ADOBE® READER MOBILE 9 SDK

User Manual

Adobe Systems Incorporated

Version 1.1
November 2, 2010

Contents

PREFACE

The ADOBE® Reader® Mobile SDK (hereafter referred to as *the SDK*) is a source code implementation of Adobe's eBook document engine that is made available to selected partners to enable them to build eBook devices and applications. The engine provides:

- Interpretation and rendering of PDF
- Interpretation and rendering of EPUB
- Support for Adobe Content Server 4 digital rights management (DRM)
- Support for integration with Adobe Digital Editions

The SDK is provided as source code; there are no binary libraries. The SDK is provided on an as-is basis. Adobe validates the SDK against two reference platforms (details below). Porting to your target platform environment is your responsibility – although Adobe is happy to provide support.

1 OVERVIEW

The SDK consists of two major pieces:

- Hobbes – This comprises the EPUB rendering facilities and DRM. It also provides the implementation on the embedding interface.
- Tetraphilia (T3) – This provides the PDF interpretation. It also provides all the low-level rendering, color management, and font handling.

All of this functionality is exposed through the *embedding interface*. The embedding interface effectively wraps all the functionality of the SDK, exposing the functionality through a set of well-defined interfaces. These interfaces are well-documented in the accompanying *Reader Mobile SDK API Specification*.

The following diagram shows a schematic representation of the SDK. Note that the embedding interface consists of two parts. There are interfaces exposed on Hobbes that are implemented in Hobbes. These include:

- Input sources
- Rendering
- Navigation
- Network Requests
- DRM
- Search
- Table of Contents support

On the host side, the host must also implement the host side of the embedding interfaces. These include:

- Raster support
- DRM Host
- Event Info
- Error Reporting

See the section "Required Host Implementations" for details about these interfaces. Also, these interfaces are covered in greater detail in the API specification.

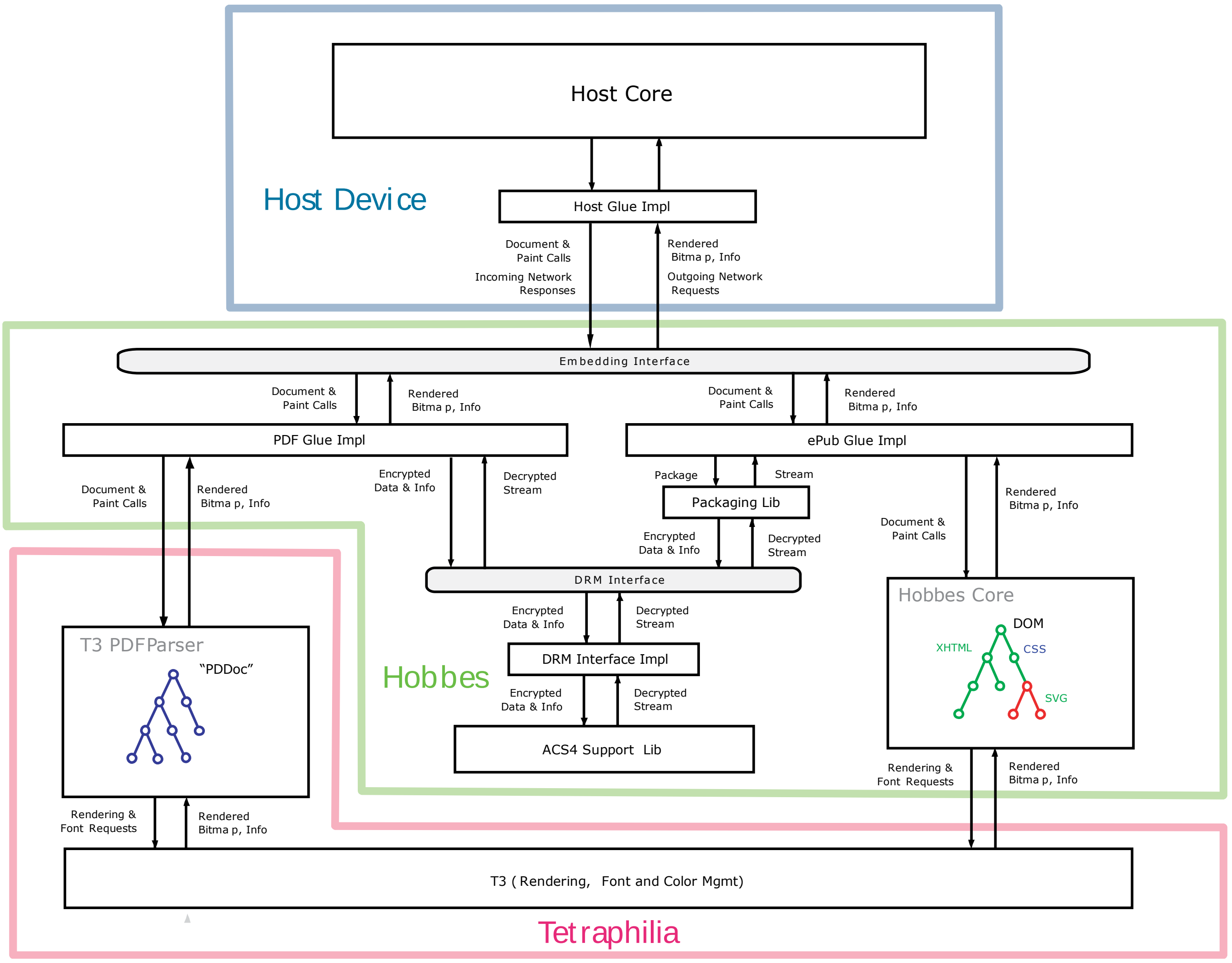

Figure 1. Schematic diagram of the Reader Mobile SDK

2 DOCUMENT FORMAT SUPPORT

2.1 PDF

PDF support in SDK 3 is roughly equivalent to PDF/A though it does not, in fact, adhere to the letter of that specification. Virtually all of PDF 1.6 is supported with the following exceptions:

- All JavaScript interactivity
- Forms
- Password-based security
- JPEG2000
- Nonseparable blend modes
- Knockout groups
- All annotations except links (which are supported)
- QuickTime and SWF multimedia annotations
- Some relatively esoteric features (for example, “figures”)

2.2 EPUB

EPUB is supported as specified in the OPS 2.0 specification with only a few minor exceptions.

3 SYSTEM REQUIREMENTS

The SDK is delivered as source code and the process of porting is your responsibility. Therefore, there are no system requirements, strictly speaking. Nonetheless, this section details the recommended minimum requirements. Note that these guidelines apply to typical PDF and EPUB files. Of course, PDF or EPUB files can be created that are so complex or large that they will not perform or render well, but these guidelines are true for “typical” files.

3.1 PROCESSOR

Adobe’s experience is that an ARM 9 processor (or equivalent) running at 200 MHz is the minimum amount of computing power that will lead to a satisfactory user experience. A 400 MHz processor (or higher) is recommended – especially if the display is larger than 600x800 pixels. The presence of a floating-point unit (FPU) will help somewhat, but most operations of the SDK are not limited by floating-point performance so while a FPU won’t hurt it doesn’t make a huge difference. The fixed-point operations of the SDK are relatively efficient.

3.2 DISPLAY

The SDK supports 8-bit grayscale and 8-bit per channel RGB. It is the responsibility of the host to transcode those representations into the native raster format, if it is different.

3.3 MEMORY

The runtime memory requirements for the SDK vary significantly depending on the size and complexity of the current page being rendered. The SDK keeps only one page in memory at a time. Typically, 10 MB of RAM are sufficient to render most any PDF or EPUB page.

Due to the stream-like nature of PDF rendering, almost any PDF page can be rendered (leaving aside very large images). However, EPUB documents are basically XML, consisting of one or more chapters where each chapter is a single XHTML file. Being XML, it is required that each chapter must be loaded entirely into memory. A consequence of this is that EPUB chapters above a certain size will lead to errors. Currently, the SDK is configured to generate an error for single XHTML chapters above 300 KB or single images above 10 MB. But this figure can be modified if the device manufacturer feels that there will be sufficient memory available to render larger documents.

4 REFERENCE PLATFORMS

Although Adobe builds and tests the SDK on a number of platforms, the SDK is built and validated against only two reference platforms, a Linux platform and Windows Mobile 6. The general process (which is fully automated) is as follows:

- The SDK (zip file) is created using a server-side script that extracts all the sources from our source code control and creates the zip archive.
- The resulting archive is copied to the reference systems. The archive is then unpacked.
- A test application (book2png, see section 9, below) is built from the unpacked sources.
- The test application is run, the results are checked, and reports are generated.

4.1 LINUX

The reference platform is a stand-alone Linux system, a Technologic Systems TS-7800 that has a 500 MHz ARM9 CPU (Marvell MV88F5182). The OS is a Debian distribution of Linux, kernel version 2.6.21. The tool chain is a CodeSourcery G++ Lite 2008 q3-41 compiler and linker, which is based on the gnu GCC 4.3.2.

4.2 WINDOWS MOBILE 6

The reference platform for Windows Mobile is a Hewlett Packard iPaq 111, which has an Intel XScale PXA310 ARM-compatible CPU running at 624 MHz. The OS is Windows Mobile 6. The executable is built with the Microsoft Visual Studio 2008 tool chain.

4.3 MACINTOSH

Adobe does not currently have a reference build for the iPhone, but we do build and validate on the Macintosh Intel desktop using the XCode 3.1 tool chain. XCode 3.2.3 and above is known not to work with the RMSDK projects as provided.

5 DELIVERABLES

The SDK is delivered as source code and documentation in a single zip file. The zip file contains:

Content	File	Comment
User Manual	RMSDK_User_Manual.pdf	This document.
API Specification	RMSDK_API_Specification.zip	Automatically generated API documentation of the embedding interface.
Release Notes	RMSDK_Release_Notes.pdf	Current notes and data on the current release of the SDK.
Source Code	In the root of the zip file	The actual source code.

6 INSTALLATION AND BUILDS

To install the SDK, you need only to unzip the contents of the zip file to an appropriate sandbox. The build files for the reference platforms are located in the root of the sources in the build folder.

