returns the variable attributes that are associated with this variable.
java.util.Map	**getAttributesandEntries**() Returns the variable attributes and their entries that are associated with this variable.
java.util.Map	**getAttributesandEntries**(boolean encoding) Returns the variable attributes and their entries that are associated with this variable.
long	**getBlockingFactor**() Gets the blocking factor for this variable.
java.lang.String	**getCompression**() Gets the string representation of the compression type and parameters set for this variable.
long[]	**getCompressionParms**() Sets the compression parameters of this variable.
long	**getCompressionPct**() Gets the compression percentage rate of this variable.
long	**getCompressionType**() Gets the compression type of this variable.
long	**getDataType**() Gets the CDF data type of this variable.

long[]	**getDimSizes**() Gets the dimensions size of this variable.
long[]	**getDimVariances**() Gets the dimension variances for this variable.
java.lang.Object	**getEntryData**(java.lang.String attrName) Gets the attribute entry data for this variable.
java.lang.Object	**getHyperData**(long recNum, long recCount, long recInterval, long[] dimIndices, long[] dimCounts, long[] dimIntervals) Reads one or more values from the current z variable.
CDFData	**getHyperDataObject**(long recNum, long recCount, long recInterval, long[] dimIndices, long[] dimCounts, long[] dimIntervals) Reads one or more values from the current z variable.
long	**getID**() Gets the ID of this variable.
long	**getMaxAllocatedRecord**() Gets the maximum allocated record number for this variable.
long	**getMaxWrittenRecord**() Gets the last written record number, beginning with 0.
CDF	**getMyCDF**() Gets the CDF object to which this variable belongs.
java.lang.String	**getName**() Gets the name of this variable.
long	**getNumAllocatedRecords**() Gets the number of records allocated for this variable.
long	**getNumDims**() Gets the number of dimensions for this variable.
long	**getNumElements**() Gets the number of elements for this variable.
long	**getNumWrittenRecords**() Gets the number of records physically written (not allocated) for this variable.
java.lang.Object	**getPadValue**() Gets the pad value for this variable.
java.lang.Object	**getRecord**(long recNum) Gets a single record from this variable.
CDFData	**getRecordObject**(long recNum) Get a single record of data from this variable.
CDFData	**getRecordsObject**(long recNum, long numRecs) Get a number of records of data from this variable.
boolean	**getRecVariance**() Gets the value of record variance.
java.lang.Object	**getScalarData**() Gets the scalar data from a non-record varying 0-dimensional variable.
java.lang.Object	**getScalarData**(long recNum) Get the scalar data from a record varying 0-dimensional variable.
CDFData	**getScalarDataObject**() Get the scalar data from a non-record varying 0-dimensional variable.
CDFData	**getScalarDataObject**(long recNum) Get the scalar data from this record varying 0-dimensional variable.
java.lang.Object	**getSingleData**(long recNum, long[] indices) Gets a single data value.
CDFData	**getSingleDataObject**(long recNum, long[] indices) Gets a single data object from this variable.

long	**getSparseRecords**() Gets the sparse record type for this variable.
void	**putEntry**(**Attribute** attr, long dataType, java.lang.Object data) Creates an attribute entry for this variable.
void	**putEntry**(java.lang.String attrName, long dataType, java.lang.Object data) Creates an attribute entry for this variable.
CDFData	**putHyperData**(long recNum, long recCount, long recInterval, long[] dimIndices, long[] dimCounts, long[] dimIntervals, java.lang.Object data) Writes one or more values from the current z variable.
CDFData	**putRecord**(long recNum, java.lang.Object data) Adds a single record to a record-varying variable.
CDFData	**putRecord**(java.lang.Object data) Adds a single record to a non-record-varying variable.
CDFData	**putRecords**(long startRec, long recCount, java.lang.Object data) Writes a number of records from the current z variable.
CDFData	**putScalarData**(long recNum, java.lang.Object data) Adds a scalar data to this variable (of 0 dimensional).
CDFData	**putScalarData**(java.lang.Object data) Adds a scalar data to this variable (of 0 dimensional).
CDFData	**putSingleData**(long recNum, long[] indices, java.lang.Object data) Adds a single data value to this variable.
java.util.Map	**readVariable**() Gets the full variable information, its specification, metadata and data.
java.util.Map	**readVariable**(boolean encoding, boolean basic, boolean metadata, boolean vardata) Gets the specified variable information: its specification, metadata and data.
java.util.Map	**readVariableData**() Gets the variable's data.
java.util.Map	**readVariableMetaData**() Gets the variable's metadata.
java.util.Map	**readVariableSpec**() Gets the variable's specification.
void	**rename**(java.lang.String newName) Renames the current variable.
void	**selectCacheSize**(long cacheSize) Sets the number of 512-byte cache buffers to be used.
void	**selectReservePercent**(long reservePercent) Sets the reserve percentage to be used for this variable.
void	**setBlockingFactor**(long blockingFactor) Sets the blocking factor for this variable.
void	**setCompression**(long cType, long[] cParms) Sets the compression type and parameters for this variable.
void	**setDimVariances**(long[] dimVariances) Sets the dimension variances for this variable.
void	**setInitialRecords**(long nRecords) Sets the number of records to be written initially for this variable.
void	**setPadValue**(java.lang.Object padValue) Sets the pad value for this variable.
void	**setRecVariance**(long recVariance) Sets the record variance for this variable.
void	**setSparseRecords**(long sparseRecords)

	Sets the sparse record type for this variable.
java.lang.String	**toString**() Gets the name of this variable.
void	**updateDataSpec**(long dataType, long numElements) Update the data specification (data type and number of elements) of the variable.

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Method Detail

create

```
public static Variable create(CDF myCDF,
              java.lang.String varName,
              long dataType,
              long numElements,
              long numDims,
              long[] dimSizes,
              long recVary,
              long[] dimVarys)
                       throws CDFException
```

Creates a variable.

The following example creates a variable called "Longitude" that is scalar (non-array) and record-varying:

```
     longitude = Variable.create(cdf, "Longitude", CDF_INT2,
                                 1L, 0L, new long [] {1},
                                 VARY,
                                 new long [] {NOVARY});
```

The following example creates a variable called "TestData" whose data is 2-dimensional (3 x 2), record variance is TURE, and dimension variances are TRUE.

```
     data = Variable.create(cdf, "TestData", CDF_INT2,
                            1L, 2L, new long [] {3,2},
                            VARY,
                            new long [] {VARY, VARY});
```

Parameters:

`myCDF` - the CDF to which this variable belongs

`varName` - the name of the variable to create

`dataType` - the CDF data type for this variable that should be one of the following:

- 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 - 2*8-byte, floating point
- CDF_TIME_TT2000 - 8-byte, signed integer

`numElements` - for CDF_CHAR and CDF_UCHAR this is the string length, 1 otherwise

`numDims` - the dimensionality

`dimSizes` - The dimension sizes. An array of length numDims indicating the size of each dimension

`recVary` - the record variance that should be either VARY or NOVARY

`dimVarys` - The dimension variance(s). Each dimension variance should be either VARY or NOVARY.

Returns:

newly created Variable object

Throws:

`CDFException` - if there is a problem creating a variable

delete

```
public void delete()
            throws CDFException
```

Deletes this variable.

Specified by:

`delete` in interface `CDFObject`

Throws:

`CDFException` - if there was an error deleting this variable

rename

```
public void rename(java.lang.String newName)
            throws CDFException
```

Renames the current variable.

Specified by:

`rename` in interface `CDFObject`

Parameters:

`newName` - the new variable name

Throws:

`CDFException` - if there was a problem renaming this variable

copy

```
public Variable copy(java.lang.String varName)
              throws CDFException
```

Copies this variable to a new variable. This method only copies the metadata associated with this variable. The duplicate method in

this class should be used if the user wants to copy a variable with data and metadata.

Parameters:

`varName` - the name of the variable to copy this variable into

Returns:

newly copied variable

Throws:

`CDFException` - if there was a problem copying a variable

copy

```
public Variable copy(CDF destCDF,
            java.lang.String varName)
              throws CDFException
```

Copies this variable into a new variable and puts it into the designated CDF file. This method only copies the metadata associated with this variable. The duplicate method in this class should be used if the user wants to copy a variable with data and metadata.

Parameters:

`destCDF` - the destination CDF into which copy this variable

`varName` - the new variable name

Returns:

newly copied variable

Throws:

`CDFException` - if there was a problem copying a variable

duplicate

```
public Variable duplicate(java.lang.String varName)
                  throws CDFException
```

Duplicates this variable to a new variable.

Note: This copies everything from the existing variable to a new variable. It includes the metadata associated with this variable, all data records as well as other information such as blocking factor/compression/sparseness/pad value.

Parameters:

`varName` - the name of the variable to duplicate this variable into

Returns:

newly duplicated variable

Throws:

`CDFException` - if there was a problem duplicating a variable

duplicate

```
public Variable duplicate(CDF destCDF,
                java.lang.String varName)
```

```
                    throws CDFException
```

Duplicates this variable and put it into the designated CDF file.

Note: This copies everything from the current variable to a new variable. It includes the metadata associated with this variable, all data records as well as other information such as blocking factor/compression/sparseness/pad value.

Parameters:

`destCDF` - the destination CDF to duplicate this variable into

`varName` - the name of the variable to duplicate this variable into

Returns:

newly duplicated variable

Throws:

`CDFException` - if there was a problem duplicating a variable

copyDataRecords

```
public void copyDataRecords(Variable destVar)
                     throws CDFException
```

Copies this variable's data to the destination variable.

Parameters:

`destVar` - the destination variable to copy data into

Throws:

`CDFException` - if there was a problem copying data records **Note:** This copies data records from the current variable to the destination variable. The metadata associated with the destination variable will be not changed.

The current CDF file MUST be saved first (by calling the save() method) before 'copying/duplicating data records' operation is performed. Otherwise the program will either fail or produce undesired results.

concatenateDataRecords

```
public void concatenateDataRecords(Variable destVar)
                            throws CDFException
```

Concatenates this variable's data records to the destination variable.

