

# Migrating Applications to Symbian OS v9.1

Mark Shackman

Version 1.1

<b>1</b>	<b>INTRODUCTION .....</b>	<b>1</b>
<b>2</b>	<b>OVERVIEW OF PLATFORM SECURITY.....</b>	<b>1</b>
	2.1 CAPABILITIES .....	2
	2.2 DATA CAGING .....	2
<b>3</b>	<b>UPDATING APPLICATIONS TO RUN UNDER SYMBIAN OS V9.1.....</b>	<b>3</b>
	3.1 PROJECT (MMP) FILE.....	3
	3.2 RESOURCE FILES .....	4
	3.3 CHANGES TO THE APPLICATION'S SOURCE CODE .....	7
<b>4</b>	<b>EXAMPLE FILES .....</b>	<b>7</b>
	4.1 APPNAME. MMP.....	7
	4.2 APPNAME_REG. RSS .....	8
	4.3 APPNAME. RSS.....	8
	4.4 APPNAME_REG. RSS .....	8
	4.5 APPNAME_LOC. RSS .....	9
<b>5</b>	<b>FURTHER INFORMATION .....</b>	<b>10</b>
	5.1 REFERENCES.....	10

## 1 Introduction

The introduction of data caging and platform security in Symbian OS v9 has necessitated making some changes in the source code of an application that is being migrated to Symbian OS v9. This paper outlines the concepts behind the data caging and platform security, and then shows how to alter the application's source code to enable it to be compiled for Symbian OS v9-based phones.

## 2 Overview of Platform Security

This section outlines some of the relevant features of the Symbian OS v9 platform security implementation. For further details, see references [1] and [2].

## 2.1 Capabilities

Platform security permits access to sensitive APIs according to capabilities. Capabilities are used to specify what functionality an application is trusted to use; they are allocated to the application at build time and are policed at run time. Once the capabilities are assigned to the application, they cannot be changed. Thus an application has a set of unalterable capabilities that describe what access the application has to the APIs.

### 2.1.1 Apps and exes under Platform Security

In Symbian OS v8.1 and previous releases, applications were compiled as . apps. These were polymorphic interface DLLs (with a single entry point) and were marked as such by having the line TARGETTYPE app in their MMP file. Whenever a . app's icon was selected, a special application framework executable, called apprun. exe, was started as a new process. It was within the main thread of this process that the . app was executed.

With platform security, the capabilities of a program are determined by the process that is running. If apprun. exe is used to run every . app, each application run this way will have the same capabilities. Therefore, each application now has to execute as a process in it's own right and hence all applications need to be built as executables (with the extension . exe).

### 2.1.2 Determining which capabilities are required

There are three suggested ways to establish which capabilities an application requires.

The first method is initially to allocate capabilities based on the general operations that the application performs. For example, an instant messaging application would probably require NetworkServices to access the Internet and ReadUserData to read the user's contacts.

The second method is to use the Symbian Developer Library, which lists the capabilities required by each API. Note that some APIs may be marked as "Dependent", meaning that the need for the capability depends on the parameters being passed (for example, accessing a file in an application's private data area requires no capabilities, but accessing a file in \sys\bin requires the AllFiles capability).

The third method is to run the application in the emulator, analysing the debug file EPOCWIND.OUT (in the system temporary directory) for capability violations. The output will state the API that caused the violation and what capability was expected.

