CIFLibCommon
A Library of Dynamic, Resizable String class and Resizable, Variable-length array template class Reference Guide

CIFLibCommon Version 3.01

Version 1.0 Fall 1996
Version 3.01 August 1999

Olivera Tosic
John D. Westbrook
Nucleic Acid Database Project
Department of Chemistry
Rutgers, The State University of New Jersey

Please direct comments on this document to jwest@ndb.rutgers.edu.

CIFLibCommon - A Library of Dynamic, Resizable String class and Resizable, Variable-length array template class

Contents

1 Introduction

The CIFLibCommon package contains several files that converge together in the CifString class and ReVarCifArray and ReVarPCifArray template classe

CifString is dynamic, resizable string class.

- CifString class provides all function as primitive string.
- Reallocaton of memory for CifString is done automaticaly by specified increment.
- CifString class provides reference counting.

ReVarCifArray is resizable, variable-length array template class, where the type of contained object can be any other class. That means we can have, for example, resizable, variable-length array of CifStrings.

ReVarPCifArray is resizable, variable-length array template class, where the type of contained object can be any primitive type, such as integer, long, etc.

- ReVarCifArray and ReVarPCifArray have the same methods. Some private methods are different because of memory allocation.
- Reallocaton of memory for array is done automaticaly by specified increment

Note: Every class that we wont to put in ReVarCifArray has to have DeleteElement() method. This method has the same functionality as destructor.

2 CifString Public Method and Member Descriptions

The following section describes the public interface to the CifString class. Example code will be provided along with each method or member.

2.1 Constructors

This subsection contains the constructors for the CifString class.

2.1.1 CifString

NAME

CifString

PROTOTYPE

```
#include "CifString.h"

CifString(unsigned n = CifString::defsize, int gb = CifString::definc);
CifString(const char *s, int gb = CifString::definc);
CifString(const char *s, unsigned n, int gb = CifString::definc);
CifString(const CifString &s);
CifString(const CifString &s,
          unsigned ofs, unsigned n, int gb = CifString::definc);
```

EXAMPLE

```
#include "CifString.h"

CifString *a1 = new CifString();
CifString *a2 = new CifString("AAA");
CifString *a3 = new CifString(a3);
```

PURPOSE

CifString(unsigned, int) Constructor to allocate a string of allocated length of n, and a reallocation increment of gb.
*CifString(const char *, int)* Create a string copying bytes from a null-terminated array s, with a reallocation increment of gb. If gb is not zero, the initial allocated length will be the next multiple of gb >= length of s. This allows the string to grow a little without needing reallocation.

*CifString(const char *, unsigned, int)* Create a string of allocated length n,reallocation increment of gb, copying bytes from s.

CifString(const CifString &) Copy constructor, which uses share semantics.

CifString(const CifString &, unsigned, unsigned, int) A constructor to create a substring of string s. The substring shares it's data with this string, starting at offset ofs, and of length n. It has reallocation inctement of gb.

RECEIVES

CifString(unsigned, int)

`n`	allocated length
`gb`	reallocation inctement

CifString(const char *, int)

`s`	the string copied into the CifString
`gb`	reallocation inctement

CifString(const char *, unsigned, int)

`s`	the string copied into the CifString
`n`	allocated length
`gb`	reallocation inctement

CifString(const CifString &)

`s`	the CifString

CifString(const CifString &, unsigned, unsigned, int)

`s`	the CifString copied
`ofs`	ofset where to start copying the CifString
`n`	allocated length
`gb`	reallocation inctement

RETURN VALUE

None

REMARKS

CifString::defsize and CifString::definc are internally defined constants, a both of them have value 16.

Allocation may fail, with string being null.

2.2 Methods

This subsection contains the public methods for the CifString class.

