### Packages

<table>
<thead>
<tr>
<th>gsfc.nssdc.cdf</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>gsfc.nssdc.cdf.util</td>
<td></td>
</tr>
</tbody>
</table>
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.

See Also:
CDF, CDFException, Entry, Variable
AHUFF_COMPRESSION, ALPHAOSF1_DECODING, ALPHAOSF1_ENCODING,
ALPHAVMSd_DECODING, ALPHAOSF1_DECODING, ALPHAVMSq_DECODING,
ALPHAVMSg_ENCODING, ALPHAOSFI_DECODING, ALPHAOSFI_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_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_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, CANNOT_ALLOCATE_RECORDS, CANNOT_CHANGE,
CANNOT_COMPRESS, CANNOT_COPY, CANNOT_SPARSEARRAYS,
CANNOT_SPARSERECORDS, CDF_, CDF_ACCESS_, CDF_ATTR_NAME_LEN,
CDF_BYTE, CDF_CACHESIZE_, CDF_CHAR, 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_INTERNAL_ERROR, CDF_MAJORITY_, CDF_MAX_DIMS,
CDF_MAX_PARMS, CDF_MIN_DIMS, CDF_NAME_, CDF_NAME_TRUNC,
CDF_NEGtoPOSfp0_MODE_, CDF_NUMATTRS_, CDF_NUMqATTRS_, 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_SCRATCHDIR_, CDF_STATUS_,
CDF_STATUSTEXT_LEN, CDF_UCHAR, CDF_UINT1, CDF_UINT2, CDF_UINT4,
CDF_VAR_NAME_LEN, CDF_VERSION_, CDF_WARN, CDF_WRITE_ERROR,
CDF_zMODE_, CDFwithSTATS_, 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, DECASTATION_DECODING,
DECASTATION_ENCODING, DEFAULT_BYTE_PADVALUE, DEFAULT_CHAR_PADVALUE,
Method Summary

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

Throws:
CDFException - if a problem occurred in creating an attribute
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

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
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 problem 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 problem deleting the entry

---

**getEntries**

public java.util.Vector *getEntries*()

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 a variable entry for this attribute

---

**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.
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.
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.

Supported dataTypes and their mappings

<table>
<thead>
<tr>
<th>CDF dataType</th>
<th>Java dataType</th>
<th>Read/Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDF_BYTE</td>
<td>java.lang.Byte</td>
<td>Y/Y</td>
</tr>
<tr>
<td>CDF_INT1</td>
<td>java.lang.Byte</td>
<td>Y/Y</td>
</tr>
<tr>
<td>CDF_UINT1</td>
<td>java.lang.Short</td>
<td>Y/Y</td>
</tr>
<tr>
<td>CDF_INT2</td>
<td>java.lang.Short</td>
<td>Y/Y</td>
</tr>
<tr>
<td>CDF_UINT2</td>
<td>java.lang.Integer</td>
<td>Y/Y</td>
</tr>
</tbody>
</table>
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.

See Also:
Attribute, CDFException, Variable

Field Summary

Fields inherited from interface gsfc.nssdc.cdf.CDFConstants
AHUFF_COMPRESSION, ALPHAOSF1_DECODING, ALPHAOSF1_ENCODING, ALPHAVMS_d_DECODING, ALPHAVMS_d_ENCODING, ALPHAVMS_q_DECODING, ALPHAVMS_q_ENCODING, ALPHAVMS_i_DECODING, ALPHAVMS_i_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_CDF_STATUS, 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_MAJORORITY, BAD_MALLOC, BAD_NEGtoPOSfp0_MODE, BAD_NUM_DIMS, BAD_NUM_ELEMS, 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, CANNOT_ALLOCATE_RECORDS, CANNOT_CHANGE,
CANNOT_COMPRESS, CANNOT_COPY, CANNOT_SPARSEARRAYS,
CANNOT_SPARSERECORDS, CDF_, CDF_ACCESS_, CDF_ATTR_NAME_LEN,
CDF_BYTE, CDF_CACHESIZE_, CDF_CHAR, 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_INTERNAL_ERROR, 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_SCRATCHDIR_, CDF_STATUS_,
CDF_STATUSTXT_LEN, CDF_UCHAR, CDF_UINT1, CDF_UINT2, CDF_UINT4,
CDF_VAR_NAME_LEN, CDF_VERSION_, CDF_WARN, CDF_WRITE_ERROR,
CDF_zMODE_, CDFwithSTATS_, 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_FLOAT_PADVALUE, DEFAULT_INT1_PADVALUE,
DEFAULT_INT2_PADVALUE, DEFAULT_INT4_PADVALUE,
DEFAULT_REAL4_PADVALUE, DEFAULT_REAL8_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, EPOCHx_FORMAT_MAX, EPOCHx_STRING_MAX,
FORCED_PARAMETER, gENTRY_, gENTRY_DATA_, gENTRY_DATASPEC_,
gENTRY_DATATYPE_, gENTRY_EXISTENCE_, gENTRY_NUMELEMS_, GET_,
GETCDFFILEBACKWARD_, GLOBAL_SCOPE, GZIP_COMPRESSION, HOST_DECODING,
HOST_ENCODING, HP_DECODING, HP_ENCODING, HUFF_COMPRESSION,
### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void <strong>close</strong>()</td>
<td>Closes this CDF file.</td>
</tr>
<tr>
<td>long <strong>confirmCDFCacheSize</strong>()</td>
<td>Gets the CDF cache size (the number of 512-byte cache buffers) set for this CDF.</td>
</tr>
<tr>
<td>long <strong>confirmCompressCacheSize</strong>()</td>
<td>Gets the number of 512-byte cache buffers being used for the compression scratch file (for the current CDF).</td>
</tr>
<tr>
<td>long <strong>confirmDecoding</strong>()</td>
<td>Gets the CDF decoding method defined for this CDF.</td>
</tr>
<tr>
<td>long <strong>confirmNegtoPosfp0</strong>()</td>
<td>Gets the -0.0 to 0.0 translation flag set for this CDF.</td>
</tr>
<tr>
<td>long <strong>confirmReadOnlyMode</strong>()</td>
<td>Gets the value of the read-only mode flag set for this CDF file.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>long <strong>confirmStageCacheSize</strong> ()</td>
<td>Gets the number of 512-byte cache buffers defined for the staging scratch file.</td>
</tr>
<tr>
<td>long <strong>confirmzMode</strong> ()</td>
<td>Gets the zMode set for this CDF.</td>
</tr>
<tr>
<td>static <strong>CDF</strong> create(java.lang.String path)</td>
<td>Creates a CDF file in the current directory.</td>
</tr>
<tr>
<td>void <strong>delete</strong> ()</td>
<td>Deletes this CDF file.</td>
</tr>
<tr>
<td>void <strong>finalize</strong> ()</td>
<td>Do the necessary cleanup when garbage collector reaps it.</td>
</tr>
<tr>
<td><strong>getAttribute</strong> (long attrNum)</td>
<td>Gets the attribute for the given attribute number.</td>
</tr>
<tr>
<td><strong>getAttribute</strong> (java.lang.String attrName)</td>
<td>Gets the attribute for the given attribute name.</td>
</tr>
<tr>
<td>long <strong>getAttributeID</strong> (java.lang.String attrName)</td>
<td>Gets the id of the given attribute.</td>
</tr>
<tr>
<td>java.util.Vector <strong>getAttributes</strong> ()</td>
<td>Gets all the global and variable attributes defined for this CDF.</td>
</tr>
<tr>
<td>java.lang.String <strong>getCompression</strong> ()</td>
<td>Gets the string representation of the compression type and parameters defined for this CDF.</td>
</tr>
<tr>
<td>long[] <strong>getCompressionParms</strong> ()</td>
<td>Gets the compression parameters set for this CDF.</td>
</tr>
<tr>
<td>long <strong>getCompressionPct</strong> ()</td>
<td>Gets the compression percentage set for this CDF.</td>
</tr>
<tr>
<td>long <strong>getCompressionType</strong> ()</td>
<td>Gets the compression type set for this CDF.</td>
</tr>
<tr>
<td>java.lang.String <strong>getCopyright</strong> ()</td>
<td>Gets the CDF copyright statement for this CDF.</td>
</tr>
<tr>
<td><strong>CDFDelegate</strong> <strong>getDelegate</strong> ()</td>
<td>This is a placeholder for future expansions/extensions.</td>
</tr>
<tr>
<td>Method Type</td>
<td>Method Name</td>
</tr>
<tr>
<td>------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>long</td>
<td>getEncoding()</td>
</tr>
<tr>
<td>static boolean</td>
<td>getFileBackward()</td>
</tr>
<tr>
<td>static int</td>
<td>getFileBackwardEnvVar()</td>
</tr>
<tr>
<td>long</td>
<td>getFormat()</td>
</tr>
<tr>
<td>java.util.Vector</td>
<td>getGlobalAttributes()</td>
</tr>
<tr>
<td>long</td>
<td>getID()</td>
</tr>
<tr>
<td>static java.lang.String</td>
<td>getLibraryCopyright()</td>
</tr>
<tr>
<td>static java.lang.String</td>
<td>getLibraryVersion()</td>
</tr>
<tr>
<td>long</td>
<td>getMajority()</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>getName()</td>
</tr>
<tr>
<td>long</td>
<td>getNumAttrs()</td>
</tr>
<tr>
<td>long</td>
<td>getNumGattrs()</td>
</tr>
<tr>
<td>long</td>
<td>getNumRvars()</td>
</tr>
<tr>
<td>long</td>
<td>getNumVars()</td>
</tr>
<tr>
<td>long</td>
<td>getNumVattrs()</td>
</tr>
<tr>
<td>Class/Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>long <code>getNumZvars</code>()</td>
<td>Gets the number of z variables in this CDF file.</td>
</tr>
<tr>
<td>java.util.Vector <code>getOrphanAttributes</code>()</td>
<td>Gets the variable attributes defined for this CDF that are not associated with any variables.</td>
</tr>
<tr>
<td>java.util.Vector <code>getRecord</code> (long recNum, long[] varIDs)</td>
<td>Retrieves a logical record that consists of single variable record(s) from an arbitrary number of CDF variables.</td>
</tr>
<tr>
<td>java.util.Vector <code>getRecord</code> (long recNum, long[] varIDs, long[] status)</td>
<td>Retrieves a logical record that consists of single variable record(s) from an arbitrary number of CDF variables.</td>
</tr>
<tr>
<td>java.util.Vector <code>getRecord</code> (long recNum, java.lang.String[] strVars)</td>
<td>Retrieves a logical record that consists of single variable record(s) from an arbitrary number of CDF variables.</td>
</tr>
<tr>
<td>java.util.Vector <code>getRecord</code> (long recNum, java.lang.String[] strVars, long[] status)</td>
<td>Retrieves a logical record that consists of single variable record(s) from an arbitrary number of CDF variables.</td>
</tr>
<tr>
<td>long <code>getStatus</code>()</td>
<td>Gets the status of the most recent CDF JNI/library function call.</td>
</tr>
<tr>
<td>static java.lang.String <code>getStatusText</code> (long statusCode)</td>
<td>Gets the status text of the most recent CDF JNI/library function call.</td>
</tr>
<tr>
<td>Variable <code>getVariable</code> (long varNum)</td>
<td>Gets the variable object for the given variable number.</td>
</tr>
<tr>
<td>Variable <code>getVariable</code> (java.lang.String varName)</td>
<td>Gets the variable object for the given variable name.</td>
</tr>
<tr>
<td>java.util.Vector <code>getVariableAttributes</code>()</td>
<td>Gets the variable attributes defined for this CDF.</td>
</tr>
<tr>
<td>long <code>getVariableID</code> (java.lang.String varName)</td>
<td>Gets the ID of the given variable.</td>
</tr>
<tr>
<td>java.util.Vector <code>getVariables</code>()</td>
<td>Gets the z variables defined for this CDF.</td>
</tr>
<tr>
<td>java.lang.String <code>getVersion</code>()</td>
<td>Gets the CDF library version that was used to create this CDF (e.g. 2.6.7, etc.).</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>static <code>CDF open</code>(java.lang.String path, long readOnly)</td>
<td>Open a CDF file.</td>
</tr>
<tr>
<td>void <code>putRecord</code>(long recNum, long[] varIDs, java.util.Vector myData)</td>
<td>Writes a logical record that consists of single variable record(s) from an arbitrary number of CDF variables.</td>
</tr>
<tr>
<td>void <code>putRecord</code>(long recNum, long[] varIDs, java.util.Vector myData, long[] status)</td>
<td>Writes a logical record that consists of single variable record(s) from an arbitrary number of CDF variables.</td>
</tr>
<tr>
<td>void <code>putRecord</code>(long recNum, java.lang.String[] strVars, java.util.Vector myData)</td>
<td>Writes a logical record that consists of single variable record(s) from an arbitrary number of CDF variables.</td>
</tr>
<tr>
<td>void <code>putRecord</code>(long recNum, java.lang.String[] strVars, java.util.Vector myData, long[] status)</td>
<td>Writes a logical record that consists of single variable record(s) from an arbitrary number of CDF variables.</td>
</tr>
<tr>
<td>void <code>rename</code>(java.lang.String path)</td>
<td>Renames the current CDF.</td>
</tr>
<tr>
<td>void <code>save</code>()</td>
<td>Saves this CDF file without closing.</td>
</tr>
<tr>
<td>void <code>selectCDFCacheSize</code>(long cacheSize)</td>
<td>Defines the number of 512-byte cache buffers to be used for the dotCDF file (for the current CDF).</td>
</tr>
<tr>
<td>void <code>selectCompressCacheSize</code>(long compressCacheSize)</td>
<td>Sets the number of 512-byte cache buffers to be used for the compression scratch file (for the current CDF).</td>
</tr>
<tr>
<td>void <code>selectDecoding</code>(long decoding)</td>
<td>Defines the CDF decoding method to be used for this CDF.</td>
</tr>
<tr>
<td>void <code>selectNegtoPosfp0</code>(long negtoPosfp0)</td>
<td>Defines whether to translate -0.0 to 0.0 for reading or writing.</td>
</tr>
<tr>
<td>void <code>selectReadOnlyMode</code>(long readOnly)</td>
<td>Sets the desired read-only mode.</td>
</tr>
</tbody>
</table>
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 **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 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 **setMajority**(long majority)

Sets the variable majority for this CDF.

java.lang.String **toString**()

Gets the name of this CDF.

<table>
<thead>
<tr>
<th>Methods inherited from class java.lang.Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>equals, getClass, hashCode, notify, notifyAll, wait, wait, wait</td>
</tr>
</tbody>
</table>

**Method Detail**

**create**

define create(java.lang.String path)
Creates a CDF file in the current directory. By default, a single-file CDF is created. If the user wants to create a multi-file CDF, the user must open a file and change its file format as following:

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

If the above example didn't change its format, it 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 a backward file for cdf1, but a V3.* file for cdf2:

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

Alternatively, use an environment variable to control the backward file creation. The environment variable `CDF_FILEBACKWARD` on Unix or Windows or `CDF $FILEBACKWARD` on Open/VMS is used. When it is set to TRUE, a V2.7 file(s) will be created automatically. In the following example, both cdf1 and cdf2 will be V2.7 if environment variable `CDF_FILEBACKWARD` (or `CDF$FILEBACKWARD`) is TRUE:

```java
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:

Throws `CDFException`
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 problem opening a file

getLibraryVersion

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

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

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

Retrieve library copyright information associated with the CDF library.

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
public long getID()

    Gets the id of this CDF file.

    Returns:
    the id of this CDF file

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.

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:
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
- MAC_DECODING
- HP_DECODING
- NeXT_DECODING
- ALPHAOSF1_DECODING
- ALPHAVMSd_DECODING
- ALPHAVMSg_DECODING
- ALPHAVMSi_DECODING

Throws:

CDFException - if there was a problem setting the requested encoding 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 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

```java
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

```java
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

```java
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 library 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.

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 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 compression
- 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.

**cParams** - 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**

```java
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
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

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.

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 setting the backward flag for this CDF file

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

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

Gets the indication of the set CDF_FILEBACKWARD environment variable.

**Returns:**
- 1 if the environment variable is set to true, 0 if not set or set to anything but true.

**Throws:**
- `CDFException` - if there was a problem

getStatus

```java
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.

```java
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:

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

---

**toString**

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

Gets the name of this CDF.

**Returns:**
the name of this CDF

---

**finalize**

```java
public void finalize()
```

throws java.lang.Throwable

Do the necessary cleanup when garbage collector reaps it.

**Throws:**
java.lang.Throwable - if there was a problem doing cleanup

---

**getDelegate**

```java
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.

---

**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
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
Gets the global attributes defined for this CDF.

**Returns:**
A vector that contains the global attributes defined in this CDF

---

**getVariableAttributes**

```java
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**

```java
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**

```java
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 rVariables. 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.

    Note: Since all CDFs opened or create with the CDFJava APIs are placed into zMODE 2, there
    are no rVariables. 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**

```java
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.

```java
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**

```java
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.

---

```java
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.

