Document Handler

Support Guide

1. Introduction

This document describes how to implement specified application interaction techniques in Series60. A Series60 subsystem called Document Handler is a utility to handle application launching and content saving, based on MIME types. It hides all the necessary tasks behind its simple interface.

Document Handler is designed for saving content and launching applications to handle content, originated from e.g. WAP Push messages, WMLBrowser download operations, Mail attachments, MMS messages or Infrared and Bluetooth receiving. It can be used by other applications as well. The main purpose of Document Handler is to hide the complexity of Symbian OS's standard interface for content handling.

Symbian OS supports two types of application launching, standalone and embedded. Launching is standalone when a handler application runs in its own process and the user can switch between the caller and the launched application freely. If the caller application embeds the handler application into its own process, the user must exit the embedded application first to get back to the original application. Embedded launching is the preferred type in Series60.

By using Document Handler, the client application developer doesn't have to know too much about application launching. It doesn't require any special support from host applications. However, handler applications have to follow some simple rules in order to support Document Handler. In this document, you'll find the instructions.

Document Handler depends heavily on Symbian OS's Application Architecture MIME-type mapping mechanism. MIME types are mapped to the appropriate application's UID. For the most basic cases, Document Handler uses the standard Application Architecture mechanism directly. Document Handler also allows non-standard application launching through the same simple interface.

In addition to launching applications, Document Handler offers an interface for saving content to application-specific places and format. Content can be either in a file or in a buffer. Document Handler takes care of all necessary tasks for its clients: finding the correct place for saving, offering uniform save as functionality, unnamed buffers are named, etc.

Figure 1 illustrates the bigger picture, and is extremely simplified. Applications mentioned in the picture are real, but not all applications involved are included. The client of Document Handler can also be an entity other than an application. For example, Document Handler also handles documents coming via WAP Push operations and there is no application involved.

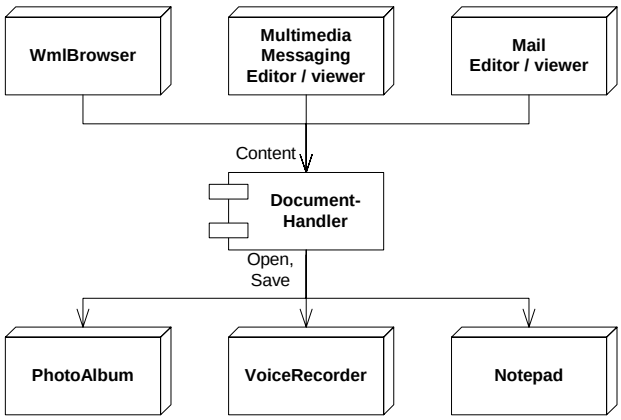

Figure 1 The bigger picture

Note: For security reasons, Document Handler discards all executable file formats, i.e. it is not possible to automatically launch received programs from Inbox, Services, etc.

2. Check List for Developers

These are the requirements for applications responsible for different kinds of MIME types in Series60. It is necessary for third-party applications to fulfil these requirements in order to be able to receive content in a Series60 device.

2.1 Make your application embeddable

In Series60 Embedded applications run in the same process as the host application. In the user interface the host application's icon is displayed instead of the embedded application's. In figure 2 a WAP page is opened in the WAP browser. The page contains an image, which in figure 3 is opened in the Photoalbum that is embedded into the browser. Menu options and the caption are from Photoalbum. Pressing cancel would lead the user always back to the WAP browser. You can find instruction for this in chapter 3.

Figure 2 WAP browser as a standalone application.

Figure 3 Photoalbum embedded into the WAP browser's process.

2.2 Register Your MIME types in the AIF File.

Applications must register their supported MIME types in the Application Information File. You can find instructions for this in chapter 3.

2.3 Make sure your mime type has a proper recogniser.

Recognisers are used to find out what the MIME type for a file is. If one is not already available in the system, the application developers have to provide a suitable recogniser. You can find instructions for this in chapter 4.

