Using ADOBE® ACROBAT® PORTFOLIOS SDK to build ACROBAT NAVIGATORS™



© <INSERT RELATIVE YEARS(S)> Adobe Systems Incorporated and its licensors. All rights reserved.

Using <INSERT NAME OF ADOBE PRODUCT WITH CORRECT TRADEMARK SYMBOLS>

This user guide is protected under copyright law, furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies that may appear in the informational content contained in this guide.

This user guide is licensed for use under the terms of the Creative Commons Attribution Non-Commercial 3.0 License. This License allows users to copy, distribute, and transmit the user guide for noncommercial purposes only so long as (1) proper attribution to Adobe is given as the owner of the user guide; and (2) any reuse or distribution of the user guide contains a notice that use of the user guide is governed by these terms. The best way to provide notice is to include the following link. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/3.0/

Adobe, the Adobe logo, <INSERT ADOBE TRADEMARKS IN ALPHABETICAL ORDER> are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

<INSERT THIRD PARTY TRADEMARKS THAT APPEAR IN THE DOCUMENT AND THAT REQUIRE ATTRIBUTION .> All other trademarks are the property of their respective owners.

Updated Information/Additional Third Party Code Information available at http://www.adobe.com/go/thirdparty.

Portions include software under the following terms:

<INSERT RELEVANT THIRD PARTY NOTICES FROM PRODUCT AUDIT>

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA.

Notice to U.S. Government End Users: The Software and Documentation are "Commercial Items," as that term is defined at 48 C.F.R. §2.101, consisting of "Commercial Computer Software" and "Commercial Computer Software Documentation," as such terms are used in 48 C.F.R. §12.212 or 48 C.F.R. §227.7202, as applicable. Consistent with 48 C.F.R. §12.212 or 48 C.F.R. §§227.7202-1 through 227.7202-4, as applicable, the Commercial Computer Software and Commercial Computer Software Documentation are being licensed to U.S. Government end users (a) only as Commercial Items and (b) with only those rights as are granted to all other end users pursuant to the terms and conditions herein. Unpublished-rights reserved under the copyright laws of the United States. Adobe agrees to comply with all applicable equal opportunity laws including, if appropriate, the provisions of Executive Order 11246, as amended, Section 402 of the Vietnam Era Veterans Readjustment Assistance Act of 1974 (38 USC 4212), and Section 503 of the Rehabilitation Act of 1973, as amended, and the regulations at 41 CFR Parts 60-1 through 60-60, 60-250, and 60-741. The affirmative action clause and regulations contained in the preceding sentence shall be incorporated by reference.

Contents

Chapter 1: About PDF Portfolios and Navigators
Navigator Concepts 1
How to Create a Custom Navigator 2
Navigator APIs 3

Chapter 2: Installing Acrobat Portfolios SDK
Installing the Portfolios SDK 4

Chapter 3: Portfolio API

Chapter 4: Portfolio Framework
Components 7
Cards 7
Managers 8
Layouts 8
Previews 8
Effects 9
Skins 9

Chapter 5: Building a Navigator
Workflow 10
Creating a Navigator Project 10
Writing and Compiling Your Navigator Application 10
Troubleshooting 11

Chapter 6: The Initial Navigator

Chapter 7: Debugging Your Navigator
Debugging a Navigator in Flash Builder 15
Debugging a Navigator with Acrobat 15

Chapter 8: Packaging Your Navigator
Exporting Your Navigator 17
Importing Your Navigator in Acrobat 18

Chapter 9: Acrobat Navigator File Format
NAV file overview 19
Component files 19
Mimetype file 19
Navigator.xml file 20
Navigator SWF file 22
Icon file 22
Locales file 23
Strings file 23

Resource files 24
Relax NG Compact Schemas 24

Chapter 10: Universal Container Format
Introduction 27
UCF overview 29
UCF container contents 32
Zip container 38
RELAX NG UCF schema 39
Example 40
Comparison of UCF and OCF 44
Digital signatures 45
Encryption 46

Chapter 1: About PDF Portfolios and Navigators

Adobe Acrobat allows you to create PDF Portfolios, which are PDF files that contain other files, called attachments, and manage those attachments. These attachments can be in different formats, of different types, and created by different applications. The original files retain their individual identities when they become part of the single PDF Portfolio file. Each attachment can be opened, read, edited, and formatted independently of the other attachments in the PDF Portfolio.

The presentation and management of the attachments in a PDF Portfolio, and any actions taken on them, such as viewing and editing, is managed by an PDF Portfolio Layout, or *Navigator*. In other words, a Navigator is a Flash front-end for the collection of content in a PDF Portfolio. It defines animations, colors, card styles, fonts, headers, footers, background, preview states, treatment of specific doc types and more. You create Navigators in Adobe Flash Builder by using the Acrobat Portfolios SDK.

Note: *This guide uses the term Navigator rather than the official term PDF Portfolio Layout. The informal term is more meaningful for developers because it corresponds to the type of file that represents a PDF Portfolio Layout.*

A Navigator can include a range of interactive features, including user interface components (widgets), animation, and localized strings. It can also contain media files—such as video clips, images, and mp3 files,—to provide a rich media experience. The Navigator interacts with PDF Portfolios through the Acrobat application.

Acrobat Pro allows you to apply PDF Portfolio Navigators to a PDF Portfolio. When applying a Navigator, you can choose from the Navigators installed with Acrobat Pro, or you can apply a custom Navigator that you or others develop. When a Navigator is applied to a PDF Portfolio, it becomes part of that PDF Portfolio. That is, it is part of the PDF document that contains the PDF Portfolio. Users of Acrobat X or Adobe Reader X can then view the PDF Portfolio using the embedded Navigator. Acrobat users have the ability to update the PDF Portfolio, and Acrobat Professional users can edit the Navigator properties.