```java
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.println("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.println("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", "Temperature"};
    int[] longitude_data = {333, 444, 555};
    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
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.
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.

```java
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
```
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

<table>
<thead>
<tr>
<th>static long</th>
<th>AHUFF_COMPRESSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>static long</td>
<td>ALPHAOSF1_DECODING</td>
</tr>
<tr>
<td>static long</td>
<td>ALPHAOSF1_ENCODING</td>
</tr>
<tr>
<td>static long</td>
<td>ALPHAVMSd_DECODING</td>
</tr>
<tr>
<td>static long</td>
<td>ALPHAVMSd_ENCODING</td>
</tr>
<tr>
<td>static long</td>
<td>ALPHAVMSg_DECODING</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>static long</td>
<td>ALPHAVMSg_ENCODING</td>
</tr>
<tr>
<td>static long</td>
<td>ALPHAVMSi_DECODING</td>
</tr>
<tr>
<td>static long</td>
<td>ALPHAVMSi_ENCODING</td>
</tr>
<tr>
<td>static long</td>
<td>ATTR_</td>
</tr>
<tr>
<td>static long</td>
<td>ATTR_EXISTENCE_</td>
</tr>
<tr>
<td>static long</td>
<td>ATTR_EXISTS</td>
</tr>
<tr>
<td>static long</td>
<td>ATTR_MAXgENTRY_</td>
</tr>
<tr>
<td>static long</td>
<td>ATTR_MAXrENTRY_</td>
</tr>
<tr>
<td>static long</td>
<td>ATTR_MAXzENTRY_</td>
</tr>
<tr>
<td>static long</td>
<td>ATTR_NAME_</td>
</tr>
<tr>
<td>static long</td>
<td>ATTR_NAME_TRUNC</td>
</tr>
<tr>
<td>static long</td>
<td>ATTR_NUMBER_</td>
</tr>
<tr>
<td>static long</td>
<td>ATTR_NUMgENTRIES_</td>
</tr>
<tr>
<td>static long</td>
<td>ATTR_NUMrENTRIES_</td>
</tr>
<tr>
<td>static long</td>
<td>ATTR_NUMzENTRIES_</td>
</tr>
<tr>
<td>static long</td>
<td><strong>ATTR_SCOPE</strong></td>
</tr>
<tr>
<td>static long</td>
<td><strong>BACKWARD</strong></td>
</tr>
<tr>
<td>static long</td>
<td><strong>BACKWARDFILEoff</strong></td>
</tr>
<tr>
<td>static long</td>
<td><strong>BACKWARDFILEon</strong></td>
</tr>
<tr>
<td>static long</td>
<td><strong>BAD_ALLOCATE_RECS</strong></td>
</tr>
<tr>
<td>static long</td>
<td><strong>BAD_ARGUMENT</strong></td>
</tr>
<tr>
<td>static long</td>
<td><strong>BAD_ATTR_NAME</strong></td>
</tr>
<tr>
<td>static long</td>
<td><strong>BAD_ATTR_NUM</strong></td>
</tr>
<tr>
<td>static long</td>
<td><strong>BAD_BLOCKING_FACTOR</strong></td>
</tr>
<tr>
<td>static long</td>
<td><strong>BAD_CACHE_SIZE</strong></td>
</tr>
<tr>
<td>static long</td>
<td><strong>BAD_CDF_EXTENSION</strong></td>
</tr>
<tr>
<td>static long</td>
<td><strong>BAD_CDF_ID</strong></td>
</tr>
<tr>
<td>static long</td>
<td><strong>BAD_CDF_NAME</strong></td>
</tr>
<tr>
<td>static long</td>
<td><strong>BAD_CDFSTATUS</strong></td>
</tr>
<tr>
<td>static long</td>
<td><strong>BAD_COMPRESSION_PARM</strong></td>
</tr>
<tr>
<td>static long</td>
<td><strong>BAD_DATA_TYPE</strong></td>
</tr>
<tr>
<td>static long</td>
<td>BAD_DECODING</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>static long</td>
<td>BAD_DIM_COUNT</td>
</tr>
<tr>
<td>static long</td>
<td>BAD_DIM_INDEX</td>
</tr>
<tr>
<td>static long</td>
<td>BAD_DIM_INTERVAL</td>
</tr>
<tr>
<td>static long</td>
<td>BAD_DIM_SIZE</td>
</tr>
<tr>
<td>static long</td>
<td>BAD_ENCODING</td>
</tr>
<tr>
<td>static long</td>
<td>BAD_ENTRY_NUM</td>
</tr>
<tr>
<td>static long</td>
<td>BAD_FNC_OR_ITEM</td>
</tr>
<tr>
<td>static long</td>
<td>BAD_FORMAT</td>
</tr>
<tr>
<td>static long</td>
<td>BAD_INITIAL_RECS</td>
</tr>
<tr>
<td>static long</td>
<td>BAD_MAJORITY</td>
</tr>
<tr>
<td>static long</td>
<td>BAD_MALLOC</td>
</tr>
<tr>
<td>static long</td>
<td>BAD_NEGtoPOSfp0_MODE</td>
</tr>
<tr>
<td>static long</td>
<td>BAD_NUM_DIMS</td>
</tr>
<tr>
<td>static long</td>
<td>BAD_NUM_ELEMS</td>
</tr>
<tr>
<td>static long</td>
<td>BAD_NUM_VARS</td>
</tr>
<tr>
<td>static long</td>
<td>BAD_READONLY_MODE</td>
</tr>
<tr>
<td>static long</td>
<td>BAD_REC_COUNT</td>
</tr>
<tr>
<td>static long</td>
<td>BAD_REC_INTERVAL</td>
</tr>
<tr>
<td>static long</td>
<td>BAD_REC_NUM</td>
</tr>
<tr>
<td>static long</td>
<td>BAD_SCOPE</td>
</tr>
<tr>
<td>static long</td>
<td>BAD_SCRATCH_DIR</td>
</tr>
<tr>
<td>static long</td>
<td>BAD_SPARSEARRAYS_PARM</td>
</tr>
<tr>
<td>static long</td>
<td>BAD_VAR_NAME</td>
</tr>
<tr>
<td>static long</td>
<td>BAD_VAR_NUM</td>
</tr>
<tr>
<td>static long</td>
<td>BAD_zMODE</td>
</tr>
<tr>
<td>static long</td>
<td>CANNOT_ALLOCATE_RECORDS</td>
</tr>
<tr>
<td>static long</td>
<td>CANNOT_CHANGE</td>
</tr>
<tr>
<td>static long</td>
<td>CANNOT_COMPRESS</td>
</tr>
<tr>
<td>static long</td>
<td>CANNOT_COPY</td>
</tr>
<tr>
<td>static long</td>
<td>CANNOT_SPARSEARRAYS</td>
</tr>
<tr>
<td>static long</td>
<td>CANNOT_SPARSERECORDS</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_ACCESS_</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_ATTR_NAME_LEN</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_BYTE</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_CACHESIZE_</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_CHAR</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_CLOSE_ERROR</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_COMPRESSION_</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_COPYRIGHT_</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_COPYRIGHT_LEN</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_CREATE_ERROR</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_DECODING_</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_DELETE_ERROR</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_DOUBLE</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_ENCODING_</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_EPOCH</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_EPOCH16</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_EXISTS</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_FLOAT</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_FORMAT_</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_INCREMENT_</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_INFO_</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_INT1</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_INT2</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_INT4</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_INTERNAL_ERROR</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_MAJORITY_</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_MAX_DIMS</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_MAX_PARMS</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_MIN_DIMS</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_NAME_</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_NAME_TRUNC</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_NEGtoPOSfp0_MODE</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_NUMATTRS</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_NUMgATTRS</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_NUMrVARS</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_NUMvATTRS</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_NUMzVARS</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_OK</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_OPEN_ERROR</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_PATHNAME_LEN</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_READ_ERROR</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_READONLY_MODE</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_REAL4</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_REAL8</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_RELEASE</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_SCRATCHDIR</td>
</tr>
<tr>
<td>static long</td>
<td>CDF_STATUS</td>
</tr>
<tr>
<td>static long</td>
<td>__CDF_STATUSTEXT_LEN</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>static long</td>
<td>__CDF_UCHAR</td>
</tr>
<tr>
<td>static long</td>
<td>__CDF_UINT1</td>
</tr>
<tr>
<td>static long</td>
<td>__CDF_UINT2</td>
</tr>
<tr>
<td>static long</td>
<td>__CDF_UINT4</td>
</tr>
<tr>
<td>static long</td>
<td>__CDF_VAR_NAME_LEN</td>
</tr>
<tr>
<td>static long</td>
<td><strong>CDF_VERSION</strong></td>
</tr>
<tr>
<td>static long</td>
<td>__CDF_WARN</td>
</tr>
<tr>
<td>static long</td>
<td>__CDF_WRITE_ERROR</td>
</tr>
<tr>
<td>static long</td>
<td><strong>CDF_zMODE</strong></td>
</tr>
<tr>
<td>static long</td>
<td><strong>CDFwithSTATS</strong></td>
</tr>
<tr>
<td>static long</td>
<td><strong>CLOSE</strong></td>
</tr>
<tr>
<td>static long</td>
<td>__COLUMN_MAJOR</td>
</tr>
<tr>
<td>static long</td>
<td><strong>COMPRESS_CACHESIZE</strong></td>
</tr>
<tr>
<td>static long</td>
<td>__COMPRESSION_ERROR</td>
</tr>
<tr>
<td>static long</td>
<td><strong>CONFIRM</strong></td>
</tr>
<tr>
<td>Type</td>
<td>Value</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>static long</td>
<td>CORRUPTED_V2_CDF</td>
</tr>
<tr>
<td>static long</td>
<td>CORRUPTED_V3_CDF</td>
</tr>
<tr>
<td>static long</td>
<td>CREATE</td>
</tr>
<tr>
<td>static long</td>
<td>CURgENTRY_EXISTENCE</td>
</tr>
<tr>
<td>static long</td>
<td>CURrENTRY_EXISTENCE</td>
</tr>
<tr>
<td>static long</td>
<td>CURzENTRY_EXISTENCE</td>
</tr>
<tr>
<td>static long</td>
<td>DATATYPE_MISMATCH</td>
</tr>
<tr>
<td>static long</td>
<td>DATATYPE_SIZE</td>
</tr>
<tr>
<td>static long</td>
<td>DECOMPRESSION_ERROR</td>
</tr>
<tr>
<td>static long</td>
<td>DECSTATION_DECODING</td>
</tr>
<tr>
<td>static long</td>
<td>DECSTATION_ENCODING</td>
</tr>
<tr>
<td>static byte</td>
<td>DEFAULT_BYTE_PADVALUE</td>
</tr>
<tr>
<td>static char</td>
<td>DEFAULT_CHAR_PADVALUE</td>
</tr>
<tr>
<td>static double</td>
<td>DEFAULT_DOUBLE_PADVALUE</td>
</tr>
<tr>
<td>static double</td>
<td>DEFAULT_EPOCH_PADVALUE</td>
</tr>
<tr>
<td>static float</td>
<td>DEFAULT_FLOAT_PADVALUE</td>
</tr>
<tr>
<td>Type</td>
<td>Value</td>
</tr>
<tr>
<td>----------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>static byte</td>
<td><code>DEFAULT_INT1_PADVALUE</code></td>
</tr>
<tr>
<td>static short</td>
<td><code>DEFAULT_INT2_PADVALUE</code></td>
</tr>
<tr>
<td>static int</td>
<td><code>DEFAULT_INT4_PADVALUE</code></td>
</tr>
<tr>
<td>static float</td>
<td><code>DEFAULT_REAL4_PADVALUE</code></td>
</tr>
<tr>
<td>static double</td>
<td><code>DEFAULT_REAL8_PADVALUE</code></td>
</tr>
<tr>
<td>static char</td>
<td><code>DEFAULT_UCHAR_PADVALUE</code></td>
</tr>
<tr>
<td>static short</td>
<td><code>DEFAULT_UINT1_PADVALUE</code></td>
</tr>
<tr>
<td>static int</td>
<td><code>DEFAULT_UINT2_PADVALUE</code></td>
</tr>
<tr>
<td>static long</td>
<td><code>DEFAULT_UINT4_PADVALUE</code></td>
</tr>
<tr>
<td>static long</td>
<td><code>DELETE_</code></td>
</tr>
<tr>
<td>static long</td>
<td><code>DID_NOT_COMPRESS</code></td>
</tr>
<tr>
<td>static long</td>
<td><code>EMPTY_COMPRESSED_CDF</code></td>
</tr>
<tr>
<td>static long</td>
<td><code>END_OF_VAR</code></td>
</tr>
<tr>
<td>static long</td>
<td><code>EPOCH_STRING_LEN</code></td>
</tr>
<tr>
<td>static long</td>
<td><code>EPOCH_STRING_LEN_EXTEND</code></td>
</tr>
<tr>
<td>static long</td>
<td><code>EPOCH1_STRING_LEN</code></td>
</tr>
<tr>
<td>Static long</td>
<td>EPOCH1_STRING_LEN_EXTEND</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------</td>
</tr>
<tr>
<td>Static long</td>
<td>EPOCH2_STRING_LEN</td>
</tr>
<tr>
<td>Static long</td>
<td>EPOCH2_STRING_LEN_EXTEND</td>
</tr>
<tr>
<td>Static long</td>
<td>EPOCH3_STRING_LEN</td>
</tr>
<tr>
<td>Static long</td>
<td>EPOCH3_STRING_LEN_EXTEND</td>
</tr>
<tr>
<td>Static long</td>
<td>EPOCHx_FORMAT_MAX</td>
</tr>
<tr>
<td>Static long</td>
<td>EPOCHx_STRING_MAX</td>
</tr>
<tr>
<td>Static long</td>
<td>FORCED_PARAMETER</td>
</tr>
<tr>
<td>Static long</td>
<td>gENTRY_</td>
</tr>
<tr>
<td>Static long</td>
<td>gENTRY_DATA_</td>
</tr>
<tr>
<td>Static long</td>
<td>gENTRY_DATASPEC_</td>
</tr>
<tr>
<td>Static long</td>
<td>gENTRY_DATATYPE_</td>
</tr>
<tr>
<td>Static long</td>
<td>gENTRY_EXISTENCE_</td>
</tr>
<tr>
<td>Static long</td>
<td>gENTRY_NUMELEMS_</td>
</tr>
<tr>
<td>Static long</td>
<td>GET_</td>
</tr>
<tr>
<td>Static long</td>
<td>GETCDIFFILEBACKWARD_</td>
</tr>
<tr>
<td>static long</td>
<td>ILLEGAL_ON_V1_CDF</td>
</tr>
<tr>
<td>static long</td>
<td>LIB_COPYRIGHT_</td>
</tr>
<tr>
<td>static long</td>
<td>LIB_INCREMENT_</td>
</tr>
<tr>
<td>static long</td>
<td>LIB_RELEASE_</td>
</tr>
<tr>
<td>static long</td>
<td>LIB_subINCREMENT_</td>
</tr>
<tr>
<td>static long</td>
<td>LIB_VERSION_</td>
</tr>
<tr>
<td>static long</td>
<td>MAC_DECODING</td>
</tr>
<tr>
<td>static long</td>
<td>MAC_ENCODING</td>
</tr>
<tr>
<td>static long</td>
<td>MULTI_FILE</td>
</tr>
<tr>
<td>static long</td>
<td>MULTI_FILE_FORMAT</td>
</tr>
<tr>
<td>static long</td>
<td>NA_FOR_VARIABLE</td>
</tr>
<tr>
<td>static long</td>
<td>NEGATIVE_FP_ZERO</td>
</tr>
<tr>
<td>static long</td>
<td>NEGtoPOSfp0off</td>
</tr>
<tr>
<td>static long</td>
<td>NEGtoPOSfp0on</td>
</tr>
<tr>
<td>static long</td>
<td>NETWORK_DECODING</td>
</tr>
<tr>
<td>static long</td>
<td>NETWORK_ENCODING</td>
</tr>
<tr>
<td>static long</td>
<td>NeXT_DECODING</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------</td>
</tr>
<tr>
<td>static long</td>
<td>NeXT_ENCODING</td>
</tr>
<tr>
<td>static long</td>
<td>NO_ATTR_SELECTED</td>
</tr>
<tr>
<td>static long</td>
<td>NO_CDF_SELECTED</td>
</tr>
<tr>
<td>static long</td>
<td>NO_COMPRESSION</td>
</tr>
<tr>
<td>static long</td>
<td>NO_DELETE_ACCESS</td>
</tr>
<tr>
<td>static long</td>
<td>NO_ENTRY_SELECTED</td>
</tr>
<tr>
<td>static long</td>
<td>NO_MORE_ACCESS</td>
</tr>
<tr>
<td>static long</td>
<td>NO_PADVALUE_SPECIFIED</td>
</tr>
<tr>
<td>static long</td>
<td>NO_SPARSEARRAYS</td>
</tr>
<tr>
<td>static long</td>
<td>NO_SPARSERECORDS</td>
</tr>
<tr>
<td>static long</td>
<td>NO_STATUS_SELECTED</td>
</tr>
<tr>
<td>static long</td>
<td>NO_SUCH_ATTR</td>
</tr>
<tr>
<td>static long</td>
<td>NO_SUCH_CDF</td>
</tr>
<tr>
<td>static long</td>
<td>NO_SUCH_ENTRY</td>
</tr>
<tr>
<td>static long</td>
<td>NO_SUCH_RECORD</td>
</tr>
<tr>
<td>static long</td>
<td>NO_SUCH_VAR</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>static long</td>
<td>NO_VAR_SELECTED</td>
</tr>
<tr>
<td>static long</td>
<td>NO_VARS_IN_CDF</td>
</tr>
<tr>
<td>static long</td>
<td>NO_WRITE_ACCESS</td>
</tr>
<tr>
<td>static long</td>
<td>NOT_A_CDF</td>
</tr>
<tr>
<td>static long</td>
<td>NOT_A_CDF_OR_NOT_SUPPORTED</td>
</tr>
<tr>
<td>static long</td>
<td>NOVARY</td>
</tr>
<tr>
<td>static long</td>
<td>NULL_</td>
</tr>
<tr>
<td>static long</td>
<td>OPEN_</td>
</tr>
<tr>
<td>static long</td>
<td>OPTIMAL_ENCODING_TREES</td>
</tr>
<tr>
<td>static long</td>
<td>PAD_SPARSERECORDS</td>
</tr>
<tr>
<td>static long</td>
<td>PRECEEDING_RECORDS_ALLOCATED</td>
</tr>
<tr>
<td>static long</td>
<td>PREV_SPARSERECORDS</td>
</tr>
<tr>
<td>static long</td>
<td>PUT_</td>
</tr>
<tr>
<td>static long</td>
<td>READ_ONLY_DISTRIBUTION</td>
</tr>
<tr>
<td>static long</td>
<td>READ_ONLY_MODE</td>
</tr>
<tr>
<td>static long</td>
<td><code>READONLYoff</code></td>
</tr>
<tr>
<td>static long</td>
<td><code>READONLYon</code></td>
</tr>
<tr>
<td>static long</td>
<td><code>rENTRY_</code></td>
</tr>
<tr>
<td>static long</td>
<td><code>rENTRY_DATA_</code></td>
</tr>
<tr>
<td>static long</td>
<td><code>rENTRY_DATASPEC_</code></td>
</tr>
<tr>
<td>static long</td>
<td><code>rENTRY_DATATYPE_</code></td>
</tr>
<tr>
<td>static long</td>
<td><code>rENTRY_EXISTENCE_</code></td>
</tr>
<tr>
<td>static long</td>
<td><code>rENTRY_NAME_</code></td>
</tr>
<tr>
<td>static long</td>
<td><code>rENTRY_NUMELEM_</code></td>
</tr>
<tr>
<td>static long</td>
<td><code>RLE_COMPRESSION</code></td>
</tr>
<tr>
<td>static long</td>
<td><code>RLE_OF_ZEROS</code></td>
</tr>
<tr>
<td>static long</td>
<td><code>ROW_MAJOR</code></td>
</tr>
<tr>
<td>static long</td>
<td><code>rVAR_</code></td>
</tr>
<tr>
<td>static long</td>
<td><code>rVAR_ALLOCATEBLOCK_</code></td>
</tr>
<tr>
<td>static long</td>
<td><code>rVAR_ALLOCATEDFROM_</code></td>
</tr>
<tr>
<td>static long</td>
<td><code>rVAR_ALLOCATEDTO_</code></td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_ALLOCATERECS</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_BLOCKINGFACTOR</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_CACHESIZE</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_COMPRESSION</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_DATA</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_DATASPEC</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_DATATYPE</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_DIMVARYS</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_EXISTENCE</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_HYPERDATA</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_INITIALRECS</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_MAXallocREC</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_MAXREC</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_NAME</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_nINDEXENTRIES</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_nINDEXLEVELS</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_nINDEXRECORDS_</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_NUMALLOCRECS_</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_NUMBER_</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_NUMELEMS_</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_NUMRECS_</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_PADVALUE_</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_RECORDS_</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_RECVARY_</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_RECVARY_</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_RESERVEPERCENT_</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_SEQDATA_</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_SEQPOS_</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_SPARSEARRAYS_</td>
</tr>
<tr>
<td>static long</td>
<td>rVAR_SPARSERECORDS_</td>
</tr>
<tr>
<td>static long</td>
<td>rVARs_CACHESIZE_</td>
</tr>
<tr>
<td>static long</td>
<td>rVARs_DIMCOUNTS_</td>
</tr>
<tr>
<td>static long</td>
<td>rVARs_DIMINDICES_</td>
</tr>
<tr>
<td>static long</td>
<td>rVARs_DIMINTERVALS_</td>
</tr>
<tr>
<td>static long</td>
<td>rVARs_DIMSIZES_</td>
</tr>
<tr>
<td>static long</td>
<td>rVARs_MAXREC_</td>
</tr>
<tr>
<td>static long</td>
<td>rVARs_NUMDIMS_</td>
</tr>
<tr>
<td>static long</td>
<td>rVARs_RECCOUNT_</td>
</tr>
<tr>
<td>static long</td>
<td>rVARs_RECDATA_</td>
</tr>
<tr>
<td>static long</td>
<td>rVARs_RECINTERVAL_</td>
</tr>
<tr>
<td>static long</td>
<td>rVARs_RECNUMBER_</td>
</tr>
<tr>
<td>static long</td>
<td>SAVE_</td>
</tr>
<tr>
<td>static long</td>
<td>SCRATCH_CREATE_ERROR</td>
</tr>
<tr>
<td>static long</td>
<td>SCRATCH_DELETE_ERROR</td>
</tr>
<tr>
<td>static long</td>
<td>SCRATCH_READ_ERROR</td>
</tr>
<tr>
<td>static long</td>
<td>SCRATCH_WRITE_ERROR</td>
</tr>
<tr>
<td>static long</td>
<td>SELECT_</td>
</tr>
<tr>
<td>static long</td>
<td>SGi_DECODING</td>
</tr>
<tr>
<td>static long</td>
<td>SGi_ENCODING</td>
</tr>
<tr>
<td>static long</td>
<td>SINGLE_FILE</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>static long</td>
<td>SINGLE_FILE_FORMAT</td>
</tr>
<tr>
<td>static long</td>
<td>SOME_ALREADY_ALLOCATED</td>
</tr>
<tr>
<td>static long</td>
<td>STAGE_CACHESIZE_</td>
</tr>
<tr>
<td>static long</td>
<td>STATUS_TEXT_</td>
</tr>
<tr>
<td>static long</td>
<td>SUN_DECODING</td>
</tr>
<tr>
<td>static long</td>
<td>SUN_ENCODING</td>
</tr>
<tr>
<td>static long</td>
<td>TOO_MANY_PARMS</td>
</tr>
<tr>
<td>static long</td>
<td>TOO_MANY_VARS</td>
</tr>
<tr>
<td>static long</td>
<td>UNKNOWN_COMPRESSION</td>
</tr>
<tr>
<td>static long</td>
<td>UNKNOWN_SPARSENESS</td>
</tr>
<tr>
<td>static long</td>
<td>UNSUPPORTED_OPERATION</td>
</tr>
<tr>
<td>static long</td>
<td>VAR_ALREADY_CLOSED</td>
</tr>
<tr>
<td>static long</td>
<td>VAR_CLOSE_ERROR</td>
</tr>
<tr>
<td>static long</td>
<td>VAR_CREATE_ERROR</td>
</tr>
<tr>
<td>static long</td>
<td>VAR_DELETE_ERROR</td>
</tr>
<tr>
<td>static long</td>
<td>VARIABLE_SCOPE</td>
</tr>
<tr>
<td>-------------</td>
<td>----------------</td>
</tr>
<tr>
<td>static long</td>
<td>VARY</td>
</tr>
<tr>
<td>static long</td>
<td>VAX_DECODING</td>
</tr>
<tr>
<td>static long</td>
<td>VAX_ENCODING</td>
</tr>
<tr>
<td>static long</td>
<td>VIRTUAL_RECORD_DATA</td>
</tr>
<tr>
<td>static long</td>
<td>zENTRY__</td>
</tr>
<tr>
<td>static long</td>
<td>zENTRY_DATA__</td>
</tr>
<tr>
<td>static long</td>
<td>zENTRY_DATASPEC__</td>
</tr>
<tr>
<td>static long</td>
<td>zENTRY_DATATYPE__</td>
</tr>
<tr>
<td>static long</td>
<td>zENTRY_EXISTENCE__</td>
</tr>
<tr>
<td>static long</td>
<td>zENTRY_NAME__</td>
</tr>
<tr>
<td>static long</td>
<td><strong>zENTRY_NUMELEMS</strong></td>
</tr>
<tr>
<td>static long</td>
<td><strong>zMODEoff</strong></td>
</tr>
<tr>
<td>static long</td>
<td><strong>zMODEon1</strong></td>
</tr>
<tr>
<td>static long</td>
<td><strong>zMODEon2</strong></td>
</tr>
<tr>
<td>static long</td>
<td><strong>zVAR</strong></td>
</tr>
<tr>
<td>static long</td>
<td><strong>zVAR_ALLOCATEBLOCK</strong></td>
</tr>
<tr>
<td>static long</td>
<td><strong>zVAR_ALLOCATEDFROM</strong></td>
</tr>
<tr>
<td>static long</td>
<td><strong>zVAR_ALLOCATEDTO</strong></td>
</tr>
<tr>
<td>static long</td>
<td><strong>zVAR_ALLOCATERECS</strong></td>
</tr>
<tr>
<td>static long</td>
<td><strong>zVAR_BLOCKINGFACTOR</strong></td>
</tr>
<tr>
<td>static long</td>
<td><strong>zVAR_CACHESIZE</strong></td>
</tr>
<tr>
<td>static long</td>
<td><strong>zVAR_COMPRESSION</strong></td>
</tr>
<tr>
<td>static long</td>
<td><strong>zVAR_DATA</strong></td>
</tr>
<tr>
<td>static long</td>
<td><strong>zVAR_DATASPEC</strong></td>
</tr>
<tr>
<td>static long</td>
<td><strong>zVAR_DATATYPE</strong></td>
</tr>
<tr>
<td>static long</td>
<td><strong>zVAR_DIMCOUNTS</strong></td>
</tr>
<tr>
<td>Variable Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>static long zVAR_DIMINDICES_</td>
<td></td>
</tr>
<tr>
<td>static long zVAR_DIMINTERVALS_</td>
<td></td>
</tr>
<tr>
<td>static long zVAR_DIMSIZES_</td>
<td></td>
</tr>
<tr>
<td>static long zVAR_DIMVARYS_</td>
<td></td>
</tr>
<tr>
<td>static long zVAR_EXISTENCE_</td>
<td></td>
</tr>
<tr>
<td>static long zVAR_HYPERDATA_</td>
<td></td>
</tr>
<tr>
<td>static long zVAR_INITIALRECS_</td>
<td></td>
</tr>
<tr>
<td>static long zVAR_MAXallocREC_</td>
<td></td>
</tr>
<tr>
<td>static long zVAR_MAXREC_</td>
<td></td>
</tr>
<tr>
<td>static long zVAR_NAME_</td>
<td></td>
</tr>
<tr>
<td>static long zVAR_nINDEXENTRIES_</td>
<td></td>
</tr>
<tr>
<td>static long zVAR_nINDEXLEVELS_</td>
<td></td>
</tr>
<tr>
<td>static long zVAR_nINDEXRECORDS_</td>
<td></td>
</tr>
<tr>
<td>static long zVAR_NUMallocRECS_</td>
<td></td>
</tr>
<tr>
<td>static long zVAR_NUMBER_</td>
<td></td>
</tr>
<tr>
<td>static long zVAR_NUMDIMS_</td>
<td></td>
</tr>
<tr>
<td>static long</td>
<td>zVAR_NUMELEMS_</td>
</tr>
<tr>
<td>static long</td>
<td>zVAR_NUMRECS_</td>
</tr>
<tr>
<td>static long</td>
<td>zVAR_PADVALUE_</td>
</tr>
<tr>
<td>static long</td>
<td>zVAR_RECCOUNT_</td>
</tr>
<tr>
<td>static long</td>
<td>zVAR_RECINTERVAL_</td>
</tr>
<tr>
<td>static long</td>
<td>zVAR_RECNUMBER_</td>
</tr>
<tr>
<td>static long</td>
<td>zVAR_RECORDS_</td>
</tr>
<tr>
<td>static long</td>
<td>zVAR_RECVAR_</td>
</tr>
<tr>
<td>static long</td>
<td>zVAR_RESERVEPERCENT_</td>
</tr>
<tr>
<td>static long</td>
<td>zVAR_SEQDATA_</td>
</tr>
<tr>
<td>static long</td>
<td>zVAR_SEQPOS_</td>
</tr>
<tr>
<td>static long</td>
<td>zVAR_SPARSEARRAYS_</td>
</tr>
<tr>
<td>static long</td>
<td>zVAR_SPARSERECORDS_</td>
</tr>
<tr>
<td>static long</td>
<td>zVARs_CACHESIZE_</td>
</tr>
<tr>
<td>static long</td>
<td>zVARs_MAXREC_</td>
</tr>
<tr>
<td>static long</td>
<td>zVARs_RECDATA_</td>
</tr>
</tbody>
</table>
static long $zVARs.RECNUMBER_