2.4 Declare default document name in resources.

For example, the Imaging Application may want to use the terms “photo” or “image” instead of generic terms like “Unnamed”. The name will be used when the caller of the Document Handler cannot supply a name for the content.

The correct place for the name is in the application's resources. If the name is not available, Document Handler uses a global name for documents without names. Example:

```
// RESOURCE DEFINITIONS
// These are standard Symbian OS resources
RESOURCE RSS_SIGNATURE { }
RESOURCE TBUF { buf = localized_default_name; }
RESOURCE EIK_APP_INFO
    {
    }
```

2.5 Check the Embedding Status of Your Application

You can check the run time embedding status of your application from CEikAppUi class member iDoorObserver. idoor observer is waiting for your task to finish. If the iDoorObserver is not NULL, your application has been launched embedded. There is no need to change your application code, but you can use some special tricks. Example:

```
if ( MyAppUi->iDoorObserver != NULL )
    {
    // Disable some menu items or do some other
    // things according to your UI specification.
    }
```

Refer to Symbian OS documentation for more information on embedding.

2.6 Delegate the exit mode to the observer.

When your application is just about to finish call

```
MyAppUi->iDoorObserver->NotifyExit( MApaEmbeddedDocObserver::TExitMode ).
```

Look at apparc.h header for exit modes.

2.7 Overload CAknDocument: OpenFileL function

Overload `CAknkDocument::OpenFileL( TBool aDoOpen, TDesC& aFilename, RFs& aFs )` function to support opening files in embedded mode. The default implementation in CAknDocument is empty. If your document class is inherited from Symbian OS CEikDocument class, you may still have to overload the function, because the implementation in CEikDocument supports only Symbian OS stream store files with application UID's. This will not work if the file is a JPEG image, for example.

Note: Remember to inherit your application's document class from CAknDocument.

2.8 Declare your application's data directory

Declare your application's data directory with a SharedData keyword. Document Handler must know the correct place in which to save your data. In case of just viewing, it is possible to save the file in d:\system\temp directory, but be warned: the file can be deleted or altered at any time. For permanent saving, you must follow the rules.
Here is how to do it:

1. Write a plain text file containing Keyword-value pair. The keyword must be SDDataDir, but the value can be any directory that is accessible in the device. For example,

```
SDDataDir=c:\system\myfirmApp
```

2. Save the file in Unicode format.

3. Name the file as <myAppid>.ini, where <myAppid> part is your application ID in hex without the "0x" part. For example, 10039110.ini

Copy the file from the installation package to \system\SharedData directory. Here is an example of a line in the application installation package descriptor:
"\MyFirmApp\group\10039110.ini"-"!:\system\SharedData\10039110.ini "

Tip: If you need separate directories for different MIME types, here is the way to do it:
Append '-' and an MIME type that you support after the normal SDDataDir key:

Example:

SDDataDir-text/vnd.myfirm.format_1=c:\system\myfirm\format_1

SDDataDir-text/vnd.myfirm.format_2=c:\system\myfirm\format_2

3. How-to Support Mime Types and Embeddability

3.1 Symbian OS MIME Support

The MIME types supported by applications are introduced in the AIF file (Application Information File). The snapshot taken from Symbian OS system documentation explains this as follows:

Multipurpose Internet Mail Extensions, MIME, define a file format for transferring non-textual data, such as graphics, audio and fax, over the Internet. The datatype_list section of the aiftool resource file lists the MIME types that the application supports, and the priority of support that each type is given. When a file is to be opened, Symbian OS launches the application, which has the highest priority support for the selected file type.

There are four priority levels, of which only **`EDataTypePriorityNormal`** or **`EDataTypePriorityLow`** should normally be used. For example, a text editor is good at editing text/plain files, and would hence be given a priority of `EDataTypePriorityNormal` for that file type. A web browser is less good at handling text files, and would be assigned the lower priority `EDataTypePriorityLow`. Hence, either application can be launched to handle a text document. However, if both applications are present; the text editor is launched by preference.

`EDataTypePriorityHigh` should only be assigned under exceptional conditions — e.g. if no other application could ever handle a particular MIME type as brilliantly.

