

Mobile Systems Software: Lecture 4

- Object creation and memory Management.
- Exception handling: TRAP, Leave and CleanupStack.
- Two Phase Construction.
- UI Application Framework.
- SDK's and IDE's.
- Compilation and Installation on Device.

HelloTextsimple.cpp

```
// helloworld.cpp
//
// KåreJ. Kristoffersen
//
#include <e32base.h>
#include <e32cons.h>

LOCAL_D CConsoleBase* console;

GLDEF_C TInt E32Main()
{
    console=CConsole::NewL(_L("HelloText"),TSize(KConsFullScreen,KConsFullScreen));
    console->Printf(_L("Hello world!\n"));
    console->Printf(_L("[ press any key ]"));
    console->Getch(); // get and ignore character
    return 0;
}
```

Hellotextsimple.mmp

```
// hellotext.mmp
```

```
//
```

```
// Copyright (c) 2000 Symbian Ltd. All rights reserved.
```

```
//
```

```
TARGET hellotextsimple.exe
```

```
TARGETTYPE exe
```

```
UID 0
```

```
SOURCE hellotextsimple.cpp
```

```
USERINCLUDE .
```

```
SYSTEMINCLUDE \epoc32\include
```

```
LIBRARY euser.lib
```

Status:

- So far programming for Symbian OS is very simple 😊
- However, it gets more complicated!

Handheld devices have limits!

- Most of the system is designed to run for years: kernel, servers, applications.
- So we don't have opportunities to reclaim leaked memory, unlike platforms that are often shut down.
- Small memory leaks accumulate over time.
- So we just need to write perfect leak free code!

How? By Managing Memory

By keeping track of all objects allocated

By making sure:

- All allocated heap memory is pointed to at by at least one pointer at all times (including if we run out of memory when constructing compound classes)
- All heap memory is freed asap after use.

We have tools to help us doing this ☺

hellotext – E32Main()

```
GLDEF_C TInt E32Main()
{
    __UHEAP_MARK;
    CTrapCleanup* cleanupStack=CTrapCleanup::New();
    TRAPD(error,consoleMainL());
    __ASSERT_ALWAYS(!error,User::Panic(_L("PEP"),error));
    delete cleanupStack;
    __UHEAP_MARKEND;
    return 0;
}
```

hellotext - consoleMainL()

```
void consoleMainL()
{
    // get a console
    console=Console::NewL(_L("Hello Text"),TSize(KConsFullScreen,KConsFullScreen));
    CleanupStack::PushL(console);
    // call function
    mainL();
    // wait for key
    console->Printf(_L("[ press any key ]"));
    console->Getch(); // get and ignore character
    // finished with console
    CleanupStack::PopAndDestroy(); // console
}
```

hellotext - mainL()

```
#include <e32base.h>
#include <e32cons.h>

LOCAL_D CConsoleBase* console;

// real main function

void mainL()
{
    console->Printf(_L("Hello world!\n"));
}
```

TRAP Harness Macros

```
TRAPD(error,consoleMainL());
```

```
if(error){ .... }
```

- If exception occurs leave, returning to last harness
- Two Macros: TRAP and TRAPD
 - TRAPD declares error as Tint
- Functions that leave end in a L
- Design where TRAPs are needed from start.
- Expensive!

Handling Exceptions

Handle resource exceptions &
Maintain pointers to heap memory,
by using 3 strategies / disciplines:

- TRAP harness & Leave.
- Cleanup Stack.
- Two Phase Construction.

Leaving Functions

Examples: CreateL(), AllocL(), NewL(), RunL().

A Leaving function is one that may need to allocate memory, open a file etc. Generally do some operation that might fail because there are insufficient resources. When you call a leaving function you must always consider what happens both when it succeeds and when it leaves. Symbian OS's cleanup framework is designed to allow you to do this.

new(Eleave)

```
CObject* thing=new(Eleave) CObject();
```

- Use overloaded C++ new operator
- Can be called from any class
- Will call User::Leave() if insufficient memory to allocate object.

Implemented as:

```
CObject* thing = new CObject;
```

```
if(!thing)
```

```
    User::Leave(KErrNoMemory);
```

new (Eleave)

Tint err;

~~TRAP(err, CreateObejectL());~~

void CreateObjectL()

{

CObject* obj=new(Eleave) CObject;

Cleanup Stack

1. Holds references to objects that (otherwise) are only pointed to by a stack pointer declared within a function that can leave.
2. If a Leave occurs, the TRAP harness will call PopAndDestroy() on each object on the stack
 - This pops each item and calls objects destructor.
 - PopAndDestroy() is atomic.