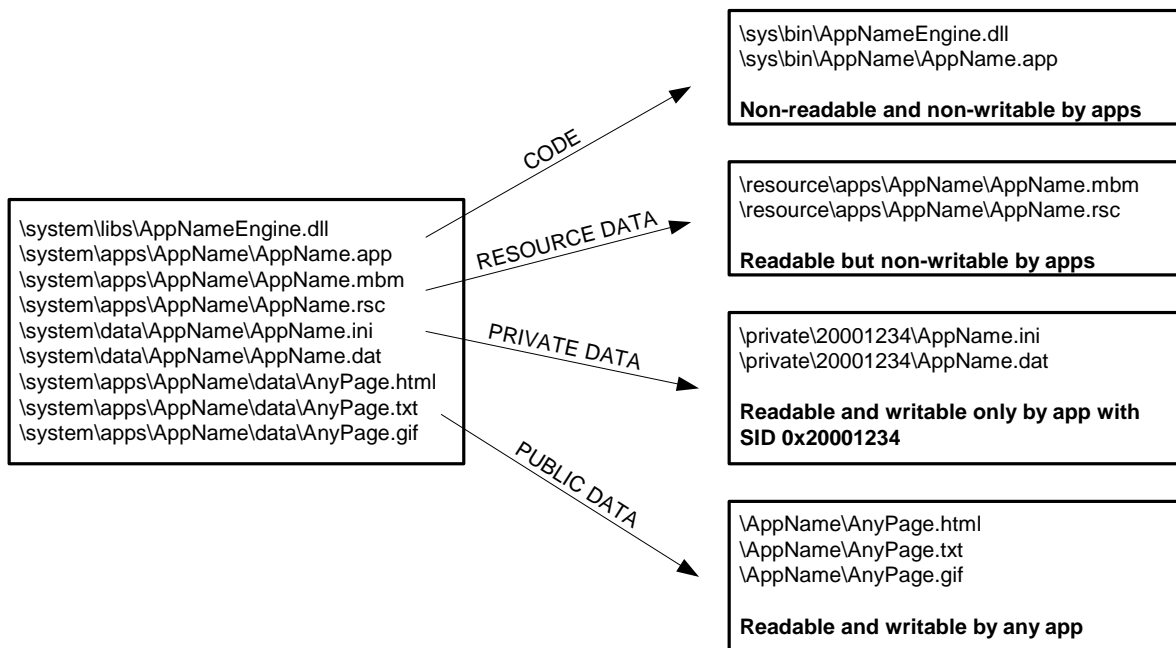
## 2.2 Data Caging

To protect the filing system, Symbian OS v9 uses data caging, meaning that parts of the filing system are "caged", or restricted, using the file path and application's capabilities to grant or deny access to files. Data caging thereby prevents programs from accessing directories holding system files, or those belonging to other programs. Additionally, all executable files are stored in, and can only be run from, \system\bin\.

To comply with the directory structure imposed by data caging, many of the files associated with an application need to be relocated. In summary, the new locations are:

Applications and DLLs	\sys\bin	
Resource files	\resource\apps	
Application-specific files	\private\<SID>\	where <SID> is the application's unique identifier

The following diagram illustrates the change in locations:



### 3 Updating applications to run under Symbian OS v9.1

This section provides step-by-step instructions on how to convert an existing DLL-type application to a new style EXE-type application.

#### 3.1 Project (MMP) file

The following changes are required to the project (MMP) file:

##### 3.1.1 Target settings

The TARGETTYPE in the MMP file changes from app to exe. The following lines should replace the existing TARGET and TARGETTYPE lines:

```
TARGET      AppName.exe
TARGETTYPE  exe
```

The TARGET filename's extension can be either .app or .exe. The second UID (the application identifier) must remain the same: 0x100039CE.

##### 3.1.2 Build the application binary to a new location

The application binary needs to be deployed in the new data-caged location. The following line should replace the TARGETPATH line associated with the TARGET:

```
TARGETPATH  \sys\bin
```

##### 3.1.3 Stack settings

The stack size should be explicitly set to a value greater than the default of 8K. Previously, applications were launched in a process initiated by apprun.exe which had a stack size of 20K. Therefore, to ensure the converted application will not have any problems with stack size, 20K is the minimum value the stack size should be set to. This is done in the MMP file with the line:

EPOCSTACKSIZE 0x5000

### 3.1.4 Libraries

The API for starting the application framework is in `ei_kcore.dll`, so `ei_kcore.lib` needs to be listed in the LIBRARY section of the MMP file:

LIBRARY ei\_kcore.lib

### 3.1.5 Capabilities

The capabilities required by the application should be established (see section 2.1.2). These are put in the MMP file using CAPABILITIES keyword, followed by the capabilities required:

CAPABILITIES ReadUserData WriteUserData

### 3.1.6 Secure ID

All applications must have a unique identifier (UID). In Symbian OS v9, the range of UIDs that can be allocated to applications has been divided as follows:

