
S60 Platform: Document Handler API Developer's Guide

Version 1.1
November 1, 2007

S60 platform

Legal notice

Copyright © 2006, 2007 Nokia Corporation. All rights reserved.

Nokia and Nokia Connecting People are registered trademarks of Nokia Corporation. Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. Other product and company names mentioned herein may be trademarks or trade names of their respective owners.

Disclaimer

The information in this document is provided “as is,” with no warranties whatsoever, including any warranty of merchantability, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification, or sample. Furthermore, information provided in this document is preliminary, and may be changed substantially prior to final release. This document is provided for informational purposes only.

Nokia Corporation disclaims all liability, including liability for infringement of any proprietary rights, relating to implementation of information presented in this document. Nokia Corporation does not warrant or represent that such use will not infringe such rights.

Nokia Corporation retains the right to make changes to this specification at any time, without notice.

License

A license is hereby granted to download and print a copy of this specification for personal use only. No other license to any other intellectual property rights is granted herein.

Contents

1.	Introduction	5
1.1	Document Handler example.....	5
2.	Document Handler basics	6
2.1	MIME types, recognizers, and handler applications	6
2.2	Embedded and standalone application launching	6
2.3	RApaLsSession.....	6
2.4	CDocumentHandler class implementation.....	6
3.	Using the Document Handler API	8
3.1	Getting started.....	8
3.2	Opening files	8
3.2.1	Handler application and the document file	9
3.2.2	Embedded file opening.....	9
3.3	Creating and saving files.....	9
3.4	Moving and copying files.....	9
3.5	Query functions	11
3.6	Changes in S60 3rd Edition	11
3.7	Observing handler application exit.....	12
3.8	DRM protection	13
3.9	Parameter passing	13
3.10	Error situations and tips and tricks.....	13
3.10.1	Updating document file for standalone handlers.....	13
3.10.2	Changing the file content.....	14
3.10.3	Determining the handler application.....	14
4.	Terms and abbreviations	16
5.	References	17
Appendix A.	Changes in the Document Handler API	18
	Evaluate this resource.....	21

Change history

December 22, 2006	Version 1.0	Initial document release
November 1, 2007	Version 1.1	Section 3.2.1 updated with issues to take into account when the document handler application is running in the background.

1. Introduction

This document describes the usage of the Document Handler API in the S60 platform. Document Handler is a concept that provides means to show different content in applications that have the capability of handling that content type. Applications such as Notepad, which can handle text format, and Image Viewer, which can handle different types of images, are called handler applications. The most convenient way to let the end user see (and possibly modify) file content using handler applications is the Document Handler API.

The Document Handler API is available in all S60 SDKs and widely used in the S60 platform. In addition to the normal usage of showing document content its file operations, including move and copy, are used in many S60 applications.

1.1 Document Handler example

The [S60 Platform: Document Handler Example](#) [3], available at [Forum Nokia](#), demonstrates MIME type recognition using its own recognizer which sets the MIME type to “`application/something-totally-new`” for all files with the extension “.new”. A handler application and a tester application are also provided. The tester application uses the Document Handler API to launch a file with the .new extension. After the MIME type has been recognized, the handler application is launched. The handler application shows the name and the beginning of the document file (the files are expected to be in text format).

The example has three components:

- HandlerApp — the handler application
- TesterApp directory — the caller application which uses the Document Handler API
- RecognizerDll directory — the recognizer

The example is compatible with all S60 editions.

2. Document Handler basics

2.1 MIME types, recognizers, and handler applications

Handler applications can handle certain MIME types. Once a document is launched, MIME type recognition takes place. After the MIME type has been recognized, a handler application (which has registered to be able to handle that MIME type) is opened and the document is passed to that handler application.

In Symbian OS, MIME type recognition is done using recognizer DLLs. The [S60 Platform: Document Handler Example](#) [3] also demonstrates the usage of recognizers. Recognizers are used for handling certain types of documents. The documents are recognized, for example, using their extension (for example, `.txt`). Recognizers are implemented as ECom plug-ins in S60 3rd Edition. The recognizer framework maintains a list of all recognizer plug-ins installed in the system. When there is an attempt to open a document, each recognizer is passed the file name and a buffer containing a few bytes from the beginning of the unrecognized file. Recognizers then return the confidence level that they recognize the file with. Once the recognition has been completed and the MIME type is known, a handler application for this MIME type is launched.