2.2.1 ToLower

NAME

ToLower

PROTOTYPE

```
#include "CifString.h"

void CifString::ToLower();
```

EXAMPLE

```
#include "CifString.h"

CifString *s = new CifString("AAA");
s->ToLower();
```

PURPOSE

ToLower() Sets text to lower cases

RECEIVES

None

RETURN VALUE

None

REMARKS

None

2.2.2 ToUpper

NAME

ToUpper

PROTOTYPE

```
#include "CifString.h"

void CifString::ToUpper();
```

EXAMPLE

```
#include "CifString.h"

CifString *s = new CifString("aaa");
s->ToUpper();
```

PURPOSE

ToUpper() Sets text to upper cases

RECEIVES

None

RETURN VALUE

None

REMARKS

None

2.2.3 Clear

NAME

Clear

PROTOTYPE

```
#include "CifString.h"

void CifString::Clear();
```

EXAMPLE

```
#include "CifString.h"

CifString *s = new CifString("abc");
s->Clear();
```

PURPOSE

Clear() Makes the string become a null string.

RECEIVES

None

RETURN VALUE

None

REMARKS

None

2.2.4 RemoveBlanks

NAME

RemoveBlanks

PROTOTYPE

```
#include "CifString.h"

void CifString::RemoveBlanks();
```

EXAMPLE

```
#include "CifString.h"

CifString *s = new CifString("a b c");
s->RemoveBlanks();
```

PURPOSE

RemoveBlanks() Removes all blanks from string.

RECEIVES

None

RETURN VALUE

None

REMARKS

None

2.2.5 LineCount

NAME

LineCount

PROTOTYPE

```
#include "CifString.h"

int CifString::LineCount();
```

EXAMPLE

```
#include "CifString.h"

CifString *s = new CifString("abc");
cout<<s->LineCount()<<endl;
```

PURPOSE

LineCount() Counts the number of lines in the string.

RECEIVES

None

RETURN VALUE

Returns number of lines in the string.

REMARKS

None

2.2.6 IsNull

NAME

IsNull

PROTOTYPE

```
#include "CifString.h"

int CifString::IsNull();
```

EXAMPLE

```
#include "CifString.h"

CifString *s = new CifString("abc");
if (s->IsNull()) . . .
```

PURPOSE

IsNull() Cheks if string references to a NULL string.

RECEIVES

None

RETURN VALUE

Returns true if string references to NULL string

REMARKS

None

2.2.7 IsUnique

NAME

IsUnique

PROTOTYPE

```
#include "CifString.h"

int CifString::IsUnique();
```

EXAMPLE

```
#include "CifString.h"

CifString *s = new CifString("abc");
if (s->IsUnique()) . . .
```

PURPOSE

IsUnique() Cheks if string references only shared data or to NULL string.

RECEIVES

None

RETURN VALUE

Returns true if string has only reference to shared data or to NULL string

REMARKS

None

2.2.8 Length

NAME

Length

PROTOTYPE

```
#include "CifString.h"

int CifString::Length();
```

EXAMPLE

```
#include "CifString.h"

CifString *s = new CifString("ABC");
cout<<s->Length()<<endl;
```

PURPOSE

Length() Determines logical length of CifStirng.

RECEIVES

None

RETURN VALUE

Returns logical length of CifStirng

REMARKS

None

2.2.9 DimLength

NAME

DimLength

PROTOTYPE

```
#include "CifString.h"

int CifString::DimLength();
```

EXAMPLE

```
#include "CifString.h"

CifString *s = new CifString("ABC");
cout<<s->DimLength()<<endl;
```

PURPOSE

DimLength() Determines allocated length of CifStirng.

RECEIVES

None

RETURN VALUE

Returns allocated length of CifStirng

REMARKS

None

2.2.10 Text

NAME

Text

PROTOTYPE

```
#include "CifString.h"

int CifString::Text();
```

EXAMPLE

```
#include "CifString.h"
CifString a;

a.Copy("ABC");
cout<<a.Text()<<endl;
```

PURPOSE

Text() Returns pointer to start of the string .