`EDataTypePriorityLastResort` should also be used sparingly. Text editors are terrible at displaying HTML, and would either have the priority `EDataTypePriorityLastResort`, or would not support the type at all.

Note:

- Given two applications with the same MIME-type priority, Symbian OS arbitrarily launches one of the applications.
- Developers can give users the ability to set their application as the default MIME-type launcher — through the user-defined mapping API (datastore.h, class CtypeStoreManager). This is usually accessed via an application menu option "Make this application the default handler".

3.2 Series60 Application Embedding

In Series60, Document Handler launches applications in embedded mode. The implementation is easy: you don't really have to do anything. The Symbian OS application framework takes care of this for you. All that is needed is to set **`embeddability`** flag in resources to **KAppEmbeddable**. See the resource code example below.

3.3 Application Info File (.aif)

The first task is to list all the MIME types an application supports. For that, we need to create an own resource file to be compiled by the aiftool included in Symbian OS SDK. In this example, we add support only one MIME type, "text/demo" which maps later to artificial ".demo" extension.

The exact syntax of the resource file and more information about keyword values can be found from standard Symbian OS system documentation under the topic of "Aiftool resource file format".

```
//
// MyFirmApp.RSS
//

#include <aiftool.rh>
RESOURCE AIF_DATA
   {
     // Example application's UID
     app_uid = 0x10009ACD;
```

```
    // Number of icons to be included in AIF file
    num_icons = 1;
    // Application capabilities.
    // Viewer applications in Series60 must support embedding
    embeddability = KAppEmbeddable;
    hidden=KAppNotHidden;
    // MIME types application supports
    datatype_list=
        {
        DATATYPE
           {
           priority = EDataTypePriorityNormal;
           type = "application/vnd.mymime";
           },
           {
            // Other MIME types can be added here
           }
        };
     } // AIF_DATA
```

This resource file is then compiled, using the aiftool, to produce the resulting MyFirmApp.AIF file.

4. Implementing recognisers

4.1 Mime Recogniser DLL (.MDL)

In Series60, we often need to open a viewer application for files. To help Symbian OS to find the proper viewer for a file, application developers need to implement MIME-type recognisers, which map file extension or file content to a handler application. The resulting DLL has .MDL extension and it is installed in c:\system\recogs directory.

Caution: In Series60, the preferred way of recognition is based on data itself, not only on the file extension. This helps to avoid security risks caused by faulty content.

By default, Series60 contains recognisers at least for the following MIME types:

- application/vnd.nokia.ringing-tone
- text/vnd.sun.j2me.app-descriptor
- application/vnd.symbian.install
- application/x-NokiaGameData
- image/vnd.nokia.ota-bitmap
- application/vnd.wap.wbxml
- application/vnd.wap.wmlc
- application/java-archive
- image/vnd.wap.wbmp
- audio/x-epoc-wve
- audio/x-sibo-wve
- image/x-epoc-mbm
- text/x-vcalendar
- text/x-vcard
- audio/x-wav
- audio/basic
- audio/midi
- image/x-wmf
- image/x-bmp
- image/tiff
- image/jpeg
- text/plain
- audio/amr
- image/png
- image/gif

Tip: In Nokia 7650, if you want the users to be able to download content for your application via WAP, you will have to provide a recogniser for all your MIME types. The accept-header of the fetch operation is dynamically generated, based on recognised MIME types in the device.