2.2 Embedded and standalone application launching

Embedded applications run in the same window group as the application launching it, so the user cannot switch between the embedded and the original application. This is the preferred way for handler applications. Most S60 applications are launched as standalone applications. The user can switch the application to the background and the foreground because each standalone application has its own window group.

2.3 RApaLsSession

Most of the Document Handler API's functionality is based on the `RApaLsSession` class. `RApaLsSession` provides a session with the application architecture server and multiple services, such as getting a list of all applications and supported MIME types.

2.4 CDocumentHandler class implementation

The only class in the Document Handler API is `CDocumentHandler`. Its public interface is defined in `documenthandler.h` (from the S60 3rd Edition SDK):

```
NONSHARABLE_CLASS(CDocumentHandler) : public CBase
{
public:
    static CDocumentHandler* NewL( CEikProcess* aProcess );
    static CDocumentHandler* NewLC( CEikProcess* aProcess );
    static CDocumentHandler* NewL( );
    static CDocumentHandler* NewLC( );
    virtual ~CDocumentHandler();
public:
    // New functions, data caging
    void OpenTempFileL(
        const TDesC& aFileName, RFile &aSharableFile);
    void SaveTempFileL(const TDesC8& aContent,
        TDataType& aDataType,
        const TDesC& aFileName,
        RFile &aSharableFile);
public:
    // New functions, parameter handling
    CAiwGenericParamList& InParamListL();
    const CAiwGenericParamList* OutParamList(); //deprecated
public:
    // New functions, open file with filehandle
```

```

TInt OpenFileL(
    RFile& aSharableFile,
    TDataType& aDataType);
TInt OpenFileEmbeddedL(
    RFile& aSharableFile,
    TDataType& aDataType,
    const CAiwGenericParamList& aParamList);
TInt OpenFileEmbeddedL(
    RFile& aSharableFile,
    TDataType& aDataType);
TInt OpenFileL(
    const TDesC& aFileName,
    TDataType& aDataType );
TInt OpenFileEmbeddedL(
    const TDesC& aFileName,
    TDataType& aDataType );
public:    // data saving
TInt SaveL(
    const TDesC8& aContent,
    TDataType& aDataType,
    const TUint aAttr );
TInt SaveL(
    const TDesC8& aContent,
    TDataType& aDataType,
    const TDesC& aName,
    const TUint aAttr );
TInt CopyL(
    const TDesC& aFileNameOld,
    const TDesC& aNameNew,
    TDataType& aDataType,
    const TUint aAttr );

TInt CopyL(
    const RFile& aFileOld,
    const TDesC& aNameNew,
    TDataType& aDataType,
    const TUint aAttr );
TInt MoveL(
    const TDesC& aFileNameOld,
    const TDesC& aNameNew,
    TDataType& aDataType,
    const TUint aAttr );
TInt SilentMoveL(
    const TDesC& aFileNameOld,
    const TDesC& aNameNew,
    const TDesC& aRootPath,
    TDataType& aDataType,
    const TUint aAttr );
public:    // query functions
TBool CanHandleL( const TDataType& aDataType );
TBool CanOpenL( const TDataType& aDataType );
TBool CanSaveL( const TDataType& aDataType );
TInt GetPath( TDes& aPath );
TInt HandlerAppUid( TUid& aUid );
void SetExitObserver( MAKnServerAppExitObserver* aObserver);
void CheckFileNameExtension(
    TDes& aFileName,
    const TDataType& aDatatype );
RApaLsSession* ApaLs();
MAKnServerAppExitObserver* ServerAppExitObserver() const;
TDocOperation DocOperation() const;
void CloseSharableFS();
void SetTempFile( const TDesC& aTempFile);
};

```

3. Using the Document Handler API

The Document Handler API is most commonly used to show the contents of a document to the end user. The Document Handler API also provides other functionalities; these are covered in this chapter.

3.1 Getting started

To be able to use the `CDocumentHandler` class, `CommonUI.lib` must be added to the application's MMP file in a `LIBRARY` statement. Also the header file `documenthandler.h` has to be included in the source code.

There are two ways to create a `CDocumentHandler` object. The `NewL` (or `NewLC`) function which takes a `CEikProcess` object as a parameter has been deprecated in S60 3rd Edition, and `CDocumentHandler` should always be created without parameters in S60 3rd Edition.