The Acrobat Portfolios SDK provides you with the tools to create your own custom Navigators. It contains two rich ActionScript APIs you use to create customized Navigators. It also includes three Flash Builder plug-in wizards that automate key steps in Navigator development:

1. The initial creation of a Navigator project
2. Launching a debugging session for the Navigator in a previewer application, or in Acrobat
3. Packaging the completed Navigator into a single file for use with PDF Portfolios

Navigator Concepts

There are a couple of concepts you need to understand when building Navigators. Navigators are applications that are called from a host application: Adobe Acrobat. They provide the presentation interface for PDF Portfolios by working in conjunction with the host application. Acrobat has two types of interfaces for the contents of a PDF Portfolio, a tabular format and a more graphical view which is controlled by the Navigator (see "Types of PDF Portfolio Interfaces" on page 2). When switching between these control is passed between the host application and the Navigator.

The Navigator has a preview concept for the contents of the PDF Portfolio. A Navigator user can *preview* the attachments that are included in the PDF Portfolio without having to leave the Navigator. Depending on the type of attachment that is part of the PDF Portfolio, the user can view and even edit the attachment's contents. Any changes made are saved in the PDF Portfolio. See "Preview the Contents of a PDF Portfolio with a Navigator" on page 2.

Types of PDF Portfolio Interfaces

Acrobat provides two ways of viewing the contents of a PDF Portfolio, a layout view and a files view.

- The layout view is a graphical representation. Attachments are shown as thumbnails. The user can navigate through the list of attachments, which can be presented in a variety of visual styles: linearly, in a grid, as a "wave," free form placement, and so on. Navigators control the display and actions of the PDF Portfolio when it is in layout view.
- The files view is a table with one row for each attachment. The row displays the details for that attachment, such as name, size, and description. If the attachment is a folder, you can double click on the folder to display its contents, also in a tabular format. This view is not controlled by the Navigator. It is controlled by Acrobat. Changes may be made to the PDF Portfolio when it is in files view.

Preview the Contents of a PDF Portfolio with a Navigator

Although Navigators were introduced with Acrobat 9, you could not preview the contents of a PDF Portfolio with a Navigator. Instead, in Acrobat 9 you used Acrobat to preview the contents of your PDF Portfolio. This could confuse the user, leaving them unsure of what application they were working in.

In Acrobat X, the new Navigators provide preview functionality. This keeps the —semi-transparent—Navigator in the background, and uses transitions and skins as clues that you are still in the Navigator context. In addition, there is a large close button; when you press the close button you exit the Navigator preview mode and return to the normal Navigator viewing mode.

When you are in preview mode, you have access to commands that act on the attachment you are previewing, including:

- See attachment information
- Go to the next or previous page of the attachment, if the attachment is a PDF file
- Extract the attachment from the PDF Portfolio

When you are in preview mode you still have access to commands for navigating within the PDF Portfolio without leaving the previewer. For example, you can go to next or previous PDF Portfolio attachment.

Note: *The Acrobat 9 style of previewing is still available to developers in the Acrobat Portfolios SDK.*

How to Create a Custom Navigator

Adobe Acrobat Portfolios SDK greatly simplifies the process of creating custom Navigators. It's wizards automate the process of creating an initial working Navigator, debugging your Navigator, and packaging the completed Navigator along with its resources as a Navigator (NAV) file.

Requirements

To build Navigators using the Acrobat Portfolios SDK the following must be installed on your system:

- Adobe Acrobat Portfolios SDK

- Adobe Flash Builder 4.0.1 or later
- Adobe Flex 4.1 SDK (available at http://www.adobe.com/cfusion/entitlement/index.cfm?e=flex4sdk)
- Adobe Acrobat Pro X or later

The development workflow

To create a custom Navigator you need to do the following:

1. Install the Adobe Acrobat Portfolios SDK plugins and AIR application in your Flash Builder programming environment. See "Installing Acrobat Portfolios SDK" on page 4 for instructions.
2. Install the acroFlashDebug plugin into Acrobat X. See "Installing Acrobat Portfolios SDK" on page 4 for instructions.
3. Run the Create Navigator Project wizard
4. Use ActionScript, Flex 4, and the Acrobat Portfolios SDK to write your Navigator
5. Test and debug your Navigator using the Debug Adobe Navigator wizards
6. Export your Navigator as a Navigator (NAV) file.
7. Import your Navigator into Acrobat, where it is available to apply to PDF Portfolios.

Navigator APIs

There are two separate application programming interfaces (APIs) you use to create custom Navigators:

- `com.adobe.portfolio.api`, known simply as the Portfolio API
- `com.adobe.portfolio`, known as the Portfolio Framework

`com.adobe.portfolio.api`, known as the Portfolio API, is a lower level API, first introduced with Acrobat 9 that has been updated to work with Acrobat X. See "Portfolio API" on page 6 for information on this API.

`com.adobe.portfolio`, known as the Portfolio Framework, is a new, higher level API, that is built on top of `com.adobe.portfolio.api`. It provides a layer of abstraction between the Navigator and its host application (Acrobat) to maximize flexibility on both sides and to give access to the host application of the Navigator's configurable properties, without the. It provides extensive support for skinning the various Navigator components. See "Portfolio Framework" on page 7 for information on this API.

When you build a Navigator using the Portfolio Framework, the representation and behavior of your Navigator's properties are controlled by the Navigator. The host application has no knowledge of how properties are stored, or what effect they have on the Navigator. The host simply allows the Navigator to specify it's properties through a collection of property interfaces defined in Portfolio Framework. The host application use these interfaces to get and set properties, with the implementation handled by the Navigator.

Chapter 2: Installing Acrobat Portfolios SDK

Before you can build custom Portfolio Navigators, you must have the required applications and install the Adobe Acrobat Portfolios SDK.

Installing the Portfolios SDK

Before you can begin developing your Navigator, install the Acrobat Portfolios SDK, install the acroFlashDebug.api plugin into Acrobat, and install the Navigator plugins for Flash Builder 4. To do this:

1. Download the Acrobat Portfolios SDK from `http://labs.adobe.com/technologies/portfoliosdk/`

   This is a zip file containing all the code files, plugins, and documentation for the Acrobat Portfolios SDK.

