# Two-phase construction

## Brief description

In Symbian C++ programming,the Two-phase construction idiom ensures that the objects being constructed and initialised do not leave before they are stored in the cleanup stack.

## Context and Problem

In programming for Symbian with C++, the pointers to heap-based objects are stored in the cleanup stack. This allows automatic cleaning of the object in case of a leave situation. A newl created object is pushed into the cleanup stack immediately after it is constructed and initialised by a C++ constructor. However, according to the C++ conventions, during the time after memory is allocated for the object and before the object's C++ constructor completes, the object cannot be stored in the cleanup stack yet, and therefore, leaving during that time should not be allowed.

## Solution, its Consequences and Liabilities

In order to avoid a leaving situation until an object is pushed into the cleanup stack, leaving of this object's C++ constructor should be prohibited. This is achieved by implementing a two-phase constructor.

According to the two-phase constructor idiom, all initialisation code that might leave, as well as any calls to the functions that might leave, is located in a separate function ConstructL(), referred to as a second-phase constructor. (For example, if base classes as well have ConstructL() methods, they should be called (explicitly) in the second-phase constructor.) This second-phase constructor is called only after the object being initialised is pushed into the cleanup stack.

For frequently used classes, both calls to the C++ constructor and to the second-phase constructor are usually encapsulated in a static special-purpose NewL() function. When the object is to be created, the NewL() function is supposed to be called instead of the C++ constructor and ConstructorL(). This factory function allocates memory for the object, pushes it into the cleanup stack, calls the second-phase constructor, and finally pops the object out of the stack. If it is envisioned that the newly created object will be used as an automatic variable (i.e. it should be kept in the cleanup stack throughout its lifetime), additional NewLC() function is provided, which is the same as the NewL() with the exception that it does not pop the object out of the stack. An example of the ConstructL(), NewL(), and NewLC() functions is provided in the listing below.

NewL() and NewLC() are the static method and can be called by the class with out creating an object. NewL() and NewLC() are kept into the public access specifier of the class. so that we can call them directly. While the default constructor (in which the initialisation code is non-leaving) and the ConstructL() method(in which the initialization code may leave) are put into the private area of the class and only the methods in public area (generally NewL() and NewLC()) can make a call to them and access them.

```
// Phase #1
CMyClass::CMyClass()
 {
 }

// Phase #2
void CMyClass::ConstructL()
 {
 // Member data initialization.
 }

// Put both phases together in one function...
CMyClass * CMyClass::NewL()
 {
 CMyClass * self = new (ELeave) CMyClass();
 CleanupStack::PushL(self);
 self->ConstructL();
 CleanupStack::Pop(self);
```

```
 return self;
 }

CMyClass * CMyClass::NewLC()
 {
 CMyClass * self = new (ELeave) CMyClass();
 CleanupStack::PushL(self);
 self->ConstructL();
 return self;
 }
```

## Future

Strictly speaking there is no need for two-phase construction and cleanup stack after Symbian moved closer to standard C++ and reimplemented leaves in terms of exceptions. Exception/Leave -safe code can be written in standard C++ easily and more conveniently, but Symbian still retains the old practices for legacy reasons. Sometime in the (hopefully) near future we could say goodbye to these idioms and use purely standard C++ to write Symbian programs.