The normal operation of the Document Handler API is to open a file. Opening a file launches the correct handler application showing the specified content. Other file operations, such as move or copy, use MIME type recognition to determine the destination directory of the files. For example, when an image is copied, it is copied to the Images directory (in S60 3rd Edition `C:\Data\Images`). This is very useful as files are correctly placed without the developer having to determine the directories.

The MIME type and the default directory are mapped together in a hard-coded manner, so files with unrecognized MIME types are always saved to the default directory. In these operations it is possible to let the Document Handler API resolve the MIME type of the file. It can also be set manually, but if the wrong MIME type is set, the file will not be in the correct directory. For example, if a text file is copied and the MIME type is set to audio ("audio/basic"), the file will be copied to `C:\Data\Sounds\Digital` and its extension is changed to `.au`.



Caution: Do not set the MIME type of your application if you are not absolutely sure about it. If the MIME type is incorrect, the file may be copied to a wrong place.

3.2 Opening files

Files are opened by calling the `OpenFileL` or `OpenFileEmbeddedL` functions of the `CDocumentHandler` class. After the file has been opened, a handler application is launched.

The following example shows how to open a file in S60 3rd Edition:

```
LIT(KTestFile, "c:\\data\\test.new")
iDocHandler = CDocumentHandler::NewL();

//the old way, used before S60 2nd Edition, FP3
/*iDocHandler = CDocumentHandler::NewL(
    CEikEnv::Static()->Process() ); */

iDocHandler->OpenTempFileL( KTestFile, iFile );
iDocHandler->OpenFileL(iFile, textType);
```

Now the `iFile` member can be closed. It has to be closed before `iDocHandler` is destroyed; otherwise a KERN-EXEC 0 panic occurs.

3.2.1 Handler application and the document file

The handler application receives the information about the document file in its `OpenFileL` function of the document (`CAknDocument`) class. See the [S60 Platform: Document Handler Example](#) [3] available at [Forum Nokia](#) for an example.

Note that if the document handler application is already running in the background, the `OpenFileL()` call for the document class is not received. The calling application must call `TAppTask::SwitchOpenFile()`. This, of course, works only if the developer has also implemented the caller application. If the file is launched from any other application (for example, S60 Web Browser or FileManager), a different approach is required: During startup, the handler application must terminate any previous instance of itself already running in the background. See [KIS000588 - Data handler applications must implement CAknAppUi::OpenFileL](#) [1] for further details.

3.2.2 Embedded file opening

A handler application can be launched in standalone mode or as an embedded application. Embedded applications (server applications in S60 3rd Edition) are discussed in the [S60 Platform: Application Framework Handbook](#) [2] available at [Forum Nokia](#). Embedded file opening is the preferred way of using the Document Handler API.

3.3 Creating and saving files

In addition to launching handler applications, the Document Handler API offers a simple interface for saving content in an application-specific place and format. The content can be either in a file or in a buffer. The Document Handler API takes care of all the necessary tasks for its clients: finding the correct place where to save the content, offering Save as functionality, etc.

The following example shows how to save content to a file and get the file path:

```
HBufC* tempFileName = HBufC::NewMaxLC( KMaxPath );
TPtr tempFileNamePtr( tempFileName->Des() );
TDataType type( KMyMimeType ); //the used MIME type
iDocHandler->SaveL( content, type, KEntryAttNormal );
iDocHandler->GetPath( tempFileNamePtr );
CleanupStack::Pop(tempFileName);
```

Now the `tempFileName` contains the path of the saved file.

3.4 Moving and copying files

Certain file types are meant to be in certain folders. With S60 3rd Edition and data caging, the Document Handler API's file moving and copying mechanisms are very useful. You do not have to worry about where to save certain files, such as images or sound files. The Document Handler API finds the correct place for known file types.

The following example shows how to copy an image file:

```
_LIT(KDestFile,"destination"); //no extension required
_LIT(KSrcFile,"c:\\data\\picture.jpg");

TDataType nullType; //MIME type not set
TInt ret = KErNone;

TRAPD(err, ret = iDocHandler->CopyL(KSrcFile,
```

```

        KDestFile,nullType, KEntryAttNormal) );

if( err || ret)
{
    //handle error
}

//Get the destination path
TFileName path;
iDocHandler->GetPath(path);
ShowMessageL(path);

```

During the copying operation, the Document Handler API recognizes that the file is an image and copies it to the Gallery (c:\Data\Images directory in S60 3rd Edition). A note about adding an image to the gallery is shown to the user.