2. Unzip the Acrobat Portfolios SDK zip file to a convenient location. This zip file contains a folder with the following subfolders:

   - `Documentation`—This folder contains this document and the API documentation for all the library classes.
   - `Tools`—This folder contains the Flash Builder wizards and the supporting plugins for them.
   - `Libs`—This folder contains the Portfolio Framework library as a SWC file.
   - `Samples`—This folder contains a FlexSkin sample and other sample code.
   - `Skins`—This folder contains FLA source files for skinning PDF Portfolios.
   - `Sources`—This folder contains a zip file with the Portfolio Framework source files that are compiled into the Portfolio Framework SWC file.

3. Find the `acroFlashDebug` file in the Acrobat Portfolios SDK folder you unzipped in step 2.

   For Windows the file is: `Tools\AcroFlashDebug\Win\AcroFlashDebug.api`

   For Macintosh the file: is`Tools\AcroFlashDebug\Mac\AcroFlashDebug.acroplugin`

4. Copy this `acroFlashDebug` file into the plugins subdirectory of Acrobat X.

   For Windows this subdirectory is normally `C:\Program Files\Adobe\Acrobat 10.0\Acrobat\plug_ins`

   For Macintosh this subdirectory is normally`Applications/Adobe Acrobat X Pro/Adobe Acrobat Pro.app/Contents/Plug_ins/`

5. Install the Navigator wizard plugins

   a. Copy the contents of the `Tools\Flashbuilder\Put contents in FB plugins dir` directory into the plugins subdirectory of your Flash Builder application.

      For Windows this is normally `C:\Program Files\Adobe\Adobe Flash Builder 4\plugins\`

      For Macintosh this is normally `/Applications/Adobe Flash Builder 4/plugins/`

   b. Copy the contents of the `Tools\Flashbuilder\Put contents in FB app dir`directory into your Flash Builder application directory.

      For Windows this is normally `C:\Program Files\Adobe\Adobe Flash Builder 4\`

For Macintosh this is normally `/Applications/Adobe Flash Builder 4/`

6. (Windows Vista/7) Register the Navigator plugin:
   a. Go to the folder where you have installed Flash Builder
   b. Select the Flash Builder executable and click the right mouse button
   c. When the menu appears, select "Run as"
   d. When the Run As dialog appears, enter that you want to run the application as Administrator, enter the administrator password and click OK.

Note: *If you are running Windows 7 and have User Access Control (UAC) turned on, you must have administrator permissions for the new plugins to be registered and to be able to debug using the preview application for Flash Builder. Whenever you update the plugins, you will need to register the updated plugins*

Chapter 3: Portfolio API

The fundamental package for implementing Navigators is *com.adobe.portfolio.api*. This package provides the low-level interfaces and classes for Navigators. For details on the contents of the Portfolio API refer to the *com.adobe.portfolio.api* API documentation.

When building a Navigator you normally use these higher-level API packages, and only need to use the lower-level *com.adobe.portfolio.api* package for a highly customized feature, if at all.

Chapter 4: Portfolio Framework

Portfolio Framework is a set of packages for implementing Navigators that are at a more abstract level than is *com.adobe.portfolio.api*. Portfolio Framework provides more powerful classes for you to use. These classes have much of the design work already coded, so they are not as flexible as the low-level API, but they give you considerable power and contain a feature rich set of classes to use. Like *com.adobe.portfolio.api* these packages all have the prefix *com.adobe.portfolio.*

When building a Navigator you normally use these higher-level API packages, and only need to use the lower-level *com.adobe.portfolio.api* package for a highly customized feature, if at all.

The classes in the higher-level API are skinnable.

Components

The package *com.adobe.portfolio.components* contains the main components used in building Navigators. These include:

- The Navigator class
- Header and background classes
- CardContainer, CardList, CardSlider, and CardTip classes
- List classes: MiniNavList, FreeFormList, LinearList
- NavSkinnableComponent and NavSkinnableContainer
- Navigation tools: FileNavigationButton, BreadCrumb, FolderBack

The base class for Acrobat X navigators, Navigator, is found in *com.adobe.portfolio.components*. It inherits from the Application class. Developers using the Portfolio Framework should extend the Navigator class instead of the Application class.

The most common component in a Navigator is a CardList with a dataProvider supplied by an instance of CollectionManager.

Cards

The items that make up a Portfolio are accessed using the interface defined in the class IAttachment. Cards are skinnable renderers for the IAttachment class. Every attachment in a Portfolio is represented by a card that shows, where possible, the user an image of the attachment, as well as metadata information about the attachment. Cards provide a preview of the data for PDF documents, video and audio attachments.

There are a variety of cards for basic file attachments, folders, previews, and so on.

Managers

The package *com.adobe.portfolio.controllers*, implements the classes for controlling the attachments, components, and other elements of the Navigator. The main classes are:

- CollectionManager—This class manages current attachments, selection, and change notifications from the Navigator's host application. It acts as the intermediary between Navigator components and an INavigatorHost.
- CommandManager—Instances of this class manage commands sent to the host.
- AttachmentDropManager—This class provides support for attachment drag and drop to user interface components that register with this class.
- PropertyManager—This class loads XML files that contain specifications of Navigator properties and propagates the information to the Navigator host.
- FlashSkinManager—Navigators can dynamically change the skins they use. Instances of the FlashSkinManager class dynamically loads SWF files and creates object instances from them to be used as visual assets by Flex components and skins.
- Font Manager, ThumbnailManager, StringManager, ResourceManager—These classes manage the use of fonts, thumbnail images, localization strings, and other resources respectively for the Navigator.