UID Range	Intended Use
0x00000000	KNUI UID
0x10000000-0x1FFFFFFF	Legacy UIDs, unused in v9
0x20000000-0x2FFFFFFF	Protected range UIDs/SIDs for v9 onwards
0xA0000000-0xFFFFFFFF	Unprotected range UIDs/SIDs for v9 onwards
0xE1000000-0xEFFFFFFF	Testing range for v9 onwards

Applications for Symbian OS v9 that are signed must have an identifier in the protected range 0x20000000-0x2FFFFFFF. This is called the Secure Identifier (SID) and is specified in the MMP file as:

SECUREID 0x20000B62

You should obtain a SID for your application from [www.symbiansigned.com](http://www.symbiansigned.com).

If the SID is not specified explicitly, the UID3 value is used instead.

### 3.1.7 Resources

Application resources are specified in the MMP file using START RESOURCE ... END blocks. These are discussed in the next section.

## 3.2 Resource files

Resource files contain details of icons and captions for the application. Prior to Symbian OS v9, these were contained in AIF and caption files; these have now been superseded by application registration information and therefore any reference to AIF files in the MMP file should be deleted.

The new application registration information comprises the mandatory registration file, which contains information about the application such as its properties, and optional localisable resource files. If a developer prefers not to use localisable resource files, the location of caption and icon definition files may also be provided in the application's UI resource file.

The following sections detail the steps required to migrate an application's resources.

### 3.2.1 The registration file

Application registration files define information about an application's name, UID and properties that is required by the application launcher or system shell. All applications must provide a registration file, called *AppName\_reg.rss*.

A registration file is a standard Symbian OS compiled resource file (.rsc). The .rss file used to build the registration file should be named *AppName\_reg.rss* and contain at least the following lines:

```
#include <appinfo.rh>
UID2 KUidAppRegistrationResourceFile
UID3 0x20000B62 // application UID (SID)
RESOURCE APP_REGISTRATION_INFO
{
    app_file="AppName"; // filename of application binary (minus extension)
}
```

To build the registration file, add the following lines to the application's MMP file:

```
START RESOURCE AppName_reg.rss
#ifdef WINSCE
TARGETPATH \private\10003a3f\apps
#else
TARGETPATH \private\10003a3f\import\apps
#endif
END
```

The registration file is used by the Application Architecture server, which has a SID of 0x10003a3f. The *import* subdirectory (if it exists) allows an application to write files into a second application's private data directory, providing that it knows the second application's SID. The *TARGETPATH* value above ensures that registration files are deployed to the Application Architecture's private data caged area.

For examples of registration files, see sections 4.2 and 4.4 below.

### 3.2.2 Default UI resource file and MBM file location

If the application provides a UI resource file, the *TARGETPATH* must be specified in the *START RESOURCE ... END* block in the MMP file as follows:

```
START RESOURCE AppName.rss
HEADER
TARGETPATH \resource\apps
END
```

If the *HEADER* keyword is supplied, a header will be generated in *\epoc32\include*.