A file can be moved to another location by using `MoveL`. It takes four parameters:

- Original file path.
- Destination.
- MIME type. If not set, the file is recognized.
- File attributes, such as read or write. The file attribute constants (like `KEntryAttNormal`) are defined in `f32file.h`.

The code for moving files is similar to the code for copying:

```

TRAPD(err, ret = iDocHandler->MoveL(KSrcFile, KDestFile,nullType,
KEntryAttNormal) );

```

If the destination file already exists, a dialog "Name destination already in use. Replace existing item of same name?" is displayed. If the answer is "No," the application asks for a new file name.

If there is no need to ask the user for a new file name or to inform him or her about the file copy operation, the `SilentMoveL` function can be used:

```

TRAPD(err, ret = iDocHandler->SilentMoveL(KSrcFile,
KDestFile,PathInfo::PhoneMemoryRootPath(),nullType, KEntryAttNormal)
);

```

The `SilentMoveL` function takes an additional parameter which defines the root path of the selected memory where file should be moved.

When a file is moved or copied and its MIME type is not one of the following, the default destination is a temporary folder:

- "image/vnd.nokia.ota-bitmap"
- "image/x-ota-bitmap"
- "application/vnd.nokia.ringing-tone"
- "audio/*"
- "image/*"
- "video/*"
- "application/vnd.rn-realmedia"
- "application/sdp"

This causes the file to be deleted when the used `CDocumentHandler` object is deleted. This can be demonstrated by modifying the [S60 Platform: Document Handler Example \[3\]](#). Thus using the copy/move methods from `CDocumentHandler` is not recommended for other content types than those listed above. As a workaround, the `PathInfo` class has a method called `OthersPath()` that returns the paths that may be used for generic content.



Caution: When copying files, the success of the copying operation depends on the MIME type. Even if the MIME type is recognized, it may be copied to a temporary directory and deleted afterwards.

3.5 Query functions

The Document Handler API contains functions for querying system capabilities concerning file handling. The `CanHandleL` and `CanOpenL` functions are used when trying to find out if a certain MIME type is supported. `CanSaveL` returns a value indicating whether content with a certain MIME type can be saved by the system.

```

_LIT8(KAudioBasicMimeType, "audio/basic");
TDataType audioDataType(KAudioBasicMimeType);
TBool result = EFalse;
TRAPD(err, result = iDocHandler->CanOpenL(audioDataType) );
//TRAPD(err, result = iDocHandler->CanSaveL(audioDataType) );
//TRAPD(err, result = iDocHandler->CanHandleL(audioDataType) );

if( err == KMimeNotSupported )
{
    //MIME type is not supported
}
else if( result && !err )
{
    //yes we can handle basic audio
}
else if( !result && !err )
{
    //we can't handle basic audio
}
else
{
    //some other error
}

```

If the MIME type is not supported by the system, a leave occurs with `KMimeNotSupported`.

3.6 Changes in S60 3rd Edition

The Document Handler API has faced several changes in S60 3rd Edition. Actually, even the creation of the `CDocumentHandler` object should be done in a different way when compared to older SDKs. Many functions which take the filename as a parameter were deprecated in S60 2nd Edition, FP3 and have been removed in S60 3rd Edition. Caller applications should switch their code to use Document Handler API functions that take file handles (`RFile`) instead of file names as parameters. The Document Handler API offers utility methods for this purpose. The `SaveTempFileL` function can be used to create a new temporary file and open a sharable file handle for it. It provides a convenient way for displaying temporary data which is not meant to be stored permanently. The created temporary file will be deleted in the destructor of `CDocumentHandler`, but the caller application can also delete it. The `OpenTempFileL` function can be used to open a file handle using existing files. Note that the caller

application is responsible for closing the handle. It is also possible to query the temporary file name using the `GetPath` function.

The following example shows how to modify Document Handler API calls:

Old code used to launch a file using `OpenBufferEmbeddedL`:

```
docHandler->OpenBufferEmbeddedL(data, dataType, aFileName, NULL);
```

In S60 3rd Edition this can be converted to the following form:

```
_LIT8(KTextMimeType, "text/");
TDataType textType(KTextMimeType);
RFile filehandle;
_LIT8(KTestData, "TestData");
_LIT(KFileName, "tempFile");
iDocHandler->SaveTempFileL(KTestData, textType,
KFileName, filehandle);
iDocHandler->OpenFileEmbeddedL(filehandle, textType);
filehandle.Close();
```

The example code created a temporary file named "tempFile" with the content of `KTestData`. After this, the file is opened in the handler application (which is a notes application for the used MIME type) and finally the file handle is closed.

