Symbian OS

Types and Declarations

v2.0 – 03 December 2009

Disclaimer

- **These slides are provided free of charge at http://www.symbianresources.com and are used during Symbian OS courses at the University of Applied Sciences in Hagenberg, Austria (http://www.fh-hagenberg.at/)**
- **Respecting the copyright laws, you are allowed to use them:**
  - **for your own, personal, non-commercial use**
  - **in the academic environment**
- **In all other cases (e.g. for commercial training), please contact andreas.jakl@fh-hagenberg.at**
- The correctness of the contents of these materials cannot be guaranteed. Andreas Jakl is not liable for incorrect information or damage that may arise from using the materials.
- Parts of these materials are based on information from Symbian Press-books published by John Wiley & Sons, Ltd. This document contains copyright materials which are proprietary to Symbian, UIQ, Nokia and SonyEricsson. “S60™” is a trademark of Nokia. “UIQ™” is a trademark of UIQ Technology. Pictures of mobile phones or applications are copyright their respective manufacturers / developers. “Symbian ™”, “Symbian OS ™” and all other Symbian-based marks and logos are trademarks of Symbian Software Limited and are used under license.

Contents

- Motivation
- Fundamental types
- Variable naming conventions
- Class naming conventions (T, C, R, M, static)

Motivation

- Most C++ frameworks define **own fundamental types**
  - For **compiler independence** using `typedef`'s
  - OpenGL: `GLint`, Symbian OS: `TInt`, …
- Symbian OS: **Class Types**
  - Each has **different characteristics**
  - Describes properties, behavior, creation on stack and/or heap, cleanup
  - Simplifies correct use
  - `TPoint`, `CFbsBitmap`, `RFile`, `MCallbackInterface`

Naming Conventions

- **Some more thoughts...**
  - Make code more **self-explaining** ("... and where's this variable coming from?")
  - Get rid of references using "`this->`" for instance variables
  - Important for **clean-up** (Instance variables stored on the heap have to be deleted in the destructor of your class)
  - **Common syntax** no matter who developed the code

Symbian OS

Fundamental Types

Fundamental Types

- Defined in `<SDK-path>\epoc32\include\e32def.h`

Standard ANSI	Symbian OS	Description
int	**TInt**	Signed (32-bit) Integer
unsigend int	**TUint**	Unsigned (32-bit) Integer
float	**TReal32**	Single-precision IEEE 754 floating-point
double	**TReal, TReal64**	Double-precision IEEE 754 floating-point
long long	**TInt64**	Uses native 64-bit support
bool	**TBool (ETrue, False)**	Equates to int due to early compilers
void*	**TAny***	“Pointer to anything”

also available: TText[8|16], TInt[8|16|32], TUint[8|16|32], TUint64

Examples

Example: TInt, TReal

```
TInt x = 5;
TReal y = (TReal)x + 0.5;
```

Example: TBool

```
TBool b = ETrue;
// Bad style: ETrue = 1, but any non-zero number should be interpreted as true!
if (b == ETrue) { ... }
// Good style:
if (b) { ... }
```

Example: void / TAny*

```
// Symbian OS uses ‘void’ for ‘nothing’ and ‘TAny*’ for ‘pointer to anyting’
void Foo();      // Returns no result
TAny* p;         // Pointer to anything
```

Symbian OS

Naming Conventions

General Overview

- **Classes**
  - Name should be a **noun**, represents an **object**
  - Name uses an **initial letter** to indicate the basic **properties**
- **Data**
  - Name should be a **noun**, represents an **object**
  - Name uses an **initial letter** to indicate their **purpose**
- **Functions**
  - Name usually a **verb**, represents an **action**
  - **Initial letter** should be **uppercase** (`Draw()`, `Intersects()`)

Variable Naming Conventions

Prefix	Category	Examples	Description
E	Enumerated constant	`EMonday`, `ESolidBrush`	Values in an enumeration – which itself should have T prefix: `EMonday` is a member of `TDayOfWeek`
K	Constant	`KMaxFileName`, `KRgbWhite`	Constants from `#define` or const `TInt`.
i	Member variable	`iDevice`, `iX`	Any non-static member variable. `i` refers to 'instance'
a	Argument	`aDevice`, `aX`	Function argument. Stands for 'argument' → `aOrigin`, not `anOrigin`!
	Automatic variable	`device`, `x`	Automatic: variable that is created automatically when required and destroyed when out of scope

Example – Variables

```
enum TStuffState                    // Declares an enumeration, prefix T
   {
   EInitialized = 0,                // Individual elements with prefix E
   EError
   };
const TInt KMaxLength = 50;         // Constant with prefix K

class TStuff                        // T type class
   {
public:
   void DoStuff(TInt aLength);      // Function argument with prefix a
private:
   TInt iLength;                    // Member (instance) variable with prefix i
   TStuffState iState;
   };

void TStuff::DoStuff(TInt aLength)
   {
   if (aLength > KMaxLength)
      iState = EError;
   else
      iLength = aLength;            // Note: no ambiguities!
   }
```

T Classes

- Remember the fundamental types (`TInt`, ...)?
- **T classes** → similar behaviour
  - **Do not have a destructor**
    - Therefore, no member data that has a destructor
  - **Contain all data internally**
    - No pointers, references or handles ("has-a" relation)
- Can be created on the stack and the heap
- Also usually used instead of a traditional C `struct`