RECEIVES

None

RETURN VALUE

Returns pointer to start of the string

REMARKS

None

2.2.11 Copy

NAME

Copy

PROTOTYPE

```
#include "CifString.h"

void CifString::Copy(const char *s)
void CifString::Copy(const CifString &s)
void CifString::Copy(const CifString *s)
```

EXAMPLE

```
#include "CifString.h"
CifString a;

a.Copy("CifString");
```

PURPOSE

*Copy(const char *)* Copys data from a null-terminated string *s* into this object.
Copy(const CifString &) Copys data from CifString *s* into this string.
*Copy(const CifString *)* Copys data from CifString *s* into this string.

RECEIVES

Copy(const char *)

s	null-terminated string

Copy(const CifString &)

s	reference to a CifString

Copy(const CifString *)

s	poiter to a CifString

RETURN VALUE

None

REMARKS

The string might be resized if necessary and can be.

2.2.12 InsertAt

NAME

InsertAt

PROTOTYPE

```
#include "CifString.h"

void CifString::InsertAt(usigned p, const char *s, usigned n)
void CifString::InsertAt(usigned p, const char *s)
void CifString::InsertAt(usigned p, const char c)
void CifString::InsertAt(usigned p,const CifString &s)
```

EXAMPLE

```
#include "CifString.h"
CifString a;

a.InsertAt("CifString");
```

PURPOSE

*InsertAt(usigned, const char *, usigned)* Inserts the text in *s*, up to *n* bytes, at position *p*.
*InsertAt(usigned, const char *)* Inserts the null-terminated text in *s*, at position *p*.
InsertAt(usigned, const char) Inserts one character in*c*, at position *p*.
InsertAt(usigned,const CifString &) Inserts the text in *s*, at position *p*.

RECEIVES

InsertAt(usigned, const char *, usigned)

`p`	starting position
`s`	null-terminated string
`n`	number of bytes copied

InsertAt(usigned, const char *)

`p`	starting position
`s`	null-terminated string

InsertAt(usigned, const char)

`p`	starting position
`c`	null-terminated string

InsertAt(usigned,const CifString &)

`p`	starting position
`s`	null-terminated string

RETURN VALUE

Returns number of elements inserted, or 0 if error.

REMARKS

CifString may grow if necessary and can. May have to truncate.

2.3 Operators

This subsection contains the operators for the CifString class.

2.3.1 operator=

NAME

operator=

PROTOTYPE

```
#include "CifString.h"

CifString &CifString::operator=(const CifString &s)
CifString &CifString::operator=(const char *s)
```

EXAMPLE

```
#include "CifString.h"
CifString a;

a="CifString";
```

PURPOSE

&operator=(const CifString &) Assigns one string to another. Checks for assignment to self. Uses share-semantics.
*&operator=(const char *)* Copy a null-terminated string into this object. The string might be resized if necessary and allowed.

RECEIVES

&operator=(const CifString &)

s	reference to a CifString

&operator=(const char *)

s	null-terminated string

RETURN VALUE

None

REMARKS

None

2.3.2 operator+=

NAME

operator+=

PROTOTYPE

```
#include "CifString.h"

void CifString::operator+=(const CifString &s)
void CifString::operator+=(const char *s)
void CifString::operator+=(const char c)
void CifString::operator+=(const int c)
void CifString::operator+=(const long c)
void CifString::operator+=(const double c)
```

EXAMPLE

```
#include "CifString.h"
CifString a;

a="CifString";
a+=1;
```

PURPOSE

operator+=(const CifString &) Adds CifString *s* to the end of this string.
*operator+=(const char *)* Adds a null terminated string *s* onto the end of this string. The null byte isn't added.
operator+=(const char) Adds a single character *c* to the end of the string.
operator+=(const int) Adds a null terminated string generated from *c* to the end of the string.
operator+=(const long) Adds a null terminated string generated from *c* to the end of the string.
operator+=(const double) Adds a to null terminated string generated from *c* the end of the string.