Note: While our internal validation builds uses OpenSSL, we are legally precluded from providing OpenSSL in source form. You will need to provide your own cryptographic library; however, if you use a library other than OpenSSL, you will need to update the cryptographic wrapper routines in the crypt (wrapper) library.

The following sections explain how Adobe builds OpenSSL on our reference platforms. Your platform and tool chain may be different, so you may need to modify these processes.

Here are a couple of notes on the build process for the SDK. First, part of Hobbes is built in a rather unconventional way. The "source" is actually XML, which is compiled using a Java tool named Velocity, which generates the C++ code, which is then compiled into binary code. For this reason, the SDK includes an entire Java Development Kit (JDK). This simplifies the build process by not having dependency on a correctly configured external JDK.

Second, the source code for the Tetraphilia section of the SDK is heavily templatized. It does not use STL, but does depend on the compiler properly supporting C++ templates. The sources have been successfully built on many different compilers, so your tool chain should work fine. However, it is an area to watch for if there are build problems. Further, one of the consequences of the extensive use of templates is that certain modules take quite a while to build. You should endeavor to build with a cross-compiler on a desktop PC if at all possible.

6.1 CONFIGURING MEMORY USAGE FOR PDF

The amount of memory used for PDF can be configured as follows. In `t3/config/T3AppTraits.h` around line 151, you can change the values passed to `setMemoryThesholds`. The parameters have the following meanings:

- The first parameter sets the amount of PDF memory usage that, when exceeded, triggers reclaiming memory from the cache.
- The second parameter sets the amount reclaimed when needed.
- The third parameter is the absolute memory limit. An error occurs if the memory usage exceeds the third value.

6.2 CONFIGURING MEMORY USAGE FOR EPUB

Memory configuration in EPUB is controlled by two sets of parameters. One is the limit on the size of an individual chapter, that is, a single XHTML file with the zip package. These limits are set via the following files:

- `config.xcconfig` on Mac OS
- `config.build` on Linux
- `windows_config.rsp` on Win32
- `windows_mobile_config.rsp` on Windows Mobile

In each case, the config file is in the appropriate build folder.

The `MAX_COMPRESSED_CHAPTER_SIZE` and `MAX_UNCOMPRESSED_CHAPTER_SIZE` are used to set the maximum sizes for the compressed and uncompressed sizes of the chapter, respectively. If `STOP_LOAD_ON_CHAPTER_TOO_LONG` is defined, the SDK will stop loading the chapter after the `MAX_UNCOMPRESSED_CHAPTER_SIZE` limit is reached. Otherwise, the SDK will issue the warning but continue to try loading the chapter.

Image size is limited in a similar way, although the define that limits it is not yet pulled out into a config file. There is a define `MAX_IMAGE_SIZE` in `xml/uft/src/uft_image.cpp` in the constructor for `BitMapImage`. For Linux builds only, the default `MAX_IMAGE_SIZE` is set to 10 MB.

6.3 CONFIGURING COLOR MANAGEMENT

First, you must decide what type of color management is desired. The choices are managed gray, unmanaged gray, managed color, and unmanaged color. The choice of gray or color should be determined by the targeted display. Use one of the managed choices if more accurate support for ICC color profiles is desired, even though it is slower. Unmanaged is faster under most circumstances.

The default setup is optimized for Linux devices with grey-scale devices and Windows Mobile devices with color displays. However, you may find that the defaults are not optimal for your setup. If you have any questions, contact readermobile_support@adobe.com

Second, pick the color traits that are needed based on the platform and color management selection. The choices are:

Linux:

- DefaultManagedGrayColorTraits
- UnmanagedGrayColorTraits
- DefaultManagedColorTraits
- UnmanagedColorTraits

Windows or Windows Mobile:

- DefaultManagedGrayColorTraits
- UnmanagedGrayColorTraits
- WinManagedColorTraits
- UnmanagedColorTraits

Mac OS X:

- DefaultManagedGrayColorTraits
- UnmanagedGrayColorTraits
- MacOSXManagedColorTraits
- UnmanagedColorTraits

Third, find the place in `t3/config/T3AppTraits.h` where `color_traits` is typedef'ed for the system and change it to the color traits selected in the second step.

Finally, be sure the necessary color context file is included at the top of `T3AppTraits`. The file needed is:

DefaultManagedGrayColorTraits and UnmanagedGrayColorTraits

- tetraphilia/color/DefaultGrayColorTraits.hpp

DefaultManagedColorTraits

- tetraphilia/color/DefaultManagedColorTraits.hpp

UnmanagedColorTraits

- tetraphilia/color/UnmanagedColorTraits.hpp

WinManagedColorTraits

- tetraphilia/color/win/WindowsManagedColorTraits.hpp

MacOSXManagedColorTraits

- tetraphilia/color/macosx/MacOSXColorTraits.hpp

6.4 FONT SUPPORT

The SDK has internal font support using what are called Multimaster fonts if the document does not embed its own font. These fonts provide simulation of serif and sans-serif fonts as well as symbols. However, the document author can (and generally should) embed their own font so they get just the look they want. For PDF, the fonts are almost always embedded (unless they are the very large CJK fonts). For EPUB, the SDK supports both TrueType and OpenType (the former is a subset of the latter). However, note that if the font is not embedded, the SDK supports only Western Roman languages (that is, not Russian, Polish, etc.) and CJK (unless this support has been disabled – see below).

By default, the default makefiles and project files enable CJK support. To disable CJK support, remove the defines `SUPPORT_JAPANESE`, `SUPPORT_SIMPLIFIED_CHINESE`, `SUPPORT_TRADITIONAL_CHINESE`, and `SUPPORT_KOREAN` from the configuration file. These are independent and can be enabled and disabled in any combination.

The configuration file also contains the names of the fonts used for CJK. If no font is available for a language, remove the corresponding `SUPPORT_<language name>` define. If only one font is available for a language, both fonts for the language can be set to the same name. A set of fonts for all four languages is included in the Linux build. Removing unneeded `SUPPORT_<language name>` defines can reduce the size of the Linux executable considerably. On Mac OS, Windows, and Windows Mobile, system fonts are used. If the named fonts for a language are not available and cannot be available on one of these systems, remove the `SUPPORT_<language name>` define for that language to avoid unnecessary attempts to access the font.

The name of the configuration file extension and location varies with the platform:

- Mac OS – `build/mac/config/config.xcconfig`
- Linux – `build/make/config.build`
- Windows Mobile – `build/vc9/windows_mobile_config.rsp`
- Windows – `build/vc9/windows_config.rsp`

There are two ways in which the RMSDK obtains CJK fonts depending on whether `USE_NATIVE_FONTS` is defined in the configuration file. `USE_NATIVE_FONTS` is defined by default for Windows, Windows Mobile, and Mac OS. `USE_NATIVE_FONTS` is not supported for Linux. When `USE_NATIVE_FONTS` is defined, the RMSDK uses the operating system to obtain and render CJK fonts. In this case, a different font can be used by simply placing the font in a directory where the OS will find it and changing the name of the font in the config file.

If `USE_NATIVE_FONTS` is not defined, any required CJK fonts must be compiled and linked with the RMSDK. If the font is a binary file, a simple console application is provided in source form to convert it into a `.h` file. This involves the following files:

- `mkhex.c` in the `fonts/build` directory, which is the conversion application.
- `mkfonts.bat` in the same directory is a Windows batch file that Adobe used with this application to create the default CJK fonts and is a good example of how to use this application.
- `copyright.txt` in the same directory contains the copyright information for the font. Change this file to the appropriate copyright for the font or fonts being converted.

After the `.h` file is created, create a simple `.c` file to include the `.h` file. This `.c` file must define symbols `<font_name>` and `<font_name>_Length`. Refer to `AdobeHeitiStd-Regular.cpp` in directory `hobbes/fonts/fonts/AdobeHeitiStd` as an example of this type of file.

Note: It is best to use the `.c` suffix for this type of file even though Adobe used `.cpp` in RMSDK. These files are simple enough to compile as either `.c` or `.cpp`; they take less memory during compilation when compiled as `.c`.

Edit the config file to change the `<language>_<style>_FONT` and `<language>_<style>_FONT_LENGTH` entries to the symbols defined in the new `.c` file. Then the new file can be compiled and linked with the RMSDK.

7 REQUIRED HOST IMPLEMENTATIONS

The following interfaces must be implemented on the host side in order for the device to be functional.

7.1 DOCUMENTCLIENT

This interface supplies callbacks that a Document instance can use to load itself from the stream, report loading completion and/or errors, request background processing, and request a DRM license (for DRM-protected documents).

7.2 RENDERERCLIENT

This interface supplies callbacks that a Renderer instance relies on. It is notified when navigation happens (for example, a link is clicked), when a part of the screen needs to be repainted, when a mouse cursor needs to be changed, and so on. It is also notified when errors occur.

7.3 SURFACE

This is an interface that represents an off-screen bitmap to which painting can be done.

7.4 STREAM

This represents a raw document file opened for random access.

7.5 EVENT SUBCLASSES

These are needed to pass mouse and navigational events to the Renderer.

7.6 DEVICE AND DEVICEPROVIDER

These are not required for Windows Mobile or desktop operating systems, where the Adobe-supplied implementation must be used. These interfaces represent a DRM-capable device. In most cases you'll have only one type of the device (representing the device on which the code is running) and an iterator (or provider) that returns only that one device.

`DeviceProvider` manages access to the device information.

`Device` provides access to hardware-based device fingerprint information and storage for the device activation. Any implementation of a custom device interface will be reviewed and approved by Adobe during the required Certification process to ensure that the DRM system works consistently.

7.7 DRMPROCESSORCLIENT

This is required only if doing network-enabled activation and fulfillment. This interface provides network access and callbacks for DRM system processes (device activation and content fulfillment).

7.8 OUTPUTSTREAM

This is required only if doing network-enabled fulfillment. This interface represents the write-only file stream that is used to write the document file with the DRM license embedded in it.

8 DRM SUPPORT

8.1 ACTIVATION SUPPORT

The simplest interface that a tethered eBook device can expose is a USB Mass Storage interface. This does not require the device manufacturer to write any drivers and it works on most operating systems out of the box. This interface is sufficient for Adobe Digital Editions to transfer books to and from the device, but by itself it is not sufficient for the device activation. This section describes how the SDK supports device identification and activation.