Cleanup Stack

```
SomeObj* x=new SomeObj;  
x->UseL(); ← May fail!  
delete x;
```

```
SomeObj* x=new(Eleave)SomeObj;  
CleanupStack::PushL(x);  
x->UseL();  
CleanupStack::PopAndDestroy(x);
```

In a nutshell

All three strategies are present in this function called NewL():

```
CClass* CClass::NewL(Tint a Int, CBase& aObj)
{
    CClass* self=new(Eleave) CClass(aInt);
    CleanupStack::PushL(self);
    self->ConstructL(aObj);
    CleanupStack::pop(self);
    return self;
}
```

Leaving on Exception

```
CClass* CClass::NewL(Tint a Int, CBase& aObj)
{
    CClass* self=new(Eleave) CClass(aInt);
    CleanupStack::PushL(self);
    self->ConstructL(aObj);
    CleanupStack::pop(self);
    return self;
}
```

Cleanup Stack

```
CClass* CClass::NewL(Tint a Int, CBase& aObj)
{
    CClass* self=new(Eleave) CClass(aInt);
    CleanupStack::PushL(self);
    self->ConstructL(aObj);
    CleanupStack::pop(self);
    return self;
}
```

Cleanup Stack

```
TRAPD(err, CreateObjectsL());
```

```
...
```

```
void CreateObjectsL()
```

```
{
```

```
    CObject* obj1=new(Eleave) CObject;
```

```
    CleanupStack::PushL(obj1);
```

```
    CObject* obj2=new(Eleave) CObject;
```

If second allocatoin fails we still have the pointer obj1. The Trap harness sees this and deletes the object from the heap.

Declaring Cleanup Stack

- Create your own for console programs with

```
GLDEF_C TInt E32Main()
```

```
{
```

```
    __UHEAP_MARK;
```

```
    CTrapCleanup* cleanupStack=CTrapCleanup::New();
```

```
    TRAPD(error,consoleMainL());
```

GUI Framework creates its own cleanup stack!!!

Two-phase construction

```
CClass* CClass::NewL(Tint a Int, CBase& aObj)
{
    CClass* self=new(Eleave) CClass(aInt);
    CleanupStack::PushL(self);
    self->ConstructL(aObj);
    CleanupStack::pop(self);
    return self;
}
```

Two-phase construction

- Construct compound classes in two phases
 1. The normal constructor `CClass::CClass()` for all safe construction.
 2. The second phase constructor `ConstructL` to safely construct things that may `Leave`.
- Factory function **newL** ties the two phases together
- Self contained classes only require single phase construction.

Two-phase construction

```
class CClass : public CBase
{
public:
    static CClass* NewL(TInt a Int, CBase& aObj);
    ~CClass();
private; //or protected
    CClass(TInt aValue);
    void ConstructL(CBase& aObj);
private:
    TInt iInt;
    CSimple* iSimple;
    CCompound iCompound;
};
```

ConstructL()

```
void CClass::ConstructL()(CBase& aObj)
{
    iSimple=new(Eleave) CSimple;
    iCompound=CCompound::NewL(Kval,AObj);
}
CClass::~~CClass()
{
    delete iSimple;
    delete iCompound;
};
```

TRAP & Leave Tips

- 95% of the time let error go all the way up to the handler.
- If don't deal with all possible errors after a TRAP propagate errors up with `User::Leave(err)`
- As TRAPs are expensive, instead of having several in sequence, end the function name with `L` and let its caller deal with the errors!
- Make sure to use `new(Eleave)` not just `new`.

Three Strategy Summary

- Use TRAP harness to encapsulate Exceptions that Leave
- Use CleanupStack to save local heap pointers.
- Use two-phase construction for compound objects: safe construction; exception raising.