Parameters:

`destVar` - the destination variable to copy data records into

Throws:

`CDFException` - if there was a problem copying data records **Note:** This copies only the data records from the current variable to the destination variable. The metadata associated with the destination variable will be not changed.

getEntryData

```
public java.lang.Object getEntryData(java.lang.String attrName)
                              throws CDFException
```

Gets the attribute entry data for this variable.

The following examples retrieves the 'Longitude' variable entry for the attribute VALIDMIN:

```
Variable var = cdf.getVariable("Longitude");
float longitude = (float) var.getEntryData("VALIDMIN");
```

Parameters:

`attrName` - the name of the attribute to get entry data from

Returns:

the attribute entry data for this variable

Throws:

`CDFException` - if there was a problem getting entry data

getSingleData

```
public java.lang.Object getSingleData(long recNum,
                                       long[] indices)
                                         throws CDFException
```

Gets a single data value. This method is useful for extracting a specific item among many items.

Let's assume that variable TestData is defined to be 1-dimensional array that has 3 elements in it. The following example extracts the last element from the second record:

```
Variable var = cdf.getVariable("TestData");
int data = (int) var.getSingleData(1L, new long [] {2});
```

Let's assume that variable TestData is defined to be 2-dimensional (3x2 - 3 rows and 2 columns) array. The following example extracts the first element of the second row from the first record:

```
Variable var = cdf.getVariable("TestData");
int data = (int) var.getSingleData(0L, new long [] {1,0});
```

Parameters:

`recNum` - the record number to retrieve data from

`indices` - the index, within a record, to extract data from

Returns:

extracted single data value

Throws:

`CDFException` - if there was a problem extracting data

getSingleDataObject

```
public CDFData getSingleDataObject(long recNum,
                                   long[] indices)
                                     throws CDFException
```

Gets a single data object from this variable. The value read is put into an CDFData object. This method is identical to the getSingleData method except that the extracted data is encapsulated inside the CDFData object along with other information such as record number, record count, record interval, dimension indices, dimension counts, and dimension intervals.

Parameters:

`recNum` - the record number to retrieve data from

`indices` - the index, within a record, to extract data from

Returns:

CDFData object containing the requested data

Throws:

`CDFException` - if there was a problem extracting data

getRecord

```
public java.lang.Object getRecord(long recNum)
                           throws CDFException
```

Gets a single record from this variable.

Let's assume that variable TestData is defined to be 2-dimensional (3x2 - 3 rows and 2 columns). The following example extracts an entire record (containing 6 elements) from the first record from a variable of data type CDF_INT4:

```
Variable var = cdf.getVariable("TestData");
int[][] data = (int [][]) var.getRecord(0L);
```

However, if a dimensional variable with all indices being invariant, e.g., 2-dimensional (1x1), the retrieved object will be different. (Since the variable has only one data value per record, it is preferred to be defined as an 0-dim, rather.). The object is not an array, instead, a single Java class item, e.g., Integer, Double, Short, etc. The following example extracts an record from the first record of a variable, 2-dim (1x1), with data type CDF_INT2:

```
Variable var = cdf.getVariable("TestVar");
short data = ((Short) var.getRecord(0L)).shortValue();
```

Parameters:

`recNum` - the record number to retrieve data from

Returns:

the requested data record

Throws:

`CDFException` - if there was a problem getting a record

getRecordObject

```
public CDFData getRecordObject(long recNum)
                        throws CDFException
```

Get a single record of data from this variable. The values read are put into an CDFData object. This method is identical to the getRecord method except that the extracted data is encapsulated inside the CDFData object along with other information such as record number, record count, record interval, dimension indices, dimension counts, and dimension intervals.

Parameters:

`recNum` - the record number to retrieve data from

Returns:

CDFData object containing the requested data record

Throws:

`CDFException` - if there was a problem getting a record

getRecordsObject

```
public CDFData getRecordsObject(long recNum,
                        long numRecs)
                          throws CDFException
```

Get a number of records of data from this variable. The values read are put into an CDFData object.

Parameters:

`recNum` - the record number to start to retrieve data from

`numRecs` - the number of records to retrieve

Returns:

CDFData object containing the requested data record(s)

Throws:

`CDFException` - if there was a problem getting the record(s)

getAllRecordsObject

```
public CDFData getAllRecordsObject()
                               throws CDFException
```

Get all written records, from record number 0 till the last written record number, from this variable. The data values read are put into an CDFData object. Note: If this variable has sparse records, then the missing record(s) is filled with FILLVAL (or PAD if FILLVAL does not exist) value.

Returns:

CDFData object containing all data record(s)

Throws:

`CDFException` - if there was a problem getting the record(s)

getAllRecords

```
public java.lang.Object getAllRecords()
                                 throws CDFException
```

Get all written records, from record number 0 till the last written record number, from this variable. The data values read are put into an Object. Note: If this variable has sparse records, then the missing record(s) is filled with FILLVAL (or PAD if FILLVAL does not exist) value.

Returns:

Object containing all data record(s)

Throws:

`CDFException` - if there was a problem getting the record(s)

getScalarData

```
public java.lang.Object getScalarData()
```

throws CDFException

Gets the scalar data from a non-record varying 0-dimensional variable.

Returns:

the variable data from this variable

Throws:

CDFException - if there was a problem getting data

getScalarData

```
public java.lang.Object getScalarData(long recNum)
                                throws CDFException
```

Get the scalar data from a record varying 0-dimensional variable.

Parameters:

recNum - The record number to retrieve data from

Returns:

the variable data from this variable

Throws:

CDFException - if there was a problem getting data

getScalarDataObject

```
public CDFData getScalarDataObject()
                            throws CDFException
```

Get the scalar data from a non-record varying 0-dimensional variable. This method is identical to the getScalarData method except that the extracted data is encapsulated inside the CDFData object along with other information such as record number, record count, record interval, dimension indices, dimension counts, and dimension intervals.

Returns:

the variable data from this variable

Throws:

CDFException - if there was a problem getting data

getScalarDataObject

```
public CDFData getScalarDataObject(long recNum)
                            throws CDFException
```

Get the scalar data from this record varying 0-dimensional variable. This method is identical to the getScalarData method except that the extracted data is encapsulated inside the CDFData object along with other information such as record number, record count, record interval, dimension indices, dimension counts, and dimension intervals.

Parameters:

recNum - the record number to retrieve data from

Returns:

the variable data from this variable

Throws:

`CDFException` - if there was a problem getting data

getHyperData

```
public java.lang.Object getHyperData(long recNum,
                              long recCount,
                              long recInterval,
                              long[] dimIndices,
                              long[] dimCounts,
                              long[] dimIntervals)
                                throws CDFException
```

Reads one or more values from the current z variable. The values are based on the current record number, current record count, current record interval, current dimension indices, current dimension counts, and current dimension intervals.

Let's assume that variable TestData is defined to be 2-dimensional (3x2 - 3 rows and 2 columns). The following example extracts the entire record (containing 6 elements) from the first, second, and third records:

```
Variable var = cdf.getVariable("TestData");
int[][][] data = (int [][][]) var.getHyperData (0L, 3L, 1L,
                                                 new long[] {0, 0},
                                                 new long[] {3, 2},
                                                 new long[] {1, 1});
```

The following example will extract the entire record from the first record:

```
Variable var = cdf.getVariable("TestData");
int[][] data = (int [][]) var.getHyperData (0L, 1L, 1L,
                                            new long[] {0, 0},
                                            new long[] {3, 2},
                                            new long[] {1, 1});
```

Note: it returns a 2-dimensional object as only one record is involved. The following example will extract the second row from the first, and third records:

```
Variable var = cdf.getVariable("TestData");
int[][] data = (int [][]) var.getHyperData (0L, 3L, 2L,
                                            new long[] {1, 0},
                                            new long[] {1, 2},
                                            new long[] {1, 1});
```

The following example will extract the first column from the first and second records:

```
Variable var = cdf.getVariable("TestData");
int[][] data = (int [][]) var.getHyperData (0L, 2L, 1L,
                                            new long[] {0, 0},
                                            new long[] {3, 1},
                                            new long[] {1, 1});
```

Parameters:

`recNum` - the record number at which data search begins

`recCount` - the number of records to read

`recInterval` - the number of records to skip between reads

`dimIndices` - the dimension index within a record at which data search begins

`dimCounts` - the number of elements to read from dimIndices

`dimIntervals` - the number of elements to skip between reads

Returns:

the variable data specified by recNum, recCount, recInterval, dimIndices, dimCounts, and dimIntervals

Throws:

`CDFException` - if there was a problem getting data

getHyperDataObject

```
public CDFData getHyperDataObject(long recNum,
                          long recCount,
                          long recInterval,
                          long[] dimIndices,
                          long[] dimCounts,
                          long[] dimIntervals)
                            throws CDFException
```

Reads one or more values from the current z variable. The values are read based on the current record number, current record count, current record interval, current dimension indices, current dimension counts, and current dimension intervals. The values read are put into an CDFData object.

Parameters:

`recNum` - the record number at which data search begins

`recCount` - the number of records to read

`recInterval` - the number of records to skip between reads

`dimIndices` - the dimension index within a record at which data search begins

`dimCounts` - the number of elements to read from dimIndices

`dimIntervals` - the number of elements to skip between reads

Returns:

CDFData object that contains the variable data specified by recNum, recCount, recInterval, dimIndices, dimCounts, and dimIntervals as well as the information passed to this method plus the number of dimensions and the number of elements for this variable.

Throws:

`CDFException` - if there was a problem getting data

readVariable

```
public java.util.Map readVariable()
```

Gets the full variable information, its specification, metadata and data. Note: Each metadata/data of epoch type data will be encoded in date/time string.

```
  Variable v = cdf.getVariable("var1");
  Map var = v.readVariable();
  // Three key-value pairs will be included with keys of as
  // "VarSpec", "VarMetaData", "VarData"

  // Key: "VarSpec" contains variable specification - another Map object
  // These include: keys of "DataType", "NumElements", "NumDims",
  // "DimSizes" and "PadValue".
  System.out.println("  specification: "+var.get("VarSpec"));

  // Key: "VarMetaData" contains variable attributes - another Map object
  // These include variable attribute name (key) and its entry (value)
  System.out.println("  meta: "+var.get("VarMetaData"));
```

```
  // Key: "VarData" has variable data - an object
  System.out.println("  data:");
  CDFUtils.printData(var.get("VarData"));
```

Returns:

A Map object that contains the full variable information.

readVariableSpec

```
public java.util.Map readVariableSpec()
```

Gets the variable's specification.