Field Detail

CDF_MIN_DIMS

public static final long CDF_MIN_DIMS

See Also:
Constant Field Values

CDF_MAX_DIMS

public static final long CDF_MAX_DIMS

See Also:
Constant Field Values

CDF_VAR_NAME_LEN

public static final long CDF_VAR_NAME_LEN

See Also:
Constant Field Values

CDF_ATTR_NAME_LEN

public static final long CDF_ATTR_NAME_LEN

See Also:
Constant Field Values

CDF_COPYRIGHT_LEN

public static final long CDF_COPYRIGHT_LEN

See Also:
    Constant Field Values

CDF_STATUSTEXT_LEN

public static final long CDF_STATUSTEXT_LEN

See Also:
    Constant Field Values

CDF_PATHNAME_LEN

public static final long CDF_PATHNAME_LEN

See Also:
    Constant Field Values

EPOCH_STRING_LEN

public static final long EPOCH_STRING_LEN

See Also:
    Constant Field Values
**EPOCH1_STRING_LEN**

public static final long EPOCH1_STRING_LEN

See Also:
   Constant Field Values

**EPOCH2_STRING_LEN**

public static final long EPOCH2_STRING_LEN

See Also:
   Constant Field Values

**EPOCH3_STRING_LEN**

public static final long EPOCH3_STRING_LEN

See Also:
   Constant Field Values

**EPOCHx_STRING_MAX**

public static final long EPOCHx_STRING_MAX

See Also:
   Constant Field Values

**EPOCHx_FORMAT_MAX**

public static final long EPOCHx_FORMAT_MAX
EPOCH_STRING_LEN_EXTEND

public static final long EPOCH_STRING_LEN_EXTEND

See Also:
Constant Field Values

EPOCH1_STRING_LEN_EXTEND

public static final long EPOCH1_STRING_LEN_EXTEND

See Also:
Constant Field Values

EPOCH2_STRING_LEN_EXTEND

public static final long EPOCH2_STRING_LEN_EXTEND

See Also:
Constant Field Values

EPOCH3_STRING_LEN_EXTEND

public static final long EPOCH3_STRING_LEN_EXTEND

See Also:
Constant Field Values
CDF_INT1

public static final long CDF_INT1

See Also:
Constant Field Values

CDF_INT2

public static final long CDF_INT2

See Also:
Constant Field Values

CDF_INT4

public static final long CDF_INT4

See Also:
Constant Field Values

CDF_UINT1

public static final long CDF_UINT1

See Also:
Constant Field Values

CDF_UINT2

public static final long CDF_UINT2
See Also:
   Constant Field Values

---

**CDF_UINT4**

public static final long CDF_UINT4

See Also:
   Constant Field Values

---

**CDF_REAL4**

public static final long CDF_REAL4

See Also:
   Constant Field Values

---

**CDF_REAL8**

public static final long CDF_REAL8

See Also:
   Constant Field Values

---

**CDF_EPOCH**

public static final long CDF_EPOCH

See Also:
   Constant Field Values
public static final long CDF_EPOCH16

See Also:
Constant Field Values

public static final long CDF_BYTE

See Also:
Constant Field Values

public static final long CDF_FLOAT

See Also:
Constant Field Values

public static final long CDF_DOUBLE

See Also:
Constant Field Values

public static final long CDF_CHAR
public static final long CDF_CHAR

See Also:
Constant Field Values

CDF_UCHAR

public static final long CDF_UCHAR

See Also:
Constant Field Values

NETWORK_ENCODING

public static final long NETWORK_ENCODING

See Also:
Constant Field Values

SUN_ENCODING

public static final long SUN_ENCODING

See Also:
Constant Field Values

VAX_ENCODING

public static final long VAX_ENCODING

See Also:
Constant Field Values
DECSTATION_ENCODING

public static final long DECSTATION_ENCODING

See Also:
    Constant Field Values

SGi_ENCODING

public static final long SGi_ENCODING

See Also:
    Constant Field Values

IBMPC_ENCODING

public static final long IBMPC_ENCODING

See Also:
    Constant Field Values

IBMRS_ENCODING

public static final long IBMRS_ENCODING

See Also:
    Constant Field Values

HOST_ENCODING
public static final long HOST_ENCODING

See Also:
Constant Field Values

MAC_ENCODING

public static final long MAC_ENCODING

See Also:
Constant Field Values

HP_ENCODING

public static final long HP_ENCODING

See Also:
Constant Field Values

NeXT_ENCODING

public static final long NeXT_ENCODING

See Also:
Constant Field Values

ALPHAOSF1_ENCODING

public static final long ALPHAOSF1_ENCODING

See Also:
ALPHAVMSd_ENCODING

public static final long ALPHAVMSd_ENCODING

See Also:
  Constant Field Values

ALPHAVMSg_ENCODING

public static final long ALPHAVMSg_ENCODING

See Also:
  Constant Field Values

ALPHAVMSi_ENCODING

public static final long ALPHAVMSi_ENCODING

See Also:
  Constant Field Values

NETWORK_DECODING

public static final long NETWORK_DECODING

See Also:
  Constant Field Values
SUN_DECODING

public static final long SUN_DECODING

See Also:
Constant Field Values

---

VAX_DECODING

public static final long VAX_DECODING

See Also:
Constant Field Values

---

DECSTATION_DECODING

public static final long DECSTATION_DECODING

See Also:
Constant Field Values

---

SGi_DECODING

public static final long SGi_DECODING

See Also:
Constant Field Values

---

IBMPC_DECODING

public static final long IBMPC_DECODING
IBMRS_DECODING

public static final long IBMRS_DECODING

See Also:
Constant Field Values

HOST_DECODING

public static final long HOST_DECODING

See Also:
Constant Field Values

MAC_DECODING

public static final long MAC_DECODING

See Also:
Constant Field Values

HP_DECODING

public static final long HP_DECODING

See Also:
Constant Field Values
NeXT_DECODING

public static final long NeXT_DECODING

See Also:

Constant Field Values

ALPHAOSF1_DECODING

public static final long ALPHAOSF1_DECODING

See Also:

Constant Field Values

ALPHAVMSd_DECODING

public static final long ALPHAVMSd_DECODING

See Also:

Constant Field Values

ALPHAVMSg_DECODING

public static final long ALPHAVMSg_DECODING

See Also:

Constant Field Values

ALPHAVMSi_DECODING

public static final long ALPHAVMSi_DECODING
See Also:
   Constant Field Values

---

**VARY**

public static final long **VARY**

See Also:
   Constant Field Values

---

**NOVARY**

public static final long **NOVARY**

See Also:
   Constant Field Values

---

**ROW_MAJOR**

public static final long **ROW_MAJOR**

See Also:
   Constant Field Values

---

**COLUMN_MAJOR**

public static final long **COLUMN_MAJOR**

See Also:
   Constant Field Values
SINGLE_FILE

public static final long SINGLE_FILE

See Also:
Constant Field Values

MULTI_FILE

public static final long MULTI_FILE

See Also:
Constant Field Values

GLOBAL_SCOPE

public static final long GLOBAL_SCOPE

See Also:
Constant Field Values

VARIABLE_SCOPE

public static final long VARIABLE_SCOPE

See Also:
Constant Field Values

READONLY on
public static final long READONLYon

See Also:
   Constant Field Values

---

READONLYoff

public static final long READONLYoff

See Also:
   Constant Field Values

---

zMODEoff

public static final long zMODEoff

See Also:
   Constant Field Values

---

zMODEon1

public static final long zMODEon1

See Also:
   Constant Field Values

---

zMODEon2

public static final long zMODEon2

See Also:
   Constant Field Values
NEGtoPOSfp0on

public static final long NEGtoPOSfp0on

See Also:
Constant Field Values

NEGtoPOSfp0off

public static final long NEGtoPOSfp0off

See Also:
Constant Field Values

BACKWARDFILEon

public static final long BACKWARDFILEon

See Also:
Constant Field Values

BACKWARDFILEoff

public static final long BACKWARDFILEoff

See Also:
Constant Field Values

CDF_MAX_PARMS
public static final long CDF_MAX_PARMS

See Also:
    Constant Field Values

---

NO_COMPRESSION

public static final long NO_COMPRESSION

See Also:
    Constant Field Values

---

RLE_COMPRESSION

public static final long RLE_COMPRESSION

See Also:
    Constant Field Values

---

HUFF_COMPRESSION

public static final long HUFF_COMPRESSION

See Also:
    Constant Field Values

---

AHUFF_COMPRESSION

public static final long AHUFF_COMPRESSION

See Also:
Constant Field Values

**GZIP_COMPRESSION**

```java
class ConstantFieldValues {
    public static final long GZIP_COMPRESSION;
}
```

See Also:
- Constant Field Values

**RLE_OF_ZEROS**

```java
class ConstantFieldValues {
    public static final long RLE_OF_ZEROS;
}
```

See Also:
- Constant Field Values

**OPTIMAL_ENCODING_TREES**

```java
class ConstantFieldValues {
    public static final long OPTIMAL_ENCODING_TREES;
}
```

See Also:
- Constant Field Values

**NO_SPARSEARRAYS**

```java
class ConstantFieldValues {
    public static final long NO_SPARSEARRAYS;
}
```

See Also:
- Constant Field Values
**NO_SPARSERECORDS**

public static final long NO_SPARSERECORDS

See Also:
- Constant Field Values

**PAD_SPARSERECORDS**

public static final long PAD_SPARSERECORDS

See Also:
- Constant Field Values

**PREV_SPARSERECORDS**

public static final long PREV_SPARSERECORDS

See Also:
- Constant Field Values

**DEFAULT_BYTE_PADVALUE**

public static final byte DEFAULT_BYTE_PADVALUE

See Also:
- Constant Field Values

**DEFAULT_INT1_PADVALUE**

public static final byte DEFAULT_INT1_PADVALUE
DEFAULT_UINT1_PADVALUE

public static final short DEFAULT_UINT1_PADVALUE

See Also:
Constant Field Values

DEFAULT_INT2_PADVALUE

public static final short DEFAULT_INT2_PADVALUE

See Also:
Constant Field Values

DEFAULT_UINT2_PADVALUE

public static final int DEFAULT_UINT2_PADVALUE

See Also:
Constant Field Values

DEFAULT_INT4_PADVALUE

public static final int DEFAULT_INT4_PADVALUE

See Also:
Constant Field Values
DEFAULT_UINT4_PADVALUE

public static final long DEFAULT_UINT4_PADVALUE

See Also:
   Constant Field Values

DEFAULT_REAL4_PADVALUE

public static final float DEFAULT_REAL4_PADVALUE

See Also:
   Constant Field Values

DEFAULT_FLOAT_PADVALUE

public static final float DEFAULT_FLOAT_PADVALUE

See Also:
   Constant Field Values

DEFAULT_REAL8_PADVALUE

public static final double DEFAULT_REAL8_PADVALUE

See Also:
   Constant Field Values

DEFAULT_DOUBLE_PADVALUE

public static final double DEFAULT_DOUBLE_PADVALUE
**DEFAULT_CHAR_PADVALUE**

```java
public static final char DEFAULT_CHAR_PADVALUE
```

See Also:
- Constant Field Values

**DEFAULT_UCHAR_PADVALUE**

```java
public static final char DEFAULT_UCHAR_PADVALUE
```

See Also:
- Constant Field Values

**DEFAULT_EPOCH_PADVALUE**

```java
public static final double DEFAULT_EPOCH_PADVALUE
```

See Also:
- Constant Field Values

**ILLEGAL_EPOCH_VALUE**

```java
public static final long ILLEGAL_EPOCH_VALUE
```

See Also:
- Constant Field Values
VIRTUAL_RECORD_DATA

public static final long VIRTUAL_RECORD_DATA

See Also:
   Constant Field Values

DID_NOT_COMPRESS

public static final long DID_NOT_COMPRESS

See Also:
   Constant Field Values

VAR_ALREADY_CLOSED

public static final long VAR_ALREADY_CLOSED

See Also:
   Constant Field Values

SINGLE_FILE_FORMAT

public static final long SINGLE_FILE_FORMAT

See Also:
   Constant Field Values

NO_PADVALUE_SPECIFIED
public static final long NO_PADVALUE_SPECIFIED

See Also:
Constant Field Values

NO_VARS_IN_CDF
public static final long NO_VARS_IN_CDF

See Also:
Constant Field Values

MULTI_FILE_FORMAT
public static final long MULTI_FILE_FORMAT

See Also:
Constant Field Values

SOME_ALREADY_ALLOCATED
public static final long SOME_ALREADY_ALLOCATED

See Also:
Constant Field Values

PRECEEDING_RECORDS_ALLOCATED
public static final long PRECEEDING_RECORDS_ALLOCATED

See Also:
Constant Field Values
CDF_OK

public static final long CDF_OK

See Also:
   Constant Field Values

ATTR_NAME_TRUNC

public static final long ATTR_NAME_TRUNC

See Also:
   Constant Field Values

CDF_NAME_TRUNC

public static final long CDF_NAME_TRUNC

See Also:
   Constant Field Values

VAR_NAME_TRUNC

public static final long VAR_NAME_TRUNC

See Also:
   Constant Field Values

NEGATIVE_FP_ZERO
public static final long NEGATIVE_FP_ZERO

See Also:
Constant Field Values

-------------

FORCED_PARAMETER

public static final long FORCED_PARAMETER

See Also:
Constant Field Values

-------------

NA_FOR_VARIABLE

public static final long NA_FOR_VARIABLE

See Also:
Constant Field Values

-------------

CDF_WARN

public static final long CDF_WARN

See Also:
Constant Field Values

-------------

ATTR_EXISTS

public static final long ATTR_EXISTS

See Also:
**BAD_CDF_ID**

```java
public static final long BAD_CDF_ID
```

See Also:
- Constant Field Values

**BAD_DATA_TYPE**

```java
public static final long BAD_DATA_TYPE
```

See Also:
- Constant Field Values

**BAD_DIM_SIZE**

```java
public static final long BAD_DIM_SIZE
```

See Also:
- Constant Field Values

**BAD_DIM_INDEX**

```java
public static final long BAD_DIM_INDEX
```

See Also:
- Constant Field Values
BAD_ENCODING

public static final long BAD_ENCODING

See Also:
   Constant Field Values

BAD_MAJORIT

public static final long BAD_MAJORIT

See Also:
   Constant Field Values

BAD_NUM_DIMS

public static final long BAD_NUM_DIMS

See Also:
   Constant Field Values

BAD_REC_NUM

public static final long BAD_REC_NUM

See Also:
   Constant Field Values

BAD_SCOPE

public static final long BAD_SCOPE


See Also:
Constant Field Values

BAD_NUM_ELEMS

public static final long BAD_NUM_ELEMS

See Also:
Constant Field Values

CDF_OPEN_ERROR

public static final long CDF_OPEN_ERROR

See Also:
Constant Field Values

CDF_EXISTS

public static final long CDF_EXISTS

See Also:
Constant Field Values

BAD_FORMAT

public static final long BAD_FORMAT

See Also:
Constant Field Values
BAD_ALLOCATE_RECS

public static final long BAD_ALLOCATE_RECS

See Also:
   Constant Field Values

BAD_CDF_EXTENSION

public static final long BAD_CDF_EXTENSION

See Also:
   Constant Field Values

NO_SUCH_ATTR

public static final long NO_SUCH_ATTR

See Also:
   Constant Field Values

NO_SUCH_ENTRY

public static final long NO_SUCH_ENTRY

See Also:
   Constant Field Values

NO_SUCH_VAR

public static final long NO_SUCH_VAR
VAR_READ_ERROR

public static final long VAR_READ_ERROR

See Also:
Constant Field Values

VAR_WRITE_ERROR

public static final long VAR_WRITE_ERROR

See Also:
Constant Field Values

BAD_ARGUMENT

public static final long BAD_ARGUMENT

See Also:
Constant Field Values

IBM_PC_OVERFLOW

public static final long IBM_PC_OVERFLOW

See Also:
Constant Field Values
TOO_MANY_VARS

public static final long TOO_MANY_VARS

See Also:
Constant Field Values

VAR_EXISTS

public static final long VAR_EXISTS

See Also:
Constant Field Values

BAD_MALLOC

public static final long BAD_MALLOC

See Also:
Constant Field Values

NOT_A_CDF

public static final long NOT_A_CDF

See Also:
Constant Field Values

CORRUPTED_V2_CDF
public static final long CORRUPTED_V2_CDF

See Also:
   Constant Field Values

VAR_OPEN_ERROR

public static final long VAR_OPEN_ERROR

See Also:
   Constant Field Values

BAD_INITIAL_RECS

public static final long BAD_INITIAL_RECS

See Also:
   Constant Field Values

BAD_BLOCKING_FACTOR

public static final long BAD_BLOCKING_FACTOR

See Also:
   Constant Field Values

END_OF_VAR

public static final long END_OF_VAR

See Also:
   Constant Field Values
BAD_CDFSTATUS

public static final long BAD_CDFSTATUS

See Also:
Constant Field Values

CDF_INTERNAL_ERROR

public static final long CDF_INTERNAL_ERROR

See Also:
Constant Field Values

BAD_NUM_VARS

public static final long BAD_NUM_VARS

See Also:
Constant Field Values

BAD_REC_COUNT

public static final long BAD_REC_COUNT

See Also:
Constant Field Values

BAD_REC_INTERVAL
public static final long BAD_REC_INTERVAL

See Also:
   Constant Field Values

BAD_DIM_COUNT

public static final long BAD_DIM_COUNT

See Also:
   Constant Field Values

BAD_DIM_INTERVAL

public static final long BAD_DIM_INTERVAL

See Also:
   Constant Field Values

BAD_VAR_NUM

public static final long BAD_VAR_NUM

See Also:
   Constant Field Values

BAD_ATTR_NUM

public static final long BAD_ATTR_NUM

See Also:
**BAD_ENTRY_NUM**

public static final long BAD_ENTRY_NUM

See Also:
    Constant Field Values

**BAD_ATTR_NAME**

public static final long BAD_ATTR_NAME

See Also:
    Constant Field Values

**BAD_VAR_NAME**

public static final long BAD_VAR_NAME

See Also:
    Constant Field Values

**NO_ATTR_SELECTED**

public static final long NO_ATTR_SELECTED

See Also:
    Constant Field Values
NO_ENTRY_SELECTED

public static final long NO_ENTRY_SELECTED

See Also:
   Constant Field Values

---

NO_VAR_SELECTED

public static final long NO_VAR_SELECTED

See Also:
   Constant Field Values

---

BAD_CDF_NAME

public static final long BAD_CDF_NAME

See Also:
   Constant Field Values

---

CANNOT_CHANGE

public static final long CANNOT_CHANGE

See Also:
   Constant Field Values

---

NO_STATUS_SELECTED

public static final long NO_STATUS_SELECTED
See Also:

  Constant Field Values

---

**NO_CDF_SELECTED**

public static final long NO_CDF_SELECTED

See Also:

  Constant Field Values

---

**READ_ONLY_DISTRIBUTION**

public static final long READ_ONLY_DISTRIBUTION

See Also:

  Constant Field Values

---

**CDF_CLOSE_ERROR**

public static final long CDF_CLOSE_ERROR

See Also:

  Constant Field Values

---

**VAR_CLOSE_ERROR**

public static final long VAR_CLOSE_ERROR

See Also:

  Constant Field Values
BAD_FNC_OR_ITEM

public static final long BAD_FNC_OR_ITEM

See Also:
Constant Field Values

ILLEGAL_ON_V1_CDF

public static final long ILLEGAL_ON_V1_CDF

See Also:
Constant Field Values

BAD_CACHE_SIZE

public static final long BAD_CACHE_SIZE

See Also:
Constant Field Values

CDF_CREATE_ERROR

public static final long CDF_CREATE_ERROR

See Also:
Constant Field Values

NO_SUCH_CDF

public static final long NO_SUCH_CDF
VAR_CREATE_ERROR

public static final long VAR_CREATE_ERROR

See Also:
Constant Field Values

READ_ONLY_MODE

public static final long READ_ONLY_MODE

See Also:
Constant Field Values

ILLEGAL_IN_zMODE

public static final long ILLEGAL_IN_zMODE

See Also:
Constant Field Values

BAD_zMODE

public static final long BAD_zMODE

See Also:
Constant Field Values
BAD_READONLY_MODE

public static final long BAD_READONLY_MODE

See Also:
Constant Field Values

CDF_READ_ERROR

public static final long CDF_READ_ERROR

See Also:
Constant Field Values

CDF_WRITE_ERROR

public static final long CDF_WRITE_ERROR

See Also:
Constant Field Values

ILLEGAL_FOR_SCOPE

public static final long ILLEGAL_FOR_SCOPE

See Also:
Constant Field Values

NO_MORE_ACCESS
public static final long NO_MORE_ACCESS

See Also:
   Constant Field Values

BAD_DECODING

public static final long BAD_DECODING

See Also:
   Constant Field Values

BAD_NEGtoPOSfp0_MODE

public static final long BAD_NEGtoPOSfp0_MODE

See Also:
   Constant Field Values

UNSUPPORTED_OPERATION

public static final long UNSUPPORTED_OPERATION

See Also:
   Constant Field Values

NO_WRITE_ACCESS

public static final long NO_WRITE_ACCESS

See Also:
   Constant Field Values
NO_DELETE_ACCESS

public static final long NO_DELETE_ACCESS

See Also:
Constant Field Values

CDF_DELETE_ERROR

public static final long CDF_DELETE_ERROR

See Also:
Constant Field Values

VAR_DELETE_ERROR

public static final long VAR_DELETE_ERROR

See Also:
Constant Field Values

UNKNOWN_COMPRESSION

public static final long UNKNOWN_COMPRESSION

See Also:
Constant Field Values

CANNOT_COMPRESS
public static final long CANNOT_COMPRESS

See Also:
   Constant Field Values

DECOMPRESSION_ERROR

public static final long DECOMPRESSION_ERROR

See Also:
   Constant Field Values

COMPRESSION_ERROR

public static final long COMPRESSION_ERROR

See Also:
   Constant Field Values

EMPTY_COMPRESSED_CDF

public static final long EMPTY_COMPRESSED_CDF

See Also:
   Constant Field Values

BAD_COMPRESSION_PARM

public static final long BAD_COMPRESSION_PARM

See Also:
**UNKNOWN_SPARSENESS**

public static final long UNKNOWN_SPARSENESS

See Also:
Constant Field Values

**CANNOT_SPARSERECORDS**

public static final long CANNOT_SPARSERECORDS

See Also:
Constant Field Values

**CANNOT_SPARSEARRAYS**

public static final long CANNOT_SPARSEARRAYS

See Also:
Constant Field Values

**TOO_MANY_PARMS**

public static final long TOO_MANY_PARMS

See Also:
Constant Field Values
NO_SUCH_RECORD

public static final long NO_SUCH_RECORD

See Also:
 Constant Field Values

CANNOT_ALLOCATE_RECORDS

public static final long CANNOT_ALLOCATE_RECORDS

See Also:
 Constant Field Values

CANNOT_COPY

public static final long CANNOT_COPY

See Also:
 Constant Field Values

SCRATCH_DELETE_ERROR

public static final long SCRATCH_DELETE_ERROR

See Also:
 Constant Field Values

SCRATCH_CREATE_ERROR

public static final long SCRATCH_CREATE_ERROR
See Also:
  Constant Field Values

---

**SCRATCH_READ_ERROR**

public static final long **SCRATCH_READ_ERROR**

See Also:
  Constant Field Values

---

**SCRATCH_WRITE_ERROR**

public static final long **SCRATCH_WRITE_ERROR**

See Also:
  Constant Field Values

---

**BAD_SPARSEARRAYS_PARM**

public static final long **BAD_SPARSEARRAYS_PARM**

See Also:
  Constant Field Values

---

**BAD_SCRATCH_DIR**

public static final long **BAD_SCRATCH_DIR**

See Also:
  Constant Field Values
**DATATYPE_MISMATCH**

public static final long DATATYPE_MISMATCH

See Also:

Constant Field Values

---

**NOT_A_CDF_OR_NOT_SUPPORTED**

public static final long NOT_A_CDF_OR_NOT_SUPPORTED

See Also:

Constant Field Values

---

**CORRUPTED_V3_CDF**

public static final long CORRUPTED_V3_CDF

See Also:

Constant Field Values

---

**ILLEGAL_EPOCH_FIELD**

public static final long ILLEGAL_EPOCH_FIELD

See Also:

Constant Field Values

---

**CREATE_**

public static final long CREATE_
See Also:
   Constant Field Values

---

**OPEN**

```java
definition
public static final long OPEN;
```

See Also:
   Constant Field Values

---

**DELETE**

```java
definition
public static final long DELETE;
```

See Also:
   Constant Field Values

---

**CLOSE**

```java
definition
public static final long CLOSE;
```

See Also:
   Constant Field Values

---

**SELECT**

```java
definition
public static final long SELECT;
```

See Also:
   Constant Field Values
CONFIRM_

public static final long CONFIRM_

See Also:
Constant Field Values

GET_

public static final long GET_

See Also:
Constant Field Values

PUT_

public static final long PUT_

See Also:
Constant Field Values

SAVE_

public static final long SAVE_

See Also:
Constant Field Values

BACKWARD_
public static final long BACKWARD_

See Also:
Constant Field Values

---

GETCDFFILEBACKWARD_

public static final long GETCDFFILEBACKWARD_

See Also:
Constant Field Values

---

NULL_

public static final long NULL_

See Also:
Constant Field Values

---

CDF_

public static final long CDF_

See Also:
Constant Field Values

---

CDF_NAME_

public static final long CDF_NAME_

See Also:
Constant Field Values
public static final long CDF_ENCODING_

See Also:
Constant Field Values

public static final long CDF_DECODING_

See Also:
Constant Field Values

public static final long CDF_MAJORITY_

See Also:
Constant Field Values

public static final long CDF_FORMAT_

See Also:
Constant Field Values

public static final long CDF_COPYRIGHT_
public static final long CDF_COPYRIGHT_

See Also:
    Constant Field Values

CDF_NUMrVARS_

public static final long CDF_NUMrVARS_

See Also:
    Constant Field Values

CDF_NUMzVARS_

public static final long CDF_NUMzVARS_

See Also:
    Constant Field Values

CDF_NUMATTRS_

public static final long CDF_NUMATTRS_

See Also:
    Constant Field Values

CDF_NUMgATTRS_

public static final long CDF_NUMgATTRS_

See Also:
Constant Field Values

CDF_NUMvATTRS_

public static final long CDF_NUMvATTRS_

See Also:
Constant Field Values

CDF_VERSION_

public static final long CDF_VERSION_

See Also:
Constant Field Values

CDF_RELEASE_

public static final long CDF_RELEASE_

See Also:
Constant Field Values

CDF_INCREMENT_

public static final long CDF_INCREMENT_

See Also:
Constant Field Values
CDF_STATUS_

public static final long CDF_STATUS_

See Also:
   Constant Field Values

CDF_READONLY_MODE_

public static final long CDF_READONLY_MODE_

See Also:
   Constant Field Values

CDF_zMODE_

public static final long CDF_zMODE_

See Also:
   Constant Field Values

CDF_NEGtoPOSfp0_MODE_

public static final long CDF_NEGtoPOSfp0_MODE_

See Also:
   Constant Field Values

LIB_COPYRIGHT_

public static final long LIB_COPYRIGHT_
See Also:
   Constant Field Values

LIB_VERSION_

public static final long LIB_VERSION_

See Also:
   Constant Field Values

LIB_RELEASE_

public static final long LIB_RELEASE_

See Also:
   Constant Field Values

LIB_INCREMENT_

public static final long LIB_INCREMENT_

See Also:
   Constant Field Values

LIB_subINCREMENT_

public static final long LIB_subINCREMENT_

See Also:
   Constant Field Values
public static final long rVARs_NUMDIMS_

See Also:
Constant Field Values

public static final long rVARs_DIMSIZES_

See Also:
Constant Field Values

public static final long rVARs_MAXREC_

See Also:
Constant Field Values

public static final long rVARs_RECDATA_

See Also:
Constant Field Values

public static final long rVARs_RECNUMBER_
rVARs_RECCOUNT_

public static final long rVARs_RECCOUNT_

See Also:
Constant Field Values

rVARs_RECINTERVAL_

public static final long rVARs_RECINTERVAL_

See Also:
Constant Field Values

rVARs_DIMINDICES_

public static final long rVARs_DIMINDICES_

See Also:
Constant Field Values

rVARs_DIMCOUNTS_

public static final long rVARs_DIMCOUNTS_

See Also:
Constant Field Values
rVARs_DIMINTERVALS_
public static final long rVARs_DIMINTERVALS_

See Also:
Constant Field Values

rVAR_
public static final long rVAR_

See Also:
Constant Field Values

rVAR_NAME_
public static final long rVAR_NAME_

See Also:
Constant Field Values

rVAR_DATATYPE_
public static final long rVAR_DATATYPE_

See Also:
Constant Field Values

rVAR_NUMELEMS_
public static final long rVAR_NUMELEMS_

See Also:

Constant Field Values

rVAR_RECVARY_

class public static final long rVAR_RECVARY_

See Also:

Constant Field Values

rVAR_DIMVARYS_

class public static final long rVAR_DIMVARYS_

See Also:

Constant Field Values

rVAR_NUMBER_

class public static final long rVAR_NUMBER_

See Also:

Constant Field Values

rVAR_DATA_

class public static final long rVAR_DATA_

See Also:

Constant Field Values
**rVAR_HYPERDATA**

public static final long rVAR_HYPERDATA_

See Also:
   Constant Field Values

**rVAR_SEQDATA**

public static final long rVAR_SEQDATA_

See Also:
   Constant Field Values

**rVAR_SEQPOS**

public static final long rVAR_SEQPOS_

See Also:
   Constant Field Values

**rVAR_MAXREC**

public static final long rVAR_MAXREC_

See Also:
   Constant Field Values

**rVAR_MAXallocREC**
public static final long rVAR_MAXallocREC_

See Also:
  Constant Field Values

rVAR_DATASPEC_
public static final long rVAR_DATASPEC_

See Also:
  Constant Field Values

rVAR_PADVALUE_
public static final long rVAR_PADVALUE_

See Also:
  Constant Field Values

rVAR_INITIALRECS_
public static final long rVAR_INITIALRECS_

See Also:
  Constant Field Values

rVAR_BLOCKINGFACTOR_
public static final long rVAR_BLOCKINGFACTOR_

See Also:
Constant Field Values

rVAR_nINDEXRECORDS_

public static final long rVAR_nINDEXRECORDS_

See Also:
Constant Field Values

rVAR_nINDEXENTRIES_

public static final long rVAR_nINDEXENTRIES_

See Also:
Constant Field Values

rVAR_EXISTENCE_

public static final long rVAR_EXISTENCE_

See Also:
Constant Field Values

zVARs_MAXREC_

public static final long zVARs_MAXREC_

See Also:
Constant Field Values
public static final long zVARs_RECDATA_

See Also:
   Constant Field Values

public static final long zVAR_

See Also:
   Constant Field Values

public static final long zVAR_NAME_

See Also:
   Constant Field Values

public static final long zVAR_DATATYPE_

See Also:
   Constant Field Values

public static final long zVAR_NUMELEMS_
See Also:
Constant Field Values

---

**zVAR_NUMDIMS**

public static final long zVAR_NUMDIMS

See Also:
Constant Field Values

---

**zVAR_DIMSIZES**

public static final long zVAR_DIMSIZES

See Also:
Constant Field Values

---

**zVAR_RECVARY**

public static final long zVAR_RECVARY

See Also:
Constant Field Values

---

**zVAR_DIMVARYS**

public static final long zVAR_DIMVARYS

See Also:
Constant Field Values
public static final long zVAR_NUMBER_  
See Also:  
Constant Field Values

public static final long zVAR_DATA_  
See Also:  
Constant Field Values

public static final long zVAR_HYPERDATA_  
See Also:  
Constant Field Values

public static final long zVAR_SEQDATA_  
See Also:  
Constant Field Values

public static final long zVAR_SEQPOS_
See Also:
Constant Field Values

zVAR_MAXREC_

public static final long zVAR_MAXREC_

See Also:
Constant Field Values

zVAR_MAXallocREC_

public static final long zVAR_MAXallocREC_

See Also:
Constant Field Values

zVAR_DATASPEC_

public static final long zVAR_DATASPEC_

See Also:
Constant Field Values

zVAR_PADVALUE_

public static final long zVAR_PADVALUE_

See Also:
Constant Field Values
zVAR_INITIALRECS_

public static final long zVAR_INITIALRECS_

See Also:
Constant Field Values

zVAR_BLOCKINGFACTOR_

public static final long zVAR_BLOCKINGFACTOR_

See Also:
Constant Field Values

zVAR_nINDEXRECORDS_

public static final long zVAR_nINDEXRECORDS_

See Also:
Constant Field Values

zVAR_nINDEXENTRIES_

public static final long zVAR_nINDEXENTRIES_

See Also:
Constant Field Values

zVAR_EXISTENCE_
public static final long zVAR_EXISTENCE_

See Also:
   Constant Field Values

zVAR_RECNUMBER_

public static final long zVAR_RECNUMBER_

See Also:
   Constant Field Values

zVAR_RECCOUNT_

public static final long zVAR_RECCOUNT_

See Also:
   Constant Field Values

zVAR_RECINTERVAL_

public static final long zVAR_RECINTERVAL_

See Also:
   Constant Field Values

zVAR_DIMINDICES_

public static final long zVAR_DIMINDICES_

See Also:
   Constant Field Values
zVAR_DIMCOUNTS_

public static final long zVAR_DIMCOUNTS_

See Also:
Constant Field Values

zVAR_DIMINTERVALS_

public static final long zVAR_DIMINTERVALS_

See Also:
Constant Field Values

ATTR_

public static final long ATTR_

See Also:
Constant Field Values

ATTR_SCOPE_

public static final long ATTR_SCOPE_

See Also:
Constant Field Values

ATTR_NAME_
public static final long ATTR_NAME_

See Also:
Constant Field Values

public static final long ATTR_NUMBER_

See Also:
Constant Field Values

public static final long ATTR_MAXENTRY_

See Also:
Constant Field Values

public static final long ATTR_NUMENTRYs_

See Also:
Constant Field Values

public static final long ATTR_MAXENTRYr_

See Also:
**ATTR_NUMrENTRIES**

public static final long **ATTR_NUMrENTRIES**

See Also:
  - Constant Field Values

**ATTR_MAXzENTRY**

public static final long **ATTR_MAXzENTRY**

See Also:
  - Constant Field Values

**ATTR_NUMzENTRIES**

public static final long **ATTR_NUMzENTRIES**

See Also:
  - Constant Field Values

**ATTR_EXISTENCE**

public static final long **ATTR_EXISTENCE**

See Also:
  - Constant Field Values
gENTRY_
public static final long gENTRY_

See Also:  
Constant Field Values

---

gENTRY_EXISTENCE_
public static final long gENTRY_EXISTENCE_

See Also:  
Constant Field Values

---

gENTRY_DATATYPE_
public static final long gENTRY_DATATYPE_

See Also:  
Constant Field Values

---

gENTRY_NUMELEMS_
public static final long gENTRY_NUMELEMS_

See Also:  
Constant Field Values

---

gENTRY_DATASPEC_
public static final long gENTRY_DATASPEC_
See Also:
   Constant Field Values

---

gENTRY_DATA_

public static final long gENTRY_DATA_

See Also:
   Constant Field Values

---

rENTRY_

public static final long rENTRY_

See Also:
   Constant Field Values

---

rENTRY_NAME_

public static final long rENTRY_NAME_

See Also:
   Constant Field Values

---

rENTRYEXISTENCE_

public static final long rENTRYEXISTENCE_

See Also:
   Constant Field Values
public static final long rENTRY_DATATYPE_

See Also:
    Constant Field Values

public static final long rENTRY_NUMELEMS_

See Also:
    Constant Field Values

public static final long rENTRY_DATASPEC_

See Also:
    Constant Field Values

public static final long rENTRY_DATA_

See Also:
    Constant Field Values

public static final long zENTRY_
See Also:

Constant Field Values

public static final long zENTRY_NAME_

See Also:

Constant Field Values

public static final long zENTRY_EXISTENCE_

See Also:

Constant Field Values

public static final long zENTRY_DATATYPE_

See Also:

Constant Field Values

public static final long zENTRY_NUMELEMS_

See Also:

Constant Field Values
**ENTRY_DATASPEC**

```java
public static final long zENTRY_DATASPEC_
```

See Also:
- Constant Field Values

**ENTRY_DATA**

```java
public static final long zENTRY_DATA_
```

See Also:
- Constant Field Values

**STATUS_TEXT**

```java
public static final long STATUS_TEXT_
```

See Also:
- Constant Field Values

**CDF_CACHESIZE**

```java
public static final long CDF_CACHESIZE_
```

See Also:
- Constant Field Values

**rVARs_CACHESIZE**
public static final long rVARs_CACHESIZE_

See Also:
Constant Field Values

---

zVARs_CACHESIZE_

public static final long zVARs_CACHESIZE_

See Also:
Constant Field Values

---

rVAR_CACHESIZE_

public static final long rVAR_CACHESIZE_

See Also:
Constant Field Values

---

zVAR_CACHESIZE_

public static final long zVAR_CACHESIZE_

See Also:
Constant Field Values

---

zVARs_RECNUMBER_

public static final long zVARs_RECNUMBER_

See Also:
Constant Field Values
rVAR_ALLOCATERECS_

public static final long rVAR_ALLOCATERECS_

See Also:
Constant Field Values

zVAR_ALLOCATERECS_

public static final long zVAR_ALLOCATERECS_

See Also:
Constant Field Values

DATATYPE_SIZE_

public static final long DATATYPE_SIZE_

See Also:
Constant Field Values

CURgENTRY_EXISTENCE_

public static final long CURgENTRY_EXISTENCE_

See Also:
Constant Field Values

CURrENTRY_EXISTENCE_
public static final long CURRENTRY_EXISTENCE

See Also: Constant Field Values

CURRENTRY_EXISTENCE

public static final long CURRENTRY_EXISTENCE

See Also: Constant Field Values

CDF_INFO

public static final long CDF_INFO

See Also: Constant Field Values

CDF_COMPRESSION

public static final long CDF_COMPRESSION

See Also: Constant Field Values

zVAR_COMPRESSION

public static final long zVAR_COMPRESSION

See Also: Constant Field Values
Constant Field Values

---

**zVAR_SPARSERECORDS_**

public static final long **zVAR_SPARSERECORDS_**

See Also:  
Constant Field Values

---

**zVAR_SPARSEARRAYS_**

public static final long **zVAR_SPARSEARRAYS_**

See Also:  
Constant Field Values

---

**zVAR_ALLOCATEBLOCK_**

public static final long **zVAR_ALLOCATEBLOCK_**

See Also:  
Constant Field Values

---

**zVAR_NUMRECS_**

public static final long **zVAR_NUMRECS_**

See Also:  
Constant Field Values
zVAR_NUMallocRECS_

public static final long zVAR_NUMallocRECS_

See Also:
   Constant Field Values

rVAR_COMPRESSION_

public static final long rVAR_COMPRESSION_

See Also:
   Constant Field Values

rVAR_SPARSERECORDS_

public static final long rVAR_SPARSERECORDS_

See Also:
   Constant Field Values

rVAR_SPARSEARRAYS_

public static final long rVAR_SPARSEARRAYS_

See Also:
   Constant Field Values

rVAR_ALLOCATEBLOCK_

public static final long rVAR_ALLOCATEBLOCK_
See Also:
Constant Field Values

rVAR_NUMRECS_

public static final long rVAR_NUMRECS_

See Also:
Constant Field Values

rVAR_NUMAllocRECS_

public static final long rVAR_NUMAllocRECS_

See Also:
Constant Field Values

rVAR_ALLOCATEDFROM_

public static final long rVAR_ALLOCATEDFROM_

See Also:
Constant Field Values

rVAR_ALLOCATEDTO_

public static final long rVAR_ALLOCATEDTO_

See Also:
Constant Field Values
zVAR_ALLOCATEDFROM_

public static final long zVAR_ALLOCATEDFROM_

See Also:
Constant Field Values

zVAR_ALLOCATEDTO_

public static final long zVAR_ALLOCATEDTO_

See Also:
Constant Field Values

zVAR_nINDEXLEVELS_

public static final long zVAR_nINDEXLEVELS_

See Also:
Constant Field Values

rVAR_nINDEXLEVELS_

public static final long rVAR_nINDEXLEVELS_

See Also:
Constant Field Values

CDF_SCRATCHDIR_

public static final long CDF_SCRATCHDIR_
See Also:
Constant Field Values

rVAR_RESERVEPERCENT_

public static final long rVAR_RESERVEPERCENT_

See Also:
Constant Field Values

zVAR_RESERVEPERCENT_

public static final long zVAR_RESERVEPERCENT_

See Also:
Constant Field Values

rVAR_RECORDS_

public static final long rVAR_RECORDS_

See Also:
Constant Field Values

zVAR_RECORDS_

public static final long zVAR_RECORDS_

See Also:
Constant Field Values
STAGE_CACHESIZE_

public static final long STAGE_CACHESIZE_

See Also:
Constant Field Values

COMPRESS_CACHESIZE_

public static final long COMPRESS_CACHESIZE_

See Also:
Constant Field Values

CDFwithSTATS_

public static final long CDFwithSTATS_

See Also:
Constant Field Values

CDF_ACCESS_

public static final long CDF_ACCESS_

See Also:
Constant Field Values
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.

See Also:

Variable, CDFException

Field Summary

Fields inherited from interface gsfc.nssdc.cdf.CDFConstants
Method Summary

```java
void delete()

See the description of the getName() method in this class.
```
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>void</strong> dump ()</td>
<td>Dump data information and values, one row at a time, to the stdErr.</td>
</tr>
<tr>
<td><strong>void</strong> dumpData ()</td>
<td>Dumps variable data, one row at a time per record.</td>
</tr>
<tr>
<td><strong>java.lang.Object</strong> getData ()</td>
<td>Returns an object that is properly dimensioned.</td>
</tr>
<tr>
<td><strong>long []</strong> getDimCounts ()</td>
<td>Gets the value of the dimension counts that represents the number of elements read or write starting at the location for a hyper get/put function.</td>
</tr>
<tr>
<td><strong>long []</strong> getDimIndices ()</td>
<td>Gets the starting dimension index within a record for a hyper get/put function.</td>
</tr>
<tr>
<td><strong>long []</strong> getDimIntervals ()</td>
<td>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.</td>
</tr>
<tr>
<td><strong>int []</strong> getDimSizes ()</td>
<td>Gets the dimension sizes of this variable.</td>
</tr>
<tr>
<td><strong>java.lang.String</strong> getName ()</td>
<td>CDFData implements CDFObject to enable CDFDelegate calls.</td>
</tr>
<tr>
<td><strong>int</strong> getnDims ()</td>
<td>Gets the dimensionality of this variable.</td>
</tr>
<tr>
<td><strong>long</strong> getRecCount ()</td>
<td>Gets the number of records to read or write for a hyper get/put function.</td>
</tr>
<tr>
<td><strong>long</strong> getRecInterval ()</td>
<td>Gets the number of records to skip for a hyper get/put function.</td>
</tr>
<tr>
<td><strong>long</strong> getRecStart ()</td>
<td>Gets the record number at which a hyper get/put function starts.</td>
</tr>
<tr>
<td><strong>void</strong> rename (java.lang.String name)</td>
<td>See the description of the getName() method in this class.</td>
</tr>
</tbody>
</table>

**Methods inherited from class java.lang.Object**

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

```java
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

getnDims

public int getnDims()

Gets the dimensionality of this variable.