8.1.1 SECURITY OVERVIEW

A note on security: The SDK treats all communications through the file system as unsecure. The SDK cannot know who is talking to it from the other side of the connection. Therefore, Digital Editions cannot trust fingerprints from such devices. So, if Digital Editions and the device are licensed to different users, DRM-protected books stored on the device cannot be read directly in Digital Editions. If the device and the instance have been authorized by the same user, then books on the device can be read directly in Digital Editions (as well as being transferrable back and forth).

The requirements for activation/authorization are that it supports:

- Devices that can only read/write to the file system
- Dual-mode devices that can work both in pure mass-storage mode and through custom drivers

Important Note:

To communicate through a shared file system, only one party can be allowed to modify the activation.xml at the same time. Fortunately, activation is a brief, easy process, so it should not be a problem in most cases.

8.1.2 MASS STORAGE MODE

Devices that wish to expose themselves to Digital Editions through mass storage must create a folder named "`.adobe-digital-editions`" and mark this folder as hidden. Communication happens through two files in that folder:

- "`device.xml`" is created by the device (that is, the host implementation). The `device.xml` file is automatically read by Digital Editions running on a PC when the device is detected. Upon reading the `device.xml` file, Digital Editions will automatically create the file "`activation.xml`".
- `activation.xml` can be read by both the device and Digital Editions.

Both of these files should be created with temporary names first (`device-tmp.xml` and `activation-tmp.xml` are recommended) and then renamed, so that whoever is reading the files never sees an incompletely written file.

If Digital Editions detects the `.adobe-digital-editions` folder, it reads `device.xml` and, if needed, creates the `activation.xml` file. The device then appears in the Digital Editions Library pane.

8.1.3 ACTIVATION.XML FILE

The `activation.xml` follows the format for the activation record of Content Server 4. A typical `activation.xml` would look like this:

```
<activationInfo xmlns="http://ns.adobe.com/adept">

  <activationToken >
    <device>deviceid</device>
    <fingerprint>base64-encoded 20-byte fingerprint</fingerprint>
    <deviceType>deviceType</deviceType>
    <activationURL>activation service URL</activationURL>
    <user>userid</user>
    <until>activation expiration time</until>
    <signature>base64-encoded signature</signature>
  </activationToken>

  ...

</activationInfo>
```

8.1.4 DEVICE.XML FILE

The `device.xml` file looks like this:

```
<deviceInfo xmlns="http://ns.adobe.com/adept">
    <deviceClass>Manufacturer + device code/name (e.g. Acme 3000)</deviceClass>
    <deviceSerial>device serial number</deviceSerial>
    <deviceName>User-friendly name of the device</deviceName>
    <deviceType>tethered or mobile</deviceType>
    <version name="hobbes" value="X.Y.ZZZ"/>
    <fingerprint>base64-encoded fingerprint</fingerprint>
</deviceInfo>
```

- The `deviceClass` field should uniquely identify the manufacturer and the model number. When combined with the `deviceSerial` (serial number), it should uniquely identify a single, specific device.
- The `deviceName` is the label that Digital Editions will show in the Library pane.
- The `deviceType` should be either tethered or mobile. The version value, `X.Y.ZZZ,` is a concatenation of version strings that can be read from the SDK using `embed::getVersionInfo` with "`hobbes.major`" (`X`), "`hobbes.minor`" (`Y`), and "`hobbes.build`" (`ZZZ`) as the arguments to `getVersionInfo`.

When a user activates this device (via the host calls to `DRMProcess`), ADE writes a new activation file with an updated activation record to the `activation.xml` file on the device.

One of the key pieces in the `device.xml` is the fingerprint, which you generate from the unique ID of the device and a random number (called the “salt”). See section 8.2, “Implementation of getFingerPrint,” for details about generating the salt and the fingerprint.

The device will need to recreate device.xml if the user somehow deletes it or it is otherwise damaged. Thus, the random-number “salt” and any other invariant information (for example, the unique ID) need to be stored somewhere outside that hidden folder. Then, when the system boots, it should recreate the device.xml file, which needs to be exactly the same each time so the user can reaccess their books. Following these steps means that, even if the user mangles the device.xml, it doesn’t matter, although the user will also need to reactivate the device.

Also note that the `device.xml` should be checked each time the device is connected to the USB connection (if it is tethered), or is wirelessly connected (if it is mobile).

Finally, note that Digital Editions automatically notes the `device.xml` (if it is in the correct folder on your device, as noted above). But if the device is a tethered device and Digital Editions activates the device, the device will NOT be activated properly for nontethered (that is, wireless) use. To use the device for wireless fulfillment, it MUST be activated via wireless, not by Digital Editions (for security reasons).

Device behavior when a device is activated multiple times should also be noted. When the user activates the same device with more than one AdobeID, the device ID associated with each user is not the same. This also means that, if a user reactivates the same device with the same AdobeID, they wouldn't be able to read a book that has restrictions to a single device.

8.2 IMPLEMENTATION OF GETFINGERPRINT

The DRM system in the Reader Mobile SDK is very important in that it provides the security for the publisher and booksellers. It is of paramount importance that the implementation be 100% secure. One must unfortunately assume that some users will maliciously try to circumvent the system. This can be prevented, but it requires that the device-software developer follow some relatively simple rules:

- The device must provide a unique ID. Ideally, this should be a hardware serial code, but if it is contained in a file, this file should absolutely NOT be accessible by the user.
- If at all possible, the `device.xml` and `activation.xml` files should also be hidden from the user.

This unique ID is used (in part) to generate the "device fingerprint". This is used to validate that the device may open the documents that have been fulfilled to the device. This validation process works as follows:

- When the device is first authorized, the device fingerprint is fetched and passed, along with the user's ID and password, to the Adobe-hosted activation server.
- The server validates the user and passes back the user's credentials that are used to communicate with the fulfillment server when the user is getting books. It also passes back the activation record for the device, which contains the user's ID and device fingerprint and is digitally signed. Since the activation record is digitally signed, it can be checked for validity by the SDK. It the activation record has been tampered with, the activation is rejected.
- When the user obtains a book through Content Server fulfillment to the device, the SDK writes the license into the book. The license contains the user's ID and is also digitally signed.
- When the user tries to open the book, the SDK requests the fingerprint from the host implementation of `getDeviceInfo`. The device must then return the fingerprint, which is guaranteed to be secure and untampered with.
- The SDK then compares the fingerprint to the fingerprint in the activation record for the device. If they do not match, the SDK will refuse to open the book, since it will appear that someone has tampered with the device.

For these reasons it is critical that the fingerprint be generated on the fly by the device using a secure, reproducible method. Note that the fingerprint is used to activate the device initially, so, for the device to usable, the fingerprint must be absolutely the same until the device is de-authorized. Therefore, to generate the fingerprint:

1. Generate the "salt" (6-8 bytes). (Or read it from the saved salt file if the file already exists.)
   The best method for generating the salt is to generate a random number. Be sure to seed the random number generator with something different each time. Otherwise, the random number

salt will be the same each time. On Linux, use `/dev/random` or `/dev/urandom;` on Windows, use `CryptGenRandom()`.

2. Save the salt into a file.

   Important: The host must save the salt because something might happen to the device.xml and that fingerprint is *required* in order to open the user's books on your device.
3. Implement the fingerprint calculation code. Read the hardware serial number and salt and combine them into a 20-byte maximum fingerprint. This has to be done so that it is not possible to come up with the desired fingerprint by manipulating the salt file. The best method to achieve this is to concatenate the serial number and the salt in a buffer and then run a SHA1 hash on it.
4. Take the first 20 bytes of the hashed values as the fingerprint. Adobe's activation server cannot currently handle fingerprints longer than 20 bytes, and will return activation errors for longer fingerprints.
5. Store the fingerprint in a static variable. That variable must always be initialized by running fingerprint calculation code and *never* by reading it from a file. Note that this means that you must regenerate the fingerprint every time the device is rebooted.
6. Implement `DeviceIterator` and `DeviceInfo` in your own code. Do NOT use the `emh::TestDeviceIteratorImpl.getFingerprint` method. Your implementation should return the fingerprint from the static variable that was initialized on step 4.
7. It is perfectly acceptable to read activation information from the `.adobe-digital-editions/activation.xml` file.
8. When generating the `device.xml` file, read the fingerprint from that same static variable that was initialized in step 4.

8.2.1 DEACTIVATING A DEVICE

Note that when the privacy-conscious user wishes to dispose of the device (for example, they are selling it), they should deactivate the device. This should be done through a hardware-specific hard reset. This hard reset should destroy the salt file and entirely remove the .adobe-digital-editions folder. This will guarantee that any new fingerprint is different.

8.3 LOANS

Both Digital Editions and the RMSDK support expiring content (that is, loans). Loans are just like any other document except that they are checked each time they are opened to ensure that the document has not expired. In addition, there is an additional section in the `activation.xml` that pertains to the loan. This section is known as the `loanToken`. Here is what an `activation.xml` on a device looks like with a single loaned book.

```
<activationInfo xmlns="http://ns.adobe.com/adept">

  <activationToken >
    <device>deviceid</device>
    <fingerprint>base64-encoded 20-byte fingerprint</fingerprint>
    <deviceType>deviceType</deviceType>
    <activationURL>activation service URL</activationURL>
    <user>userid</user>
    <until>activation expiration time</until>
    <signature>base64-encoded signature</signature>
  </activationToken>

  ...
  <loanToken>
    <time>2009-06-10T11:59:43-07:00</time>
    <user>urn:uuid:c09d8d88-1498-4ced-9d09-6068d512441a</user>
    <operatorURL>http://london.corp.adobe.com/fulfillment</operatorURL>
    <licenseURL>https://nasigningservice.adobe.com/licensesign</licenseURL>
    <loan>b843f5fc-71a1-478f-9c41-a6a24f5055c6-00000000</loan>
    <signature>yGU80Ci9 ...  NRrznSbc=</signature>
  </loanToken>

</activationInfo>
```

When the book is first borrowed from the library using Digital Editions, it receives the `loanToken`. When the book is transferred to a tethered device, Digital Editions automatically updates the `activation.xml` with the current loan tokens.