Layouts

Layouts, as their name implies, are a set of classes that control the layout of the cards that make up the Portfolio controlled by the Navigator and provide a means for users to flip through the attachments contained in a Portfolio. Each attachment in the Portfolio is represented by a card that presents the user with an image of the attachment (where possible), as well as metadata concerning the attachment. There a several different layout options supplied:

- Grid layout—This layout presents the contained attachments in a grid format.
- Click through—The layout presents the cards in the Portfolio one at a time in a horizontal line.
- Linear layout—The layout presents the cards in the Portfolio in a horizontal line showing the currently selected document in the context of the set of documents, and displaying a notepad for each document.
- Free form layout—This layout provides a way to display attachments anywhere and in any orientation the author desires upon the background.
- A table row format—This layout is used for for the files view of a Portfolio. It displays attachments (and folders) in a list, with each row in the list displaying the visible fields of the attachment or folder.
- A wave style layout—As the name implies this layout presents the cards in the Navigator using a wavelike animation effect when transitioning from one document to the next; as you move from one document to the next the set of thumbnails representing the documents in the Portfolio moves in a sine wave.

Previews

This package, *com.adobe.portfolio.previews*, implements the classes for previewing attachments in the Portfolio, including the ability to preview audio and video attachments.

Effects

Utilitity classes for effects and transitions for Navigators are contained in *com.adobe.portfolio.effects*.

Skins

The Navigators that come with Acrobat X support a set of skins. These skins can be further customized to change fonts and color palettes. Additionally, you can create designer skins that can be applied to your custom Navigator in Flash Builder. These can be used with the Portfolio Framework components.

The classes supporting skins are contained in *com.adobe.portfolio.skins*

Chapter 5: Building a Navigator

This section explains how to develop a Navigator that provides a user interface for viewing and working with attachments in a PDF Portfolio.

Workflow

Creating a Navigator for a PDF Portfolio involves the following tasks:

1 "Creating a Navigator Project" on page 10 in Flash Builder and setting up the proper environment for it

2 "Troubleshooting" on page 11 your Navigator

3 "Writing and Compiling Your Navigator Application" on page 10 as a SWF file

4 "Packaging Your Navigator" on page 17 to bundle up the SWF file and any needed resources into a "Acrobat Navigator File Format" on page 19 file

5 "Importing Your Navigator in Acrobat" on page 18 in Acrobat

Creating a Navigator Project

You create a new Navigator project using the create Navigator project wizard that is part of the Acrobat Portfolios SDK.

1 Open the Flash Builder 4 application.

2 Go to File > New > Other.

3 In the "New" dialog box "select Adobe Navigator > Navigator Project and click Next.

4 In the Adobe Acrobat Navigator window enter a project name for your new Navigator project.

The folder field shows you where the files for your new Navigator project will be located.

5 Click Finish.

The wizard generates all the necessary files for your Navigator project, including a navfiles folder. This folder contains the additional files that are packaged with the .SWF file you build to create the .NAV Navigator file.

You are now ready to extend and customize your Navigator project

Writing and Compiling Your Navigator Application

If you use the New Navigator Project wizard to create your Navigator project, It provides you with a working Navigator. This Navigator implements the key elements of a Navigator:

- It creates an instance of the Navigator class
- It adds standard visual components to the Navigator: a background, header, a skinnable data container for the attachments, a MiniNavList component, a preview component, and scroller.

- It adds non-visual components such as an AttachmentDropManager to handle drag-and-drop addition of attachments.
- It adds states (loading, empty, and normal) and transition effects for state changes.
- It handles the initialize event, so that the Navigator is properly connected to Acrobat.

To customize your Navigator you can change the properties of the initial components, add to or replace these components with alternatives, and apply your own skins.

Troubleshooting

This section provides general troubleshooting tips for finding and fixing problem with your Navigator.

Troubleshooting your code independent of Acrobat

The cleanest way to troubleshoot your code is to use the debugging capabilities and tools within Flash Builder. There is a Navigator wizard that automates the setup for debugging your code independent of Acrobat. Instead, the code is run using AIR. See "Debugging a Navigator in Flash Builder" on page 15 for the steps involved.

Troubleshooting problems loading your Navigator

If your Navigator uses assets such as images and localized text strings, make sure you import those assets into the ActionScript file.

The most common reason Acrobat reports an error while loading a Navigator is because the mimetype file is not the first file in the Navigator file (see "Mimetype file" on page 19). To avoid this problem completely always use the export wizard to export your Navigator. This properly packages your Navigator files into a proper Navigator format file.

If you have manually created the Navigator file, you can use WinZip or a hex editor to examine the order of the files. To use WinZip to examine and repair the order of files in a Navigator file:

1. In WinZip, open the Navigator file.
2. Select View > Sort > Original Order.
3. If the mimetype file is not the first one in the archive, delete everything from the archive except the mimetype.
4. Add the remaining component files you deleted in the previous step (so that the mimetype file is now the first file).
5. Close the Navigator file.
6. Install the Navigator file in Acrobat (see "Importing Your Navigator in Acrobat" on page 18).

Troubleshooting your code within Acrobat

There is a Navigator wizard that automates most of the setup for debugging your code within Acrobat. See "Debugging a Navigator with Acrobat" on page 15"Debugging a Navigator in Flash Builder" on page 15 for the steps involved.

Chapter 6: The Initial Navigator

The following code is the initial `main.mxml` code generated by the New Navigator Project wizard. This wizard creates a working Navigator that you can customize.