```java
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 dimensionality of temperature

Returns:
the dimensionality of this variable

---

gDimSizes

class public int[] getDimSizes()  

Gets the dimension sizes of this variable. For example, 3 X 10 (3 rows and 10 columns) two-
dimensional 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

---

gRecStart

class 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

---

gRecCount

class 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**

```java
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**

```java
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**

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

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

**Returns:**
the dimension counts for this variable

---

**getDimIntervals**

```java
generic 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**

```java
generic 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.

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.

```java
CDFData data;
long[] dimIndices   = {0,0};
long[] dimIntervals = {3,5};
long[] dimSizes     = {1,1};
```

```java
data = var.getHyperDataObject(0L,                 // record start
                              1,          // record counts
                              1,          // record interval
                              dimIndices,
                              dimSizes,
                              dimIntervals);
```

```java
data.dumpData();
```
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: /nDims:[sizes] recStart/recCount/recInterval/dimIndices/dimsSizes/dimIntervals/dataArraySignature

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
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

<table>
<thead>
<tr>
<th>void</th>
<th>cdflib(CDF theCDF, CDFObject cdfObject, java.util.Vector cmds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Defines the method that is responsible for acting as the gateway between the CDF Java code and the CDF library.</td>
</tr>
</tbody>
</table>

Method Detail

cdflib
public 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
DEFAULT_DOUBLE_PADVALUE, DEFAULT_EPOCH_PADVALUE,
DEFAULT_FLOAT_PADVALUE, DEFAULT_INT1_PADVALUE,
DEFAULT_INT2_PADVALUE, DEFAULT_INT4_PADVALUE,
DEFAULT_REAL4_PADVALUE, DEFAULT_REAL8_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, EPOCHx_FORMAT_MAX, EPOCHx_STRING_MAX,
FORCED_PARAMETER, gENTRY__, gENTRY_DATA__, gENTRY_DATASPEC__,
gENTRY_DATATYPE__, gENTRY_EXISTENCE__, gENTRY_NUMELEMS__, GET_,
GETCDFFILEBACKWARD__, GLOBAL_SCOPE, GZIP_COMPRESSION, HOST_DECODING,
HOST_ENCODING, HP_DECODING, HP_ENCODING, HUFF_COMPRESSION,
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,
LIB_COPYRIGHT__, LIB_INCREMENT__, LIB_RELEASE__, LIB_subINCREMENT__,
LIB_VERSION__, MAC_DECODING, MAC_ENCODING, MULTI_FILE,
MULTI_FILE_FORMAT, NA_FOR_VARIABLE, NEGATIVE_FP_ZERO,
NEGtoFP0off, NEGtoFP0on, NETWORK_DECODING, NETWORK_ENCODING,
NeXT_DECODING, NeXT_ENCODING, NO_ATTR_SELECTED, NO_CDF_SELECTED,
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, NOT_A_CDF, NOT_A_CDF_OR_NOT_SUPPORTED, NOVARY,
NULL__, OPEN__, OPTIMAL_ENCODING_TREES, PAD_SPARSERECORDS,
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__, rVAR__,
RLE_COMPRESSION, RLE_OF_ZEROS, ROW_MAJOR, rVAR__, rVAR_ALLOCATEBLOCK__,
rVAR_ALLOCATEDFROM__, rVAR_ALLOCATEDTO__, rVAR_ALLOCATERS__,
rVAR_BLOCKINGFACTOR__, rVAR_CACHESIZE__, rVAR_COMPRESSION__,
rVAR_DATA__, rVAR_DATASPEC__, rVAR_DATATYPE__, rVAR_DIMVARYS__,
rVAR_EXISTENCE__, rVAR_HYPERDATA__, rVAR_INITIALRECS__,
Constructor Summary
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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>long</td>
<td>getCurrentStatus()</td>
</tr>
<tr>
<td></td>
<td>Gets the status code that caused CDFException.</td>
</tr>
<tr>
<td>static java.lang.String</td>
<td>getStatusMsg(long statusCode)</td>
</tr>
<tr>
<td></td>
<td>Get the status text message for the given status code.</td>
</tr>
</tbody>
</table>

### Methods inherited from class java.lang.Throwable

- fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace, toString

### Methods inherited from class java.lang.Object

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

### Constructor Detail

CDFException

```java
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
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

CDFNativeLibrary()

Method Summary

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
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.
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

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>delete()</td>
<td>Deletes the current object.</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>getName()</td>
<td>Returns the name of the current object.</td>
</tr>
<tr>
<td>void</td>
<td>rename(java.lang.String name)</td>
<td>Renames the current object.</td>
</tr>
</tbody>
</table>

### Method Detail

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

Returns the name of the current object.

**Returns:**
the name of the current object

---

rename

public 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

public void delete()
throws CDFException

Deletes the current object.

**Throws:**
CDFException - if an error occurs deleting the current object
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 2005/08/15 15:26:44 liu Exp $

Field Summary

<table>
<thead>
<tr>
<th>static int</th>
<th>ALL_VALUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>static int</td>
<td>NAMED_VALUES</td>
</tr>
<tr>
<td>static int</td>
<td>NO_REPORTS</td>
</tr>
<tr>
<td>static int</td>
<td>NO_VALUES</td>
</tr>
<tr>
<td>static int</td>
<td>NRV_VALUES</td>
</tr>
<tr>
<td>static int</td>
<td>REPORT_ERRORS</td>
</tr>
<tr>
<td>static int</td>
<td>REPORT_INFORMATION</td>
</tr>
<tr>
<td>static int</td>
<td>REPORT_WARNINGS</td>
</tr>
<tr>
<td>static int</td>
<td>RV_VALUES</td>
</tr>
</tbody>
</table>

Fields inherited from interface gsfc.nssdc.cdf.CDFConstants

AHUFF_COMPRESSION, ALPHAOSF1_DECODING, ALPHAOSF1_ENCODING,
ALPHAVMsD_DECODING, ALPHAVMsD_ENCODING, ALPHAVMsQ_DECODING,
ALPHAVMsQ_ENCODING, ALPHAVMSI_DECODING, ALPHAVMSI_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_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_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, CANNOT_ALLOCATE_RECORDS, CANNOT_CHANGE,
CANNOT_COMPRESS, CANNOT_COPY, CANNOT_SPARSEARRAYS,
CANNOT_SPARSERECORDS, CDF_, CDF_ACCESS_, CDF_ATTR_NAME_LEN,
CDF_BYTE, CDF_CACHESIZE_, CDF_CHAR, 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,
Constructor Summary

CDFTools()

Method Summary

static void skeletonCDF(java.lang.String skeletonName, java.lang.String cdfName, boolean delete, boolean log, boolean neg2posfp0, boolean statistics, int zMode, int reportType, int cacheSize)

skeletonCDF 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 cacheSize)

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

public static final int NRV_VALUES

See Also:
Constant Field Values

public static final int RV_VALUES

See Also:
Constant Field Values

public static final int ALL_VALUES

See Also:
Constant Field 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
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 .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.
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 512-byte buffers to be used for the CDF's dotCDF file, staging file, and compression scratch file. If this qualifier is absent, default cache sizes chosen by the CDF library are used. The cache sizes are specified with a comma-separated list of pairs where is the number of cache buffers and is the type of file. The file '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' 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
java.lang.InterruptedException

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 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 .skt in the current directory

cdfName - The pathname of the CDF from which the skeleton table will be created. 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 512-byte buffers to be used for the CDF's dotCDF file, staging file, and compression scratch file. If this qualifier is absent, default cache sizes chosen by the CDF library are used. The cache sizes are specified with a comma-separated list of pairs where is the number of cache buffers and is the type of file. The file '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' 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
java.lang.InterruptedException
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
Constructor Summary

CDFUtils()
## Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static boolean <code>cdfFileExists</code> (java.lang.String fileName)</td>
<td>Checks the existence of the given CDF file name.</td>
</tr>
<tr>
<td>static long <code>getDataTypeValue</code> (java.lang.String cdfDataType)</td>
<td>Gets the long value of the given CDF data type in string.</td>
</tr>
<tr>
<td>static long <code>getLongCompressionType</code> (java.lang.String compressionType)</td>
<td>Gets the long representation of the given CDF compression type in string.</td>
</tr>
<tr>
<td>static long <code>getLongEncoding</code> (java.lang.String encodingType)</td>
<td>Gets the long value of the given CDF encoding type in string.</td>
</tr>
<tr>
<td>static long <code>getLongFormat</code> (java.lang.String formatType)</td>
<td>Gets the long value of the given CDF file format in string.</td>
</tr>
<tr>
<td>static long <code>getLongMajority</code> (java.lang.String majorityType)</td>
<td>Gets the long value of the given CDF majority.</td>
</tr>
<tr>
<td>static long <code>getLongSparseRecord</code> (java.lang.String sparseRecordType)</td>
<td>Gets the long value of the given sparse record type in string.</td>
</tr>
<tr>
<td>static long <code>getNumElements</code> (long dataType, java.lang.Object data)</td>
<td>Gets the number of elements contained in the given data object.</td>
</tr>
<tr>
<td>static java.lang.String <code>getSignature</code> (java.lang.Object obj)</td>
<td>Gets the java signature of the given object.</td>
</tr>
<tr>
<td>static java.lang.String <code>getStringCompressionType</code> (CDF cdf)</td>
<td>Gets the string representation of the given CDF file's compression type.</td>
</tr>
<tr>
<td>static java.lang.String <code>getStringCompressionType</code> (long compressionType)</td>
<td>Gets the string representation of the given CDF compression type.</td>
</tr>
<tr>
<td>static java.lang.String <code>getStringCompressionType</code> (Variable var)</td>
<td>Gets the string representation of the given variable's compression type.</td>
</tr>
<tr>
<td>static java.lang.String <code>getStringData</code> (java.lang.Object data)</td>
<td>Returns the string value of the given data.</td>
</tr>
<tr>
<td>static java.lang.String <code>getStringData</code> (java.lang.Object data, int epochType)</td>
<td>Returns the string value of the given data.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>String getStringData(java.lang.Object data, java.lang.String separator)</code></td>
<td>Returns the string of the value of the given data.</td>
</tr>
<tr>
<td><code>String getStringData(java.lang.Object data, java.lang.String separator, int epochType)</code></td>
<td>Returns the string of the value of the given data.</td>
</tr>
<tr>
<td><code>String getStringDataType(Entry entry)</code></td>
<td>Gets the string value of the CDF data type for the given entry.</td>
</tr>
<tr>
<td><code>String getStringDataType(long cdfDataType)</code></td>
<td>Gets the string representation of the given CDF data type.</td>
</tr>
<tr>
<td><code>String getStringDataType(Variable var)</code></td>
<td>Gets the string value of the CDF data type for the given variable.</td>
</tr>
<tr>
<td><code>String getStringDecoding(CDF cdf)</code></td>
<td>Gets the string value of the given CDF file's decoding type.</td>
</tr>
<tr>
<td><code>String getStringDecoding(long decodingType)</code></td>
<td>Gets the string value of the given CDF decoding type.</td>
</tr>
<tr>
<td><code>String getStringEncoding(CDF cdf)</code></td>
<td>Get the string value of the given CDF's encoding type.</td>
</tr>
<tr>
<td><code>String getStringEncoding(long encodingType)</code></td>
<td>Gets the string value of the given CDF encoding type.</td>
</tr>
<tr>
<td><code>String getStringFormat(CDF cdf)</code></td>
<td>Gets the string value of the given CDF's file format.</td>
</tr>
<tr>
<td><code>String getStringFormat(long formatType)</code></td>
<td>Gets the string value of the given CDF's file format.</td>
</tr>
<tr>
<td><code>String getStringMajority(CDF cdf)</code></td>
<td>Gets the string value of the given CDF file's majority.</td>
</tr>
<tr>
<td><code>String getStringMajority(long majorityType)</code></td>
<td>Gets the string value of the given CDF majority.</td>
</tr>
<tr>
<td><code>String getStringSparseRecord(long sparseRecordType)</code></td>
<td>Gets the string value of the given sparse record type.</td>
</tr>
<tr>
<td><code>String getStringSparseRecord(Variable var)</code></td>
<td>Gets the string value of the given variable's sparse record type.</td>
</tr>
</tbody>
</table>
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, java.io.PrintWriter outWriter)
Prints the value of the given data to the place designated by PrintWriter that can be a file, System.out, System.err, and etc.

static void **printData**(java.lang.Object data, java.io.PrintWriter outWriter, int which)

---

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)).
Parameters:

    obj - the object from which Java signature is retrieved

Returns:

    Java signature of the given object
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

---

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)

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.
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, System.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)
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.

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 parsed

    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 parsed
    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.

---

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 parsed
    separator - the delimiter for array elements
    epochType - Epoch or Epoch16 data type indicator
        == 1 for EPOCH, == 2 for EPOCH16, == 0 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.
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.

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.

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:
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**

```java
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_REAL4
  - CDF_FLOAT
  - CDF_REAL8
  - CDF_DOUBLE
  - CDF_EPOCH
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.
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.

---

g 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
- MAC_ENCODING
- HP_ENCODING
- NeXT_ENCODING
- ALPHAOSF1_ENCODING
- ALPHAVMSd_ENCODING
- ALPHAVMSg_ENCODING
- ALPHAVMSi_ENCODING

**Returns:**
string representation of encodingType. The returned value is one of the following:
- NETWORK
- SUN
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
- MAC
- HP
- NeXT
- ALPHAOSF1
- ALPHAVMSd
- ALPHAVMSg
- ALPHAVMSi

Returns:
long representation of encodingType. The returned value is one of the following:
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
- MAC_DECODING
- HP_DECODING
- NeXT_DECODING
- ALPHAOSF1_DECODING
- ALPHAVMSd_DECODING
- ALPHAVMSg_DECODING
- ALPHAVMSi_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

---

**getLongSparseRecord**

**public static long getLongSparseRecord(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

class 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
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
Method Summary
### create

```java
public static Entry create(Attribute myAttribute, long id, long dataType, java.lang.Object data)
    Creates a new global or variable attribute entry.
```

### delete

```java
void delete()
    Deletes this entry.
```

### getData

```java
java.lang.Object getData()
    Gets the data for this entry.
```

### getDataType

```java
long getDataType()
    Gets the CDF data type of this entry.
```

###getID

```java
long getID()
    Gets the ID of this entry.
```

###getName

```java
java.lang.String getName()
    Gets the name of this entry.
```

###getNumElements

```java
long getNumElements()
    Gets the number of elements in this entry.
```

###putData

```java
void putData(long dataType, java.lang.Object data)
    Put the entry data into the CDF.
```

###rename

```java
void rename(java.lang.String name)
    This method is here as a placeholder since the Entry class implements the
    CDFObject interface that includes "rename".
```

###updateDataSpec

```java
void updateDataSpec(long dataType, long numElements)
    Update the data specification (data type and number of elements) of the entry.
```
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_UINT1 - 1-byte, unsigned integer
data - the entry data to be added

**Returns:**
newly created attribute entry

**Throws:**

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 - 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.
**getNumElements**

```java
public long getNumElements()
```

Gets the number of elements in this entry. For CDF_CHAR, it returns the number of characters stored.

<table>
<thead>
<tr>
<th>Entry data</th>
<th>Number of elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>20.8</td>
<td>1</td>
</tr>
<tr>
<td>10 20 30</td>
<td>3</td>
</tr>
<tr>
<td>20.8 20.9</td>
<td>2</td>
</tr>
<tr>
<td>&quot;Upper Limits&quot;</td>
<td>12</td>
</tr>
</tbody>
</table>

**Returns:**
the number of elements stored in this entry

---

**getData**

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

Gets the data for this entry.

**Returns:**
the data for this entry

---

**getID**

```java
public longgetID()
```
Gets the ID of this entry.

**Returns:**
the ID/number of this entry

---

### getName

**Description:**
public java.lang.String getName()

This method 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

**Description:**
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 -- 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.

    Throws:
        CDFException

putData

public void putData(long dataType,
                     java.lang.Object data)
    throws CDFException

    Put the entry data into the CDF.