Returns:

A Map object that contains the variable's specification with the key "VarSpec". The value is another Map object.

readVariableMetaData

```
public java.util.Map readVariableMetaData()
```

Gets the variable's metadata.

Returns:

A Map object that contains the variable's metadata with the key "VarMetaData". The value is another Map object if it has metadata.

readVariableData

```
public java.util.Map readVariableData()
```

Gets the variable's data.

Returns:

A Map object that contains the variable's specification with the key "VarData".

readVariable

```
public java.util.Map readVariable(boolean encoding,
                          boolean basic,
                          boolean metadata,
                          boolean vardata)
```

Gets the specified variable information: its specification, metadata and data.

```
  Variable v = cdf.getVariable("var1");
  Map var = v.readVariable(true, true, true, true);
  // Three key-value pairs might be included with keys of as
  // "VarSpec" and "VarMetaData" and/or "VarData" if there are data.

  // Key: "VarSpec" contains variable specification - another Map object
  // These include: keys of "DataType", "NumElements", "NumDims",
  // "DimSizes" and "PadValue".
  System.out.println("  specification: "+var.get("VarSpec"));

  // Key: "VarMetaData" contains variable attributes - another Map object
```

```
  // These include variable attribute name (key) and its entry (value)
  System.out.println("  meta: "+var.get("VarMetaData"));

  // Key: "VarData" has variable data - an object
  System.out.println("  data:");
  CDFUtils.printData(var.get("VarData"));
```

Parameters:

`encoding` - A true/false flag indicates whether to encode epoch data type from metadata to date/time string.

`basic` - A true/false flag indicates whether to return the specification of the variable

`metadata` - A true/false flag indicates whether to return the metadata from the variable

`vardata` - true/false flag indicates whether to return the data from the variable

Returns:

A Map object that contains the requested variable information.

putEntry

```
public void putEntry(java.lang.String attrName,
            long dataType,
            java.lang.Object data)
              throws CDFException
```

Creates an attribute entry for this variable.

The following example creates a variable entry for the variable "Longitude" associated with the attribute "VALIDMIN":

```
     Variable longitude = cdf.getVariable("Longitude");
     longitude.putEntry("VALIDMIN", CDF_INT2, new Short((short)180));
```

Parameters:

`attrName` - the attribute to which this attribute entry is attached

`dataType` - the CDF data type of the entry data - see the description of the create method in this class for a list of the CDF data types supported

`data` - the attribute entry data to be added

Throws:

`CDFException` - if a problem occurs putting an entry

See Also:

`Attribute`, `Entry`

putEntry

```
public void putEntry(Attribute attr,
            long dataType,
            java.lang.Object data)
              throws CDFException
```

Creates an attribute entry for this variable. The following example creates a variable entry for the variable "Longitude" associated with the attribute "VALIDMIN":

```
     Variable longitude = cdf.getVariable("Longitude");
     Attribute validMin = Attribute.create(cdf, "VALIDMIN",
                                           VARIABLE_SCOPE);
     Entry.create(validMin, longitude.getID(), CDF_INT2,
```

```
                new Short((short)10));
            OR
    longitude.putEntry(validMin, CDF_INT2, new Short((short)180));
```

Parameters:

`attr` - the attribute to which this attribute entry is attached

`dataType` - the CDF data type of the entry data - see the description of the create method in this class for a list of the CDF data types supported

`data` - the attribute entry data to be added

Throws:

`CDFException` - if a problem occurs putting an entry

See Also:

`Attribute`, `Entry`

putSingleData

```
public CDFData putSingleData(long recNum,
                    long[] indices,
                    java.lang.Object data)
                      throws CDFException
```

Adds a single data value to this variable. This method is used to specify a particular element in a record (if a record is comprised of multiple elements). If a record contains 3 elements, the following example will write the second element to record number 0, leaving the first and third elements unwritten.

```
    longitude = cdf.getVariable("Longitude");
    longitude.putSingleData(0L, new long[] {1}, new Short((short)200));
           or
    longitude.putSingleData(0L, new long[] {1}, longitudeData[1]);
```

Parameters:

`recNum` - the record number to which this data belongs

`indices` - the index (location) in the specified record

`data` - the data to be added

Returns:

CDFData object containing the user specified data

Throws:

`CDFException` - if there was an error writing data

putScalarData

```
public CDFData putScalarData(long recNum,
                    java.lang.Object data)
                      throws CDFException
```

Adds a scalar data to this variable (of 0 dimensional). This method should be used if a variable is defined as record-varying and non-array. The following example will write data to record number 0.

```
    longitude = cdf.getVariable("Longitude");
    longitude.putScalarData(0L, new Short((short)200));
           or
```

```
    longitude.putScalarData(0L, longitudeData[0]);
```

Parameters:

`recNum` - the record number to which this data belongs

`data` - the data to be added

Returns:

CDFData object containing the user specified data

Throws:

`CDFException` - if there was an error writing data

putScalarData

```
public CDFData putScalarData(java.lang.Object data)
                      throws CDFException
```

Adds a scalar data to this variable (of 0 dimensional). This method should be used if a variable is defined as non-record-varying and non-array. Note that there'll be only one record exist if a variable is defined as non-record-varying. The following example will write data to record number 0.

```
    longitude = cdf.getVariable("Longitude");
    longitude.putScalarData(new Short((short)200));
           or
    longitude.putScalarData(longitudeData[0]);
```

Parameters:

`data` - the data to be added

Returns:

CDFData object containing the user specified data

Throws:

`CDFException` - if there was an error writing data

putRecord

```
public CDFData putRecord(long recNum,
                java.lang.Object data)
                  throws CDFException
```

Adds a single record to a record-varying variable. This method should be used if a record contains one or more elements.

```
    The following example adds a scalar data to record number 0:
        longitude = cdf.getVariable("Longitude");
        longitude.putRecord(0L, new Short((short)200));

    The following example adds multiple elements (array) to record
    number 0:
        short [] longitudeData = {10, 20, 30};
        longitude = cdf.getVariable("Longitude");
        longitude.putRecord(0L, longitudeData);
```

Parameters:

`recNum` - the record number to which this data belongs

`data` - the data to be added

Returns:

CDFData object containing the user specified data

Throws:

CDFException - if there was a problem writing data

putRecord

```
public CDFData putRecord(java.lang.Object data)
                  throws CDFException
```

Adds a single record to a non-record-varying variable. This method should be used if a record contains one element or multiple elements.

```
    The following example adds a scalar data to record number 0:
        longitude = cdf.getVariable("Longitude");
        longitude.putRecord(new Short((short)200));

    The following example adds multiple elements (array) to record
    number 0:
        short [] longitudeData = {10, 20, 30};
        longitude = cdf.getVariable("Longitude");
        longitude.putRecord(longitudeData);
```

Parameters:

data - the data to be added

Returns:

CDFData object containing the user specified data

Throws:

CDFException - if there was a problem writing data

putHyperData

```
public CDFData putHyperData(long recNum,
                    long recCount,
                    long recInterval,
                    long[] dimIndices,
                    long[] dimCounts,
                    long[] dimIntervals,
                    java.lang.Object data)
                      throws CDFException
```

Writes one or more values from the current z variable. The values are written based on the current record number, current record count, current record interval, current dimension indices, current dimension counts, and current dimension intervals. The values read are put into an CDFData object. Although this method returns a CDFData object, it is not necessary to capture the return value to a CDFData variable.

Let's assume that variable TestData is defined to be 2-dimensional (3x2 - 3 rows and 2 columns). The following example writes the entire record (containing 6 elements) to the first, second, and third records:

```
     long [][][] testData = {{{10,20},{30,40},{50, 60}},
                             {{15,25},{45,55},{75, 85}}
                             {{90,95},{96,97},{2147483648L,4294967295L}}
                             };
     testData.putHyperData (0L, 3L, 1L,
                            new long[] {0, 0},
                            new long[] {3, 2},
                            new long[] {1, 1}, testData);
```

The following example will write the first two rows of testData to the first, third, and fifth records:

```
testData.putHyperData (0L, 3L, 2L,
                       new long[] {0, 0},
                       new long[] {2, 2},
                       new long[] {1, 1}, testData);
```

Parameters:

`recNum` - the record number at which data write begins

`recCount` - the number of records to write

`recInterval` - the number of records to skip between writes

`dimIndices` - the dimension index within a record at which data write begins

`dimCounts` - the number of elements to write from dimIndices

`dimIntervals` - the number of elements to skip between writes

`data` - the data to be written

Returns:

CDFData object that contains the variable data specified by recNum, recCount, recInterval, dimIndices, dimCounts, and dimIntervals as well as the information passed to this method plus the number of dimensions and the number of elements for this variable.

Throws:

`CDFException` - if there was a problem writing data

putRecords

```
public CDFData putRecords(long startRec,
                 long recCount,
                 java.lang.Object data)
                   throws CDFException
```

Writes a number of records from the current z variable. The values are written from the starting record number. Although this method returns a CDFData object, it is not necessary to capture the return value to a CDFData variable.

Let's assume that variable TestData is defined to be 2-dimensional (3x2 - 3 rows and 2 columns). The following example writes three records to the variable as the first, second, and third record:

```
long [][][] testData = {{{10,20},{30,40},{50, 60}},
                        {{15,25},{45,55},{75, 85}}
                        {{90,95},{96,97},{2147483648L,4294967295L}}
                        };
testData.putRecords (0L, 3L, testData);
```

Parameters:

`startRec` - the starting record number at which data write begins

`recCount` - the number of records to write

`data` - the data to be written

Returns:

CDFData object that contains the variable data specified by recNum, recCount, recInterval, dimIndices, dimCounts, and dimIntervals as well as the information passed to this method plus the number of dimensions and the number of elements for this variable.