RECEIVES

operator+=(const CifString &)

`s`	reference to a CifString

operator+=(const char *)

`c`	null-terminated string

operator+=(const char)

`c`	one character

operator+=(const int)

`c`	integer value

operator+=(const long)

`c`	long integer value

operator+=(const double)

`c`	double value

RETURN VALUE

None

REMARKS

The string may grow if necessary and can be.

2.3.3 operator+

NAME

operator+

PROTOTYPE

```
#include "CifString.h"

CifString CifString::operator+(const CifString &a, const CifString &b)
```

EXAMPLE

```
#include "CifString.h"

CifString a, b, c;

a.Copy("ab");
b.Copy("cd");
c=a+b;
```

PURPOSE

operator+(const CifString &, const CifString &) Adds two strings together and returns the result.

RECEIVES

operator+(const CifString &, const CifString &)

`a`	reference to a CifString
`b`	reference to a CifString

RETURN VALUE

Returns a CifString as a result of concatenation of two input CifStings.

REMARKS

None

2.4 Comparison operators

This subsection contains the comparison operators for the CifString class.

2.4.1 operator==

NAME

operator==

PROTOTYPE

```
#include "CifString.h"

int CifString::operator==(const CifString &a, const CifString &b)
```

EXAMPLE

```
#include "CifString.h"

CifString a, b, c;

a.Copy("ab");
b.Copy("cd");
if (a==b)
   cout<<"a==b"<<endl;
else
   cout<<"a!=b"<<endl;
```

PURPOSE

operator==(const CifString &, const CifString &) Cheks if two input CifString are the same.

RECEIVES

operator==(const CifString &, const CifString &)

`a`	reference to a CifString
`b`	reference to a CifString

RETURN VALUE

Returns 1 if CifStrings are the same, otherwise 0.

REMARKS

None

2.4.2 operator!=

NAME

operator!=

PROTOTYPE

```
#include "CifString.h"

int CifString::operator!=(const CifString &a, const CifString &b)
```

EXAMPLE

```
#include "CifString.h"

CifString a, b, c;

a.Copy("ab");
b.Copy("cd");
if (a!=b)
   cout<<"a!=b"<<endl;
else
   cout<<"a==b"<<endl;
```

PURPOSE

operator!=(const CifString &, const CifString &) Cheks if two input CifString are different.

RECEIVES

operator!=(const CifString &, const CifString &)

`a`	reference to a CifString
`b`	reference to a CifString

RETURN VALUE

Returns 1 if CifStrings are different, otherwise 0.

REMARKS

None

2.4.3 operator>

NAME

operator>

PROTOTYPE

```
#include "CifString.h"

int CifString::operator$>$(const CifString &a, const CifString &b)
```

EXAMPLE

```
#include "CifString.h"

CifString a, b, c;

a.Copy("ab");
b.Copy("cd");

if (a>b) ...
```

PURPOSE

operator>(const CifString &, const CifString &) Cheks if first string is greater then second.

RECEIVES

operator>(const CifString &, const CifString &)

`a`	reference to a CifString
`b`	reference to a CifString

RETURN VALUE

Returns 1 if the first CifStrings is greater than second, otherwise 0.

REMARKS

None

2.4.4 operator>=

NAME

operator>=

PROTOTYPE

```
#include "CifString.h"

int CifString::operator$>=$(const CifString &a, const CifString &b)
```

EXAMPLE

```
#include "CifString.h"

CifString a, b, c;

a.Copy("ab");
b.Copy("cd");

if (a>=b) ...
```

PURPOSE

operator>=(const CifString &, const CifString &) Cheks if first string is greater then or equal to second.

RECEIVES

operator>=(const CifString &, const CifString &)

`a`	reference to a CifString
`b`	reference to a CifString

RETURN VALUE

Returns 1 if the first CifStrings is greater than or equal to second, otherwise 0.