For details of the changes, see Appendix A, "Changes in the Document Handler API."

3.7 Observing handler application exit

Applications interested in the lifetime of their embedded applications can register themselves to receive a notification of the embedded application exit.

In S60 3rd Edition a new server application framework is used for this purpose. The observer class should derive from `MAknServerAppExitObserver` and implement the method `HandleServerAppExit`. Applications must also register themselves to take notifications using the `CDocumentHandler::SetExitObserver` function.

S60 1st and 2nd Edition applications wanting to receive notifications about embedded application exits derive their observer class from `MApaEmbeddedDocObserver`, implement the method `NotifyExit`, and register themselves as observers using the `CDocumentHandler::SetExitObserver` function.

The following example shows how to use the `NotifyExit` function without the application-specific document handling implementations:

```
void CTestAppAppUi::NotifyExit(TExitMode aMode)
{
    switch (aMode)
    {
        case EKeepChanges:
            {
                // Changes to the embedded document must be saved.
            }
            break;
        case ERevertToSaved:
            {
                // Revert to the saved version of the document
            }
            break;
        case ENoChanges:
            {
                // No changes have been made to the embedded document.
            }
            break;
    }
}
```

```

case EEmpty:
    {
        // The embedded document is empty.
    }
    break;
default:
    //Unknown mode"
    break;
}

```

In S60 3rd Edition the `HandleServerAppExit` function can be implemented in a similar manner.



Note: The exit mechanism only works for embedded handler applications.

3.8 DRM protection

S60 platform provides OMA Digital Rights Management (DRM) technology to enable authorization of content users and to prevent unauthorized modification of content. If DRM-protected files with restricted rights are opened, there will be a query asking if the user wants to open the file or not.

3.9 Parameter passing

The parameter passing mechanism from the caller application to the document handler (populating them and reading them in a handler application) is currently not supported by S60 SDKs, because the `CGenericParamList` and `CAiwGenericParamList` classes are not public.

3.10 Error situations and tips and tricks

The Document Handler API uses a leave mechanism to indicate error situations. For example, if the MIME type of the file is not recognized or the Document Handler API cannot find a handler application for the recognized MIME type, the Document Handler API causes a leave which should be handled by the caller application. Also other possible errors (for example, DRM rights issues or unexpected system errors) are indicated with a leave.

One of the most common errors is `KErrInUse` (-14). This occurs especially while trying to open a file in write mode. This is because the Document Handler framework opens the file in shareable read mode and shares the session with the handler application. Therefore, it is not possible for the application to open the file in write mode. In order to delete the original file and rename the temporary file, the handler application must wait for the `OpenFileL()` method to return. It is not possible to delete or manipulate the file within `OpenFileL()`.

3.10.1 Updating document file for standalone handlers

If the handler application is already running, the document file is not updated automatically. There is a special technique for this issue. The application UI class has a `void OpenFileL(const TDesC& aFileName)` function which is called by the launcher application if the handler application is already running. It is up to the launcher application to do this. See the [S60 Platform: Document Handler Example](#) [3] for implementation details. The technique involves the following steps:

1. Check if the handler application is running. If you don't know the handler application's UID, you can use `CDocumentHandler` to get it. This is discussed in Section 3.5.
2. If the handler application is running, update the document file by calling `TApaTask` class's `SwitchOpenFile(const TDesC& aFileName)` function.



Tip: If you are using a standalone handler and the handler application is already running, update the document file as shown in the [S60 Platform: Document Handler Example](#) [3].

Note that `TApaTaskList` can only be used for searching GUI applications. For non-GUI applications, `TFindProcess` should be used. Use the `ExitCategory` function to check whether the application has actually terminated or not. A zero-length category means that the application is still running.