For access to the returnable loaned content, the activation info for the device should contain the `loanToken` with the loaned resource listed in it. The `loanToken` is issued by the same ACS4 server that fulfilled the loaned content. During fulfillment of a loan, `DRMProcessor` receives the updated `loanToken` from the ACS4 server. `DRMProcessor` then replaces the existing `loanToken` for that server (if any) in the device activation info with the newly received `loanTokens`. From the device perspective, during fulfillment, an `embed::DeviceInfo::setActivationInfo` call will be made to update the activation record with the current set of loan tokens for the user.

When the loan is returned, `loanToken` is also updated (using `embed::DeviceInfo::setActivationInfo` as well) and the loan ID for the returned resource is removed. This prevents returned loaned content from working. Therefore, it is important to make sure that *every* call to `embed::DeviceInfo::setActivationInfo` actually updates the device activation info even after device activation is done. This applies to the `activation.xml` file for the Digital Editions mass-storage device interface.

8.4 MULTIPLE FIXED PARTITIONS

The current SDK does not support multiple fixed partitions. This is potentially feasible, but it is currently seen as an edge case and is not supported. The problem with removable partitions is that the file structure that would be created would be left on the removable card after the card was removed and would confuse Digital Editions if the card is plugged into the card reader on the PC itself.

8.5 ACTIVESYNC

The Reader Mobile SDK provides support for Window Mobile's ActiveSync through the implementation (in the SDK) of `embed::DeviceInfo`. Device manufacturers *must* use this implementation to ensure that everyone's activations are compatible. The device needs only to create an instance of `embed::DeviceInfo`. This sets up the registry in the right state. More specifically, the values for "fingerprint" (base64-encoded) and "type" ("mobile" or "tethered") are written in `HKEY_LOCAL_MACHINE\Software\Adobe\Adept\Device key`. Digital Editions then reads that key to be able to recognize the device and handle synchronizations.

8.6 FULFILLMENT SUPPORT

The SDK supports fulfillment of eBooks from Content Server 4 enabled sites. In the desktop Digital Editions, this functionality is handled outside the SDK, so the SDK has been enhanced to provide this support. This is implemented in the SDK via the `DRMProcessor` and `DRMHost` interfaces. The host needs to implement `DRMHost` and instantiate a `DRMProcessor` object (via the static factory method) and make the appropriate calls.

For more information, see `embed/public/embed_viewer.h` and the *RMSDK API Specification*.

8.7 ACTIVATION WORKFLOWS

8.7.1 INITSIGNINWORKFLOW

The `initSignInWorkflow` function allows authentication in more than one way. In this example, it authenticates using a specified username and password:

```
unsigned int dpdrm::DRMProcessor::initSignInWorkflow( unsigned int workflows,
  const dp::String & authority,
  const dp::String & user,
  const dp::String & password )
```

If the authority is specified as AdobeID, this function behaves identically to the deprecated `initAdobeIDSignInWorkflow`.

For authorities (vendors) other than AdobeID, the authority specifies the password format, but the user should still be a human-readable username.

To facilitate authentication methods besides username and password, there is an override of `initSignInWorkflow` that takes raw data as `authData` to be passed on to the authority's web service by the activation service. For example:

```
unsigned int dpdrm::DRMProcessor::initSignInWorkflow( unsigned int workflows,
const dp::String & authority,
  const dp::String & user,
  const dp::Data & authData )
```

Note: Although the user name is passed into this workflow, this is strictly to provide a label for the activation record on the device. The user string is not included in the activation request to the activation server.

In either case, after initializing the SignIn workflow, the rest of the process of activation proceeds as before by calling startWorkflows, and then process the results through a DRMProcessorClient callback. Most likely you will want to specify the workflows flags as DW_AUTH_SIGN_IN | DW_ACTIVATE in startWorkflows.

8.7.2 ACTIVATING LINKED ACCOUNTS

The workflow flag DW_GET_CREDENTIAL_LIST contacts the activation server for the specified user, gets a list of users from linked accounts, and—if the DW_ACTIVATE flag is specified—activates all accounts that are not already activated on the current device. The flags are used in `dpdrm::DRMProcssor::initWorkflows`.

8.7.3 JOINING ACCOUNTS

You can also get the URL that allows a user to join a new account to the current account:

```
unsigned int dpdrm::DRMProcessor::initJoinAccountsWorkflow( unsigned int workflows,
const dp::String & userID,
  const dp::String & newUserID )
```

This contacts the activation server and authenticates with the credentials for the `userID`, which must be present on the device for the DRMProcessor, and negotiate a session and resulting URL on the activation server. Next, the launchBrowser callback for the DRMProcessorClient is called with the negotiated URL, and a web browser should be launched going to the URL by the RMSDK implementer. The user can enter the required credentials to authenticate for the newUserID, or choose to cancel. The activation server generates a .acsm file containing either a success or failure message which, when opened, will be reported to the DRMProcessorClient using either workflowsDone or reportWorkflowError as appropriate.

Even if successful, this workflow will not activate the device to the account indicated by newUserID. Instead it is the responsibility of the RMSDK implementer to start an Activate Linked Accounts workflow upon successful completion of the Join Accounts workflow.

8.7.4 OPENING A BOOK

To take advantage of the Activating Linked Accounts and Joining Accounts workflows, the RMSDK listen for an E_ADEPT_CORE_USER_NOT_ACTIVATED error, which is reported to the `DPDocumentClient::reportDocumentError` callback. For this error, the second token (space separated) is the `userID` for the license in the book.

To respond to this error, the RMSDK implementer should first check whether a network connection is available. If a network connection is not available, the error should be reported to the user. If a network connection is available, an 'Activate Linked Accounts' workflow should be started to see whether the user has already joined accounts. After successful completion, DRMProcessor::getActivations can be used to determine whether the needed userID has been activated for the device.

If the userID has not been activated for the device, the implementer should, if the device has a browser that is available, start a 'Join Accounts' workflow and, after successful completion, then immediately start an 'Activate Linked Accounts' workflow.

8.8 PASSHASH-PROTECTED BOOKS

8.8.1 OPENING A SIDE-LOADED PASSHASH-PROTECTED BOOK

When a device attempts to open a passhash-protected book, it expects the book's passhash to already exist in the activation record. So, if the book is copied from the device where it was fulfilled to another device, the device might not be able to open the book.

If none of the existing passhashes can be used to open the book, RMSDK generates the E_ADEPT_CORE_PASSHASH_NOT_FOUND error, which is reported to the `DPDocumentClient::reportDocumentError` callback. The error string also contains the URL of the operator that issued the license. Depending on the settings on the operator's fulfillment server, this URL might also contain the unique ID of the distributor that originated the transaction (for example, http://vancouverbc.corp.adobe.com/fulfillment?distid=5a4e29f3-dc27-4246-8431-afb8a621ea30).

When this error is encountered, the client should do the following:

1. Ask the user for username and password.

2. Calculate the passhash from this input using the `DRMProcessor::calculatePassHash()` function.
3. Add the calculated passhash to the activation record using the `DRMProcessor::addPassHash()` function.
4. If this call succeeds, retry opening the book.

8.8.2 Fulfillment of Passhash-Protected Books

When a passhash-protected book is being fulfilled and the license cannot be opened with stored passhashes, the DRMProcessor calls the `DRMProcessorClient::requestPasshash` callback, passing an instance that implements the `dpdrm::FulfillmentItem` interface.

The implementation of the `dpdrm::FulfillmentItem` interface should do the following:

1. Ask the user for input.
2. Calculate the passhash from this input using the `DRMProcessor::calculatePassHash()` function.
3. Call `DRMProcessor::providePasshash`, passing the passhash. (Do not call `DRMProcessor::addPassHash`.) The DRMProcessor keeps calling `DRMProcessorClient::requestPasshash` until the correct passhash is provided.
4. If a correct passhash is not received, cancel the loop, and call `DRMProcessor::providePasshash` with an empty passhash. In this case, the DRMProcessor reports the E_ADEPT_PASSHASH_CANCELLED error for the fulfillment workflow.

Note: RMSDK limits the number of passhashes per user, so multiple incorrect answers may preempt previously stored passhashes that are required for opening other documents.

9 RMSDK USAGE

9.1 HIT TESTING

A common usage for devices that have some method for the user to point and click is to select text. The following steps indicate how this can be done for selecting a range of text.

1. Get a Location object for the initial selection point `hitTest(x, y)` for the initial point selected (that is, beginning of the drag).
2. Get a `Location` object from `hitTest(x2, y2)` for the end of the selection.
3. Call `getText` with the two locations.
4. If you wish to highlight the selection, simply call `addHighlight(Location1, Location 2)`.

9.2 HIT TESTING AND SELECTING TEXT

The initial release (9.0) of the RMSDK does not include a built-in API for selecting a single word. However, with a little work, this can be achieved, as follows:

1. Assume that the user clicked on the 'd' in Adobe.
2. Get a `Location` object (Loc_1) from `hitTest(x, y)` for the point clicked.
3. Get a `Location` object (Loc_2) from `hitTest(0, y)` or start of screen/page for start of line.
4. Get a `Location` object (Loc_3) from `hitTest(max_x, y)` or end of screen/page for end of line.
5. `getText()` for Location 2, 1 (gets line text up to point clicked).
6. `getText()` for Location 2, 3 (get line text after point clicked).
7. From (4) and (5), get start of word ("Ad") and end of word ("obe") by searching for the last space character and the first space character, respectively.
8. Call `findText(Loc_2, Loc_1, .., "Ad", Loc_Start, Loc_Tmp)` search for start of word.
9. Then call `findText (Loc1, Loc_3, …, "obe", Loc_Tmp, Loc_End)`. `Loc_Start` and `Loc_End` now contain the start and end of "Adobe".
10. You can now get the text with `getText(Loc_Start, Loc_End)` and highlight it with `addHighlight(Loc_Start, Loc_End)`.

9.3 PERMISSIONS IMPLEMENTATION

All device and application implementations are required to properly implement document permissions for all documents.

For devices and applications supporting only display functions, permissions are already handled by the Reader Mobile SDK, and no further effort is required.

For devices and applications supporting printing, excerpting, and play (reading out loud), the implementation must check for the presence of the appropriate permission before allowing the function to be performed. See the `get_rights` API in the *Reader Mobile API Specification* for more information.