    Throws:
        CDFException
public class Epoch
extends java.lang.Object
implements CDFConstants

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 = " , at ";
System.out.println(Epoch.encodex(ep,format));
# Field Summary

Fields inherited from interface gsfc.nssdc.cdf.CDFConstants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AHUFF_COMPRESSION</td>
<td>Alpha Huff compression</td>
</tr>
<tr>
<td>ALPHAOSF1_DECODING</td>
<td>Alpha OSF-1 decoding</td>
</tr>
<tr>
<td>ALPHAOSF1_ENCODING</td>
<td>Alpha OSF-1 encoding</td>
</tr>
<tr>
<td>ALPHAVMSd_DECODING</td>
<td>Alpha VMSd decoding</td>
</tr>
<tr>
<td>ALPHAVMSd_ENCODING</td>
<td>Alpha VMSd encoding</td>
</tr>
<tr>
<td>ALPHAVMSq_DECODING</td>
<td>Alpha VMSq decoding</td>
</tr>
<tr>
<td>ALPHAVMSq_ENCODING</td>
<td>Alpha VMSq encoding</td>
</tr>
<tr>
<td>ALPHAVMSi_DECODING</td>
<td>Alpha VMSi decoding</td>
</tr>
<tr>
<td>ALPHAVMSi_ENCODING</td>
<td>Alpha VMSi encoding</td>
</tr>
<tr>
<td>ATTR_</td>
<td>Attribute</td>
</tr>
<tr>
<td>ATTR_EXISTENCE_</td>
<td>Attribute existence</td>
</tr>
<tr>
<td>ATTR_EXISTS</td>
<td>Attribute exists</td>
</tr>
<tr>
<td>ATTR_MAXgENTRY_</td>
<td>Attribute max g entry</td>
</tr>
<tr>
<td>ATTR_MAXrENTRY_</td>
<td>Attribute max r entry</td>
</tr>
<tr>
<td>ATTR_MAXzENTRY_</td>
<td>Attribute max z entry</td>
</tr>
<tr>
<td>ATTR_NAME_</td>
<td>Attribute name</td>
</tr>
<tr>
<td>ATTR_NAME_TRUNC</td>
<td>Attribute name truncated</td>
</tr>
<tr>
<td>ATTRNUMBER_</td>
<td>Attribute number</td>
</tr>
<tr>
<td>ATTR_NUMgENTRIES_</td>
<td>Attribute num g entries</td>
</tr>
<tr>
<td>ATTR_NUMrENTRIES_</td>
<td>Attribute num r entries</td>
</tr>
<tr>
<td>ATTR_NUMzENTRIES_</td>
<td>Attribute num z entries</td>
</tr>
<tr>
<td>ATTR_SCOPE_</td>
<td>Attribute scope</td>
</tr>
<tr>
<td>BACKWARD_</td>
<td>Backward</td>
</tr>
<tr>
<td>BACKWARDFILEoff</td>
<td>Backward file off</td>
</tr>
<tr>
<td>BACKWARDFILEon</td>
<td>Backward file on</td>
</tr>
<tr>
<td>BAD_ALLOCATE_RECS</td>
<td>Bad allocate records</td>
</tr>
<tr>
<td>BAD_ARGUMENT</td>
<td>Bad argument</td>
</tr>
<tr>
<td>BAD_ATTR_NAME</td>
<td>Bad attribute name</td>
</tr>
<tr>
<td>BAD_ATTR_NUM</td>
<td>Bad attribute number</td>
</tr>
<tr>
<td>BAD_BLOCKING_FACTOR</td>
<td>Bad blocking factor</td>
</tr>
<tr>
<td>BAD_CACHE_SIZE</td>
<td>Bad cache size</td>
</tr>
<tr>
<td>BAD_CDF_EXTENSION</td>
<td>Bad CDF extension</td>
</tr>
<tr>
<td>BAD_CDF_ID</td>
<td>Bad CDF ID</td>
</tr>
<tr>
<td>BAD_CDF_NAME</td>
<td>Bad CDF name</td>
</tr>
<tr>
<td>BAD_CDFSTATUS</td>
<td>Bad CDF status</td>
</tr>
<tr>
<td>BAD_COMPRESSION_PARM</td>
<td>Bad compression parameter</td>
</tr>
<tr>
<td>BAD_DATA_TYPE</td>
<td>Bad data type</td>
</tr>
<tr>
<td>BAD_DECODING</td>
<td>Bad decoding</td>
</tr>
<tr>
<td>BAD_DIM_COUNT</td>
<td>Bad dim count</td>
</tr>
<tr>
<td>BAD_DIM_INDEX</td>
<td>Bad dim index</td>
</tr>
<tr>
<td>BAD_DIM_INTERVAL</td>
<td>Bad dim interval</td>
</tr>
<tr>
<td>BAD_DIM_SIZE</td>
<td>Bad dim size</td>
</tr>
<tr>
<td>BAD_ENCODING</td>
<td>Bad encoding</td>
</tr>
<tr>
<td>BAD_ENTRY_NUM</td>
<td>Bad entry number</td>
</tr>
<tr>
<td>BAD_FNC_OR_ITEM</td>
<td>Bad function or item</td>
</tr>
<tr>
<td>BAD_FORMAT</td>
<td>Bad format</td>
</tr>
<tr>
<td>BAD_INITIAL_RECS</td>
<td>Bad initial records</td>
</tr>
<tr>
<td>BAD_MAJORITY</td>
<td>Bad majority</td>
</tr>
<tr>
<td>BAD_MALLOC</td>
<td>Bad malloc</td>
</tr>
<tr>
<td>BAD_NEGtoPOSfp0_MODE</td>
<td>Bad neg to pos fp 0 mode</td>
</tr>
<tr>
<td>BAD_NUM_DIMS</td>
<td>Bad num dims</td>
</tr>
<tr>
<td>BAD_NUM_ELEMs</td>
<td>Bad num elements</td>
</tr>
<tr>
<td>BAD_NUM_VARS</td>
<td>Bad num vars</td>
</tr>
<tr>
<td>BAD_READONLY_MODE</td>
<td>Bad readonly mode</td>
</tr>
<tr>
<td>BAD_REC_COUNT</td>
<td>Bad rec count</td>
</tr>
<tr>
<td>BAD_REC_INTERVAL</td>
<td>Bad rec interval</td>
</tr>
<tr>
<td>BAD_REC_NUM</td>
<td>Bad rec num</td>
</tr>
<tr>
<td>BAD_SCOPE</td>
<td>Bad scope</td>
</tr>
<tr>
<td>BAD_SCRATCH_DIR</td>
<td>Bad scratch dir</td>
</tr>
<tr>
<td>BAD_SPARSEARRAYS_PARM</td>
<td>Bad sparse arrays parameter</td>
</tr>
<tr>
<td>BAD_VAR_NAME</td>
<td>Bad var name</td>
</tr>
<tr>
<td>BAD_VAR_NUM</td>
<td>Bad var num</td>
</tr>
<tr>
<td>BAD_zMODE</td>
<td>Bad z mode</td>
</tr>
<tr>
<td>CANNOT_ALLOCATE_RECORDS</td>
<td>Cannot allocate records</td>
</tr>
<tr>
<td>CANNOT_CHANGE</td>
<td>Cannot change</td>
</tr>
<tr>
<td>CANNOT_COMPRESS</td>
<td>Cannot compress</td>
</tr>
<tr>
<td>CANNOT_COPY</td>
<td>Cannot copy</td>
</tr>
<tr>
<td>CANNOT_SPARSEARRAYS</td>
<td>Cannot sparse arrays</td>
</tr>
<tr>
<td>CANNOT_SPARSERECORDS</td>
<td>Cannot sparse records</td>
</tr>
<tr>
<td>CDF_</td>
<td>CDF</td>
</tr>
<tr>
<td>CDF_ACCESS_</td>
<td>CDF access</td>
</tr>
<tr>
<td>CDF_ATTR_NAME_LEN</td>
<td>CDF attribute name length</td>
</tr>
<tr>
<td>CDF_BYTE</td>
<td>CDF byte</td>
</tr>
<tr>
<td>CDF_CACHE_SIZE</td>
<td>CDF cache size</td>
</tr>
<tr>
<td>CDF_CHAR</td>
<td>CDF char</td>
</tr>
<tr>
<td>CDF_CLOSE_ERROR</td>
<td>CDF close error</td>
</tr>
<tr>
<td>CDF_COMPRESSION</td>
<td>CDF compression</td>
</tr>
<tr>
<td>CDF_COPYRIGHT</td>
<td>CDF copyright</td>
</tr>
<tr>
<td>CDF_COPYRIGHT_LEN</td>
<td>CDF copyright length</td>
</tr>
<tr>
<td>CDF_CREATE_ERROR</td>
<td>CDF create error</td>
</tr>
<tr>
<td>CDF_DECODING</td>
<td>CDF decoding</td>
</tr>
<tr>
<td>CDF_DELETE_ERROR</td>
<td>CDF delete error</td>
</tr>
<tr>
<td>CDF_DOUBLE</td>
<td>CDF double</td>
</tr>
<tr>
<td>CDF_ENCODING</td>
<td>CDF encoding</td>
</tr>
<tr>
<td>CDF_EPOCH</td>
<td>CDF epoch</td>
</tr>
<tr>
<td>CDF_EPOCH16</td>
<td>CDF epoch 16</td>
</tr>
<tr>
<td>CDF_EXISTS</td>
<td>CDF exists</td>
</tr>
<tr>
<td>CDF_FLOAT</td>
<td>CDF float</td>
</tr>
<tr>
<td>CDF_FORMAT_</td>
<td>CDF format</td>
</tr>
<tr>
<td>CDF_INCREMENT</td>
<td>CDF increment</td>
</tr>
<tr>
<td>CDF_INFO_</td>
<td>CDF info</td>
</tr>
<tr>
<td>CDF_INT1</td>
<td>CDF int 1</td>
</tr>
<tr>
<td>CDF_INT2</td>
<td>CDF int 2</td>
</tr>
<tr>
<td>CDF_INT4</td>
<td>CDF int 4</td>
</tr>
<tr>
<td>CDF_INTERNAL_ERROR</td>
<td>CDF internal error</td>
</tr>
<tr>
<td>CDF_MAJORITY_</td>
<td>CDF majority</td>
</tr>
<tr>
<td>CDF_MAX_DIMS</td>
<td>CDF max dims</td>
</tr>
<tr>
<td>CDF_MAX_PARMS</td>
<td>CDF max params</td>
</tr>
<tr>
<td>CDF_MIN_DIMS</td>
<td>CDF min dims</td>
</tr>
<tr>
<td>CDF_NAME_</td>
<td>CDF name</td>
</tr>
<tr>
<td>CDF_NAME_TRUNC</td>
<td>CDF name truncated</td>
</tr>
<tr>
<td>CDF_NEGtoPOSfp0_MODE_</td>
<td>CDF neg to pos fp 0 mode</td>
</tr>
<tr>
<td>CDF_NUMATTRS_</td>
<td>CDF num attrs</td>
</tr>
<tr>
<td>CDF_NumqATTRS_</td>
<td>CDF num q attrs</td>
</tr>
<tr>
<td>CDF_NumrVARS_</td>
<td>CDF num r vars</td>
</tr>
<tr>
<td>CDF_NumvATTRS_</td>
<td>CDF num v attrs</td>
</tr>
<tr>
<td>CDF_NUMzVARS_</td>
<td>CDF num z vars</td>
</tr>
<tr>
<td>CDF_OK</td>
<td>CDF ok</td>
</tr>
<tr>
<td>CDF_OPEN_ERROR</td>
<td>CDF open error</td>
</tr>
<tr>
<td>CDF_PATHNAME_LEN</td>
<td>CDF pathname length</td>
</tr>
<tr>
<td>CDF_READ_ERROR</td>
<td>CDF read error</td>
</tr>
<tr>
<td>CDF_READONLY_MODE</td>
<td>CDF read only mode</td>
</tr>
<tr>
<td>CDF_REAL4</td>
<td>CDF real 4</td>
</tr>
<tr>
<td>CDF_REAL8</td>
<td>CDF real 8</td>
</tr>
<tr>
<td>CDF_RELEASE_</td>
<td>CDF release</td>
</tr>
<tr>
<td>CDF_SCRATCHDIR_</td>
<td>CDF scratch dir</td>
</tr>
<tr>
<td>CDF_STATUS_</td>
<td>CDF status</td>
</tr>
<tr>
<td>CDF_STATUSTEXT_LEN</td>
<td>CDF status text length</td>
</tr>
<tr>
<td>CDF_UNICODE</td>
<td>CDF unicode</td>
</tr>
<tr>
<td>CDF_UINT1</td>
<td>CDF uint 1</td>
</tr>
<tr>
<td>CDF_UINT2</td>
<td>CDF uint 2</td>
</tr>
<tr>
<td>CDF_UINT4</td>
<td>CDF uint 4</td>
</tr>
<tr>
<td>CDF_VAR_NAME_LEN</td>
<td>CDF var name length</td>
</tr>
<tr>
<td>CDF_VERSION_</td>
<td>CDF version</td>
</tr>
<tr>
<td>CDF_WARN</td>
<td>CDF warn</td>
</tr>
<tr>
<td>CDF_WRITE_ERROR</td>
<td>CDF write error</td>
</tr>
</tbody>
</table>
**Constructor Summary**

*Epoch* ()

**Method Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static long[]</td>
<td><strong>breakdown</strong> (double epoch)</td>
</tr>
<tr>
<td></td>
<td>Breaks an EPOCH value down into its component parts.</td>
</tr>
<tr>
<td>static double</td>
<td><strong>compute</strong> (long year, long month, long day, long hour, long minute, long second, long msec)</td>
</tr>
<tr>
<td></td>
<td>Computes an EPOCH value based on its component parts.</td>
</tr>
<tr>
<td>static java.lang.String</td>
<td><strong>encode</strong> (double epoch)</td>
</tr>
<tr>
<td></td>
<td>Converts an EPOCH value into a readable date/time string.</td>
</tr>
<tr>
<td>static java.lang.String</td>
<td><strong>encode1</strong> (double epoch)</td>
</tr>
<tr>
<td></td>
<td>Converts an EPOCH value into a readable date/time string.</td>
</tr>
<tr>
<td>static java.lang.String</td>
<td><strong>encode2</strong> (double epoch)</td>
</tr>
<tr>
<td></td>
<td>Converts an EPOCH value into a readable date/time string.</td>
</tr>
<tr>
<td>static java.lang.String</td>
<td><strong>encode3</strong> (double epoch)</td>
</tr>
<tr>
<td></td>
<td>Converts an EPOCH value into a readable date/time string.</td>
</tr>
<tr>
<td>static java.lang.String</td>
<td><strong>encodex</strong> (double epoch, java.lang.String formatString)</td>
</tr>
<tr>
<td></td>
<td>Converts an EPOCH value into a readable date/time string using the specified format.</td>
</tr>
<tr>
<td>static double</td>
<td><strong>parse</strong> (java.lang.String inString)</td>
</tr>
<tr>
<td></td>
<td>This function parses an input date/time string and returns an EPOCH value.</td>
</tr>
<tr>
<td>static double</td>
<td><strong>parse1</strong> (java.lang.String inString)</td>
</tr>
<tr>
<td></td>
<td>This function parses an input date/time string and returns an EPOCH value.</td>
</tr>
<tr>
<td>static double</td>
<td><strong>parse2</strong> (java.lang.String inString)</td>
</tr>
<tr>
<td></td>
<td>This function parses an input date/time string and returns an EPOCH value.</td>
</tr>
<tr>
<td>static double</td>
<td><strong>parse3</strong> (java.lang.String inString)</td>
</tr>
<tr>
<td></td>
<td>This function parses an input date/time string and returns an EPOCH value.</td>
</tr>
</tbody>
</table>

Methods inherited from class java.lang.Object
Constructor Detail

Epoch

public Epoch()

Method Detail

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 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 encodeEPOCH.

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
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:    yyyymmddd.ttttttt
Examples:  19950508.0000000
            19671231.58     (== 19671213.5800000)

The expected format is the same as that produced by encodeEPOCH1.

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 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:   yyyymmdddhhmmss
Examples: 19950508000000
            19671231235959

The expected format is the same as that produced by encodeEPOCH2.

Parameters:
    inString - the epoch in string representation

Returns:
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 encodeEPOCH3.

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

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

---

**encode1**

```java
public static java.lang.String encode1(double epoch)
```

Converts an EPOCH value into a readable date/time string.

**Format:** yyyymmd.tttt

**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

---

**encode2**

```java
public static java.lang.String encode2(double epoch)
```

Converts an EPOCH value into a readable date/time string.

**Format:** yyyymmdhhmss

**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**(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

---

**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**

```java
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.

**breakdown**

```java
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
gsfc.nssdc.cdf.util

Class Epoch16

gsfc.nssdc.cdf.util.Epoch16

java.lang.Object

All Implemented Interfaces:

CDFConstants

public class Epoch16
extends java.lang.Object
implements CDFConstants

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, nanoseconds 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));
// Print out the date using format
String format = " , at ";
System.out.println(Epoch16.encodex(epoch16,format));
### Field Summary

Fields inherited from interface gsfc.nssdc.cdf. **CDFConstants**

<table>
<thead>
<tr>
<th>Field Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>AHUFF_COMPRESSION, ALPHAOFS1_DECODING, ALPHAOFS1_ENCODING, ALPHAVMSd_DECODING, ALPHAVMSd_ENCODING, ALPHAVMSq_DECODING, ALPHAVMSg_ENCODING, ALPHAVMSi_DECODING, ALPHAVMSi_ENCODING, ATTR, ATTR_EXISTENCE, ATTR_EXISTS, ATTR_MAXqENTRY, ATTR_MAXrENTRY, ATTR_MAXzENTRY, ATTR_NAME, ATTR_NAME_TRUNC, ATTR_NUMBER, ATTR_NUMqENTRIES, ATTR_NUMrENTRIES, ATTR_NUMzENTRIES, ATTR_SCOPE, BACKWARD, BACKWARDFILEoff, BACKWARDFILEon, BAD_ALLOCATE_RECS, BAD_ARGUMENT, BAD_ATTR_NAME, BAD_ATTRIBUTE, BAD_BLOCKING_FACTOR, BAD_CACHE_SIZE, BAD_CDF_EXTENSION, BAD_CDF_ID, BAD_CDF_NAME, BAD_CDFSTATUS, 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_MAJOR, BAD_MALLOC, BAD_NEGtoPOSfp0_MODE, BAD_NUM_DIMS, BAD_NUM_ELEMS, 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 zIndex, CANNOT_ALLOCATE_RECORDS, CANNOT_CHANGE, CANNOT_COMPRESS, CANNOT_COPY, CANNOT_SPARSEARRAYS, CANNOT_SPARSERECORDS, CDF, CDF_ACCESS, CDF_ATTR_NAME_LEN, CDF_BYTE, CDF_CACHESIZE, CDF_CHAR, 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_INTERNAL_ERROR, CDF_MAJOR, CDF_MAJORITY, CDF_MAX_DIMS, CDF_MAX_PARMS, CDF_MIN_DIMS, CDF_NAME, CDF_NAME_TRUNC, CDF_NEGtoPOSfp0_MODE, CDF_NUMATTRS, CDF_NUMqATTRS, 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_SCRATCHDIR, CDF_STATUS</td>
</tr>
</tbody>
</table>
Constructor Summary

**Epoch16 ()**

Method Summary

<table>
<thead>
<tr>
<th>static long[]</th>
<th><strong>breakdown</strong> (java.lang.Object epoch)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Breaks an EPOCH16 value down into its component parts.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static double</th>
<th><strong>compute</strong> (long year, long month, long day, long hour, long minute, long second, long msec, long usec, long nsec, long psec, java.lang.Object epoch)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Computes an EPOCH16 value based on its component parts.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static java.lang.String</th>
<th><strong>encode</strong> (java.lang.Object epoch)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Converts an EPOCH16 value into a readable date/time string.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static java.lang.String</th>
<th><strong>encode1</strong> (java.lang.Object epoch)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Converts an EPOCH16 value into a readable date/time string.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static java.lang.String</th>
<th><strong>encode2</strong> (java.lang.Object epoch)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Converts an EPOCH16 value into a readable date/time string.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static java.lang.String</th>
<th><strong>encode3</strong> (java.lang.Object epoch)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Converts an EPOCH16 value into a readable date/time string.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static java.lang.String</th>
<th><strong>encodex</strong> (java.lang.Object epoch, java.lang.String formatString)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Converts an EPOCH16 value into a readable date/time string using the specified format.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static java.lang.String</th>
<th><strong>parse</strong> (java.lang.String inString)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This function parses an input date/time string and returns an EPOCH16 value.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>static java.lang.String</th>
<th><strong>parse1</strong> (java.lang.String inString)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This function parses an input date/time string and returns an EPOCH16 value.</td>
</tr>
</tbody>
</table>
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.