NewL and NewLC

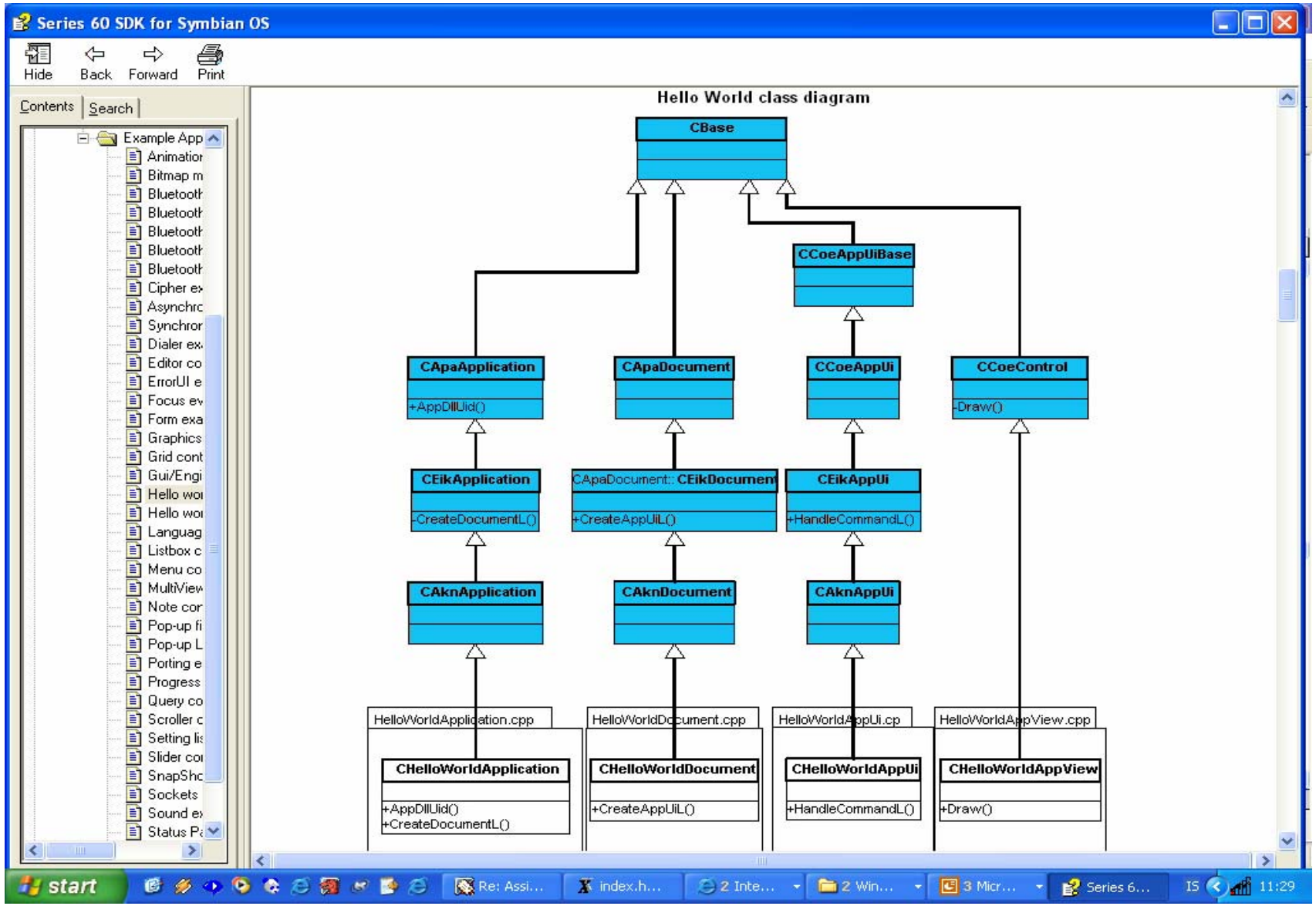
```
CZ* CZ::NewL()
{
    CZ* self=NewLC();
    CleanupStack.pop();
    return self;
}
```

```
CZ* CZ::NewLC()
{
    CZ* self=new(Eleave)CZ;
    CleanupStack::PushL(self);
    self.ConstructL();
    return self();
}
```

**Difference:
NewLC lets
pointer stay
on Cleanup
Stack!**

Application Framework

1. Fundamental Symbian UI program structure.
2. What happens when a program is executed?



Compilation and Installation

Directory Overview (SDKs)

C:\Symbian\7.0s\Series60_v20_CW\ or
C:\Symbian\UIQ_21\

Both contains the epoc32 directory containing compilers and subdirectories for the various target platforms (including the emulator).

Today we will use the SDK for UIQ.

EPOCROOT

Set the Environment Variable EPOCROOT

Set epocroot=\Symbian\UIQ_21\

The epocroot must be set to the directory in the SDK which contains the epoc32 directory.

(this is only important when compiling on the command prompt!)