9.4 ANNOTATIONS AND BOOKMARKS

Bookmarks and annotations are host-side functionality. There is no support for them in the SDK because they tend to be specific to the host implementation. However, implementing this support is relatively easy.

Annotations and bookmarks are stored in XML format. An annotation is the same as a bookmark except that an annotation has some text (that is, a content element) in it. Here is a description of the annotation XML syntax:

```
<annotationSet xmlns:xhtml="http://www.w3.org/1999/xhtml"
xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns="http://ns.adobe.com/digitaleditions/annotations">
<publication>
   … Publication metadata such as <dc:identifier>, <dc:title>, <dc:creator> etc…
</publication>
<annotation>
    <dc:identifier>…  annotation identifier (UUID) …</dc:identifier>
    <dc:date>… W3C Date Annotation was created …</dc:date>
    <dc:creator>… User Identifier (UUID) …</dc:creator>
    <target>
       <fragment start="… Location …" end="… Location …">
           <text>… optional snippet of text if DRM allows …</text>
       </fragment>
    </target>
    <content>
        <dc:date>… W3C Date that the user comment was last Modified…</dc:date>
        <text>… Text of user comment …</text>
     </content>
</annotation>
… Additional annotation elements as needed …
</annotationSet>
```

The tags in the annotation should be self-explanatory. The target fragment start and end attributes are a serialized version of the Location (use the `getBookmark()` method on the `Location` object).

Basically, taking Digital Editions as an example, the annotations and bookmarks are stored in a folder that is a subfolder of the My Digital Editions folder. That folder contains a file with the same name as the file being annotated but with the extension “annot”. That file contains some very simple XML with the annotations and bookmarks for that document. In Digital Editions, an entry in the manifest.xml for a given book points to the annotation file.

To create bookmarks:

1. Detect the mouse selection when the user clicks and/or drags to create the bookmark/annotation.
2. Retrieve the highlighted region via the APIs (as indicated above).

3. Create the annotation file in an appropriate location (we encourage you to use the Digital Editions approach). The file is XML. See the ones created by DE and use the same format.
4. Store the serialized Location info (that is, the bookmark) for the start and end locations, creation date, and so on.
5. If you support annotations (as opposed to bookmarks) then allow the user to enter whatever text they want and store that too. Close the annotations file.
6. Repeat the process for each bookmark or annotation the user wants to make, adding each to the same annotations file (one for each document).

Retrieving the information is the inverse of the preceding process:

1. Open the annotation file.
2. Iterate through the annotations.
3. Retrieve the information and build your user-interface tree.
4. When the user clicks on an element in the tree, retrieve the serialized location information.
5. Use it to navigate to that location in the document.

9.5 SWITCHING BETWEEN CTS AND THE OLD TEXT ENGINE

CTS (Core Text Solution) is a collection of components for font access, text layout, and rasterization that provide an end-to-end solution to text display, are world-ready, support rich styling, and fit on small devices.

CTS is new in RMSDK 9.2 and provides better typography than the old text engine. However, it has slightly longer rendering times.

To switch between CTS and the old text engine, set the inheritable CSS property `adobe-text-layout` on any block-level element, such as `div`, body, and so on.

The syntax is:

```
adobe-text-layout optimizeQuality | optimizeSpeed
```

The `optimizeQuality` setting uses the new, CTS-based text engine; `optimizeSpeed` uses the old, minimal text engine. For example, the following CSS rule sets the text engine to be the old engine for all text in the document:

```
Body {
  Adobe-text-layout: optimizeSpeed;
}
```

10 THIRD-PARTY TOOLS

10.1 BUILDING OPENSSL FOR WINDOWS MOBILE

10.1.1 DISCLAIMERS

This chapter describes the steps required for building OpenSSL port for Windows Mobile 6. When this was written, OpenSSL version 0.9.8i was the latest stable release. Some steps require patching the source files. These steps may not be required in the future, when these changes become part of OpenSSL and/or WCECompat.

This chapter uses some default installation paths on a Windows XP 32-bit machine. You may need to modify the paths and environment variables on other versions of Windows or when the required packages or sources are installed to custom locations.

10.1.2 ACKNOWLEDGEMENT

The information provided in this chapter and tools required for OpenSSL support on Windows CE come from various sources. Special thanks to:

- Essemer Pty Ltd (http://www.essemer.com.au) for initial development of WCECompat;
- Maurice Kalinowski (maurice.kalinowski (AT) nokia.com) for updating WCECompat and creating patch for OpenSSL;
- SymbolicTools (their website http://symbolictools.de is dead) for developing PocketConsole and also creating PocketCmd based on ReactOS command-line interpreter;
- Mike Croucher (http://www.walkingrandomly.com) for information about setting up PocketCmd and PocketConsole and hosting the files.

10.1.3 PREREQUISITES

- Visual Studio 2008
- Windows Mobile 6 SDK
- ActivePerl: http://www.activestate.com/ActivePerl
- GNU patch utility for Windows. for example, http://gnuwin32.sourceforge.net/packages/patch.htm
- Utility to unpack Unix tarballs, for example, http://7-zip.org/

10.1.4 GETTING THE SOURCES

OpenSSL

1. Download OpenSSL sources from http://www.openssl.org/source/openssl-0.9.8i.tar.gz.
2. Extract `openssl-0.9.8i.tar.gz` to `C:\` - this should create `c:\openssl-0.9.8i` folder with the sources.
3. Visit http://marc.info/?l=openssl-dev&m=122536319510494&w=2 and save the attached file "`wince.patch`" into `C:\openssl-0.9.8i\wince.patch.txt`. Make sure that this file has MS-DOS (CR-LF) line endings.
4. Run the cmd.exe and go to the `C:\openssl-0.9.8i` folder.

5. Patch the OpenSSL with the following command:

```
patch.exe -u -p1 < wince.patch.txt
```

Windows CE compatibility files (WCECompat)

1. Download WCECompat from http://github.com/mauricek/wcecompat/tree/master. Note that the download button may not work in Internet Explorer 6.0, but it works with Firefox 3.0.
2. Extract the archive into c:\wcecompat.
3. Open `C:\wcecompat\include\stdlib.h` in a text editor and change line #39 from

   ```
   extern float fmodf(float);
   ```

   to

   ```
   extern float fmodf(float, float);
   ```

10.1.5 Building

10.1.5.1 WCECompat

1. Run `cmd.exe` and go to the `c:\wcecompat` folder.
2. Configure make files using the following commands:

   ```
   set OSVERSION=WCE502
   set TARGETCPU=ARMV4I
   c:\perl\bin\perl config.pl
   ```

   This needs to be done just once.
3. Create the `C:\wcecompat\build.bat` file with the following content:

   ```
   set OSVERSION=WCE502
   set TARGETCPU=ARMV4I
   call "C:\Program Files\Microsoft Visual Studio 9.0\VC\bin\vcvars32.bat"
   set PATH=C:\Program Files\Microsoft Visual Studio 9.0\VC\ce\bin\x86_arm;%PATH%
   set INCLUDE=C:\Program Files\Microsoft Visual Studio
   9.0\VC\ce\Include\%TARGETCPU%;C:\Program Files\Windows Mobile 6
   SDK\PocketPC\Include\%TARGETCPU%;C:\Program Files\Windows Mobile 6
   SDK\PocketPC\Include;C:\Program Files\Microsoft Visual Studio 9.0\VC\ce\atlmfc\include
   set LIB=C:\Program Files\Windows Mobile 6 SDK\PocketPC\LIB\%TARGETCPU%;C:\Program
   Files\Microsoft Visual Studio 9.0\VC\ce\Lib\%TARGETCPU%
   nmake %1 %2 %3 %4 %5 %6 %7 %8 %9
   ```
4. Open `c:\wcecompat\makefile`, go to line 3, and remove the `/WX` compiler option from `CFLAGS`.
5. Run build.bat without parameters to build WCECompat.
6. Copy `C:\wcecompat\lib\wcecompactex.lib` to the `thirdparty\openssl\public\wince\lib\` directory in the SDK source tree.

7. Copy `C:\wcecompat\include\errno.h` to the `thirdparty\openssl\public\wince\include\wcecompat\` directory in the SDK source tree.
8. Copy `C:\wcecompat\include\sys\types.h` to the `thirdparty\openssl\public\wince\include\wcecompat\sys\` directory in the SDK source tree.
9. If you need to clean up the build output, run `build.bat` with the `clean` parameter.

10.1.5.2 OpenSSL

1. Run the `cmd.exe` and go to `c:\openssl-0.9.8i` folder
2. Configure make files by the following commands:

```
set OSVERSION=WCE502
set TARGETCPU=ARMV4I
c:\perl\bin\perl Configure VC-CE
ms\do_ms
```

   This needs to be done just once.
3. Create the `C:\openssl-0.9.8i\build.bat` file with the following content:

```
set WCECOMPAT=c:\wcecompat

set OSVERSION=WCE502

set TARGETCPU=ARMV4I

call "C:\Program Files\Microsoft Visual Studio 9.0\VC\bin\vcvars32.bat"

set PATH=C:\Program Files\Microsoft Visual Studio 9.0\VC\ce\bin\x86_arm;%PATH%

set INCLUDE=C:\Program Files\Microsoft Visual Studio
9.0\VC\ce\Include\%TARGETCPU%;C:\Program Files\Windows Mobile 6
SDK\PocketPC\Include\%TARGETCPU%;C:\Program Files\Windows Mobile 6
SDK\PocketPC\Include;C:\Program Files\Microsoft Visual Studio 9.0\VC\ce\atlmfc\include

set LIB=C:\Program Files\Windows Mobile 6 SDK\PocketPC\LIB\%TARGETCPU%;C:\Program
Files\Microsoft Visual Studio 9.0\VC\ce\Lib\%TARGETCPU%

nmake -f ms\ce.mak %1 %2 %3 %4 %5 %6 %7 %8 %9
```

4. Open `C:\openssl-0.9.8i\ms\ce.mak` for editing
   a. Remove the `/WX` compiler option from `CFLAGS` on line #19
   b. Change line #27 from
      `EX_LIBS=winsock.lib $(WCECOMPAT)/lib/wcecompatex.lib`
      to
      `EX_LIBS=ws2.lib $(WCECOMPAT)/lib/wcecompatex.lib`
5. Run `build.bat` without parameters to build OpenSSL.
6. The output binaries are in the `out32_ARMV4I` folder.
7. Copy all the files from the `C:\openssl-0.9.8i\inc32\openssl directory` to the `thirdparty/openssl/public/vc71/include/openssl directory` in the SDK source tree.
8. Copy `libeay32.lib and ssleay32.lib from openssl-0.9.8i\out32_ARMV4I` to the `thirdparty\openssl\public\wince\lib\` directory in the SDK source tree.