REMARKS

None

2.4.5 operator<

NAME

operator<

PROTOTYPE

```
#include "CifString.h"

int CifString::operator$<$(const CifString &a, const CifString &b)
```

EXAMPLE

```
#include "CifString.h"

CifString a, b, c;

a.Copy("ab");
b.Copy("cd");

if (a<b) ...
```

PURPOSE

operator<(const CifString &, const CifString &) Cheks if first string is less then second.

RECEIVES

operator<(const CifString &, const CifString &)

`a`	reference to a CifString
`b`	reference to a CifString

RETURN VALUE

Returns 1 if the first CifStrings is less than second, otherwise 0.

REMARKS

None

2.4.6 operator<=

NAME

operator<=

PROTOTYPE

```
#include "CifString.h"

int CifString::operator$<=$(const CifString &a, const CifString &b)
```

EXAMPLE

```
#include "CifString.h"
CifString a, b, c;

a.Copy("ab");
b.Copy("cd");

if (a<=b) ...
```

PURPOSE

operator<=(const CifString &, const CifString &) Cheks if first string is less then or equal to second.

RECEIVES

operator<=(const CifString &, const CifString &)

`a`	reference to a CifString
`b`	reference to a CifString

RETURN VALUE

Returns 1 if the first CifStrings is less than or equal to second, otherwise 0.

REMARKS

None

3 ReVarCifArray Public Method and Member Descriptions

The following section describes the public interface to the ReVarCifArray class. Example code will be provided along with each method or member.

3.1 Constructors

This subsection contains the constructors for the ReVarCifArray class.

3.1.1 ReVarCifArray

NAME

ReVarCifArray

PROTOTYPE

```
#include "ReVarCifArray.h"

ReVarCifArray();
ReVarCifArray(unsigned n, int gb);
ReVarCifArray(const ReVarCifArray<TYPE$> &s);
ReVarCifArray(const TYPE *s,unsigned n );
```

EXAMPLE

```
#include "ReVarCifArray.h"
#include "CifString.h"

ReVarCifArray<CifString> * a1 = new ReVarCifArray<CifString>();
ReVarCifArray<CifString> * a2 = new ReVarCifArray<CifString>(8,4);
ReVarCifArray<CifString> * a3 = new ReVarCifArray<CifString>(a2);
```

PURPOSE

ReVarCifArray() Default constructor. Constructs a resizable array with initial length 1 and a reallocation increment of 1.
ReVarCifArray(unsigned, int) Constructs a resizable array with initial length n and a reallocation increment of gb.
ReVarCifArray(const ReVarCifArray<TYPE> &) Copy constructor.
*ReVarCifArray(const TYPE *, unsigned)* Constructor that creates resizable array and copies the data from the low-level C array. Defaults to not allowing any resizing.

RECEIVES

ReVarCifArray()

None

ReVarCifArray(unsigned, int)

`n`	allocated length
`gb`	reallocation inctement

ReVarCifArray(const ReVarCifArray<TYPE> &)

`s`	copied ReVarCifArray

ReVarCifArray(const TYPE *, unsigned)

`s`	low-level C array
`n`	allocated length

RETURN VALUE

None

REMARKS

None

3.2 Methods

This subsection contains the public methods for the ReVarCifArray class.

3.2.1 Add

NAME

Add

PROTOTYPE

```
#include "ReVarCifArray.h"

unsigned ReVarCifArray<TYPE>::Add(const TYPE &s);
```

EXAMPLE

```
#include "ReVarCifArray.h"
#include "CifString.h"

ReVarCifArray<CifString> a = new ReVarCifArray<CifString>();
a->Add("elem1");
```

PURPOSE

Add() Add an element s onto the end of the arary.

RECEIVES

None

RETURN VALUE

Returns 1 upon success, or if error.