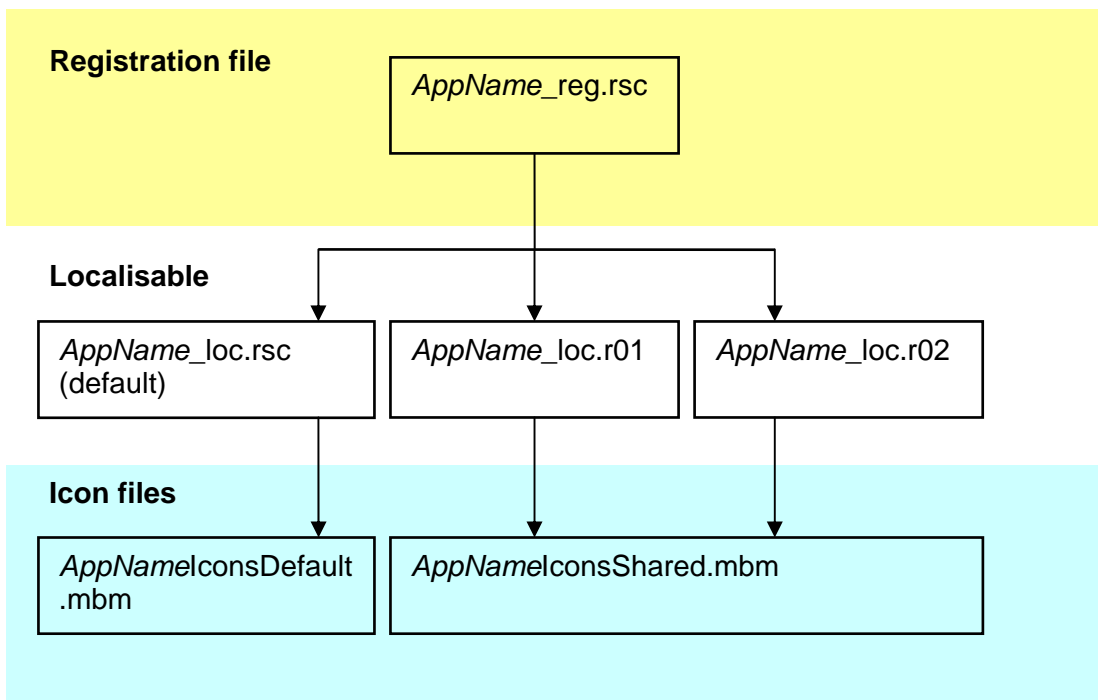
If the application provides an MBM bitmap file, the *TARGETPATH* must be specified in the *START BITMAP ... END* block in the MMP file, similarly to the above example.

### 3.2.3 Application information

The application information is provided in two categories:

- Non-localisable application information (provided by the registration file)
- Localisable application information (localisable resource file and icon file)

The diagram below shows the new file structure used to provide localisable application information.



The registration file (see section 3.2.1) "points" to one of three localisable resource files provided. If the current language is 01 or 02 (see reference [3] for discussion of localisation), the system will load the corresponding version of *AppName\_loc*, otherwise *AppName\_loc.rsc* will be loaded.

Note that *AppName\_loc.r01* and *AppName\_loc.r02* share the same icon file.

### 3.2.3.1 Localisable resource file

Localisable application information can either be provided in a separate resource file (see example *AppName\_loc.rsc* in section 4.5 below), or as a resource within the application UI resource file (see *AppName.rsc* in section 4.3 below).

When providing the information as a resource within the application UI resource file, the registration file's `APP_REGISTRATION_INFO` struct must contain the following lines:

```

Localisable_resource_file = "\\resource\\apps\\AppName";
Localisable_resource_id = R_LAI;
  
```

where `R_LAI` is a named `LOCALISABLE_APP_INFO` resource struct within the UI resource file.

The `Localisable_resource_file` field should not include a drive in the path, or a file-extension, but it should provide a full directory path and a file name (without extension).

To build a resource file for language 01, add the following to the MMP file:

```

START RESOURCE AppName_loc.rsc
TARGETPATH \resource\apps
LANG 01
END
  
```

Icon files referenced in the localisable resource should also be built to `\resource\apps\`.

### 3.3 Changes to the application's source code

All Symbian OS v9 applications are stand-alone executables, therefore the old `GLDEF_C TInt E32Dll()` function should be removed. Instead, a new entry point, called `E32Main()`, must be provided.

For applications, `E32Main()` must call the `EikStart::RunApplication()` function, passing as an argument a pointer-to-function which creates an instance of the `CApaApplication`-derived class (this is the same function that is exported at ordinal 1 for DLL-apps). This is shown in the following code extract:

```
#include <eikstart.h>

GLDEF_C TInt E32Main()
{
    return EikStart::RunApplication(NewApplication);
}
```

A small number of APIs have been updated for Symbian OS v9. Any such changes should be evident in the compilation output, and you should refer to the Symbian Developer Library for details of the new parameters.

## 4 Example files

This section contains example files.