This code includes the following components and functionality:

- An attachment drop manager that handles drag-and-drop functionality for your Navigator
- A MiniNavList component that is the gets the focus by default
- A NavPreview component that allows you to preview the contents of the PDF Portfolio
- A background component
- A scroller and scroll group that handles scrolling and clipping of the visual display
- A set of component states
- A set of transition effects

```
<?xml version="1.0" encoding="utf-8"?>
<!--
/*****************************************************************************
*
* ADOBE SYSTEMS INCORPORATED
*  Copyright 2010 Adobe Systems Incorporated
*  All Rights Reserved.
*
* NOTICE:  Adobe permits you to use, modify, and distribute this
* file in accordance with the terms of the Adobe license agreement
* accompanying it.  If you have received this file from a source
* other than Adobe, then your use, modification, or distribution
* of it requires the prior written permission of Adobe.
*****************************************************************************/
-->
<components:Navigator xmlns:fx="http://ns.adobe.com/mxml/2009"
            xmlns:s="library://ns.adobe.com/flex/spark"
            xmlns:mx="library://ns.adobe.com/flex/mx"
            xmlns:supportClasses="com.adobe.portfolio.components.supportClasses.*"
            xmlns:components="com.adobe.portfolio.components.*"
            xmlns:layouts="com.adobe.portfolio.layouts.*"
            xmlns:managers="com.adobe.portfolio.managers.*"
            currentState="loading" backgroundColor="#515151" frameRate="24"
            fontFamily=""
            >
  <fx:Script>
    <![CDATA[
      import com.adobe.portfolio.cards.BasicCard;
      import com.adobe.portfolio.cards.CardFactory;
      import com.adobe.portfolio.managers.ThumbnailManager;

      import mx.binding.utils.BindingUtils;
      import mx.events.FlexEvent;
      import mx.events.StateChangeEvent;
      import mx.managers.IFocusManagerComponent;
```

```
      /** A component that's suitable for receiving focus by default.
       */
      override public function get defaultFocusItem():IFocusManagerComponent
      {
        return miniNav;
      }

      private var loadingHasCompleted:Boolean = false;
      public function loadingComplete():void
      {
        if (loadingHasCompleted) {
          return;
        }
        loadingHasCompleted = true;
        scroller.validateNow();

        // These bindings cause so much thrashing on load that the intro experience
        // is never smooth. If we set them here rather than in mxml, then we can delay
        // that thrashing.
        BindingUtils.bindProperty(cardList, "dataProvider", collectionManager, "rootItems");
        BindingUtils.bindProperty(miniNav, "dataProvider", collectionManager, "rootItems");
        BindingUtils.bindProperty(navPreview, "dataProvider", collectionManager,
"currentItems");
      }
    ]]>
  </fx:Script>
  <fx:Declarations>
    <!-- Place non-visual elements (e.g., services, value objects) here -->
    <fx:Number id="fadeDuration">500</fx:Number>
    <managers:AttachmentDropManager dropTarget="{this}"/>
  </fx:Declarations>
  <components:states>
    <s:State name="loading"/>
    <s:State name="empty"/>
    <s:State name="normal"/>
  </components:states>
  <components:transitions>
    <s:Transition fromState="loading" toState="*">
      <s:Sequence targets="{[scroller, this]}">
        <s:Fade target="{scroller}" duration="{fadeDuration}"/>
        <s:CallAction target="{this}" functionName="loadingComplete"/>
      </s:Sequence>
    </s:Transition>
    <s:Transition fromState="normal" toState="empty">
      <s:Fade target="{miniNav}" duration="{fadeDuration}"/>
    </s:Transition>
    <s:Transition fromState="empty" toState="normal">
      <s:Fade target="{miniNav}" duration="{fadeDuration}"/>
    </s:Transition>
  </components:transitions>
  <s:Scroller id="scroller" alpha="1.0" alpha.loading="0.0" left="0" right="0" top="0"
bottom="0" horizontalScrollPolicy="auto" verticalScrollPolicy="auto">
    <!-- This is the group that gets clipped/scrolled when the nav window is too small -->
    <s:Group id="scrollGroup" width="100%" height="100%" clipAndEnableScrolling="true">
      <!-- This is the group that defines a minimum width/height, thus forcing it's parent
group to clip/scroll -->
      <s:Group id="innerGroup" minWidth="600" minHeight="{miniNav.height + header.height +
```

```
350}" width="100%" height="100%">
        <components:Background id="bkgnd0" width="100%" height="100%"/>
        <components:Header id="header" width="100%" top="0" heightIfPresent="100"/>

        <components:CardList id="cardList" left="0" top="{header.height}" right="0"
bottom="{miniNav.height}" tabEnabled="true" contentBackgroundAlpha="0" borderVisible="false"
                itemRendererFunction="{CardFactory.getItemRendererFunction(true, true, true,
BasicCard.STATE_LARGE, CardFactory.DROP_SHADOW_BOTTOM, null, 1, true, true)}">
          <components:layout>
            <layouts:OneUpLayout/>
          </components:layout>
        </components:CardList>

        <components:MiniNavList id="miniNav" bottom="0" left="0" right="0"
heightIfPresent="180"
                    showSlammer="true" alpha="0.0" alpha.normal="1.0" enabled.empty="false"
                    itemRendererFunction="{CardFactory.getItemRendererFunction(false, true,
true, null, null, null, 1, true, true)}"
                    />
        <components:NavPreview id="navPreview" width="100%" height="100%"
                   previewFolderSize="{new Point(innerGroup.width-60, innerGroup.height-60)}"
                   previewCardSize="{new Point(innerGroup.width-160, innerGroup.height-120)}"
                     rootDataGroup="{cardList.bindableDataGroup}"
                     itemRendererFunction="{CardFactory.getItemRendererFunction()}"/>
      </s:Group>
    </s:Group>
  </s:Scroller>
</components:Navigator>
```

Chapter 7: Debugging Your Navigator

There are two environments in which to debug your project. Normally you debug the application within Flash Builder using AIR. Debugging with AIR is the easiest method and allows you to test and examine all aspect of your code other than any interactions with Acrobat.

If you need to debug the interactions of your project with Acrobat itself, you can launch a debug session using Acrobat.