3.10.2 Changing the file content

When a file is opened, the following function of the handler application is called:

```
CAknDocument::OpenFileL(CFileStore*& aFileStore, RFile& aFile)
```

Here `aFile` is opened in read mode. It is not possible for the client application to make changes to the file directly. The solution for this is to open a new temporary file with the content of the file, edit the temporary file, and replace it with the original file. Another way of doing this is to extract the file name and delegate the processing to a separate engine or the application UI class:

```
void CHelloWorldBasicDocument::OpenFileL(
    CFileStore*& aFileStore, RFile& aFile)
{
    aFileStore=NULL;
    TFileName iFileName;
    aFile.FullName(iFileName);
    Process()->SetMainDocFileName(iFileName);
    TInt fileMode = EFileWrite|EFileShareExclusive;
    SetAppFileMode(fileMode);
    aFile.Close();
    iAppUi->LoadFile(iFileName);
}
```

The problem with an already open file handle does not occur in S60 1st or 2nd Edition. The difference between S60 2nd Edition and S60 3rd Edition is that in S60 2nd Edition the `OpenFileL` function is called with the file name instead of the handle:

```
CAknDocument::OpenFileL(TBool aDoOpen, const TDesc&
    aFileName, RFs& aFs)
```

If the file that is being opened is in a private folder or there is no need to open it in write mode, a duplicate of the file handle can be maintained for future use.

3.10.3 Determining the handler application

The MIME type can be used to figure out which applications can handle it. The `CDocumentHandler` class has a convenient `HandlerAppUID` function which returns the handler application UID for a certain file (or to be more specific, the MIME type of the file). Unfortunately, the function is declared only in S60 3rd Edition. In earlier editions `RApaLsSession` can be used to find out the handler application.

```

#include <apmstd.h> //TDataType, apmime.lib
#include <DocumentHandler.h> //CDocumentHandler, commonui.lib
#include <apgcli.h> //RApaLsSession, apgrfx.lib
...
    LIT(KTestFile, "C:\\test.txt");
    //In real life don't hard code paths!
    TDataType empty = TDataType();
    //Open a file
    iDocHandler->OpenFileL(KTest,empty );
    TUid handlerUid;
    TInt err = KErrNone;
    #ifdef __SERIES60_3X__ //If we're in S60 3rd Edition
        err = iDocHandler->HandlerAppUid(handlerUid);
    #else //Otherwise use RApaLsSession
        RApaLsSession apaLs;
        User::LeaveIfError( apaLs.Connect() );
        err = apaLs.AppForDocument(path, handlerUid, empty);
        apaLs.Close();
    #endif
//now handlerUid contains the UID of the handler application

```

The UID can be used, for example, to update the document file once the handler application is running. See Section 3.10.1 for an example.

4. Terms and abbreviations

Term or abbreviation	Meaning
Application UID	Application unique identifier.
AVKON	UIKON extended GUI.
AVKON Open File Service	Service provided by AVKON. Used for launching a handler application embedded in opening a file.
Basic operation	The Save and Open operations implemented by the default handler in the standard Symbian OS way.
Caller (client) application	Application that uses the Document Handler API.
Common UI	UI-dependent common components library in S60 platform.
DRM	Digital Rights Management.
ECom	The ECom framework provides facilities to resolve and load the appropriate implementations at run time.
Embedded application	An embedded application works in the same window group as the application launching it. The user cannot switch between the embedded and original application.
Embedded launching	A service provided by another application is launched within the application the user was originally working with. An embedded application is chained to the same window group as the original application.
Handler application	An application responsible for certain MIME type(s) and thus able to handle file content.
MIME	Multi-Purpose Internet Mail Extensions
Recognizer	Recognizers are used for identifying data type. Once the data type has been identified, the data can be passed to the application that best handles it. Applications specify the MIME types they support. The plug-in recognizer architecture allows additional data recognizers to be created and added.
Standalone launching	A separate application that is launched to provide a service.
UID	See Application UID.

5. References

- [1] [KIS000588 - Data handler applications must implement CAknAppUi::OpenFileL](#) available at <http://www.wiki.forum.nokia.com>
- [2] [S60 Platform: Application Framework Handbook](#) available at [Forum Nokia](#).
- [3] [S60 Platform: Document Handler Example](#) available at [Forum Nokia](#)
- [4] Server application framework, a Symbian library paper available at <http://developer.symbian.com/main/tools/appcode/cpp/ServiceExplorerdemo.jsp>

Appendix A. Changes in the Document Handler API

Table 1, Table 2, and Table 3 list the changes that the Document Handler API has faced since S60 1st Edition. Note that the Document Handler API has been cleaned in S60 3rd Edition by removing deprecated methods. Take care not to use them when developing for older platform versions. Otherwise the application cannot be compiled for S60 3rd Edition and because of the binary break, the application cannot be run on S60 3rd Edition devices. Embedded file opening in the Document Handler API is changed to use the Server Application Framework (for details, see the Server application framework paper published by Symbian [4]).