REMARKS

None

3.2.2 InsertAt

NAME

InsertAt

PROTOTYPE

```
#include "ReVarCifArray.h"

unsigned ReVarCifArray<TYPE>::InsertAt(unsigned p, const TYPE *s, unsigned n);
```

EXAMPLE

```
#include "ReVarCifArray.h"
#include "CifString.h"

ReVarCifArray<CifString> a = new ReVarCifArray<CifString>();
a->InsertAt("elem1");
```

PURPOSE

InsertAt() Inserts the data pointed to by s into the array. Up to n elements are inserted, (truncating if necesary), starting at position p, (counted by zero).

RECEIVES

None

RETURN VALUE

Returns number of elements inserted, or 0 if error.

REMARKS

None

3.2.3 Length

NAME

Length

PROTOTYPE

```
#include "ReVarCifArray.h"

unsigned ReVarCifArray<TYPE>::Length();
```

EXAMPLE

```
#include "ReVarCifArray.h"
#include "CifString.h"

ReVarCifArray<CifString> a = new ReVarCifArray<CifString>();
a->Add("elem1");
a->Add("elem2");
cout<<a.Length()<<endl;
```

PURPOSE

Length() Returns logical length of array.

RECEIVES

None

RETURN VALUE

Returns logical length of array.

REMARKS

None

3.2.4 DimLength

NAME

DimLength

PROTOTYPE

```
#include "ReVarCifArray.h"

unsigned ReVarCifArray<TYPE>::DimLength();
```

EXAMPLE

```
#include "ReVarCifArray.h"
#include "CifString.h"

ReVarCifArray<CifString> a = new ReVarCifArray<CifString>();
a->Add("elem1");
a->Add("elem2");
cout<<a.DimLength()<<endl;
```

PURPOSE

DimLength() Returns allocated length of array.

RECEIVES

None

RETURN VALUE

Returns allocated length of array.

REMARKS

None

3.2.5 Data

NAME

Data

PROTOTYPE

```
#include "ReVarCifArray.h"

TYPE *ReVarCifArray<TYPE>::Data();
```

EXAMPLE

```
#include "ReVarCifArray.h"
#include "CifString.h"

ReVarCifArray<CifString> a = new ReVarCifArray<CifString>();
a->Add("elem1");
a->Add("elem2");

ReVarCifArray<CifString> b = new ReVarCifArray<CifString>(a->Data(),10);
```

PURPOSE

Data() Returns poiter to actual data.

RECEIVES

None

RETURN VALUE

Returns poiter to actual data.

REMARKS

None

3.2.6 Clear

NAME

Clear

PROTOTYPE

```
#include "ReVarCifArray.h"

void ReVarCifArray<TYPE>::Clear();
```

EXAMPLE

```
#include "ReVarCifArray.h"
#include "CifString.h"

ReVarCifArray<CifString> a = new ReVarCifArray<CifString>();
a->Add("elem1");
a->Add("elem2");
a->Clear()
```

PURPOSE

Clear() Inserts the data pointed to by s into the array. Up to n elements are inserted, (truncating if necesary), starting at position p, (counted by zero).

RECEIVES

None

RETURN VALUE

Returns number of elements inserted, or 0 if error.

REMARKS

None

3.3 Operators

This subsection contains the operators for the ReVarCifArray class.

3.3.1 operator=

NAME

operator=

PROTOTYPE

```
#include "ReVarCifArray.h"

ReVarCifArray<TYPE> &ReVarCifArray<TYPE>::operator=(const ReVarCifArray<TYPE> &s)
```

EXAMPLE

```
#include "ReVarCifArray.h"
#include "CifString.h"

ReVarCifArray<CifString> a;
ReVarCifArray<CifString> b;

a.Add("elem1");
a.Add("elem2");
b=a;
```

PURPOSE

&operator=(const ReVarCifArray &) Copies input array *s* into self.

RECEIVES

&operator=(const ReVarCifArray<TYPE> &)

s	reference to a ReVarCifArray

RETURN VALUE

Returns copied array.