Debugging a Navigator in Flash Builder

To launch a debugging session from within Flash Builder:

1. Open Flash Builder 4 with the workspace for your Navigator project.
2. Select your Navigator project in the Package Explorer and drill down to the main source file for your Navigator (*e.g.*, MyNavigator > src > (default package) > main.mxml). Double click this file to open it in Flash Builder.
3. Set the initial breakpoints you want (Run > Toggle Breakpoint).
4. Go to Run > Debug > Other... This brings up the Debug Configurations dialog box.
5. In the Debug Configurations dialog box in the left panel select Adobe Navigator and either double click the mouse or click the New button to create a new launch configuration for your Navigator.
6. Select the icon for the new launch configuration you have created for your Navigator project.
7. Click the Debug button in the lower right corner.

Your Navigator now starts. It halts at any break points you have set and Flash Builder switches to the Flash Debug perspective when the first breakpoint is encountered.

Note: *You may get a warning message that pops up telling you that "The installed Adobe Flash Player is not a debugger...". This is a spurious warning. Simply click OK On Windows you can eliminate the warning message by installing and launching, as Administrator, Adobe Flash Player 10.1 Debugger. You can download Adobe Flash Player 10.1 Debugger from* `http://www.adobe.com/support/flashplayer/downloads.html`*launching Adobe Flash Player 10.1 Debugger once, as Administrator, will set it as the default for SWF files and eliminate the warning message*

Debugging a Navigator with Acrobat

If your Navigator seems to work properly within Flash Builder but not with Acrobat, you need to debug your Navigator with Acrobat, instead of AIR.

Preliminary Steps

You first need to let Acrobat know about your Navigator. You only need to do this once for each Navigator you build.

1. Export your Navigator from Flash Builder as a Debug Navigator (see "Exporting Your Navigator" on page 17).
2. Open Acrobat Pro
3. Click the Create PDF Portfolio button on the welcome screen.

4 In the Create PDF Portfolio window that appears click the Import Custom Layout... button, then find and select your Navigator file.

5 Click Finish

6 Close Acrobat Pro

You have now registered your new Navigator with Acrobat and are ready to launch the debug version in Acrobat.

Launching the Debugger

Once you register your Navigator with Acrobat Pro, you can launch a debugging session using Acrobat and the integrated debugger in Flash Builder.

Note: *After registering your Navigator with Acrobat Pro, you must close Acrobat Pro to complete that process before you can launch a debugging session using Acrobat Pro.*

1 Open Flash Builder 4

2 Select your Navigator project in the Package Explorer and drill down to the main source file for your Navigator. E.g., MyNavigator > src > (default package) > MyNavigator.mxml). Double click this file to open it in Flash Builder

3 Set the initial breakpoints you want (Run > Toggle Breakpoint)

4 Go to Run > Debug > Other... This brings up the Debug Configurations dialog box. In the Debug Configurations dialog box in the left panel select Adobe Navigator and either double click the mouse or click the New button to create a new launch configuration for your Navigator. If you already have a launch configuration for debugging your Navigator in AIR, we recommend that you create a new launch configuration, using the New button to create a separate launch configuration for running the debugger with Acrobat.

5 Select the icon for the new launch configuration you have created for your Navigator project.

6 Click the Navigator tab on the right to open that tab.

7 Uncheck the Use Previewer check box.

8 Enter the path to Acrobat in the Host Application entry field.

For Windows the path to Acrobat is normally `C:\Program Files\Adobe\Acrobat 10.0\Acrobat\Acrobat.exe`

For Macintosh the path to Acrobat is normally `Applications/Adobe Acrobat X Pro/Adobe Acrobat Pro`

9 Click the Apply button.

10 Click the Debug button in the lower right corner.

11 Acrobat starts. In Acrobat, click Create a PDF Portfolio again.

12 In the Create PDF Portfolio dialog that appears, click Finished.

13 The Acrobat Portfolio window comes up and your Navigator appears in the list on the right. Select it to run it.

14 When your application hits a breakpoint you have set, Flash Builder displays the Confirm Perspective Switch dialog box. You can now debug your application using the Flash Builder debugger.

Note: *Once you are finished debugging your Navigator in Acrobat, make sure to shut down Acrobat. Otherwise, the debug session will stay connected to Acrobat and Acrobat will not work properly.*

Chapter 8: Packaging Your Navigator

Once you have completed the coding and testing of your Navigator you are ready to use it to create an Acrobat portfolio.

Acrobat can use any SWF file that implements the Acrobat INavigator interface as a Navigator, but this is not recommended. By packaging your Navigator SWF file into a Navigator file, you can add your Navigator to list of available Navigators in Acrobat and customize how your Navigator is presented in the Acrobat User Interface. For example, you can supply an icon for your Navigator.

A Navigator (.NAV) file collects into a single, special type of ZIP file, all the resources required by a PDF Portfolio Navigator. The mandatory objects in the Navigator file are:

- A mimetype file that specifies the mime type of the Navigator: *application/vnd.adobe.pdf-navigator*
- A *navigator.xml* manifest that contains the basic information about the Navigator for use by Acrobat
- The SWF file that manages display of a PDF portfolio

Additionally, the Navigator file may contain any of the following optional contents:

- An icon file
- A locales file
- One or more strings files
- One or more resource files, such as images and font files
- Flash themes and skins

Exporting Your Navigator

Once you have compiled your Navigator and tested it thoroughly with the debugger, you are ready to export it for use with Acrobat.

1. Open the Flash Builder 4 application.
2. Select your Navigator project in the Package Explorer.
3. Build the project. If you want to export a release version of the Navigator, make sure you have first compiled a release version of the Navigator SWF (File > Export Release Build).
4. Go to File > Export > Other... This will bring up the Export wizard.
5. Open the Adobe Navigator folder, select either “Debug Navigator” or “Release Navigator” and press “Next >”.

   The difference between “Debug Navigator” and “Release Navigator” is whether the compiled navigator SWF file is taken from the bin-debug or the bin-release subdirectory for you project. Flash Builder automatically compiles the debug version when needed. To export the release version you must first compile a release version. To do this, go to `File > Export Release Build` and press Finish. This will compile a release version of the SWF file. You then need to follow the steps above to export the release version as a NAV file package.
6. Leave the default boxes on the left checked. If desired, edit the filename and destination that appears in the Destination entry field and press “Finished”.

