



Contents

- [1 Memory Management](#)
- [2 Memory Management Modules](#)
- [3 Stack and Heap](#)
- [4 Leaves](#)
- [5 Cleanup Stack](#)
- [6 Two Phase Construction](#)
- [7 Memory Leaks](#)

Memory Management

An essential characteristic of Symbian OS is that it is designed for devices where the amount of memory and resources available is limited. So, it needs a very efficient memory management. Run-time errors may occur in any application due to lack of resources: for example, a machine runs out of memory or fails to access a hardware resource. These errors are known as exceptions and it is impossible to prevent them occurring by modifying the program. Therefore, programs should be able to recover from exceptions when they occur.

So, key requirements for Symbian OS for good memory management are:

- Program efficiently so that your program does not use RAM unnecessarily
- Release resources as soon as possible (remember resources are limited)
- Cope with out-of-memory errors, in fact you need to do that in every operation where

memory is allocated

- When out-of-memory situation arises in the middle of an operation, go back to an acceptable

and stable state and make sure that you clean up all resources allocated during that operation.

Memory Management Modules

- Stack and Heap
- Leaves
- Cleanup Stack
- Two Phase Construction
- Memory Leaks
- Panics
- Debugging

Stack and Heap

On a Stack:

- objects are deleted automatically
- default size is 8Kb

On a Heap:

- objects must be deleted by programmer using delete
- size dependent on device but generally > 0.5Mb

Example:

```
TInt i = 0;  
CMyObj* obj = new (ELeave) CMyObj;
```

Here, i is stored on a stack and obj is pointed to heap memory.

When objects are declared on the Stack, they are deleted automatically when the object goes out of scope. The default stack size is 8Kb, but it is possible to increase this by use of the `epocstacksize` statement in the project (.mmp file). Use of the project file will be covered in a subsequent module. (Note: changing the stack size will have no effect when using the Emulator).

By contrast, when an object is allocated dynamically on the Heap, it needs to be specifically deleted by the programmer using the `delete` keyword. If it isn't a memory leak occurs. The size of heap will vary according to the memory available on a given device, but should exceed 0.5Mb.

Leaves

Leaves

- Leaves are used instead of C++ exceptions
- When there is a resource failure, the code ?Leaves?
- This propagates up the call stack until handled
- Leaves are conceptually equivalent to throwing an exception
- The new operator has been overloaded to Leave if insufficient memory is available
- use `new (ELeave)`
- memory is released as normal using `delete`
- Functions that can leave should end in ?L?

Leave Examples

- Dynamic memory allocation:

```
return new (ELeave) TInt8[100];
```

- Raising a Leave

Memory_Management

```
User::Leave(KErrNotFound); // from e32std.h
```

- Leaving if no Memory

```
User::LeaveNoMemory();
```

- Leaving if NULL

```
void CMyClass::SetCallbackL(MNotify* aNotify)
{
User::LeaveIfNull(aNotify);
...
}
```

- Leaving if an error occurs

```
RFs fileServer; // handle to file server
TInt error = fileServer.Connect();
User::LeaveIfError(error);
```

Cleanup Stack

Cleanup Stack

Cleanup stack support for non-CBase classes

Using Cleanup Stack

- To put items on the cleanup stack use

```
CleanupStack::PushL(ptr) for pointers ? memory will be deleted in
the event of a Leave
```

```
CleanupClosePushL(handle) for handles ? handle will be closed in
the event of a Leave
```

- To remove items from the cleanup stack use

```
CleanupStack::Pop(pointer) to remove the top item
```

```
CleanupStack::PopAndDestroy(pointer) to remove and
delete/close the item
```

- Push an object to the cleanup stack if

? that object is referred to by a local pointer only AND

? a function that can leave is called during the lifetime of that object.

- Do not put member variables of a class on the cleanup stack.

Example:

Here, if ConstructL() Leaves, self will be automatically deleted:

```
CMyClass* CMyClass::NewL(TInt aBufSize)
{
    CMyClass* self = new (ELeave) CMyClass;
    CleanupStack::PushL(self);
    self->ConstructL(aBufSize);
    CleanupStack::Pop(self);
    return self;
}
```

- Items should be popped from the cleanup stack when a leave can no longer occur
- If a function needs to leave an object on the cleanup stack after exit, its name must have a ?C? at the end.

Two Phase Construction

Two-phase construction

NewL() and NewLC()

These static functions perform both construction phases in one go. Providing NewL() or NewLC() functions as part of a class makes it easier to use that class. NewLC() would typically be used where one function contains a series of automatic variables (which point to heap memory), to save pushing each one onto the cleanup stack.

Memory Leaks

Memory leaks

Memory Leaks Debugging

How to log heap allocations, Symbian 9

--