9. If you need to clean up the build output, run `build.bat` with `vclean` parameter.

Note: This batch file builds the static libraries as well as test executables, which links these libraries statically. If you need to build dynamic libraries and executables for testing them, use `ms\cedll.mak` instead of `ms\ce.mak`.

10.1.6 Testing on Windows Mobile 6

The OpenSSL test programs are console applications, which write to the standard output. By default, Windows Mobile 6 does not support console IO. To add the console support to a Windows Mobile 6 device or emulator for testing purposes, you need to download the following archive: http://www.walkingrandomly.com/images/downloads/Pocketcmd_pocketConsole.zip. It contains the installers for PocketConsole and PocketCmd. You may need to reboot your device after installation of PocketConsole.

Set the value of the `HKEY_LOCAL_MACHINE\Drivers\Console\OutputTo` registry entry on the device/emulator to zero before running PocketCmd. Visit http://ceregeditor.mdsoft.pl/ to get CeRegEditor, a free desktop program to edit the registry of WindowsCE-based device.

You must copy the `openssl.conf` and server.pem files from `c:\openssl-0.9.8i\apps` to the OpenSSL folder on the device/emulator before running `ssltest.exe`

10.2 Building OpenSSL on Linux

To build OpenSSL on Adobe's reference Linux platform:

1. Get source if needed:
   a. Go to `http://www.openssl.org/source/`
   b. Select `openssl-0.9.8i.tar.gz` and save the file.
   c. Unpack `openssl-0.9.8i.tar.gz`
2. Build OpenSSL:
   a. Go to the `openssl` directory (`openssl-0.9.8i` in the unpacked files).
   b. Run "`perl ./Configure linux-generic32 -DL_ENDIAN`".
   c. Edit the resulting makefile, changing the line starting "`CC=`" to "`CC=arm-none-linux-gnueabi-gcc`".
   d. Run "`make`" to execute the Makefile and build OpenSSL.
   e. Copy all the files in the `include/openssl` directory to the `thirdparty/openssl/public/linux_arm_codesourcery/include/openssl` directory in the SDK source tree.
   f. Copy `libcrypto.a` and `libssl.a` to the `thirdparty/openssl/public/linux_arm_codesourcery/lib` directory in the SDK source tree.

10.3 Building OpenSSL on Windows Desktop

1) Get source if needed:
   a. Go to `http://www.openssl.org/source/`.

   b. Select `openssl-0.9.8i.tar.gz` and save the file.
   c. Unpack `openssl-0.9.8i.tar.gz.`

2) Build OpenSSL:
   a. Go to the `openssl` directory (`openssl-0.9.8i` in the unpacked files).
   b. Follow the directions in the INSTALL.W32.
   c. In the first step, use a temp directory for "some/openssl/dir".
   d. Use MASM for the assembler.
   e. Open \openssl-0.9.8i\ms\nt.mak, go to line 19, and remove the /WX compiler option from CFLAG.
   f. Open a new command prompt window using "All Programs > Microsoft Visual Studio 2008->Visual Studio Tools->Visual Studio 2008 Command Prompt".
   g. Go to the openssl directory in the new command prompt window.
   h. Execute the commands specified in INSTALL.W32, including the test and install steps. You should build a static version of the library.
   i. Go to the temp directory selected in the first step.
   j. Copy all the files from the include\openssl directory to the thirdparty\openssl\public\vc71\include\openssl directory in the SDK source tree.
   k. Copy lib\libeay32.lib and `lib\ssleay32.lib` to the `thirdparty\openssl\public\vc71\lib\` directory in the SDK source tree.

10.4 Building libcurl on Linux

libcurl is used for transferring files with URL syntax on the Mac OS and Linux in the book2png test program. Platform APIs are used for this purpose on Windows and Windows Mobile. The Mac OS and some versions of Linux already contain libcurl, but some embedded Linux implementations do not. The following instructions may be used to build libcurl for these versions of Linux.

To build libcurl on Adobe's reference Linux platform:

1. Get source if needed:
   a. Go to http://curl.haxx.se/download.
   b. Select `curl-7.18.2.tar.gz` and save the file.
   c. Create the directory `thirdparty/curl` within the sources you received.
   d. Unpack `curl-7.18.2.tar.g` into `thirdparty/curl`.
2. Build libcurl:
   a. Go to the `thirdparty/curl/curl-7.18.2` directory.
   b. Create a shell script in that directory containing the following. You will need to change lines 2-10 appropriately if you are not using the CodeSourcery tools. Here is the shell script.

```
! /bin/sh
export PATH=$PATH:/opt/CodeSourcery/Sourcery_G++_Lite/bin
export CPPFLAGS="-pipe -msoft-float -fPIC"
export AR=arm-none-linux-gnueabi-ar

export AS=arm-none-linux-gnueabi-as

export LD=arm-none-linux-gnueabi-ld

export RANLIB=arm-none-linux-gnueabi-ranlib

export CC=arm-none-linux-gnueabi-gcc

export NM=arm-none-linux-gnueabi-nm

export LDFLAGS="-static-libgcc -Wl,-rpath=/opt/CodeSourcery/Sourcery_G++_Lite/arm-none-
linux-gnueabi/libc/lib:/opt/CodeSourcery/Sourcery_G++_Lite/arm-none-linux-
gnueabi/libc/usr/lib -Wl,--dynamic-linker,/opt/CodeSourcery/Sourcery_G++_Lite/arm-none-
linux-gnueabi/libc/lib/ld-linux.so.3"

./configure --target=arm-none-linux \

            --host=arm-none-linux \

            --build=i586-pc-linux-gnu \

            --prefix=/tmp/usr/local \

            --exec-prefix=/usr/local \

            --disable-shared

make

mkdir -p ../public/linux_arm_codesourcery/lib

cp lib/.libs/libcurl.a ../public/linux_arm_codesourcery/lib

cp src/curl ../public/linux_arm_codesourcery/lib

mkdir -p ../public/linux_arm_codesourcery/include/curl

cp include/curl/curl.h ../public/linux_arm_codesourcery/include/curl

cp include/curl/mprintf.h ../public/linux_arm_codesourcery/include/curl


cp include/curl/curlver.h ../public/linux_arm_codesourcery/include/curl

cp include/curl/easy.h ../public/linux_arm_codesourcery/include/curl

cp include/curl/multi.h ../public/linux_arm_codesourcery/include/curl
```

c. Execute the shell script.

11 BOOK2PNG

book2png is a small console application that Adobe builds on all supported platforms. It is provided along with the SDK as both a testing aid and as an example of how to integrate with the SDK. Adobe uses it extensively for our own testing and validation efforts.

The general usage is to invoke it with a series of command-line arguments, including a document and more optional arguments. The output is a series of PNG files that are the pixel-for-pixel output of each page of the document. This section lists the available features of book2png.

General usage:

```
            book2png [option ...] [--] filename
```

11.1 SYNTAX AND TERMINOLOGY

Options can start with '-' or '--'.

'--' ends option processing; in other words, if you use -- preceding an option, all options after that option are ignored.

Words in CAPS are variable args.

Pages versus *screens*: EPUB, being reflowable, does not have set pages; it is laid out as a series of “screens”. PDF has explicit pagination but, when reflowed, each page can result in one or more “screens” per original page.

Over-the-air: When the following commands speak of “over-the-air”, it refers to information coming and going via the network interfaces. The SDK doesn’t know how the information comes or goes; it could be a wireless connection or an ethernet cable. But typically these connections are “over-the-air” (that is, wireless), so this terminology is used.

11.2 EXAMPLES

Here are examples of common actions that you might do with book2png.

book2png [options] docfile

Loads and renders a PDF or EPUB document to a sequence of PNG images. Also (by default) generates an HTML wrapper for viewing convenience.

book2png [options] -auth <authProvider> **-user** <username> **-password** <password>

Activates the device using the specified provider authentication, username, and password. You can alternatively activate using this with the **authData** option instead of **user** and **password**.

book2png [options] -fulfill <ACSM>

Processes a fulfillment token: request the license(s), download the book(s), and write the license(s) to the book(s)

11.3 GENERAL OPTIONS

11.3.1 RENDERING OPTIONS

alpha

Renders to a bitmap with an 8-bit alpha (transparency) channel.

blend-with-background

Blends rendering onto the surface.

coff-start <N>

Same as **page-start** (for historical reasons).

depth <N>

Specifies the bit depth for grayscale dithering (default is 3).

dpi <R>

Sets the number of CSS pixels per inch for EPUB (default: 96). This affects how absolute CSS units (pt, in, mm, cm) are converted to CSS pixels. It affects the EPUB default font size, among other things.

find <text>

Before starting rendering, navigates to the first occurrence of the specified text. Text with no embedded spaces or dashes does not need to be enclosed in quotation marks; however, quotation marks are always allowed around the text. For example: **-find "this text".**

fit

Fits the PDF size to the viewport; this is the equivalent to Acrobat's Fit Page setting. Cannot be used with the **scale** option. Can be used only with nonreflowed PDF. The scale factor is calculated automatically to fit one PDF page into one PNG using the values specified by the **width** and **height** options.

font-size small | medium | large | x-large

Sets the default font size for EPUB. The default is **medium, which** corresponds to 12 points.

gamma <N>

Specifies the gamma for the grayscale transfer function (default is 1).

go-to <bookmark>

Before starting rendering, navigates to the specified bookmark.

go-to-end

Before starting rendering, navigates to the end of the document.

height <H>

Sets the viewport height to the specified height in logical (document) units (default: 792 points). See **scale** for information on logical units. For EPUB, this number multiplied by the scale factor will typically be the height of the PNG files produced. For PDF, it has an effect only when used with the **fit** option. See **width** for more information.

margin-hor <H>

Adds the specified number of logical (document) units to the existing horizontal margin. See **scale** for information on logical units.

margin-vert <V>

Adds the specified number of logical (document) units to the existing vertical margin. See **scale** for information on logical units.

monochrome

Renders to an 8-bit grayscale bitmap.

no-html

Don't produce an HTML wrapper.

no-png

Don't produce either HTML or PNG.

out-html <name>

Uses the specified name for the HTML wrapper (default is book.html). If the file already exists, it is overwritten.

out-page <template>

Uses the specified template for the resulting PNG files using printf formatting notation (default is "page000n.png").

page-start <N>

Before starting rendering, navigates to the specified page number. The first page of the document is always 1.

page-count <N>

Renders at most the specified number of screens (which might not be the same as pages). Will produce at most the specified number of PNG files.

render pages | no

What to render. The default is **pages** and **no** turns off rendering.

reflow

Turns on PDF reflow mode.

scale <S>

Sets the scaling factor that determines how many physical pixels there will be per logical (document) unit.