   This will create your Navigator (NAV) file, ready for use.

Importing Your Navigator in Acrobat

Once you have exported the NAV file that contains your Navigator you can import it into Acrobat. To do this:

1. Open Adobe Acrobat Pro X.
2. From the Welcome screen click on Create PDF Portfolio.
3. In the Create PDF Portfolio window that appears, click on the Import Custom Layout button in the left panel, under Choose a layout for your Portfolio.
4. In the Open file dialog that appears locate your NAV file, select it, and click Open.
5. Make certain that your Navigator is selected and click Finish.

Acrobat opens your Navigator. You can now add the portfolio files.

Verifying Your Navigator

You should verify that your PDF Portfolio Navigator is working properly before distributing it in PDF Portfolios being shared with others.

To verify that your Navigator works properly:

1. Using Acrobat X Pro, open an existing PDF Portfolio or create a new PDF Portfolio containing a variety of files.
2. Click the Edit on the tool bar.
3. Make sure that your Navigator is available from the Edit Portfolio panel on the right and that it appears with the correct icon, name, and description.
4. Select your Navigator, and then save your PDF Portfolio. The Navigator will be embedded in your PDF document.
5. Open the portfolio you have just saved.
6. Examine your the portfolio in layout and details modes to make sure it is working as you intend.

   Note: *The details mode is part of Acrobat; it is not controlled by your customized Navigator.*
7. If your Navigator is going to be used with portfolios that are not locked, make sure that you can edit and delete attachments without problem.
8. Copy the file to another computer, and then open the file using Adobe Reader X.
9. Verify that your Navigator is present in the layout view.

Chapter 9: Acrobat Navigator File Format

In Acrobat 9, PDF Portfolios (previously known as packages and portable collections) were extended to enable Flash-based presentations called PDF Portfolio layouts, or Navigators. While Adobe includes standard Navigators with Acrobat, you can create your own Navigator to provide a custom view using a Flash component.

A Navigator file collects into a single file the resources required by a PDF Portfolio layout. The core piece of a Navigator file is a SWF file that implements the presentation of a PDF Portfolio. The Navigator file also includes meta data as well as auxiliary resources required by the SWF file, such as images.

When a user selects a Navigator for use in a PDF Portfolio, Acrobat extracts the data it needs from the Navigator file and inserts it into the PDF file.

In Acrobat X there are five standard Navigators, and Acrobat also supports custom Navigators. All editions of Acrobat 9, and later, and Adobe Reader 9, and later can display a PDF Portfolio using the embedded custom Navigator.A Navigator (NAV) file is a Universal Container Format (UCF)-based format that contains an XML manifest, the Navigator SWF file, and other resources required by the Navigator, such as images, localized strings, and private data. The SWF file is a Flex- or Flash-based component that interacts with a host authoring or viewing application by using an ActionScript API. For more information about the UCF format, see "Universal Container Format" on page 27.

Acrobat lets a user choose a Navigator and modify its properties, while Adobe Reader simply displays the PDF Portfolio contents by loading and running the embedded Navigator SWF.

NAV file overview

The MIME type of a Navigator file is application/vnd.adobe.pdf-navigator, and the extension is .nav. The NAV file is a Zip file conforming to the "Universal Container Format" on page 27. A Navigator file must contain a file named mimetype as defined by UCF, a file named navigator.xml, and a Navigator SWF file. A Navigator file may contain an icon file, a locales file, one or more strings files, and resource files.

Component files

UCF file names are case sensitive. Further restrictions can be found in "Universal Container Format" on page 27. Relative URIs must be used to create references between files in a Navigator. The navigator.xml file is considered to be in a root directory. Other resources may be in this directory or in other subdirectories, excluding the META-INF subdirectory, which is reserved by UCF. A URI that resolves to a location other than the root directory or its subdirectories is not permitted and will be ignored.

Mimetype file

"Universal Container Format" on page 27 describes the mimetype file in more detail, and should be considered the normative reference. This section summarizes the rules for this file.

- A file called mimetype must be the first file stored in the Zip file.
- It must be uncompressed.

- Its contents must be the string application/vnd.adobe.pdf-navigator.

Navigator.xml file

The navigator.xml file is the starting point for processing a Navigator file. It contains basic information about the Navigator that allows an authoring application to describe the Navigator without loading the Navigator SWF file. The only required data in this file is a reference to the Navigator SWF file.

The following is an example of a navigator.xml file.

```
<navigator src="MyNavigator.swf" loadAs="default" id="urn:MyNavigator" version="9.0.0.0"
apiVersion="10.0" xmlns="http://ns.adobe.com/pdf/navigator/2007">
  <name sID="navName"/>
  <description sID="navDescription"/>
  <category sID="navCategory"/>
  <icon src="AX_ClickThrough_Md_N.png"/>
  <locales src="locales.xml"/>
  <initialFields>
    <field name="FileName" type="filename" visible="true" editable="true">
      <displayName sID="fieldFileName"/>
    </field>
    <field name="Created" type="creationDate" visible="true" editable="false">
      <displayName sID="fieldCreated"/>
    </field>
    <field name="ModDate" type="modDate" visible="true" editable="false">
      <displayName sID="fieldModified"/>
    </field>
    <field name="Size" type="size" visible="true" editable="false">
      <displayName sID="fieldSize"/>
    </field>
    <field name="CompressedSize" type="compressedSize" visible="true" editable="false">
      <displayName sID="fieldCompressed"/>
    </field>
    <field name="Description" type="desc" visible="true" editable="true">
      <displayName sID="fieldDescription"/>
    </field>
    <field name="adobe:DisplayName" type="text" visible="false" editable="true">
      <displayName sID="fieldDisplayName"/>
    </field>
    <field name="adobe:Tags" type="text" visible="false" editable="true">
      <displayName sID="fieldTags"/>
    </field>
    <field name="adobe:AspectRatio" type="number" visible="false" editable="true">
      <displayName sID="fieldAspectRatio"/>
    </field>
  </initialFields>
</navigator>
```

The navigator.xml file is an XML 1.0 file whose namespace is http://ns.adobe.com/pdf/navigator/2007. The root element for this file is <navigator>, which includes several attributes described later in this section.