REMARKS

None

3.3.2 operator[]

NAME

operator[]

PROTOTYPE

```
#include "ReVarCifArray.h"

TYPE &ReVarCifArray<TYPE>::operator[](const ReVarCifArray<TYPE> &s)
```

EXAMPLE

```
#include "ReVarCifArray.h"
#include "CifString.h"

ReVarCifArray<CifString> a;
ReVarCifArray<CifString> b;

a.Add("elem1");
a.Add("elem2");
cout<<a[0].Text()<<endl;
```

PURPOSE

&operator[](const ReVarCifArray &) Returns element from array which is on position i (start counting from 0).

RECEIVES

&operator[](const ReVarCifArray<TYPE> &)

`s`	reference to a ReVarCifArray

RETURN VALUE

Returns one element from array.

REMARKS

None

4 ReVarPCifArray Public Method and Member Descriptions

The following section describes the public interface to the ReVarPCifArray class. Example code will be provided along with each method or member.

4.1 Constructors

This subsection contains the constructors for the ReVarPCifArray class.

4.1.1 ReVarPCifArray

NAME

ReVarPCifArray

PROTOTYPE

```
#include "ReVarPCifArray.h"

ReVarPCifArray();
ReVarPCifArray(unsigned n, int gb);
ReVarPCifArray(const ReVarPCifArray<TYPE$> &s);
ReVarPCifArray(const TYPE *s,unsigned n );
```

EXAMPLE

```
#include "ReVarPCifArray.h"

ReVarPCifArray<int> * a1 = new ReVarPCifArray<int>();
ReVarPCifArray<int> * a2 = new ReVarPCifArray<int>(8,4);
ReVarPCifArray<int> * a3 = new ReVarPCifArray<int>(a2);
```

PURPOSE

ReVarPCifArray() Default constructor. Constructs a resizable array with initial length 1 and a reallocation increment of 1.
ReVarPCifArray(unsigned, int) Constructs a resizable array with initial length *n* and a reallocation increment of *gb*.
ReVarPCifArray(const ReVarPCifArray<TYPE> &) Copy constructor.
*ReVarPCifArray(const TYPE *, unsigned)* Constructor that creates resizable array and copies the data from the low-level C array. Defaults to not allowing any resizing.

RECEIVES

ReVarPCifArray()

None

ReVarPCifArray(unsigned, int)

`n`	allocated length
`gb`	reallocation inctement

ReVarPCifArray(const ReVarPCifArray<TYPE> &)

`s`	copied ReVarPCifArray

ReVarPCifArray(const TYPE *, unsigned)

`s`	low-level C array
`n`	allocated length

RETURN VALUE

None

REMARKS

None

4.2 Methods

This subsection contains the public methods for the ReVarPCifArray class.

4.2.1 Add

NAME

Add

PROTOTYPE

```
#include "ReVarPCifArray.h"

unsigned ReVarPCifArray<TYPE>::Add(const TYPE &s);
```

EXAMPLE

```
#include "ReVarPCifArray.h"
#include "CifString.h"

ReVarPCifArray<int> a = new ReVarPCifArray<int>();
a->Add(123);
```

PURPOSE

Add() Add an element s onto the end of the arary.

RECEIVES

None

RETURN VALUE

Returns 1 upon success, or if error.

REMARKS

None

4.2.2 InsertAt

NAME

InsertAt

PROTOTYPE

```
#include "ReVarPCifArray.h"

unsigned ReVarPCifArray<TYPE>::InsertAt(unsigned p, const TYPE *s, unsigned n);
```

EXAMPLE

```
#include "ReVarPCifArray.h"
#include "CifString.h"

ReVarPCifArray<int> a = new ReVarPCifArray<int>();
a->InsertAt(111);
```

PURPOSE

InsertAt() Inserts the data pointed to by s into the array. Up to n elements are inserted, (truncating if necesary), starting at position p, (counted by zero).

RECEIVES

None

RETURN VALUE

Returns number of elements inserted, or 0 if error.