4.2 Implementing MDL

New MIME recognizers can be implemented by tuning existing ones. Here is a source code example for an imaginary mime type.

```
/*********************************************************/
```

```
/*                                                            */
/*                   MyRecognizer.h                           */
/*                                                            */
/************************************************************/

// includes
#include <apmrec.h>  // For CApaDataRecognizerType

// Mime type string and the extension
_LIT8( KMyMimeType, "application/vnd.mymime" );
_LIT( KDotMymime,".mym" );

// File header to look for from the data
_LIT8( KMyMimeHeader, "#!MY" );

// TUid of the recognizer
const TUid KUidMyMimeRecognizer( { 0x100530 } );

class CMyRecognizer : public CApaDataRecognizerType
    {

    public: // from CApaDataRecognizerType
        CMyRecognizer();
        virtual TUint PreferredBufSize();
        virtual TDataType SupportedDataTypeL( TInt aIndex ) const;

    private: // from CApaDataRecognizerType
        virtual void DoRecognizeL(const TDesC& aName,
                                  const TDesC8& aBuffer );
    // New funtions
    private:
        // Check the file name extension
        TBool NameRecognized( const TDesC& aName );
       // Look into the data
        TBool HeaderRecognized( const TDesC8& aName );
    };
// End of file
/************************************************************/
/*                                                            */
/*                   MyRecognizer.cpp                         */
/*                                                            */
/************************************************************/

// Includes
#include "MyRecognizer.h"


//
// Constructor
//
MyRecognizer::MyRecognizer()
    :CApaDataRecognizerType(
    KUidMyMimeRecognizer,
    CApaDataRecognizerType::EHigh )
    {
    iCountDataTypes = 1;
    }


//
// Preferred buffer size.
//
TUint MyRecognizer::PreferredBufSize()
    {
    return 128;
    }
//
```

```
// For the framework for collecting the supported mime types list.
//
TDataType MyRecognizer::SupportedDataTypeL( TInt aIndex ) const
    {
    switch( aIndex )
        {
        case 0:
        default:
           return TDataType( KMyMimeType );
           break;
         }
     }

//
// The framework calls this function for recognition
//
void MyRecognizer::DoRecognizeL(
    const TDesC& aName,
    const TDesC8& aBuffer )
    {
    TBool     nameOk( EFalse );
    TBool     headerOk( EFalse );
    iConfidence = ENotRecognized;

    if ( aBuffer.Length() < 10 )
        return;

    // First try the name, then the data.
    nameOk  = NameRecognized( aName );
    headerOk = HeaderRecognized( aBuffer );

    if ( nameOk && headerOk )
        {
        iConfidence = ECertain;
        }
      else if ( !nameOk && headerOk )
       {
       iConfidence = EProbable;
       }
       else if ( nameOk && !headerOk )
       {
       iConfidence = EPossible;
       }
      else
          return;
     iDataType = TDataType( KMyMimeType );
    };
//
// Check if the file header can be recognized
//
TBool MyRecognizer::HeaderRecognized( const TDesC8& aBuffer )
    {
    if ( aBuffer.Find( KMyMimeHeader ) )
        return ETrue;
    else
        return EFalse;
    }

//
// Check if the file name has ".mym" extension
//
TBool MyRecognizer::NameRecognized( const TDesC& aName )
    {
    TBool ret = EFalse;
    if ( aName.Length() > 5 )
        {
        TInt dotPos = aName.LocateReverse( '.' );
```

```
        if (dotPos != KErrNotFound)
            {
            TInt extLength = aName.Length() - dotPos;
            HBufC* ext = aName.Right( extLength ).AllocL();
            CleanupStack::PushL( ext );
            if ( ext->CompareF( KMyMimeHeader ) == 0 )
                {
                ret = ETrue;
                }
            CleanupStack::PopAndDestroy(); // ext
            }
        }
    return ret;
    }

//
// The gate function - ordinal 1
//
EXPORT_C CApaDataRecognizerType* CreateRecognizer()
    {
    CApaDataRecognizerType* thing = new MyRecognizer();
    return thing; // NULL if new failed
    }

//
// DLL entry point
//
GLDEF_C TInt E32Dll(TDllReason /*aReason*/)
    {
    return KErrNone;
    }

// End of file

/*************************************************************/
/*                                                           */
/*                   MyRecognizer.mmp                        */
/*                                                           */
/*************************************************************/
TARGET          MyRecognizer.MDL
TARGETTYPE      MDL
UID             0x10003A19 0x100530
TARGETPATH      \system\recogs\
SOURCEPATH      ..\src
SOURCE          MyRecocnizer.cpp
USERINCLUDE     ..\inc
SYSTEMINCLUDE   \epoc32\include
LIBRARY         EUSER.LIB
LIBRARY         APMIME.LIB
```