T Classes – Example

TPoint definition from e32cmn.h

```
class TPoint
        {
public:
        enum TUninitialized { EUninitialized };
        /** Constructs default point, initialising its iX and iY members to zero. */
        TPoint(TUninitialized) {}
        inline TPoint();
        inline TPoint(TInt aX, TInt aY);
        IMPORT_C TBool operator==(const TPoint& aPoint) const;
        // [...]
        IMPORT_C TPoint operator-() const;
        IMPORT_C void SetXY(TInt aX, TInt aY);
        IMPORT_C TSize AsSize() const;
public:
        /** The x coordinate. */
        TInt iX;
        /** The y coordinate. */
        TInt iY;
        };
```

Note the naming conventions for an enum

Usage example

```
void CMyControl::Draw(const TRect &aRect) const
    {
    CWindowGc& gc = SystemGc ();
    gc.DrawLine (TPoint (0, 0), iLastPoint);
    }
```

C Classes

- **Properties of C classes** (‘C’ for ‘cleanup’)
  - Usually created on the **heap**
    (they’re often too large for stack themselves)
  - Usually **own pointers** to large objects, resources, ...
- **Derive from CBase** (directly or indirectly)
  - Safe construction / Destruction
  - Zero initialization

C Classes – Characteristics

- **Safe construction / destruction**
  - CBase defines **virtual destructor**
    - C++ therefore calls destructor in correct order
    - Also required for the cleanup stack (later module)
  - Declares a private copy constructor and assignment operator
    - Prevents errors
    - If required: derived class has to declare it
- **Zero initialization**
  - CBase overloads new-operator
  - Zero-initializes all member data

C Classes – Example

```
class CSprite : public CBase
   {
   public:           // Constructors and destructors
      static CSprite* NewL( TInt aXVelocity, /* ... */, CFbsBitmap* aImage, CFbsBitmap* aMask);
      static CSprite* NewLC( TInt aXVelocity, /* ... */, CFbsBitmap* aImage, CFbsBitmap* aMask);
      virtual ~CSprite();

   public:           // New functions (omitted for clarity)

   private:          // Constructors
      CSprite( TInt aXVelocity, /* ... */, CFbsBitmap* aImage, CFbsBitmap* aMask);
      void ConstructL();

   private:          // Data
      TPoint iPosition;
      const CFbsBitmap * const iImage;
      const CFbsBitmap * const iMask;
};
```

R Classes

- Own an **external resource handle**, e.g.:
  - Server session (`RFs` – file server session)
  - Memory (`RArray`, `RBuf`)
- **Initialization:**
  - **Open()**, Create() or Initialize()
  - **Close()** or Reset() **instead of destructor**
  - No automated Close() through the destructor!

R Classes – Example

- Note: leaves and the cleanup stack will be covered in the following modules

```
void CMyClass::SendCachedDataL()
   {
   RSocketServ socketServer;

   // Connect to the SocketServer
   User::LeaveIfError( socketServer.Connect() );
   // Make sure Close() is called at the end
   CleanupClosePushL( socketServer );

   // …

   CleanupStack::PopAndDestroy();  // Calls Close() on the socket server object
   }
```

M Classes

- **M class**
  - M is for “mixin”
  - Used for **defining interface classes**
  - Declares pure **virtual functions**
  - Should not contain members or constructors
- **Implementing class**
  - Usually derives from `CBase` and the interface
  - Only form of multiple inheritance encouraged on Symbian OS

M Classes – Example

```
// Interface definition
class MMdaAudioPlayerCallback
      {
public:
      virtual void MapcInitComplete(TInt aError, const TTimeIntervalMicroSeconds& aDuration) = 0;
      virtual void MapcPlayComplete(TInt aError) = 0;
      };

// Implementing class
class CSoundPlayer : public CBase, public MMdaAudioPlayerCallback
      {
      // [...]
protected:  // Functions from base classes
      void MapcInitComplete( TInt aError, const TTimeIntervalMicroSeconds& aDuration );
      void MapcPlayComplete( TInt aError );
      // [...]
      }
```

The CBase-derivation always has to be first!

Static Classes

- Static classes provide utility code
- Can not be instantiated
- No prefix letter

Examples

```
TInt computerMoveX = Math::Rand(iSeed) % iGridSize.iWidth;
User::After(1000);                    // Suspends the current thread for 1000 microseconds
Mem::FillZ(&targetData, 12);          // Zero-fills 12-byte block starting from &targetData
```

Summary – Classes

Prefix	Category	Examples	Description
T	Type	`TDesC`, `TPoint`, `TFileName`	No destructor, “has-a” data owned internally
C	Class	`CBase`, `CActive`, `CFbsBitmap`	Any class has to be derived from `CBase`. Always allocated on the heap. Member data automatically initialized with zero. Important for cleanup stack.
R	Resource	`RFile`, `RTimer`, `RWriteStream`, `RWindow`	Owns resources other than on the default heap. Usually allocated as members or automatic variables. Usually require `Close()` to free resources.
M	Mixin, interface	`MGraphicsDeviceMap`, `MEikMenuObserver`	Interface consisting of virtual functions. Implementation derives from it. Only approved use of multiple inheritance.
	Static class	`User`, `Math`, `Mem`	Consists purely of static functions, cannot be instantiated into an object.

When to use which class type?

Class contains no member data	Usually an interface → M class Sometimes C class
Class contains only static functions (or factory classes)	Static class → no prefix
Member data has no destructor / does not need special cleanup	Only contains native types (T classes) or “uses-a” data → T class
Size of the data contained by the class will be large (> 512 bytes)	Stack is limited → avoid T class → typically a C class
Class owns data that needs cleanup	Usually a C class. Also R-classes, which are seldom implemented yourself

Try it for your own

… let’s move to the Challenges!