S60 3rd Edition
<p>Added:</p> <pre>TInt CopyL(const RFile& aFileOld, const TDesC& aNameNew, TDataType& aDataType, const TUint aAttr); void CheckFileNameExtension(TDes& aFileName, const TDataType& aDatatype); void SetTempFile(const TDesC& aTempFile); TInt RecognizeAndCheckFileL(RFile& aFileHandle, TDataType& aDataType, TUid& aUid);</pre>
<p>Deprecated:</p> <pre>void ListSupportedMimeTypesL();</pre>
<p>Removed:</p> <p>OutParamList</p> <p>SharableRFS and SetSourceFile methods</p> <p>RecognizeAndCheckBufL</p> <p>Methods that were deprecated in S60 2nd Edition, FP3</p>

Table 1: Changes to the Document Handler API in S60 3rd Edition

S60 2nd Edition, Feature Pack 3**Added:**

Failure reason `KDRMErrPreviewRights`

`NewL` and `NewLC` which take no parameters. Marked the `CEikProcess` version to be deprecated in S60 3rd Edition.

```
void OpenTempFileL(const TDesC& aFileName, RFile &aSharableFile);
```

```
void SaveTempFileL(const TDesC8& aContent, TDataType& aDataType,
const TDesC& aFileName, RFile &aSharableFile);
```

```
CAiwGenericParamList& InParamListL();
```

```
const CAiwGenericParamList* OutParamList(); // Note: Does not work in S60
2nd Ed, FP3
```

```
TInt OpenFileL(RFile& aSharableFile, TDataType& aDataType);
```

```
TInt OpenFileEmbeddedL(RFile& aSharableFile, TDataType& aDataType,
const CAiwGenericParamList& aParamList);
```

```
TInt OpenFileEmbeddedL(RFile& aSharableFile, TDataType& aDataType);
```

```
TInt MoveL(const TDesC& aFileNameOld, const TDesC& aNameNew,
TDataType& aDataType, const TUint aAttr );
```

```
TInt SilentMoveL(const TDesC& aFileNameOld, const TDesC& aNameNew,
const TDesC& aRootPath, TDataType& aDataType, const TUint aAttr );
```

```
TInt HandlerAppUid( TUid& aUid );
```

```
void SetExitObserver( MAknServerAppExitObserver* aObserver );
```

```
void SetSourceFile( TFileName aSourceFile ); //marked as temporary
```

```
RFs& SharableRFs();
```

```
void CloseSharableFS();
```

Deprecated:

`OpenBufferL` method. Instead, save the content with `SaveTempL` and use `OpenFileL` to open that file.

`OpenFileXXX` functions which take the filename as a parameter. Use `RFile` versions of the functions instead.

`GetOutputParams` method

Methods that use the `MApaEmbeddedDocObserver` class

Table 2: Changes to the Document Handler API in S60 2nd Edition, Feature Pack 3

S60 2nd Edition, Feature Pack 1
<p>Added:</p> <p>Failure reasons such as <code>KDRMErrNoRights</code></p>
S60 2nd Edition
<p>Added:</p> <pre> TInt OpenBufferL(const TDesC8& aContent, TDataType& aDataType, const CGenericParamList& aParamList); TInt OpenBufferL(const TDesC8& aContent, TDataType& aDataType, const TDesC& aName, const TUint aAttr, const CGenericParamList& aParamList); TInt OpenFileL(const TDesC& aFileName, TDataType& aDataType, const CGenericParamList& aParamList); TInt OpenBufferEmbeddedL(const TDesC8& aContent, TDataType& aDataType, const CGenericParamList& aParamList); TInt OpenBufferEmbeddedL(const TDesC8& aContent, TDataType& aDataType, const TDesC& aName, const TUint aAttr, const CGenericParamList& aParamList); TInt OpenFileEmbeddedL(const TDesC& aFileName, TDataType& aDataType, const CGenericParamList& aParamList); const CGenericParamList* GetOutputParams(); </pre>

Table 3: Additions to the Document Handler API in S60 2nd Edition and S60 2nd Edition, Feature Pack 1

Evaluate this resource

Please spare a moment to help us improve documentation quality and recognize the resources you find most valuable, by [rating this resource](#).