By default, the scaling factor is 1, which means that a logical document unit is rendered to 1 pixel in the PNG file. The logical unit in PDF documents is 1 point. The logical unit in EPUB documents is one CSS "px" unit.

This option cannot be used with the **fit** option, which calculates the scaling factor automatically on for each page.

type application/pdf | application/epub+zip

Sets the mime type for the document to load. Valid mime types are "application/pdf" and "application/epub+zip".

width <W>

Sets the viewport width to the specified width in logical (document) units (default: 612 points). See **scale** for information on logical units.

For typical EPUB and reflowed PDF, this number multiplied by the scale factor will be the width of the PNG files produced. For non-reflowed PDF, it has an effect only when used with the **fit** option.

book2png sets the document to the viewport size according to the **width** and **height** options specified, but then queries the document for its natural size. XHTML and reflowed PDF honor the external viewport size, but nonreflowed PDF disregards it. book2png uses the viewport size from the document for rendering. When the **fit** option is given, the scale factor is automatically calculated to fit the document's desired viewport into the width and height specified by the options.

11.3.2 Authentication Options

add-hash <operatorURL> <passHash>

Adds the specified base64-encoded password hash for the specified fulfillment operator's URL.

add-pass <operatorURL> <user> <password>

Calculates and adds a password hash for the specified username and password for the specified fulfillment operator's URL.

async-password

Specifies that the password to access an already password-encrypted PDF file will be requested asynchronously after setURL returns.

auth <authProvider>

Activates the device using the specified provider. This must be used either with both the **user** and **password** options, or with the **authData** option. The authProvider can either be **AdobeID** or the vendor name of the authentication provider. For example, **-auth AdobeID**

authData <authData>

Specifies base64-encoded authentication data for device activation used together with the **auth** option.

deactivate

Removes activation information from the device.

joined-accounts

Uses joined accounts to try to reopen the book after failure.

password <password>

Specifies the password to use for device activation along with the **auth** and **user** options.

pdf-password <password>

Specifies the password to access an already password-encrypted PDF file.

remove-hash <operatorURL> <passHash>

Removes the specified base64-encoded password hash from the specified operator URL.

remove-pass <operatorURL> <user> <password>

Calculates and removes the password hash for the specified username and password.

user <username>

Specifies the username to use for device activation with the **auth** and **password** options.

11.3.3 Fulfillment and License Options

fulfill <ACSM>

Fulfills using the specified .acsm file (fulfillment token).

license <id> <type> <file>

Reads the license for the specified resource ID of the specified type from the specified file. Can be used to process raw (unfulfilled) files if the license is available.

return-loan <id>

Gives back the loan that has the specified identifier.

write <newfile>

Writes the document with the license in it. Can be used to add the license specified by the license option into a file.

11.3.4 Command and Test Options

command-file <command-file-name>

Runs a series of commands from the specified file. Refer to the "Command-File Capabilities" section for valid commands. To read commands from stdin instead, see the –i option.

force

Forces all specified options to occur, if possible, even if the document did not fully load.

i

Changes to interactive mode, which prints the book2png> prompt to stdout, allows multiple commands to be entered from stdin, one after each prompt, and doesn't exit when an error occurs. The exit command will stop book2png.

resources <dir>

Uses the specified folder (directory) to read resources, for example, a user-defined cascading style sheet (.css).

run perm | nored

Runs the specified test. The test can be:

- perm - Prints the document's permissions.
- nored - Searches for red pixels in each bitmap produced; for the test to pass, no red pixels must be present.

11.3.5 Reporting Options

These options all produce output to stdout unless otherwise specified.

help

Prints the usage statement and exits.

hit-test <x> <y>

Hit-tests a point located at x,y on every screen before rendering, then prints the bookmarks for the locations found at these points.

link-info

Prints link information for every screen before rendering.

msgs-on-stdout

Sends messages from RMSDK to stdout (the default is B2PLOG_LOGERROR).

query-bookmark

Before rendering (but after navigation option processing), prints the current location's bookmark.

query-ccount

Prints the document's content count (usually the number of 'pages' in the document).

query-coff

Before rendering (but after navigation option processing), prints the document's starting page offset (content offset).

query-dimensions

Prints the document's natural width and height.

query-meta <name>

Prints the specified metadata from the document. To query Dublin Core metadata, use "DC." (uppercase) as a prefix (for example, **-query-meta DC.title**).

query-page

Prints the document's starting page name based on the document settings.

query-pages

Prints the document’s total page count.

query-toc

Prints the document's table of content.

redirect-stdout <file>

Redirects stdout to the specified file. If the file already exists, it is overwritten.

redirect-B2PLOG_LOGERROR <file>

Redirects stderr, which is where most processing error messages go, to the specified file. If the file already exists, it is overwritten.

time

Prints the rendering time for each screen to stdout as the pages are rendered.

time-detail <file>

Prints to the specified file the rendering time for each command as it completes processing. If the file already exists, it is overwritten.

verbose

Prints additional messages. For over-the-air operations, prints all the requests and responses.

version hobbes | hobbes.build

Prints the version of the specified module. Note that version **hobbes** returns "9.0" because this release is Reader Mobile 9.0 SDK. The command version **hobbes.build** returns the hobbes version plus the build number, for example, “9.22567”.

11.4 Command-File Capabilities

11.4.1 General

The book2png command processor has no looping or branching capabilities. The intent is to have each command map to a single call over the embedding interface to the extent possible.

Each command is on a separate line of the command file.

The command file is in UTF-8 encoding. However, all characters outside of quoted strings are in the ASCII subset of Unicode (range 0x0 to 0x7F).

A command file is invoked with the "-command-file <command file name> "option on the book2png command line. To read commands from stdin instead of from a file, specify the "-i" option.

11.4.2 Command Line

A command line consists of a command followed by 0 to 7 parameters. The command consists of up to 32 ASCII characters at the start of the line. Whitespace is used as the separator between the command and the first parameter and between each pair of parameters. Extra whitespace at the end of the line is ignored, but no other characters are allowed on the line after the final parameter. Whitespace at the start of the line is also ignored. Whitespace consists of any combination of spaces (Unicode 0x20) and tabs (Unicode 0x09).

A command line starting with a # (Unicode 0x23) is a comment line and is ignored. Blank command lines are also ignored.

11.4.3 Parameters

The parameters may be integers, real numbers, quoted strings, or locations. Integers, real numbers, and quoted strings used for input parameters may be constants or calculated values. Locations and all output parameters must specify a calculated value.

Type	Description
Constant integers	Consist of an optional sign followed by up to 10 decimal digits. An integer must fit in a 32-bit twos-complement number.
Constant real numbers	Consist of an optional sign, up to twelve significant digits, and an optional decimal point at any place in the number. Leading or trailing 0s may be used if the decimal point must be outside the significant digits. A real number must be representable as a 64-bit IEEE floating-point number.
Constant quoted strings	Consist of up to 127 characters within a pair of double-quotes (" - Unicode 0x22). A backslash (/ - Unicode 0x5C) is used as an escape character. The only valid escape sequences are a backslash followed by a double-quote (\") or a pair of backslashes (\\).
Calculated values	Outputs from commands are placed in these and can be used as inputs to later commands. A calculated value is represented by a single letter followed by a single digit. The letters used for integer, real number, quoted string, location, or matrix calculated values are I, R, S, L, and M respectively. Only 10 calculated values of each type are available at a time. Calculated values can be reused.

11.4.4 Interaction with book2png Options

The following options may be used in conjunction with -**command-file** on the line calling book2png. They have the usual behavior as long as a document file is also specified on the command line; otherwise, these are ignored:

alpha	margin-hor	query-coff	scale
async-password	margin-vert	query-dimensions	time
blend-with-background	monochrome	query-meta	time-detail
depth	msgs-on-stdout	query-page	type
dpi	no-html	query-pages	verbose
font-size	no-png	query-toc	version
force	out-html	reflow	width
gamma	out-page	redirect-stdout	write
height	pdf-password	redirect-B2PLOG_LOGERRR	
joined-accounts	query-bookmark	resources	
license	query-ccount		

The following options, if used anywhere on the command line, execute immediately and exit without attempting to perform any other options or commands:

help

deactivate

The following options on the line calling book2png are *not* compatible with -**command-file**; they cause an error and exit:.

add-hashadd-pass	fulfill	page-start	return-loan
auth	go-to	password	run
authData	go-to-end	remove-hash	user
coff-start	hit-test	remove-pass	
find	link-info	render no	
fit	page-count	render pages	

11.4.5 Aborting Commands Using abortCount

Several commands take an `abortCount` parameter. This parameter aborts the processing of that command by returning a false `canContinueProcessing` callback. This is done by counting the number of calls to `canContinueProcessing` with any foreground processing kinds of codes (any code, except for PK_BACKGROUND). Normally this parameter should be set to 0, in which case `canContinueProcessing` always returns true. However, if a positive number is given, then, after the number of calls specified with this parameter, this function starts to return false, thereby requesting an operation to stop prematurely. After an operation is aborted, the document and renderer can be in inconsistent state, for example, if navigation is aborted, the renderer may have navigated or not navigated to the requested location.