Throws:

`CDFException` - if there was a problem writing data

getMyCDF

```
public CDF getMyCDF()
```

Gets the CDF object to which this variable belongs.

Returns:

the CDF object to which this variable belongs

getCompressionType

```
public long getCompressionType()
```

Gets the compression type of this variable.

Returns:

the compression type of this variable

getCompressionPct

```
public long getCompressionPct()
```

Gets the compression percentage rate of this variable.

Returns:

the compression percentage rate of this variable

getCompressionParms

```
public long[] getCompressionParms()
```

Sets the compression parameters of this variable. This is only applicable for the GZIP compression method.

Returns:

the compression parameters of this variable

setCompression

```
public void setCompression(long cType,
                  long[] cParms)
                    throws CDFException
```

Sets the compression type and parameters for this variable.

Parameters:

`cType` - the compression type

`cParms` - the compression parameters that go with cType

Throws:

`CDFException` - if a problem occurs setting compression type and parameters

getCompression

```
public java.lang.String getCompression()
                                throws CDFException
```

Gets the string representation of the compression type and parameters set for this variable.

Returns:

the string representation of the compression type and parameters for this variable

Throws:

`CDFException` - if a problem occurs getting the compression type and parameters

getNumDims

```
public long getNumDims()
```

Gets the number of dimensions for this variable.

Returns:

the number of dimensions for this variable

getDimSizes

```
public long[] getDimSizes()
```

Gets the dimensions size of this variable.

Returns:

the dimension size of this variable

getNumElements

```
public long getNumElements()
```

Gets the number of elements for this variable. For CDF_CHAR and CDF_UCHAR this is the number of characters in the string. For all other types this defaults to 1.

Returns:

the number of elements for this variable

getName

```
public java.lang.String getName()
```

Gets the name of this variable.

Specified by:

`getName` in interface `CDFObject`

Returns:

the name of this variable

getID

```
public long getID()
```

Gets the ID of this variable.

Returns:

the ID of this variable

toString

```
public java.lang.String toString()
```

Gets the name of this variable.

Overrides:

`toString` in class `java.lang.Object`

Returns:

the name of this variable

setRecVariance

```
public void setRecVariance(long recVariance)
                    throws CDFException
```

Sets the record variance for this variable.

Parameters:

`recVariance` - the record variance that should be either VARY or NOVARY.

Throws:

`CDFException` - if a problem occurs setting the record variance

getRecVariance

```
public boolean getRecVariance()
```

Gets the value of record variance.

Returns:

True if this variable is record varying, False otherwise

setDimVariances

```
public void setDimVariances(long[] dimVariances)
                     throws CDFException
```

Sets the dimension variances for this variable.

Parameters:

`dimVariances` - the dimension variances for this variable

Throws:

`CDFException` - if a problem occurs setting the dimension variances

getDimVariances

```
public long[] getDimVariances()
```

Gets the dimension variances for this variable.

Returns:

the dimension variances for this variable

getDataType

```
public long getDataType()
```

Gets the CDF data type of this variable.

Returns:

the CDF data type of this variable

deleteRecords

```
public void deleteRecords(long firstRec,
                  long lastRec)
                    throws CDFException
```

Deletes a range of records from this variable. While non-sparse variable records after the last deleted record will be renumbered, the sparse variable records will not.

Parameters:

`firstRec` - the first record to be deleted

`lastRec` - the last record to be deleted

Throws:

`CDFException` - if a problem occurs deleting records

deleteRecordsRenumber

```
public void deleteRecordsRenumber(long firstRec,
                          long lastRec)
                            throws CDFException
```

Deletes a range of records from this variable. All records after the last deleted record are renumbered for both the sparse or non-sparse variable after the deletion.

Parameters:

`firstRec` - the first record to be deleted

`lastRec` - the last record to be deleted

Throws:

`CDFException` - if a problem occurs deleting records

allocateBlock

```
public void allocateBlock(long firstRec,
                  long lastRec)
                    throws CDFException
```

Allocates a range of records for this variable.

Parameters:

`firstRec` - the first record to be allocated

`lastRec` - the last record to be allocated

Throws:

`CDFException` - if a problem occurs allocating records

allocateRecords

```
public void allocateRecords(long num0toRecords)
                      throws CDFException
```

Allocates a number of records, starting from record number 0.

Parameters:

`num0toRecords` - the number of records to be allocated

Throws:

`CDFException` - if a problem occurs allocating records

getNumWrittenRecords

```
public long getNumWrittenRecords()
                           throws CDFException
```

Gets the number of records physically written (not allocated) for this variable.

Returns:

the number of records written physically

Throws:

`CDFException` - if a problem occurs getting the number of records written physically

getMaxWrittenRecord

```
public long getMaxWrittenRecord()
                          throws CDFException
```

Gets the last written record number, beginning with 0.

Returns:

the last written record number

Throws:

`CDFException` - if a problem occurs getting the last written record number

getNumAllocatedRecords

```
public long getNumAllocatedRecords()
                            throws CDFException
```

Gets the number of records allocated for this variable.

Returns:

the number of records allocated

Throws:

`CDFException` - if a problem occurs getting the number of records allocated

getMaxAllocatedRecord

```
public long getMaxAllocatedRecord()
                           throws CDFException
```

Gets the maximum allocated record number for this variable.

Returns:

the maximum allocated record number

Throws:

`CDFException` - if a problem occurs getting the maximum allocated record number

setPadValue

```
public void setPadValue(java.lang.Object padValue)
                 throws CDFException
```

Sets the pad value for this variable. This pad value is used, when storing data, for undefined values.

Parameters:

`padValue` - the pad value to be used for undefined values

Throws:

`CDFException` - if a problem occurs setting the pad value

checkPadValueExistence

```
public boolean checkPadValueExistence()
```

```
                                    throws CDFException
```

Checks if the pad value has been defined for this variable.

Returns:

Whether the user-defined pad value exists. It is either true or false.

- true - pad value has been specified.
- false - pad value is not specified.

Throws:

`CDFException` - if a problem occurs checking the existence of the pad value Note: The getPadValue() method will return a pad value, whether it is the user defined or the default.

getPadValue

```
public java.lang.Object getPadValue()
                                throws CDFException
```

Gets the pad value for this variable.

Returns:

the pad value set for this variable

Throws:

`CDFException` - if a problem occurs reading the pad value

setSparseRecords

```
public void setSparseRecords(long sparseRecords)
                      throws CDFException
```

Sets the sparse record type for this variable.

Parameters:

`sparseRecords` - sparse record type that should be one of the following types:

- NO_SPARSERECORDS - The variable doesn't have sparse records.
- PAD_SPARSERECORDS - The variable has pad-missing records.
- PREV_SPARSERECORDS - The variable has previous-missing records.

Throws:

`CDFException` - if a problem occurs setting the sparse record type

getSparseRecords

```
public long getSparseRecords()
```

Gets the sparse record type for this variable.

Returns:

one of the following sparse record type is returned:

- NO_SPARSERECORDS - means that no sparse records are defined
- PAD_SPARSERECORDS - means that the variable's pad value is used when reading values from a missing record
- PREV_SPARSERECORDS - means that values from the previous existing records are used when reading values from a missing record

setBlockingFactor

```
public void setBlockingFactor(long blockingFactor)
                       throws CDFException
```

Sets the blocking factor for this variable. The blocking factor has no effect for Non-Record varying (NRV) variables or muti-file CDFs.

Parameters:

`blockingFactor` - the blocking factor - a value of zero (0) indicates that the default blocking factor should be used

Throws:

`CDFException` - if a problem occurs setting the blocking factor

getBlockingFactor

```
public long getBlockingFactor()
                       throws CDFException
```

Gets the blocking factor for this variable.

Returns:

the blocking factor set this variable

Throws:

`CDFException` - if a problem occurs getting the blocking factor set for this variable

setInitialRecords

```
public void setInitialRecords(long nRecords)
                       throws CDFException
```

Sets the number of records to be written initially for this variable.

Parameters:

`nRecords` - the number of records to be written initially

Throws:

`CDFException` - if a problem occurs writing initial records

selectCacheSize

```
public void selectCacheSize(long cacheSize)
                     throws CDFException
```

Sets the number of 512-byte cache buffers to be used. This operation is not applicable for a single-file CDF.

Parameters:

`cacheSize` - the number of 512-byte cache buffers

Throws:

`CDFException` - if a problem occurs allocating cache buffers

confirmCacheSize

```
public long confirmCacheSize()
                     throws CDFException
```

Gets the number of 512-byte cache buffers defined for this variable.

Returns:

the number of 512-byte cache buffers set for this variable

Throws:

CDFException - if a problem occurs getting the number of cache buffers set for this variable

selectReservePercent

```
public void selectReservePercent(long reservePercent)
                         throws CDFException
```

Sets the reserve percentage to be used for this variable. This operation is only applicable to compressed z Variables. The Concepts chapter in the CDF User's Guide describes the reserve percentage scheme used by the CDF library.

Parameters:

reservePercent - the reserve percentage to be used

Throws:

CDFException - if a problem occurs setting a reserve percentage

confirmReservePercent

```
public long confirmReservePercent()
                          throws CDFException
```

Gets the reserve percentage set for this variable. This operation is only applicable to compressed z Variables.

Returns:

the reserve percentage set for this variable

Throws:

CDFException - if a problem occurs getting the reserve percentage

confirmPadValue

```
public long confirmPadValue()
                    throws CDFException
```

Checks the existence of an explicitly specified pad value for the current z variable. If an explicit pad value has not been specified, the informational status code NO_PADVALUE_SPECIFIED is returned. Otherwise, CDF_OK is returned.

Returns:

Existence of pad value. If no pad value is specified for this variable, NO_PADVALUE_SPECIFIED is returned. If a pad value has been specified, then CDF_OK is returned.

Throws:

CDFException - if a problem occurs checking the existence of pad value. Note: for NO_PADVALUE_SPECIFIED, getPadValue method will return the default pad value.

getAllocatedFrom

```
public long getAllocatedFrom(long recNum)
                      throws CDFException
```

Inquires the next allocated record at or after a given record for this variable.

Parameters:

recNum - The record number at which to begin searching for the next allocated record. If this record exists, it will be considered the next allocated record.