Type 'cd %EPOCROOT%' to reach the Symbian root directory.

Target Binaries

UDEB for debugging

UREL for release

WINSCW

ARM4

ARMI

THUMB

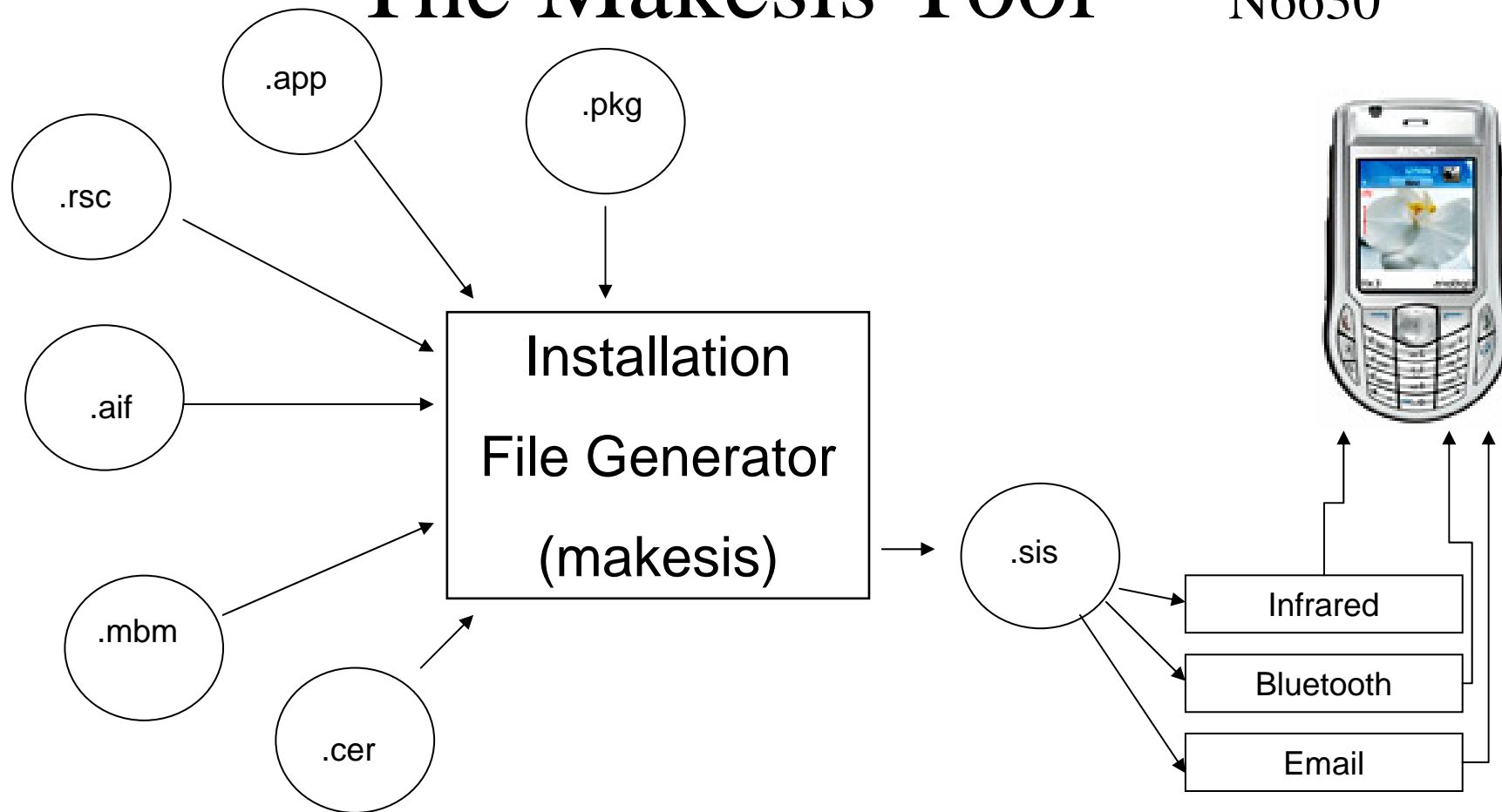
VC6

WINC

WINS

The Makesis Tool

N6630



Folder structure:

Group:

- Project files: .mmp, .pkg, .rss,...

inc:

- Header files: .h, .hrh

sis:

- Installation files: .pkg, .sis

src:

- Source files: .cpp

helloworldplus.mmp

```
TARGET      HelloWorldPlus.app
TARGETTYPE  app
```

```
// change the second number here to change the UID for this application
```

```
UID         0x100039CE 0x10005b95
```

```
TARGETPATH  \system\apps\helloworldplus
```

```
SOURCEPATH  ..\src
```

```
SOURCE      HelloWorldPlus.cpp
```

```
SOURCE      HelloWorldPlusApplication.cpp
```

```
SOURCE      HelloWorldPlusAppView.cpp
```

```
SOURCE      HelloWorldPlusAppUi.cpp
```

```
SOURCE      HelloWorldPlusDocument.cpp
```

```
SOURCE      Car.cpp
```

```
SOURCEPATH  ..\group
```

```
RESOURCE    HelloWorldPlus.rss
```

```
RESOURCE    HelloWorldPlus_caption.rss
```

```
USERINCLUDE  ..\inc
```

```
SYSTEMINCLUDE \epoc32\include
```

```
LIBRARY      euser.lib
```

```
LIBRARY      apparc.lib
```

```
LIBRARY      cone.lib
```

```
LIBRARY      eikcore.lib
```

```
LIBRARY      avkon.lib
```

```
LIBRARY      commonengine.lib
```

Number must match the one in
helloworldplusapplication.cpp and
helloworldplus.pkg

Number common for *all* GUI
applications.

helloworldplusapplication.cpp

```
/* Copyright (c) 2002, Nokia. All rights reserved */


#include "HelloWorldPlusDocument.h"
#include "HelloWorldPlusApplication.h"

// local constants
static const TUid KUidHelloWorldPlusApp = {0x10005b95};

CApaDocument* CHelloWorldPlusApplication::CreateDocumentL()
{
    // Create an HelloWorldPlus document, and return a pointer to it
    CApaDocument* document = CHelloWorldPlusDocument::NewL(*this);
    return document;
}

TUid CHelloWorldPlusApplication::AppDllUid() const
{
    // Return the UID for the HelloWorldPlus application
    return KUidHelloWorldPlusApp;
}
```

IMPORTANT:
Must be the
same as in
.mmp file and
.pkg file.



Compiling for the phone:

The emulator version **cannot** run on
the phone !!!

```
bldmake bldfiles
```

```
abld build armi urel
```

... Or, do it in the CodeWarrior IDE!

helloworldplus.pkg

```
; targetapp (HelloWorldPlus)
; target directory \Symbian\Series60_1_2_B\epoc32\release\winsb\udeb\z\system\apps\helloworldplus\
; Header
#{ "HelloWorldPlus" },(0x10005b95),1,0,0,TYPE=SISAPP

;Supports Series 60 v 2.0
;This line indicates that this installation is for the Series 60 platform v0.9
;This line must appear _exactly_ as shown below in the sis file
;If this line is missing or incorrect, the sis file will not be able
;to be installed on Series 60 v0.9 platforms
(0x101F7964), 0, 0, 0, { "Series60ProductID" }

; mmpfile: HelloWorldPlus.mmp
; targets
; target: HelloWorldPlus.app
"\Symbian\7.0s\Series60_v21_CW\epoc32\release\armi\urel\HELLOWORLDPLUS.APP"-
"!:\system\apps\HelloWorldPlus\HelloWorldPlus.app"
; resources
; resource: ..\group\HelloWorldPlus.rss
;
"\Symbian\7.0s\Series60_v21_CW\epoc32\data\z\system\apps\HELLOWORLDPLUS\HELLOWORLDPLUS.RSC"-
"!:\system\apps\HelloWorldPlus\HelloWorldPlus.rsc"
; resource: ..\group\HelloWorldPlus_caption.rss
"\Symbian\7.0s\Series60_v21_CW\epoc32\data\z\system\apps\helloworldplus\HelloWorldPlus_caption.rsc"-
"!:\system\apps\HelloWorldPlus\HelloWorldPlus_caption.rsc"
; bitmaps
; languages
; aif files
19-09-2005
; end of pkg file
```

Kåre Kristoffersen, Innovation, ITU.

CodeWarrior

In the **File Menu** choose "**Open Project from mmp file**".

For **Vendor** choose **Symbian**

For **SDK** choose **UIQ_21**

Click **Next**

Browse to: C:\Symbian\UIQ_21\UIQExamples\papers\pep\hellotext\hellotext.mmp

Click **Finish**

In the **Project Menu** choose **Run**.

Now the application is compiled, linked and launched.