REMARKS

None

4.2.3 Length

NAME

Length

PROTOTYPE

```
#include "ReVarPCifArray.h"

unsigned ReVarPCifArray<TYPE>::Length();
```

EXAMPLE

```
#include "ReVarPCifArray.h"
#include "CifString.h"

ReVarPCifArray<int> a = new ReVarPCifArray<int>();
a->Add(1);
a->Add(2);
cout<<a.Length()<<endl;
```

PURPOSE

Length() Returns logical length of array.

RECEIVES

None

RETURN VALUE

Returns logical length of array.

REMARKS

None

4.2.4 DimLength

NAME

DimLength

PROTOTYPE

```
#include "ReVarPCifArray.h"

unsigned ReVarPCifArray<TYPE>::DimLength();
```

EXAMPLE

```
#include "ReVarPCifArray.h"
#include "CifString.h"

ReVarPCifArray<int> a = new ReVarPCifArray<int>();
a->Add(1);
a->Add(2);
cout<<a.DimLength()<<endl;
```

PURPOSE

DimLength() Returns allocated length of array.

RECEIVES

None

RETURN VALUE

Returns allocated length of array.

REMARKS

None

4.2.5 Data

NAME

Data

PROTOTYPE

```
#include "ReVarPCifArray.h"

TYPE *ReVarPCifArray<TYPE>::Data();
```

EXAMPLE

```
#include "ReVarPCifArray.h"
#include "CifString.h"

ReVarPCifArray<int> a = new ReVarPCifArray<int>();
a->Add(1);
a->Add(2);
```

PURPOSE

Data() Returns poiter to actual data

RECEIVES

None

RETURN VALUE

Returns poiter to actual data.

REMARKS

None

4.2.6 Clear

NAME

Clear

PROTOTYPE

```
#include "ReVarPCifArray.h"

void ReVarPCifArray<TYPE>::Clear();
```

EXAMPLE

```
#include "ReVarPCifArray.h"
#include "CifString.h"

ReVarPCifArray<int> a = new ReVarPCifArray<int>();
a->Add(1);
a->Add(2);
a->Clear()
```

PURPOSE

Clear() Inserts the data pointed to by s into the array. Up to n elements are inserted, (truncating if necesary), starting at position p, (counted by zero).

RECEIVES

None

RETURN VALUE

Returns number of elements inserted, or 0 if error.

REMARKS

None

4.3 Operators

This subsection contains the operators for the ReVarPCifArray class.

4.3.1 operator=

NAME

operator=

PROTOTYPE

```
#include "ReVarPCifArray.h"

ReVarPCifArray<TYPE> &ReVarPCifArray<TYPE>::operator=(const ReVarPCifArray<TYPE> &s)
```

EXAMPLE

```
#include "ReVarPCifArray.h"
#include "CifString.h"

ReVarPCifArray<int> a;
ReVarPCifArray<int> b;

a.Add(1);
a.Add(2);
b=a;
```

PURPOSE

&operator=(const ReVarPCifArray &) Copies input array *s* into self.

RECEIVES

&operator=(const ReVarPCifArray<TYPE> &)

s	reference to a ReVarPCifArray

RETURN VALUE

Returns copied array.

REMARKS

None

4.3.2 operator[]

NAME

operator[]

PROTOTYPE

```
#include "ReVarPCifArray.h"

TYPE &ReVarPCifArray<TYPE>::operator[](const ReVarPCifArray<TYPE> &s)
```

EXAMPLE

```
#include "ReVarPCifArray.h"
#include "CifString.h"

ReVarPCifArray<int> a;
ReVarPCifArray<int> b;

a.Add(1);
a.Add();
cout<<a[0]<<endl;
```

PURPOSE

&operator[](const ReVarPCifArray &) Returns element from array which is on position i (start counting from 0).

RECEIVES

&operator[](const ReVarPCifArray<TYPE> &)

s	reference to a ReVarPCifArray

RETURN VALUE

Returns one element from array.

REMARKS

None