Returns:

the number of the next allocated record

Throws:

CDFException - if a problem occurs getting the number of the next allocated record

getAllocatedTo

```
public long getAllocatedTo(long firstRec)
                    throws CDFException
```

Inquires the last allocated record (before the next unallocated record) at or after a given record for this variable.

Parameters:

firstRec - the record number at which to begin searching for the last allocated record

Returns:

the number of the last allocated record

Throws:

CDFException - if a problem occurs getting the number of the last allocated record

updateDataSpec

```
public void updateDataSpec(long dataType,
                  long numElements)
                    throws CDFException
```

Update the data specification (data type and number of elements) of the variable.

Parameters:

dataType - the data type of the entry data

numElements - the number of elements fo the entry data

Throws:

CDFException - An exception is thrown if the operation fails

getAttributes

```
public java.util.Vector getAttributes()
```

Returns the variable attributes that are associated with this variable.

```
     The following example describes how to retrieve all the variable
     attributes that are associated with a particular variable.

        Variable v = cdf.getVariable("myVariable");
        Vector   attrs = v.getAttributes();
        if (attrs.size() > 0) {
            for (Enumeration e=attrs.elements(); e.hasMoreElements();) {
                 Attribute a = (Attribute) e.nextElement();
                 // manipulate the attribute
            }
        }
```

Returns:

Returns the variable attributes that are associated with this variable.

getAttributesandEntries

```
public java.util.Map getAttributesandEntries()
```

Returns the variable attributes and their entries that are associated with this variable.

```
     The following example describes how to retrieve all the variable
     attributes and their entries that are associated with a particular
     variable.

       Variable v = cdf.getVariable("myVariable");
       Map vaes = v.getAttributesandEntries();
       if (vaes != null) {
         for (Object key : vaes.keySet()) {
           System.oit.println("Attribute:"+key+" entry:"+vaes.get(key));
         }
       }
```

Note: Each attribute's entry, an object, can be a single data or an array of values. CDF Epoch type data will be encoded in date/time string.

Returns:

Returns a Map object that contains variable attribute names as keys and entries as values that are associated with this variable. All keys and values are in object type. If there is no attribute, then null is returned;

getAttributesandEntries

```
public java.util.Map getAttributesandEntries(boolean encoding)
```

Returns the variable attributes and their entries that are associated with this variable.

```
     The following example describes how to retrieve all the variable
     attributes and their entries that are associated with a particular
     variable.

       Variable v = cdf.getVariable("myVariable");
       Map vaes = v.getAttributesandEntries(true);
       if (vaes != null) {
         for (Object key : vaes.keySet()) {
           System.oit.println("Attribute:"+key+" entry:"+vaes.get(key));
         }
       }
```

Note: Each attribute's entry, an object, can be a single data or an array of values.

Parameters:

`encoding` - A true/false flag indicates whether to encode epoch type data to date/time string.

Returns:

Returns a Map object that contains variable attribute names as keys and entries as values that are associated with this variable. All keys and values are in object type. If there is no attribute, then null is returned;

dumpDataWithFormat

```
public void dumpDataWithFormat(java.lang.Object data)
```

Dump the data, either the variable value or metadata, with or without the format. The format comes from the FORMAT variable attribute.

Parameters:

`data` - An object, likely an attribute entry, for output

dumpDataWithFormat

```
public void dumpDataWithFormat(java.lang.Object data,
                               java.io.PrintWriter outWriter)
```

Dump the data, either the variable value or metadata, with or without the format. The format comes from the FORMAT variable attribute.

Parameters:

`data` - An object, likely an attribute entry, for output

`outWriter` - the print writer to which formatted representations of the object/data is printed as a text-output stream

dumpDataWithFormat

```
public void dumpDataWithFormat(CDFData data)
```

Dump the data, either the variable value or metadata, with the format. The format comes from the FORMAT variable attribute.

Parameters:

`data` - A CDFData object, likely variable data, for output

Hierarchy For All Packages

Package Hierarchies:
gsfc.nssdc.cdf, gsfc.nssdc.cdf.util

Class Hierarchy

- java.lang.Object
  - gsfc.nssdc.cdf.**Attribute** (implements gsfc.nssdc.cdf.CDFConstants, gsfc.nssdc.cdf.CDFObject)
  - gsfc.nssdc.cdf.**CDF** (implements gsfc.nssdc.cdf.CDFConstants, gsfc.nssdc.cdf.CDFObject)
  - gsfc.nssdc.cdf.**CDFData** (implements gsfc.nssdc.cdf.CDFConstants, gsfc.nssdc.cdf.CDFObject)
  - gsfc.nssdc.cdf.**CDFNativeLibrary** (implements gsfc.nssdc.cdf.CDFDelegate)
  - gsfc.nssdc.cdf.**CDFTools** (implements gsfc.nssdc.cdf.CDFConstants)
  - gsfc.nssdc.cdf.util.**CDFTT2000** (implements gsfc.nssdc.cdf.CDFConstants)
  - gsfc.nssdc.cdf.util.**CDFUtils** (implements gsfc.nssdc.cdf.CDFConstants)
  - gsfc.nssdc.cdf.**Entry** (implements gsfc.nssdc.cdf.CDFConstants, gsfc.nssdc.cdf.CDFObject)
  - gsfc.nssdc.cdf.util.**Epoch** (implements gsfc.nssdc.cdf.CDFConstants)
  - gsfc.nssdc.cdf.util.**Epoch16** (implements gsfc.nssdc.cdf.CDFConstants)
  - gsfc.nssdc.cdf.util.**EpochNative**
  - java.lang.Throwable (implements java.io.Serializable)
    - java.lang.Exception
      - gsfc.nssdc.cdf.**CDFException** (implements gsfc.nssdc.cdf.CDFConstants)
  - gsfc.nssdc.cdf.**Variable** (implements gsfc.nssdc.cdf.CDFConstants, gsfc.nssdc.cdf.CDFObject)

Interface Hierarchy

- gsfc.nssdc.cdf.**CDFConstants**
- gsfc.nssdc.cdf.**CDFDelegate**
- gsfc.nssdc.cdf.**CDFObject**

Deprecated API

Contents

Deprecated Methods

Method and Description
gsfc.nssdc.cdf.CDF.create(String, int) *Use setFileBackward(long) method to set the file backward flag and create(String) to create file instead.*

Overview Package Class Tree Deprecated Index Help

Prev Next Frames No Frames All Classes

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

Z

A B C D E F G H I L M N O P R S T U V Z

Overview Package Class Tree Deprecated Index Help

Prev Next Frames No Frames All Classes

How This API Document Is Organized

This API (Application Programming Interface) document has pages corresponding to the items in the navigation bar, described as follows.

Overview

The Overview page is the front page of this API document and provides a list of all packages with a summary for each. This page can also contain an overall description of the set of packages.

Package

Each package has a page that contains a list of its classes and interfaces, with a summary for each. This page can contain six categories:

- Interfaces (italic)
- Classes
- Enums
- Exceptions
- Errors
- Annotation Types

Class/Interface

Each class, interface, nested class and nested interface has its own separate page. Each of these pages has three sections consisting of a class/interface description, summary tables, and detailed member descriptions:

- Class inheritance diagram
- Direct Subclasses
- All Known Subinterfaces
- All Known Implementing Classes
- Class/interface declaration
- Class/interface description
- Nested Class Summary
- Field Summary
- Constructor Summary
- Method Summary
- Field Detail
- Constructor Detail
- Method Detail

Each summary entry contains the first sentence from the detailed description for that item. The summary entries are alphabetical, while the detailed descriptions are in the order they appear in the source code. This preserves the logical groupings established by the programmer.

Annotation Type

Each annotation type has its own separate page with the following sections:

- Annotation Type declaration

- Annotation Type description
- Required Element Summary
- Optional Element Summary
- Element Detail

Enum

Each enum has its own separate page with the following sections:

- Enum declaration
- Enum description
- Enum Constant Summary
- Enum Constant Detail

Tree (Class Hierarchy)

There is a Class Hierarchy page for all packages, plus a hierarchy for each package. Each hierarchy page contains a list of classes and a list of interfaces. The classes are organized by inheritance structure starting with `java.lang.Object`. The interfaces do not inherit from `java.lang.Object`.

- When viewing the Overview page, clicking on "Tree" displays the hierarchy for all packages.
- When viewing a particular package, class or interface page, clicking "Tree" displays the hierarchy for only that package.

Deprecated API

The Deprecated API page lists all of the API that have been deprecated. A deprecated API is not recommended for use, generally due to improvements, and a replacement API is usually given. Deprecated APIs may be removed in future implementations.

Index

The Index contains an alphabetic list of all classes, interfaces, constructors, methods, and fields.

Prev/Next

These links take you to the next or previous class, interface, package, or related page.

Frames/No Frames

These links show and hide the HTML frames. All pages are available with or without frames.

All Classes

The All Classes link shows all classes and interfaces except non-static nested types.

Serialized Form

Each serializable or externalizable class has a description of its serialization fields and methods. This information is of interest to re-implementors, not to developers using the API. While there is no link in the navigation bar, you can get to this information by going to any serialized class and clicking "Serialized Form" in the "See also" section of the class description.

Constant Field Values

The Constant Field Values page lists the static final fields and their values.

This help file applies to API documentation generated using the standard doclet.

Packages

Package	Description
gsfc.nssdc.cdf	
gsfc.nssdc.cdf.util	

Overview Package Class Tree Deprecated Index Help

Prev Next Frames No Frames All Classes

Package gsfc.nssdc.cdf

Interface Summary

Interface	Description
CDFConstants	This class defines the constants used by the CDF library and CDF Java APIs, and it mimics the cdf.h include file from the cdf distribution.
CDFDelegate	This class defines the method that is responsible for acting as the gateway between the CDF Java code and the CDF library.
CDFObject	CDFObject provides the base interface for all CDF objects.

Class Summary

Class	Description
Attribute	This class contains the methods that are associated with either global or variable attributes.
CDF	The CDF class is the main class used to interact with a CDF file.
CDFData	This class acts as the glue between the Java code and the Java Native Interface (JNI) code.
CDFNativeLibrary	This class implements the method that act as the gateway between the CDF Java APIs and the CDF library.
CDFTools	CDFTools.java Created: Tue Nov 24 16:14:50 1998
Entry	This class describes a CDF global or variable attribute entry.
Variable	The **Variable** class defines a CDF variable.