Methods inherited from class java.lang.Object

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

Constructor Detail

Epoch16

public Epoch16()

Method Detail

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**

```java
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
19671231.580000000000000
```

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**

```java
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.

**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
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**

```java
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`
**encode**

```java
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**

```java
public static java.lang.String encode1(java.lang.Object epoch)
```

Converts an EPOCH16 value into a readable date/time string.

**Format:** yyyymmdd.tttttttttttttttttttttttttttttttttttttttttt

**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:         yyyymmddhhmss
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

-----------------------------------

encodex

public static java.lang.String encodex(java.lang.Object epoch,
Converting 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**

```java
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

**Returns:**
**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
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:**

```java
// 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
```
### Constructor Summary

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>EpochNative()</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>static long[]</td>
</tr>
<tr>
<td>breakdown</td>
</tr>
<tr>
<td>static double</td>
</tr>
<tr>
<td>compute</td>
</tr>
<tr>
<td>static java.</td>
</tr>
<tr>
<td>lang.String</td>
</tr>
<tr>
<td>encode</td>
</tr>
<tr>
<td>static java.</td>
</tr>
<tr>
<td>lang.String</td>
</tr>
<tr>
<td>encode1</td>
</tr>
<tr>
<td>static java.</td>
</tr>
<tr>
<td>lang.String</td>
</tr>
<tr>
<td>encode2</td>
</tr>
<tr>
<td>static java.</td>
</tr>
<tr>
<td>lang.String</td>
</tr>
<tr>
<td>encode3</td>
</tr>
<tr>
<td>static java.</td>
</tr>
<tr>
<td>lang.String</td>
</tr>
<tr>
<td>encodex</td>
</tr>
<tr>
<td>static double</td>
</tr>
<tr>
<td>parse</td>
</tr>
<tr>
<td>static double</td>
</tr>
<tr>
<td>parse1</td>
</tr>
</tbody>
</table>

- `breakdown(double epoch)`: Mirrors EPOCHbreakdown from the CDF library.
- `compute(long year, long month, long day, long hour, long minute, long second, long msec)`: Mirrors computeEPOCH from the CDF library.
- `encode(double epoch)`: Mirrors encodeEPOCH from the CDF library.
- `encode1(double epoch)`: Mirrors encodeEPOCH1 from the CDF library.
- `encode2(double epoch)`: Mirrors encodeEPOCH2 from the CDF library.
- `encode3(double epoch)`: Mirrors encodeEPOCH3 from the CDF library.
- `encodex(double epoch, java.lang.String format)`: Mirrors encodeEPOCHx from the CDF library.
- `parse(java.lang.String sEpoch)`: Mirrors parseEPOCH from CDF library.
- `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.

Methods inherited from class java.lang.Object  
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

EpochNative

public EpochNative()
compute

```java
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.

breakdown

```java
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

```java
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

```java
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.

```
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.

```
public static java.lang.String encode3(double epoch)
```

Mirrors encodeEPOCH3 from the CDF library. See Section 8.6 of the
encodex

```java
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

parse

```java
public static double parse(java.lang.String sEpoch)
```

MIRRORS parseEPOCH from CDF library. See Section 8.8 of the
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

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
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.
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`
Method Summary

void allocateBlock(long firstRec, long lastRec)

Allocates a range of records for this variable.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>allocateRecords(long num0toRecords)</code></td>
<td>Allocates a number of records, starting from record number 0.</td>
</tr>
<tr>
<td><code>checkPadValueExistence()</code></td>
<td>Checks if the pad value has been defined for this variable.</td>
</tr>
<tr>
<td><code>concatenateDataRecords(Variable destVar)</code></td>
<td>Concatenates this variable's data records to the destination variable.</td>
</tr>
<tr>
<td><code>confirmCacheSize()</code></td>
<td>Gets the number of 512-byte cache buffers defined for this variable.</td>
</tr>
<tr>
<td><code>confirmPadValue()</code></td>
<td>Checks the existence of an explicitly specified pad value for the current variable.</td>
</tr>
<tr>
<td><code>confirmReservePercent()</code></td>
<td>Gets the reserve percentage set for this variable.</td>
</tr>
<tr>
<td><code>copy(CDF destCDF, java.lang.String varName)</code></td>
<td>Copies this variable into a new variable and puts it into the designated CDF file.</td>
</tr>
<tr>
<td><code>copy(java.lang.String varName)</code></td>
<td>Copies this variable to a new variable.</td>
</tr>
<tr>
<td><code>copyDataRecords(Variable destVar)</code></td>
<td>Copies this variable's data to the destination variable.</td>
</tr>
<tr>
<td><code>create(CDF myCDF, java.lang.String varName, long dataType, long numElements, long numDims, long[] dimSizes, long recVary, long[] dimVarys)</code></td>
<td>Creates a variable.</td>
</tr>
<tr>
<td><code>delete()</code></td>
<td>Deletes this variable.</td>
</tr>
<tr>
<td><code>deleteRecords(long firstRec, long lastRec)</code></td>
<td>Deletes a range of records from this variable.</td>
</tr>
<tr>
<td><code>duplicate(CDF destCDF, java.lang.String varName)</code></td>
<td>Duplicates this variable and put it into the designated CDF file.</td>
</tr>
<tr>
<td><code>duplicate(java.lang.String varName)</code></td>
<td>Duplicates this variable to a new variable.</td>
</tr>
<tr>
<td>Type</td>
<td>Method Name</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td><code>long</code></td>
<td><code>getAllocatedFrom(long recNum)</code></td>
</tr>
<tr>
<td><code>long</code></td>
<td><code>getAllocatedTo(long firstRec)</code></td>
</tr>
<tr>
<td><code>java.util.Vector</code></td>
<td><code>getAttributes()</code></td>
</tr>
<tr>
<td><code>long</code></td>
<td><code>getBlockingFactor()</code></td>
</tr>
<tr>
<td><code>java.lang.String</code></td>
<td><code>getCompression()</code></td>
</tr>
<tr>
<td><code>long[]</code></td>
<td><code>getCompressionParms()</code></td>
</tr>
<tr>
<td><code>long</code></td>
<td><code>getCompressionPct()</code></td>
</tr>
<tr>
<td><code>long</code></td>
<td><code>getCompressionType()</code></td>
</tr>
<tr>
<td><code>long</code></td>
<td><code>getDataType()</code></td>
</tr>
<tr>
<td><code>long[]</code></td>
<td><code>getDimSizes()</code></td>
</tr>
<tr>
<td><code>long[]</code></td>
<td><code>getDimVariances()</code></td>
</tr>
<tr>
<td><code>java.lang.Object</code></td>
<td><code>getEntryData(java.lang.String attrName)</code></td>
</tr>
<tr>
<td><code>java.lang.Object</code></td>
<td><code>getHyperData(long recNum, long recCount, long recInterval, long[] dimIndices, long[] dimCounts, long[] dimIntervals)</code></td>
</tr>
<tr>
<td><code>CDFData</code></td>
<td><code>getHyperDataObject(long recNum, long recCount, long recInterval, long[] dimIndices, long[] dimCounts, long[] dimIntervals)</code></td>
</tr>
<tr>
<td>Type</td>
<td>Method</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>long</td>
<td><code>getID()</code></td>
</tr>
<tr>
<td>long</td>
<td><code>getMaxAllocatedRecord()</code></td>
</tr>
<tr>
<td>long</td>
<td><code>getMaxWrittenRecord()</code></td>
</tr>
<tr>
<td>CDF</td>
<td><code>getMyCDF()</code></td>
</tr>
<tr>
<td>java.lang. String</td>
<td><code>getName()</code></td>
</tr>
<tr>
<td>long</td>
<td><code>getNumAllocatedRecords()</code></td>
</tr>
<tr>
<td>long</td>
<td><code>getNumDims()</code></td>
</tr>
<tr>
<td>long</td>
<td><code>getNumElements()</code></td>
</tr>
<tr>
<td>long</td>
<td><code>getNumWrittenRecords()</code></td>
</tr>
<tr>
<td>java.lang. Object</td>
<td><code>getPadValue()</code></td>
</tr>
<tr>
<td>java.lang. Object</td>
<td><code>getRecord(long recNum)</code></td>
</tr>
<tr>
<td>CDFData</td>
<td><code>getRecordObject(long recNum)</code></td>
</tr>
<tr>
<td>CDFData</td>
<td><code>getRecordsObject(long recNum, long numRecs)</code></td>
</tr>
<tr>
<td>boolean</td>
<td><code>getRecVariance()</code></td>
</tr>
<tr>
<td>java.lang. Object</td>
<td><code>getScalarData()</code></td>
</tr>
<tr>
<td>java.lang. Object</td>
<td><code>getScalarData(long recNum)</code></td>
</tr>
<tr>
<td>CDFData</td>
<td><strong>getScalarDataObject</strong> ()</td>
</tr>
<tr>
<td>---------</td>
<td>--------------------------</td>
</tr>
<tr>
<td></td>
<td>Get the scalar data from a non-record varying 0-dimensional variable.</td>
</tr>
<tr>
<td>CDFData</td>
<td><strong>getScalarDataObject</strong> (long recNum)</td>
</tr>
<tr>
<td></td>
<td>Get the scalar data from this record varying 0-dimensional variable.</td>
</tr>
<tr>
<td>java.lang.Object</td>
<td><strong>getSingleData</strong> (long recNum, long[] indices)</td>
</tr>
<tr>
<td></td>
<td>Gets a single data value.</td>
</tr>
<tr>
<td>CDFData</td>
<td><strong>getSingleDataObject</strong> (long recNum, long[] indices)</td>
</tr>
<tr>
<td></td>
<td>Gets a single data object from this variable.</td>
</tr>
<tr>
<td>long</td>
<td><strong>getSparseRecords</strong> ()</td>
</tr>
<tr>
<td></td>
<td>Gets the sparse record type for this variable.</td>
</tr>
<tr>
<td>void</td>
<td><strong>putEntry</strong> (Attribute attr, long dataType, java.lang.Object data)</td>
</tr>
<tr>
<td></td>
<td>Creates an attribute entry for this variable.</td>
</tr>
<tr>
<td>void</td>
<td><strong>putEntry</strong> (java.lang.String attrName, long dataType, java.lang.Object data)</td>
</tr>
<tr>
<td></td>
<td>Creates an attribute entry for this variable.</td>
</tr>
<tr>
<td>CDFData</td>
<td><strong>putHyperData</strong> (long recNum, long recCount, long recInterval, long[] dimIndices, long[] dimCounts, long[] dimIntervals, java.lang.Object data)</td>
</tr>
<tr>
<td></td>
<td>Writes one or more values from the current z variable.</td>
</tr>
<tr>
<td>CDFData</td>
<td><strong>putRecord</strong> (long recNum, java.lang.Object data)</td>
</tr>
<tr>
<td></td>
<td>Adds a single record to a record-varying variable.</td>
</tr>
<tr>
<td>CDFData</td>
<td><strong>putRecord</strong> (java.lang.Object data)</td>
</tr>
<tr>
<td></td>
<td>Adds a single record to a non-record-varying variable.</td>
</tr>
<tr>
<td>CDFData</td>
<td><strong>putScalarData</strong> (long recNum, java.lang.Object data)</td>
</tr>
<tr>
<td></td>
<td>Adds a scalar data to this variable (of 0 dimensional).</td>
</tr>
<tr>
<td>CDFData</td>
<td><strong>putScalarData</strong> (java.lang.Object data)</td>
</tr>
<tr>
<td></td>
<td>Adds a scalar data to this variable (of 0 dimensional).</td>
</tr>
<tr>
<td>CDFData</td>
<td><strong>putSingleData</strong> (long recNum, long[] indices, java.lang.Object data)</td>
</tr>
<tr>
<td></td>
<td>Adds a single data value to this variable.</td>
</tr>
<tr>
<td>void</td>
<td><strong>rename</strong> (java.lang.String newName)</td>
</tr>
<tr>
<td></td>
<td>Renames the current variable.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| `selectCacheSize` | (long cacheSize)  
Sets the number of 512-byte cache buffers to be used. |
| `selectReservePercent` | (long reservePercent)  
Sets the reserve percentage to be used for this variable. |
| `setBlockingFactor` | (long blockingFactor)  
Sets the blocking factor for this variable. |
| `setCompression` | (long cType, long[] cParms)  
Sets the compression type and parameters for this variable. |
| `setDimVariances` | (long[] dimVariances)  
Sets the dimension variances for this variable. |
| `setInitialRecords` | (long nRecords)  
Sets the number of records to be written initially for this variable. |
| `setPadValue` | (java.lang.Object padValue)  
Sets the pad value for this variable. |
| `setRecVariance` | (long recVariance)  
Sets the record variance for this variable. |
| `setSparseRecords` | (long sparseRecords)  
Sets the sparse record type for this variable. |
| `toString` | ()  
Gets the name of this variable. |
| `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`
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:

```java
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.

```java
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_UNIT2 - 2-byte, unsigned integer
  - CDF_INT4 - 4-byte, signed integer
- CDF_UINT4 - 4-byte, unsigned 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

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**

```java
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

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
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.

**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.

**Parameters:**
- destVar - the destination variable to copy data into

**Throws:**
- CDFException - if there was a problem copying data records
**concatenateDataRecords**

```java
public void concatenateDataRecords(Variable destVar)
    throws CDFException
```

Concatenates this variable's data records to the destination variable.

**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.

**Parameters:**
- `destVar` - the destination variable to copy data records into

**Throws:**
- `CDFException` - if there was a problem copying data records

---

**getEntryData**

```java
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:

```java
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**

```java
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:

```java
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:

```java
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 the entire record (containing 6 elements) from the first record:

    Variable var = cdf.getVariable("TestData");
    int[][][] data = (int [][][]) var.getRecord(0L);

Parameters:
    recNum - the record number to retrieve data from

Returns:
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:
CDFObject containing the requested data record

Throws:
CDFException - if there was a problem getting a record

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

Throws:
CDFException - if there was a problem getting a record
Returns:
CDFObject containing the requested 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,
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:

```java
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:

```java
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:

```java
Variable var = cdf.getVariable("TestData");
int[][] data = (int[][]) var.getHyperData(0L, 3L, 2L,
new long[]{0, 0},
new long[]{3, 2},
new long[]{1, 1});
```
The following example will extract the first column from the first and second records:

```java
Variable var = cdf.getVariable("TestData");
int[][] data = (int[][]) var.getHyperData(0L, 2L, 1L,
    new long[] {0,
        new long[] {0,
            new long[] {1,
                new long[] {0,
                    new long[] {1,
                        new long[] {1}}}}},
            new long[] {1,
                    new long[] {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

```java
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

---

putEntry

```java
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":

```java
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

```java
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":

```java
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)10));
```
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

Add 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:

```java
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**

```java
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:

```java
longitude = cdf.getVariable("Longitude");
longitude.putRecord(new Short((short)200));
```
The following example adds multiple elements (array) to record number 0:

```java
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**

```java
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 a 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:

```java
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});
```

The following example will write the first two rows of testData to the first, third, and fifth records:

```java
testData.putHyperData (0L, 3L, 2L,
                       new long[] {0, 0},
                       new long[] {2, 2},
                       new long[] {1, 1});
```

**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

---

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

cGetCompression

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.

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**

```java
public boolean getRecVariance()
```

Gets the value of record variance.

**Returns:**

True if this variable is record varying, False otherwise

---

**setDimVariances**

```java
public void setDimVariances(long[] dimVariances)
```

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**

```java
public long[] getDimVariances()
```

Gets the dimension variances for this variable.

**Returns:**

the dimension variances for this variable
### getDataType

```java
class getDataType {
    public long getDataType() {
        // Gets the CDF data type of this variable.
        // Returns:
        //  the CDF data type of this variable
    }
}
```

### deleteRecords

```java
class deleteRecords {
    public void deleteRecords(long firstRec, long lastRec) throws CDFException {
        // Deletes a range of records from this variable.
        // Parameters:
        //   firstRec - the first record to be deleted
        //   lastRec - the last record to be deleted
        // Throws:
        //   CDFException - if a problem occurs deleting records
    }
}
```

### allocateBlock

```java
class 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
    }
}
```
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

setMaxWrittenRecord

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**

```java
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**

```java
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. While the getPadValue() method always returns a pad value, it may simply be the default pad value (albeit the pad value was never defined by the user).

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.

Note: The system default pad value is returned if getPadValue() is called.

Throws:
    CDFException - if a problem occurs checking the existence of the pad value

getPadValue

public java.lang.Object getPadValue()
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
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)
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**

```java
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**

```java
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.

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.

Throws:  
CDFException

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.