11.4.6 Command Set

This section lists the supported commands and how they relate to the API in the embedding interface. Each section starts with the command name followed by the parameters that it takes. The direction (input or output) and type of each parameter is shown between < and >.

addSelectHighlight	startLocation *(input, location)* endLocation *(input, location)* index *(output, integer)*

Executes the `Renderer::addHighlight` API passing HT_SELECTION as the highlightType. The locations specified by startLocation and endLocation are passed in as the start and end respectively. Returns the highlight index in the calculated value specified by the index parameter.

calcFitMatrix	docWidth *(input,real)* docHeight *(input,real)* desWidth *(input, real)* desHeight *(input,real)* ctm *(output,matrix)*

Calculates a transformation matrix, ctm, that can be used to draw the document with docWidth and docHeight dimensions into a surface that has dimensions desWidth and desHeight.

ceil	realNumber *(input,real)* intNumber *(output,integer)*

Converts a real to an integer, rounding up.

cnvtrBookmkToLoc	bookmark *(input, string)* location *(output, location)*

Executes the `Document::getLocationFromBookmark` API, passing in the value specified by bookmark and placing the result into the calculated value specified by the location parameter.

cnvtLocToBookmk	location *(input, location)* bookmark *(output, string)*

Executes the `Location::getBookmark` API on the location specified by the location parameter and places the result into the calculated value specified by the bookmark parameter.

createMatrix	a *(input,real)* b *(input,real)* c *(input,real)* d *(input,real)* e *(input,real)* f *(input,real)* matrix *(output,matrix)*

Creates a transformation matrix.

createSearchFlags

caseMatching *(input, string)*
direction *(input, string)*
wholeWords *(input, string)*
wrap *(input, string)*
searchFlags *(output, integer)*

Creates the searchFlags needed for findText. This does not call any embedding interface APIs. The caseMatching parameter must be "matchCase" or "ignoreCase". The direction parameter must be "forward" or "backward". The wholeWords parameter must be "wholeWords" or "partialWordsOK". The wrap parameter must be "wrap" or "noWrap". The result is put into searchFlags.

findText

stringToFind *(input, string)*
searchFlags *(input, integer)*
startSearch *(input,string)*
endSearch *(input, String)*
abortCount *(input, integer))*
startStringFound *(output, location)*
endStringFound *(output, location)*

Executes the `Document::findText` API, passing stringToFind and searchFlags as parameters, startSearch as start, and endSearch as end. The searchFlags can be created using the createSearchFlags command. The start and end of the string found are placed into startStringFound and endStringFound, respectively. If the stringToFind was not found, "String <stringToFind > not found" is written to stdout. abortCount is described at the beginning of this section..

floor

realNumber *(input,real)*
intNumber *(output,integer)*

Converts a real to an integer, rounding down.

get

beginning *(output, location)*

Executes the `Document::getBeginning` API and places the result into the calculated value specified by the beginning parameter.

getDocumentEnd

end *(output, location)*

Executes the `Document::getEnd` API and places the result into the calculated value specified by the end parameter.

getLinkLocations

index *(input, integer)*
startLocation *(output, location)*
endLocation *(output, location)*

Executes the `Renderer::getLinkInfo` API, passing in the value specified by the index parameter as the linkIndex. The values returned in the srcStart and srcEnd are placed into the calculated values specified by the startLocation parameter and the endLocation parameter, respectively.

getLinkTargetLocation index *(input, integer)*
location *(output, location)*

Executes the `Renderer::getLinkInfo` API, passing in the value specified by the index parameter as the linkIndex and placing the value returned as the target into the calculated value specified by the location parameter.

getLocation currentLocation *(output, location)*

Executes the `Render::getCurrentLocation` API and places the result into the calculated value specified by the currentLocation parameter.

getLocationFromPagePosition pagePosition *(input, real)*
location *(output, location)*

Executes the `Document::getLocationFromPagePosition` API, passing the value from the pagePosition parameter as pos and placing the result into the calculated value specified by the location parameter.

getNavigationMatrix ctm *(output,matrix)*

Executes the `Renderer::getNavigationMatrix` API and returns the result.

getNaturalSize width *(output,real)*
height *(output,real)*

Executes the `Renderer::getNaturalSize` API and returns width and height.

getPageNumbersForScreen start *(output, integer)*
end *(output, integer)*

Executes the `Renderer::getPageNumbersForScreen` API, placing the value returned in start and end into the calculated values specified by the start and end parameters, respectively.

getRangeInfo start *(input, Location)*
end *(input, Location)*

Executes the `Render::getRangeInfo` API and saves the result. Only a single range info is saved at any time. An additional call to this command will overwrite the old range info. The range info becomes invalid on a call to any command that access the Renderer.

getRangeInfoBox index *(input, integer)*
xMin *(output, double)*
yMin *(output, double)*
xMax *(output, double)*
yMax *(output, double)*

Executes the `RangeInfo::getBox` API on the currently saved range info. The index specifies the box in case more than one box is needed to cover the range. No commands that access the Renderer may be called between the getRangeInfo command that created the range and this command.

getScreenEnd endOfScreen *(output, location)*

Executes the `Render::getScreenEnd` API and places the result into the calculated value specified by the endOfScreen parameter.

getScreenStart beginningOfScreen *(output, location)*

Executes the `Render::getScreenBeginning` API and places the result into the calculated value specified by the beginningOfScreen parameter.

navigateActivate

Executes the `Render::handleEvent` API with an event of EK_NAVIGATE kind and NAVIGATE_ACTIVATE type.

navigateFirst abortCount *(input, integer)*

Executes the `Render::handleEvent` API with an event of EK_NAVIGATE kind and NAVIGATE_FIRST type. abortCount is described at the beginning of this section.

navigateLast abortCount *(input, integer)*

Executes the `Render::handleEvent` API with an event of EK_NAVIGATE kind and NAVIGATE_LAST type. abortCount is described at the beginning of this section.

navigateLeave

Executes the `Render::handleEvent` API with an event of EK_NAVIGATE kind and NAVIGATE_LEAVE type.

navigateNext abortCount *(input, integer)*

Executes the `Render::handleEvent` API with an event of EK_NAVIGATE kind and NAVIGATE_NEXT type. abortCount is described at the beginning of this section.

navigatePrevious abortCount *(input, integer)*

Executes the `Render::handleEvent` API with an event of EK_NAVIGATE kind and NAVIGATE_PREVIOUS type. abortCount is described at the beginning of this section.

navigateToLocation location *(input, location)*
abortCount *(input, integer)*

Executes the `Render::navigateToLocation` API, passing the value from the location parameter as the loc. abortCount is described at the beginning of this section.

next

Executes the `Render::nextScreen` API.

previous

Executes the `Render::previousScreen` API.

printInteger comment *(input, string)*
number *(input, integer)*

Prints a line to stdout consisting of the string from the comment parameter followed by the value of the integer in the number parameter. This does not call any embedding interface APIs.

printMatrix comment *(input, string)*
matrix *(input,matrix)*

Prints a line to stdout consisting of the string from the comment parameter followed by six real values of the matrix parameter components. This does not call any embedding interface APIs.

printReal comment *(input, string)*
number *(input, real)*

Prints a line to stdout consisting of the string from the comment parameter followed by the value of the real number in the number parameter. This does not call any embedding interface APIs.

printString comment *(input, string)*
string *(input, string)*

Prints a line to stdout consisting of the string from the comment parameter followed by the string from the string parameter. This does not call any embedding interface APIs.

render abortCount *(input, integer)*

Executes the `Render::paint` API. It uses the bitmap format determined by the book2png command-line options -monochrome and -alpha and the dimensions determined by the options -width, -height, and -scale. abortCount is described at the beginning of this section.

renderRect format *(input, string)*
xMin *(input, integer)*
yMin *(input, integer)*
xMax *(input,integer)*
yMax *(input, integer)*
abortCount *(input, integer)*

Executes the `Render::paint` API. It is similar to the render command, but provides more parameters to control bitmap format and the region to be painted.

setEnvironmentMatrix ctm *(input,matrix)*

Executes the `Render::setEnvironmentMatrix` API with the specified parameter.

setFontSize sizeFactor *(input, integer)*

Executes the `Render::setScaleFactor` API, passing the value from the sizeFactor parameter as the factor.

setNavigationMatrix ctm *(input,matrix)*

Executes the `Render::setNavigationMatrix` API with the specified parameter.

setPagingModeToFlowPages

Executes the `Render::setPaging`Mode API, passing PM_FLOW_PAGES as the pagingMode.

setPagingModeToHardPages

Executes the `Render::setPagingMode` API, passing PM_HARD_PAGES as the pagingMode.

setPagingModeToScrollPages

Executes the `Render::setPagingMode` API, passing PM_SCROLL_PAGES as the pagingMode.

setSelectHighlightColor index *(input, integer)*
color *(input, integer)*

Executes the `Render::setHighlightColor` API, passing HT_SELECTION as the highlight type and the value from the color parameter as the color.

setViewportSize width *(input, real)*
height *(input, real)*

Executes the `Renderer::setViewport` API with the specified parameters.

11.5 Usage Examples

To view protected content, you need to authorize the computer, fulfill the content, and then view the content. The following three examples demonstrate these steps.

11.6 Authorize Computer Example

```
book2png -auth AdobeID -user username@example.com -password password
```

11.7 Fulfill protected content Example

If you have a running ACS4 server instance, you can set up the sample (PHP) store to generate signed URLs. This example uses one of our internal servers.

```
curl "<Signed URL to a URLLink.acsm service>" > test.acsm

book2png -fulfill ./test.acsm
```

This names the content fulfilled0001.epub (or .pdf) and places it in the default directory for the DRMProcessor for the device, usually the Digital Editions folder.

11.8 Render first page of content

This command creates a file named page0000.png, which is the rendering of the first page using the default view size (612x792):

```
book2png -page-count 1 -page-start 0 ./fulfilled0001.epub
```

12 WINEMBED

The sources include a simple Windows app named WinEmbed. This provides a simple application that allows viewing PDF and EPUB files. It is provided simply as-is as a test application that is sometimes useful for debugging and learning how to implement the embedding interface. There is currently no equivalent MacEmbed or LinuxEmbed.

There are no command-line arguments for WinEmbed. A fair number of the keystroke commands can be found in `embed/hosts/win/win_embed_host.cpp` in the function `emh::WinRendererHost::windowP`.