Exception Summary

Exception	Description
CDFException	This class defines the informational, warning, and error messages that can arise from CDF operations.

Package gsfc.nssdc.cdf.util

Class Summary

Class	Description
CDFTT2000	This class contains the handy utility routines (methods) called by the CDF applications to handle epoch data of CDF's CDF_TIME_TT2000 data type.
CDFUtils	This class contains the handy utility routines (methods) called by the core CDF Java APIs.
Epoch	This class contains the handy utility routines (methods) called by the CDF applications to handle epoch data of CDF's CDF_EPOCH data type.
Epoch16	This class contains the handy utility routines (methods) called by the CDF applications to handle epoch data of CDF's CDF_EPOCH16 data type.
EpochNative	The Epoch class is a Java wrapper to the CDF epoch handling routines.

Overview Package Class Tree Deprecated Index Help

Prev Package Next Package Frames No Frames All Classes

Hierarchy For Package gsfc.nssdc.cdf

Package Hierarchies:
All Packages

Class Hierarchy

- java.lang.Object
  - gsfc.nssdc.cdf.**Attribute** (implements gsfc.nssdc.cdf.CDFConstants, gsfc.nssdc.cdf.CDFObject)
  - gsfc.nssdc.cdf.**CDF** (implements gsfc.nssdc.cdf.CDFConstants, gsfc.nssdc.cdf.CDFObject)
  - gsfc.nssdc.cdf.**CDFData** (implements gsfc.nssdc.cdf.CDFConstants, gsfc.nssdc.cdf.CDFObject)
  - gsfc.nssdc.cdf.**CDFNativeLibrary** (implements gsfc.nssdc.cdf.CDFDelegate)
  - gsfc.nssdc.cdf.**CDFTools** (implements gsfc.nssdc.cdf.CDFConstants)
  - gsfc.nssdc.cdf.**Entry** (implements gsfc.nssdc.cdf.CDFConstants, gsfc.nssdc.cdf.CDFObject)
  - java.lang.Throwable (implements java.io.Serializable)
    - java.lang.Exception
      - gsfc.nssdc.cdf.**CDFException** (implements gsfc.nssdc.cdf.CDFConstants)
  - gsfc.nssdc.cdf.**Variable** (implements gsfc.nssdc.cdf.CDFConstants, gsfc.nssdc.cdf.CDFObject)

Interface Hierarchy

- gsfc.nssdc.cdf.**CDFConstants**
- gsfc.nssdc.cdf.**CDFDelegate**
- gsfc.nssdc.cdf.**CDFObject**

All Classes

Attribute
CDF
CDFConstants
CDFData
CDFDelegate
CDFException
CDFNativeLibrary
CDFObject
CDFTools
CDFTT2000
CDFUtils
Entry
Epoch
Epoch16
EpochNative
Variable

Constant Field Values

Contents

gsfc.nssdc.*

gsfc.nssdc.cdf.CDFConstants

Modifier and Type	Constant Field	Value
public static final long	AHUFF_COMPRESSION	3L
public static final long	ALPHAOSF1_DECODING	13L
public static final long	ALPHAOSF1_ENCODING	13L
public static final long	ALPHAVMSd_DECODING	14L
public static final long	ALPHAVMSd_ENCODING	14L
public static final long	ALPHAVMSg_DECODING	15L
public static final long	ALPHAVMSg_ENCODING	15L
public static final long	ALPHAVMSi_DECODING	16L
public static final long	ALPHAVMSi_ENCODING	16L
public static final long	ARM_BIG_DECODING	18L
public static final long	ARM_BIG_ENCODING	18L
public static final long	ARM_LITTLE_DECODING	17L
public static final long	ARM_LITTLE_ENCODING	17L
public static final long	ATTR_	85L
public static final long	ATTR_EXISTENCE_	95L
public static final long	ATTR_EXISTS	-2001L
public static final long	ATTR_MAXgENTRY_	89L
public static final long	ATTR_MAXrENTRY_	91L
public static final long	ATTR_MAXzENTRY_	93L
public static final long	ATTR_NAME_	87L
public static final long	ATTR_NAME_TRUNC	-1001L
public static final long	ATTR_NUMBER_	88L
public static final long	ATTR_NUMgENTRIES_	90L
public static final long	ATTR_NUMrENTRIES_	92L
public static final long	ATTR_NUMzENTRIES_	94L
public static final long	ATTR_SCOPE_	86L
public static final long	BACKWARD_	1010L
public static final long	BACKWARDFILEoff	0L
public static final long	BACKWARDFILEon	1L
public static final long	BAD_ALLOCATE_RECS	-2015L
public static final long	BAD_ARGUMENT	-2022L
public static final long	BAD_ATTR_NAME	-2044L
public static final long	BAD_ATTR_NUM	-2042L
public static final long	BAD_BLOCKING_FACTOR	-2031L

public static final long	BAD_CACHE_SIZE	-2063L
public static final long	BAD_CDF_EXTENSION	-2016L
public static final long	BAD_CDF_ID	-2002L
public static final long	BAD_CDF_NAME	-2049L
public static final long	BAD_CDFSTATUS	-2034L
public static final long	BAD_CHECKSUM	-2225L
public static final long	BAD_COMPRESSION_PARM	-2097L
public static final long	BAD_DATA_TYPE	-2003L
public static final long	BAD_DECODING	-2079L
public static final long	BAD_DIM_COUNT	-2039L
public static final long	BAD_DIM_INDEX	-2005L
public static final long	BAD_DIM_INTERVAL	-2040L
public static final long	BAD_DIM_SIZE	-2004L
public static final long	BAD_ENCODING	-2006L
public static final long	BAD_ENTRY_NUM	-2043L
public static final long	BAD_FNC_OR_ITEM	-2058L
public static final long	BAD_FORMAT	-2014L
public static final long	BAD_INITIAL_RECS	-2030L
public static final long	BAD_MAJORITY	-2007L
public static final long	BAD_MALLOC	-2026L
public static final long	BAD_NEGtoPOSfp0_MODE	-2081L
public static final long	BAD_NUM_DIMS	-2008L
public static final long	BAD_NUM_ELEMS	-2011L
public static final long	BAD_NUM_STRINGS	-2238L
public static final long	BAD_NUM_VARS	-2036L
public static final long	BAD_READONLY_MODE	-2073L
public static final long	BAD_REC_COUNT	-2037L
public static final long	BAD_REC_INTERVAL	-2038L
public static final long	BAD_REC_NUM	-2009L
public static final long	BAD_SCOPE	-2010L
public static final long	BAD_SCRATCH_DIR	-2111L
public static final long	BAD_SPARSEARRAYS_PARM	-2110L
public static final long	BAD_VAR_NAME	-2045L
public static final long	BAD_VAR_NUM	-2041L
public static final long	BAD_zMODE	-2072L
public static final long	BADDATE_LEAPSECOND_UPDATED	-2235L
public static final double	BeginUnixTimeEPOCH	6.21672192E13
public static final double	BeginUnixTimeEPOCH16	6.21672192E10
public static final long	BLOCKINGFACTOR_TOO_LARGE	1014L
public static final long	BLOCKINGFACTOR_TOO_SMALL	1011L
public static final long	BLOCKINGFACTOR_TOO_SMALL2	1012L
public static final long	CANNOT_ALLOCATE_RECORDS	-2103L
public static final long	CANNOT_CHANGE	-2051L
public static final long	CANNOT_COMPRESS	-2091L
public static final long	CANNOT_CONVERT_WIDECHAR	-2239L
public static final long	CANNOT_COPY	-2104L
public static final long	CANNOT_INSERT_RECORDS	-2233L
public static final long	CANNOT_SPARSEARRAYS	-2100L
public static final long	CANNOT_SPARSERECORDS	-2099L
public static final long	CDF_	1L
public static final long	CDF_ACCESS_	201L

public static final long	CDF_ATTR_NAME_LEN	64L
public static final long	CDF_ATTR_NAME_LEN256	256L
public static final long	CDF_BYTE	41L
public static final long	CDF_CACHESIZE_	117L
public static final long	CDF_CHAR	51L
public static final long	CDF_CHECKSUM_	156L
public static final long	CDF_CLOSE_ERROR	-2055L
public static final long	CDF_COMPRESSION_	130L
public static final long	CDF_COPYRIGHT_	7L
public static final long	CDF_COPYRIGHT_LEN	256L
public static final long	CDF_CREATE_ERROR	-2066L
public static final long	CDF_DECODING_	4L
public static final long	CDF_DELETE_ERROR	-2088L
public static final long	CDF_DOUBLE	45L
public static final long	CDF_ENCODING_	3L
public static final long	CDF_EPOCH	31L
public static final long	CDF_EPOCH16	32L
public static final long	CDF_EXISTS	-2013L
public static final long	CDF_FLOAT	44L
public static final long	CDF_FORMAT_	6L
public static final long	CDF_INCREMENT_	15L
public static final long	CDF_INFO_	129L
public static final long	CDF_INT1	1L
public static final long	CDF_INT2	2L
public static final long	CDF_INT4	4L
public static final long	CDF_INT8	8L
public static final long	CDF_INTERNAL_ERROR	-2035L
public static final long	CDF_LEAPSECONDLASTUPDATED_	159L
public static final long	CDF_MAJORITY_	5L
public static final long	CDF_MAX_DIMS	10L
public static final long	CDF_MAX_PARMS	5L
public static final long	CDF_MIN_DIMS	0L
public static final long	CDF_NAME_	2L
public static final long	CDF_NAME_TRUNC	-1002L
public static final long	CDF_NEGtoPOSfp0_MODE_	19L
public static final long	CDF_NUMATTRS_	10L
public static final long	CDF_NUMgATTRS_	11L
public static final long	CDF_NUMrVARS_	8L
public static final long	CDF_NUMvATTRS_	12L
public static final long	CDF_NUMzVARS_	9L
public static final long	CDF_OK	0L
public static final long	CDF_OPEN_ERROR	-2012L
public static final long	CDF_PATHNAME_LEN	512L
public static final long	CDF_READ_ERROR	-2074L
public static final long	CDF_READONLY_MODE_	17L
public static final long	CDF_REAL4	21L
public static final long	CDF_REAL8	22L
public static final long	CDF_RELEASE_	14L
public static final long	CDF_SAVE_ERROR	-2083L
public static final long	CDF_SCRATCHDIR_	149L
public static final long	CDF_STATUS_	16L

