Collection classes

Symbian OS does not support the Standard Template Library (STL) for a number of reasons—primarily because of STL's large footprint. However, Symbian OS does provide a number of templated collection classes, so that developers do not need to write their own arrays, linked lists and so on.

There are many classes available. CArray and RArray are essentially dynamic arrays. Then we have lists, TSglQue single linked, TDblQue double linked, CCirBuf a circular buffer. And a balanced tree implementation, TBtree.

What types of collection are available?

RArray classes: - use flat storage only, - support sorting and searching using a comparator function and can ensure uniqueness, - provide specializations for common types, for instance, integers.

CArray classes: - provide a choice of either flat or segmented storage, - support sorting and searching using a key specification and can ensure uniqueness, - provide several variants, for instance, fixed or variable size elements, packed data - are generally slower and can be less easy to use than the RArray classes.

Descriptor arrays: - provide a choice of either flat or segmented storage - can contain 8-bit or 16-bit descriptors - support sorting and searching and can ensure uniqueness - provide variants that can store any type of descriptor or can hold pointers to the data.

Linked lists: - support iterators for scanning through the list - are available as singly and doubly linked lists; the link object must be a member of the linked class.

TFixedArray: - used when the number of elements is fixed and known at compile time - should be used instead of traditional C++ arrays because they provide bounds checking.

TArray: - used for representing any array data in a generic way.

RArray and RPointerArray Types

Since RArrays are easier to use than CArrays, we will discuss them first. An RArray is a simple array of fixed-length objects, while an RPointerArray is an array of pointers to objects. It is worth noting from the outset that RArrays impose a limitation of 640 bytes on the size of their elements. Usually this is not an issue, since most objects are likely to be smaller than this limit, and RPointerArray can be used to point to arbitrarily large objects anyway. Generally, therefore, owing to their greater efficiency, flexibility and ease of use, RArrays are recommended over CArray types.

RArrays and RPointerArrays tend to be constructed on the stack or simply nested directly into other heap-based objects.They are resource based classes hence you have to close before leaving. Using the cleanup stack remember the CleanupClosePushL() instead of PushL().

Using RArray

```
struct Networkdata
{
 TBuf<10> LAC;
   TBuf<15> CELLID;
};
```

// RArray of type Networkdata.

```
RArray<data> DesArray;
```

// Filling data into structure

```
struct Networkdata db;
db.LAC.Copy(_L("1121") );
db.CELLID.Copy(_L("152"));
```

// Appending Networkdata in DesArray.

```
DesArray.Append( db );
```

Once an array is finished with, it must be reset before being allowed to go out of scope (or before the object in which it is nested is deleted). Both RArray and RPointerArray implement a Reset() method that frees all of the memory allocated for storing the elements.

In case of RPointerArray you could use method ResetAndDestroy() to free memory, allocated by the elements.

```
DesArray.Reset();
```


(C) Copyright Nokia 2013. All rights reserved.

CArray Types

All of the CArray types use buffers to store their data. Buffers (derived from CBufBase) provide access to regions of memory and are in some ways similar to descriptors in that respect. However, while descriptors are intended to store data objects whose maximum size is not expected to alter much, buffers are expected to grow and shrink dynamically during the lifetime of the program. There are two buffer types 1) **flat** and 2) **segmented** and these two types give rise to two basic subtypes of CArray.

Flat buffers store their entire data within a single heap cell. Once full, any subsequent append operation requires a new heap cell to be allocated that is large enough to contain the original and new data. Once the allocation has completed, all of the old data is copied to the new cell, and the old cell is released back to the global heap.

Segmented buffers store their data in a doubly-linked list of smaller segments, each of which is a separate heap cell of fixed size. Once all of the segments have been allocated, a new segment will be allocated and added into the list, with the old data remaining in place, and without the need for copying. While this can reduce the memory thrashing associated with frequent reallocation, accessing data is less efficient than flat buffers, since the list of segments must be traversed, plus more memory is consumed by the need to store linked list pointers. It can also lead to memory fragmentation.

Why Use RArray Instead of CArrayX?

The original CArrayX classes use CBufBase, which allows a varied dynamic memory layout for the array using either flat or segmented array buffers. However, CBufBase works with byte buffers and requires a TPtr8 object to be **constructed** for **every array access**. This results in a **performance overhead**, even for a simple flat array containing fixedlength elements. Furthermore, for every method which accesses the array, there are a **minimum of two assertions to check** the parameters, even in release builds

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.