### 4.1 *AppName.mmp*

```
TARGET                AppName.exe
TARGETTYPE            exe
TARGETPATH            \sys\bin
UID                   0x100039CE 0x20000B62
SECUREID              0x20000B62
EPOCSTACKSIZE        0x5000
SOURCEPATH            .
SOURCE                AppName.cpp
USERINCLUDE           .
SYSTEMINCLUDE         \epoc32\include

// Application exe registration resource file
START_RESOURCE        AppName_reg.rss
#ifdef WINS32
    TARGETPATH        \private\10003a3f\apps
#else
    TARGETPATH        \private\10003a3f\import\apps
#endif
LANG                  sc
END

// Resource file
START_RESOURCE        AppName.rss
HEADER
TARGETPATH            \resource\apps
END

LIBRARY               euser.lib apparc.lib eikcore.lib
```

## 4.2 *AppName\_reg.rss*

This version of the file contains localisable application information as a resource within the application UI resource file.

```
#include <appinfo.rh>
#include <appname.rsg>

UID2 KUIDAppRegistrationResourceFile
UID3 0x20000B62 // application UID (SID)

RESOURCE APP_REGISTRATION_INFO
{
    app_file = "AppName";
    //
    localisable_resource_file = "\\resource\\apps\\appname";
    localisable_resource_id = R_LAI;
}
```

## 4.3 *AppName.rss*

This version of the file contains localisable application information as a resource within the application UI resource file.

```
#include <ekon.rh>
#include <ekon.rsg>
#include <appinfo.rh>

NAME APPN

RESOURCE RSS_SIGNATURE { }
RESOURCE TBUF r_appname_default_file { buf="default file name"; }
RESOURCE EIK_APP_INFO
{
    hotkeys=r_appname_hotkeys;
    menubar=r_appname_menubar;
    toolbar=r_appname_toolbar;
}
RESOURCE LOCALISABLE_APP_INFO r_lai
{
    short_caption = "AppName";
}
RESOURCE TOOLBAR r_appname_toolbar { }
RESOURCE HOTKEYS r_appname_hotkeys { }
RESOURCE MENU_BAR r_appname_menubar { }
```

## 4.4 *AppName\_reg.rss*

This version of the file contains localisable application information in a separate resource file.

```
#include <appinfo.rh>

UID2 KUIDAppRegistrationResourceFile
UID3 0x20000B62 // application UID (SID)

RESOURCE APP_REGISTRATION_INFO
{
    app_file = "AppName";
    //
    localisable_resource_file = "\\resource\\apps\\AppName_loc";
}
```

```
//
hidden = KAppNotHidden;
embeddability = KAppNotEmbeddable;
newfile = KAppDoesNotSupportNewFile;
launch = KAppLaunchInForeground;
group_name = "AppNameGroup";
//
default_screen_number = 2;
//
datatype_list =
{
    DATATYPE { priority=EDatatypePriorityNormal; type="image/jpeg"; },
    DATATYPE { priority=EDatatypePriorityNormal; type="image/gif"; }
};
//
file_ownership_list =
{
    FILE_OWNERSHIP_INFO {file_name="//temp//AppNameTempFile.txt"; },
    FILE_OWNERSHIP_INFO {file_name="//temp//AppName.txt"; }
};
}
```

#### 4.5 AppName\_loc.rss

This version of the file contains localisable application information in a separate resource file. RLS files have been used to demonstrate how resource files for different languages (AppName\_loc.r01 and AppName\_loc.rsc, shown below) can be built from a single RSS file.

```
#include <appinfo.rh>
#ifdef LANGUAGE_01
#include "AppName01.rls"
#else
#include "AppNamesc.rls"
#endif
RESOURCE LOCALISABLE_APP_INFO
{
    short_caption = STRING_r_short_caption;
    caption_and_icon =
    {
        CAPTION_AND_ICON_INFO
        {
            caption = STRING_r_caption;
            number_of_icons = 2; // each icon is a bitmap/mask pair
            icon_file = STRING_r_icon_file;
        }
    };
//
view_list =
{
    VIEW_DATA
    {
        uid = 268123123;
        screen_mode = 0x00;
        caption_and_icon =
        {
            CAPTION_AND_ICON_INFO
            {
                caption = STRING_r_view_268123123_caption;
                number_of_icons = 1; // each icon is a bitmap/mask pair
            }
        }
    }
};
```