public static final long	CDF_STATUSTEXT_LEN	80L
public static final long	CDF_TIME_TT2000	33L
public static final long	CDF_UCHAR	52L
public static final long	CDF_UINT1	11L
public static final long	CDF_UINT2	12L
public static final long	CDF_UINT4	14L
public static final long	CDF_VAR_NAME_LEN	64L
public static final long	CDF_VAR_NAME_LEN256	256L
public static final long	CDF_VERSION_	13L
public static final long	CDF_WARN	-2000L
public static final long	CDF_WRITE_ERROR	-2075L
public static final long	CDF_zMODE_	18L
public static final long	CDFwithSTATS_	200L
public static final long	CHECKSUM_	1012L
public static final long	CHECKSUM_ERROR	-2226L
public static final long	CHECKSUM_NOT_ALLOWED	-2227L
public static final long	CLOSE_	1004L
public static final long	COLUMN_MAJOR	2L
public static final long	COMPRESS_CACHESIZE_	155L
public static final long	COMPRESSION_ERROR	-2093L
public static final long	CONFIRM_	1006L
public static final long	CORRUPTED_V2_CDF	-2028L
public static final long	CORRUPTED_V3_CDF	-2223L
public static final long	CREATE_	1001L
public static final long	CURgENTRY_EXISTENCE_	126L
public static final long	CURrENTRY_EXISTENCE_	127L
public static final long	CURzENTRY_EXISTENCE_	128L
public static final long	DATATYPE_MISMATCH	-2112L
public static final long	DATATYPE_SIZE_	125L
public static final long	DECOMPRESSION_ERROR	-2092L
public static final long	DECSTATION_DECODING	4L
public static final long	DECSTATION_ENCODING	4L
public static final byte	DEFAULT_BYTE_PADVALUE	-127
public static final char	DEFAULT_CHAR_PADVALUE	32
public static final double	DEFAULT_DOUBLE_PADVALUE	-1.0E30
public static final double	DEFAULT_EPOCH_PADVALUE	0.0
public static final double	DEFAULT_EPOCH16_PADVALUE	0.0
public static final float	DEFAULT_FLOAT_PADVALUE	-1.0000000150474662E30f
public static final byte	DEFAULT_INT1_PADVALUE	-127
public static final short	DEFAULT_INT2_PADVALUE	-32767
public static final int	DEFAULT_INT4_PADVALUE	-2147483647
public static final long	DEFAULT_INT8_PADVALUE	-9223372036854775807L
public static final float	DEFAULT_REAL4_PADVALUE	-1.0000000150474662E30f
public static final double	DEFAULT_REAL8_PADVALUE	-1.0E30
public static final long	DEFAULT_TT2000_PADVALUE	-9223372036854775807L
public static final char	DEFAULT_UCHAR_PADVALUE	32
public static final short	DEFAULT_UINT1_PADVALUE	254
public static final int	DEFAULT_UINT2_PADVALUE	65534
public static final long	DEFAULT_UINT4_PADVALUE	4294967294L
public static final long	DELETE_	1003L
public static final long	DID_NOT_COMPRESS	1002L

public static final long	EMPTY_COMPRESSED_CDF	-2096L
public static final long	END_OF_VAR	-2032L
public static final long	EPOCH_STRING_LEN	24L
public static final long	EPOCH_STRING_LEN_EXTEND	36L
public static final long	EPOCH1_STRING_LEN	16L
public static final long	EPOCH1_STRING_LEN_EXTEND	24L
public static final long	EPOCH2_STRING_LEN	14L
public static final long	EPOCH2_STRING_LEN_EXTEND	14L
public static final long	EPOCH3_STRING_LEN	24L
public static final long	EPOCH3_STRING_LEN_EXTEND	36L
public static final long	EPOCH4_STRING_LEN	23L
public static final long	EPOCH4_STRING_LEN_EXTEND	32L
public static final long	EPOCHx_FORMAT_MAX	68L
public static final long	EPOCHx_STRING_MAX	50L
public static final long	FILLED_TT2000_VALUE	-9223372036854775808L
public static final long	FORCED_PARAMETER	-1006L
public static final long	FUNCTION_NOT_SUPPORTED	-2236L
public static final long	gENTRY_	96L
public static final long	gENTRY_DATA_	101L
public static final long	gENTRY_DATASPEC_	100L
public static final long	gENTRY_DATATYPE_	98L
public static final long	gENTRY_EXISTENCE_	97L
public static final long	gENTRY_NUMELEMS_	99L
public static final long	GET_	1007L
public static final long	GETCDFCHECKSUM_	1013L
public static final long	GETCDFFILEBACKWARD_	1011L
public static final long	GETCDFVALIDATE_	1015L
public static final long	GETLEAPSECONDSENVVAR_	1016L
public static final long	GLOBAL_SCOPE	1L
public static final long	GZIP_COMPRESSION	5L
public static final long	HOST_DECODING	8L
public static final long	HOST_ENCODING	8L
public static final long	HP_DECODING	11L
public static final long	HP_ENCODING	11L
public static final long	HUFF_COMPRESSION	2L
public static final long	IA64VMSd_DECODING	20L
public static final long	IA64VMSd_ENCODING	20L
public static final long	IA64VMSg_DECODING	21L
public static final long	IA64VMSg_ENCODING	21L
public static final long	IA64VMSi_DECODING	19L
public static final long	IA64VMSi_ENCODING	19L
public static final long	IBM_PC_OVERFLOW	-2023L
public static final long	IBMPC_DECODING	6L
public static final long	IBMPC_ENCODING	6L
public static final long	IBMRS_DECODING	7L
public static final long	IBMRS_ENCODING	7L
public static final long	ILLEGAL_EPOCH_FIELD	-2224L
public static final double	ILLEGAL_EPOCH_VALUE	-1.0
public static final long	ILLEGAL_FOR_SCOPE	-2076L
public static final long	ILLEGAL_IN_zMODE	-2071L
public static final long	ILLEGAL_ON_V1_CDF	-2060L

public static final long	ILLEGAL_TT2000_VALUE	-9223372036854775805L
public static final long	IS_A_NETCDF	-2228L
public static final long	LIB_COPYRIGHT_	20L
public static final long	LIB_INCREMENT_	23L
public static final long	LIB_RELEASE_	22L
public static final long	LIB_subINCREMENT_	24L
public static final long	LIB_VERSION_	21L
public static final long	MAC_DECODING	9L
public static final long	MAC_ENCODING	9L
public static final long	MD5_CHECKSUM	1L
public static final long	MULTI_FILE	2L
public static final long	MULTI_FILE_FORMAT	1007L
public static final long	NA_FOR_VARIABLE	-1007L
public static final long	NEGATIVE_FP_ZERO	-1004L
public static final long	NEGtoPOSfp0off	0L
public static final long	NEGtoPOSfp0on	-1L
public static final long	NETWORK_DECODING	1L
public static final long	NETWORK_ENCODING	1L
public static final long	NeXT_DECODING	12L
public static final long	NeXT_ENCODING	12L
public static final long	NO_ATTR_SELECTED	-2046L
public static final long	NO_CDF_SELECTED	-2053L
public static final long	NO_CHECKSUM	0L
public static final long	NO_COMPRESSION	0L
public static final long	NO_DELETE_ACCESS	-2087L
public static final long	NO_ENTRY_SELECTED	-2047L
public static final long	NO_MORE_ACCESS	-2077L
public static final long	NO_PADVALUE_SPECIFIED	1005L
public static final long	NO_SPARSEARRAYS	0L
public static final long	NO_SPARSERECORDS	0L
public static final long	NO_STATUS_SELECTED	-2052L
public static final long	NO_SUCH_ATTR	-2017L
public static final long	NO_SUCH_CDF	-2067L
public static final long	NO_SUCH_ENTRY	-2018L
public static final long	NO_SUCH_RECORD	-2102L
public static final long	NO_SUCH_VAR	-2019L
public static final long	NO_VAR_SELECTED	-2048L
public static final long	NO_VARS_IN_CDF	1006L
public static final long	NO_WRITE_ACCESS	-2086L
public static final long	NONE_CHECKSUM	0L
public static final long	NOT_A_CDF	-2027L
public static final long	NOT_A_CDF_OR_NOT_SUPPORTED	-2113L
public static final long	NOVARY	0L
public static final long	NULL_	1000L
public static final long	OPEN_	1002L
public static final long	OPTIMAL_ENCODING_TREES	0L
public static final long	OTHER_CHECKSUM	2L
public static final long	PAD_SPARSERECORDS	1L
public static final long	PPC_DECODING	9L
public static final long	PPC_ENCODING	9L
public static final long	PRECEEDING_RECORDS_ALLOCATED	1009L

public static final long	PREV_SPARSERECORDS	2L
public static final long	PUT_	1008L
public static final long	READ_ONLY_DISTRIBUTION	-2054L
public static final long	READ_ONLY_MODE	-2070L
public static final long	READONLYoff	0L
public static final long	READONLYon	-1L
public static final long	rENTRY_	102L
public static final long	rENTRY_DATA_	108L
public static final long	rENTRY_DATASPEC_	107L
public static final long	rENTRY_DATATYPE_	105L
public static final long	rENTRY_EXISTENCE_	104L
public static final long	rENTRY_NAME_	103L
public static final long	rENTRY_NUMELEMS_	106L
public static final long	rENTRY_NUMSTRINGS_	162L
public static final long	rENTRY_STRINGSDATA_	160L
public static final long	RLE_COMPRESSION	1L
public static final long	RLE_OF_ZEROs	0L
public static final long	ROW_MAJOR	1L
public static final long	rVAR_	35L
public static final long	rVAR_ALLOCATEBLOCK_	140L
public static final long	rVAR_ALLOCATEDFROM_	143L
public static final long	rVAR_ALLOCATEDTO_	144L
public static final long	rVAR_ALLOCATERECS_	123L
public static final long	rVAR_BLOCKINGFACTOR_	51L
public static final long	rVAR_CACHESIZE_	120L
public static final long	rVAR_COMPRESSION_	137L
public static final long	rVAR_DATA_	42L
public static final long	rVAR_DATASPEC_	48L
public static final long	rVAR_DATATYPE_	37L
public static final long	rVAR_DIMVARYS_	40L
public static final long	rVAR_EXISTENCE_	54L
public static final long	rVAR_HYPERDATA_	43L
public static final long	rVAR_INITIALRECS_	50L
public static final long	rVAR_MAXallocREC_	47L
public static final long	rVAR_MAXREC_	46L
public static final long	rVAR_NAME_	36L
public static final long	rVAR_nINDEXENTRIES_	53L
public static final long	rVAR_nINDEXLEVELS_	148L
public static final long	rVAR_nINDEXRECORDS_	52L
public static final long	rVAR_NUMallocRECS_	142L
public static final long	rVAR_NUMBER_	41L
public static final long	rVAR_NUMELEMS_	38L
public static final long	rVAR_NUMRECS_	141L
public static final long	rVAR_PADVALUE_	49L
public static final long	rVAR_RECORDS_	152L
public static final long	rVAR_RECORDS_RENUMBER_	157L
public static final long	rVAR_RECVARY_	39L
public static final long	rVAR_RESERVEPERCENT_	150L
public static final long	rVAR_SEQDATA_	44L
public static final long	rVAR_SEQPOS_	45L
public static final long	rVAR_SPARSEARRAYS_	139L

public static final long	rVAR_SPARSERECORDS_	138L
public static final long	rVARs_CACHESIZE_	118L
public static final long	rVARs_DIMCOUNTS_	33L
public static final long	rVARs_DIMINDICES_	32L
public static final long	rVARs_DIMINTERVALS_	34L
public static final long	rVARs_DIMSIZES_	26L
public static final long	rVARs_MAXREC_	27L
public static final long	rVARs_NUMDIMS_	25L
public static final long	rVARs_RECCOUNT_	30L
public static final long	rVARs_RECDATA_	28L
public static final long	rVARs_RECINTERVAL_	31L
public static final long	rVARs_RECNUMBER_	29L
public static final long	SAVE_	1009L
public static final long	SCRATCH_CREATE_ERROR	-2107L
public static final long	SCRATCH_DELETE_ERROR	-2106L
public static final long	SCRATCH_READ_ERROR	-2108L
public static final long	SCRATCH_WRITE_ERROR	-2109L
public static final long	SELECT_	1005L
public static final long	SGi_DECODING	5L
public static final long	SGi_ENCODING	5L
public static final long	SINGLE_FILE	1L
public static final long	SINGLE_FILE_FORMAT	1004L
public static final long	SOME_ALREADY_ALLOCATED	1008L
public static final long	STAGE_CACHESIZE_	154L
public static final long	STATUS_TEXT_	116L
public static final long	STRING_NOT_UTF8_ENCODING	1013L
public static final long	SUN_DECODING	2L
public static final long	SUN_ENCODING	2L
public static final long	TOO_MANY_PARMS	-2101L
public static final long	TOO_MANY_VARS	-2024L
public static final long	TRY_TO_READ_NONSTRING_DATA	-2237L
public static final long	TT2000_0_STRING_LEN	30L
public static final long	TT2000_1_STRING_LEN	19L
public static final long	TT2000_2_STRING_LEN	14L
public static final long	TT2000_3_STRING_LEN	29L
public static final long	TT2000_4_STRING_LEN	30L
public static final long	TT2000_CDF_MAYNEEDUPDATE	1010L
public static final long	TT2000_TIME_ERROR	-2229L
public static final long	TT2000_USED_OUTDATED_TABLE	-2234L
public static final long	UNABLE_TO_PROCESS_CDF	-2230L
public static final long	UNKNOWN_COMPRESSION	-2090L
public static final long	UNKNOWN_SPARSENESS	-2098L
public static final long	UNSUPPORTED_OPERATION	-2082L
public static final long	VALIDATE_	1014L
public static final long	VALIDATEFILEoff	0L
public static final long	VALIDATEFILEon	-1L
public static final long	VAR_ALREADY_CLOSED	1003L
public static final long	VAR_CLOSE_ERROR	-2056L
public static final long	VAR_CREATE_ERROR	-2068L
public static final long	VAR_DELETE_ERROR	-2089L
public static final long	VAR_EXISTS	-2025L

public static final long	**VAR_NAME_TRUNC**	-1003L
public static final long	**VAR_OPEN_ERROR**	-2029L
public static final long	**VAR_READ_ERROR**	-2020L
public static final long	**VAR_SAVE_ERROR**	-2084L
public static final long	**VAR_WRITE_ERROR**	-2021L
public static final long	**VARIABLE_SCOPE**	2L
public static final long	**VARY**	-1L
public static final long	**VAX_DECODING**	3L
public static final long	**VAX_ENCODING**	3L
public static final long	**VIRTUAL_RECORD_DATA**	1001L
public static final long	**zENTRY_**	109L
public static final long	**zENTRY_DATA_**	115L
public static final long	**zENTRY_DATASPEC_**	114L
public static final long	**zENTRY_DATATYPE_**	112L
public static final long	**zENTRY_EXISTENCE_**	111L
public static final long	**zENTRY_NAME_**	110L
public static final long	**zENTRY_NUMELEMS_**	113L
public static final long	**zENTRY_NUMSTRINGS_**	163L
public static final long	**zENTRY_STRINGSDATA_**	161L
public static final long	**ZLIB_COMPRESS_ERROR**	-2231L
public static final long	**ZLIB_UNCOMPRESS_ERROR**	-2232L
public static final long	**zMODEoff**	0L
public static final long	**zMODEon1**	1L
public static final long	**zMODEon2**	2L
public static final long	**zVAR_**	57L
public static final long	**zVAR_ALLOCATEBLOCK_**	134L
public static final long	**zVAR_ALLOCATEDFROM_**	145L
public static final long	**zVAR_ALLOCATEDTO_**	146L
public static final long	**zVAR_ALLOCATERECS_**	124L
public static final long	**zVAR_BLOCKINGFACTOR_**	75L
public static final long	**zVAR_CACHESIZE_**	121L
public static final long	**zVAR_COMPRESSION_**	131L
public static final long	**zVAR_DATA_**	66L
public static final long	**zVAR_DATASPEC_**	72L
public static final long	**zVAR_DATATYPE_**	59L
public static final long	**zVAR_DIMCOUNTS_**	83L
public static final long	**zVAR_DIMINDICES_**	82L
public static final long	**zVAR_DIMINTERVALS_**	84L
public static final long	**zVAR_DIMSIZES_**	62L
public static final long	**zVAR_DIMVARYS_**	64L
public static final long	**zVAR_EXISTENCE_**	78L
public static final long	**zVAR_HYPERDATA_**	67L
public static final long	**zVAR_INITIALRECS_**	74L
public static final long	**zVAR_MAXallocREC_**	71L
public static final long	**zVAR_MAXREC_**	70L
public static final long	**zVAR_NAME_**	58L
public static final long	**zVAR_nINDEXENTRIES_**	77L
public static final long	**zVAR_nINDEXLEVELS_**	147L
public static final long	**zVAR_nINDEXRECORDS_**	76L
public static final long	**zVAR_NUMallocRECS_**	136L
public static final long	**zVAR_NUMBER_**	65L

public static final long	zVAR_NUMDIMS_	61L
public static final long	zVAR_NUMELEMS_	60L
public static final long	zVAR_NUMRECS_	135L
public static final long	zVAR_PADVALUE_	73L
public static final long	zVAR_RECCOUNT_	80L
public static final long	zVAR_RECINTERVAL_	81L
public static final long	zVAR_RECNUMBER_	79L
public static final long	zVAR_RECORDS_	153L
public static final long	zVAR_RECORDS_RENUMBER_	158L
public static final long	zVAR_RECVARY_	63L
public static final long	zVAR_RESERVEPERCENT_	151L
public static final long	zVAR_SEQDATA_	68L
public static final long	zVAR_SEQPOS_	69L
public static final long	zVAR_SPARSEARRAYS_	133L
public static final long	zVAR_SPARSERECORDS_	132L
public static final long	zVARs_CACHESIZE_	119L
public static final long	zVARs_MAXREC_	55L
public static final long	zVARs_RECDATA_	56L
public static final long	zVARs_RECNUMBER_	122L

gsfc.nssdc.cdf.CDFTools

Modifier and Type	Constant Field	Value
public static final int	ALL_VALUES	3
public static final int	NAMED_VALUES	4
public static final int	NO_REPORTS	0
public static final int	NO_VALUES	0
public static final int	NRV_VALUES	1
public static final int	REPORT_ERRORS	1
public static final int	REPORT_INFORMATION	4
public static final int	REPORT_WARNINGS	2
public static final int	RV_VALUES	2

Serialized Form

Package gsfc.nssdc.cdf

Class gsfc.nssdc.cdf.CDFException extends java.lang.Exception implements Serializable

Serialized Fields

myStatus

```
long myStatus
```

Hierarchy For Package gsfc.nssdc.cdf.util

Package Hierarchies:
All Packages

Class Hierarchy

- java.lang.Object
  - gsfc.nssdc.cdf.util.**CDFTT2000** (implements gsfc.nssdc.cdf.CDFConstants)
  - gsfc.nssdc.cdf.util.**CDFUtils** (implements gsfc.nssdc.cdf.CDFConstants)
  - gsfc.nssdc.cdf.util.**Epoch** (implements gsfc.nssdc.cdf.CDFConstants)
  - gsfc.nssdc.cdf.util.**Epoch16** (implements gsfc.nssdc.cdf.CDFConstants)
  - gsfc.nssdc.cdf.util.**EpochNative**

Overview Package Class Tree Deprecated Index Help

Prev Next Frames No Frames All Classes