<navigator> element attributes

The <navigator> element of the navigator.xml file can contain the following attributes.

- xmlns—(Required) The namespace for the navigator.xml file.
- src—(Required) The location of the Navigator SWF file expressed as a URI relative to navigator.xml.
- id—(Optional) A string representing a unique ID for the Navigator, expressed as a URI. The ID follows the same rules as XML namespace names. IDs can be human-readable or a URN based on a machine-generated GUID, as long as they are likely to be unique. GUIDs are defined by RFC 4122.
- loadAs—(Optional) If the value is "module", the Navigator SWF file is to be loaded as a Flex module. If the value is "default" or is omitted, the Navigator SWF file is loaded as an ordinary SWF application. Modules are similar to Flash Runtime Shared Libraries (RSLs) in that they separate code from an application into separately loaded SWF files.
- version—(Optional) A string that represents a numerical version number of the form m[.n[.p[.q]]] (m, n, p, and q are non-negative integers. If not present, n, p, and q have a default value of 0). If there are multiple Navigators with the same ID, the one with the latest version number is loaded. If a Navigator does not include a version number, any Navigator with the same ID and any version number has precedence over it. Acrobat's behavior is undefined when there are two Navigators with the same ID and version number.
- apiVersion—(Required) This attribute represents a combination of the Navigator API version and the embedded Flash Player version. If the version requested is later than the version provided by Acrobat, the Navigator must not be loaded.

<navigator> child elements

A <navigator> element may also have the following child elements.

The first three elements, <name>, <description>, and <category>, have the same structure. If one of these element contains CharData, this data specifies non-localized text. However, if the element has an sID attribute, the value of this attribute must be an ID. The ID points to a string in the string table defined in the strings file. If the ID is not present in the string table, then the CharData contained in the element is used. The use of the sID attribute enables a Navigator to support multiple locales. The value of the sID attribute must be a string that obeys the rules for XML 1.0 name tokens.

- name—(Optional) A name displayed to the user when choosing a Navigator.
- description—(Optional) A description of the Navigator available to the user when choosing a Navigator.
- category—(Optional) A category name used to group Navigators.
- icon—(Optional) The location of a bitmap that represents the Navigator, expressed as a URI relative to navigator.xml. The <icon> element, if present, must include an src attribute that specifies the icon location as a relative URI
- locales—(Optional) The locales supported by this Navigator, and the location of the text strings associated with each locale.

  The <locales> element specifies the location of a locales file or includes locale information directly. The src attribute, if present, specifies the locales file location as a relative URI. The contents of the locales file is described below. If the src attribute is present, the <locales> element should have no children. Otherwise, the <locales> element may have one or more child <locale> elements. The <locale> element can appear as a <locales> child or in a locales file.
- initialView—(Optional) The initial layout of the view that contains the Navigator, which determines whether only the Navigator is shown or a preview is shown as well.

<initialView> has two optional attributes:

- splitDirection—(Optional) The direction of the split between the Navigator and the preview. Its value must be one of horizontal, vertical, or none. The default value is none, in which case the preview is not visible. The Navigator portion of the window is on the top or left if the window is split.
- splitPosition—(Optional) The initial position of the splitter bar, specified as a percentage of the available window area provided to the Navigator. Values can range from 0 to 100. If this attribute is not specified, an initial position is automatically chosen. This attribute is ignored if the value of splitDirection is none.

- initialFields—(Optional) The collection fields that the Navigator expects.

  The <initialFields> element contains zero or more <field> elements. If any of these fields do not exist in the portfolio schema, they are added in the specified order. Existing fields are reordered if necessary to conform to the order of the <field> elements. Existing fields not listed as <field> elements are moved to the end of the portfolio schema.

  A field is uniquely identified by its name, specified by the name attribute of the <field> element. Names are arbitrary Unicode strings, but they are not localized and are not displayed to the user. The name displayed to the user is specified by the <displayName> child element of a <field> element.

  The <field> element includes the following attributes.

  - name—(Required) A unique name that identifies the field. The name is an arbitrary Unicode string, and is not restricted to the set of characters in an XML name.
  - type—(Optional) The type of the field value, corresponding to the values of the Subtype field in a PDF collection field dictionary.

    The allowed values are text, date, number, or one of the following types corresponding to embedded file meta data: desc (file description), modDate (file modification date), creationDate (file creation date), size (file size). The default value is text.
  - visible—(Optional) Indicates whether the field is initially visible, specified as a boolean value. The default value is true.
  - editable—(Optional) Whether the user can edit the value of this field for each file, specified as a boolean value. The default value is false.

  The <field> element may have the an optional child element: displayName. This is the name displayed to the user when fields are visible. <displayName> has the same structure and attribute as the <name> element that is a child of <navigator>. The default value is the field name.

Navigator SWF file

The navigator SWF file manages the display of a PDF Portfolio's contents. It may also provide customization support for the Navigator. A navigator SWF file may refer to resources using URIs. However, only relative URIs, which reference other files in the Navigator, may be used.

Icon file

The icon file is a small image used to visually represent the Navigator in Acrobat. Both PNG and JPEG images are supported, but PNG is the recommended format for this icon. The recommended size is 42 x 42 pixels.

Locales file

The following is an example of a locales file.

```
<locales xmlns="http://ns.adobe.com/pdf/navigator/locales/2007">
  <locale id="en_US" strings="strings/en_US/strings.asfx"/>
  <locale id="fr_FR" strings="fr_FR.asfx"/>
  <locale id="de_DE" strings="de_DE.asfx"/>
</locales>
```

The <locales> element and its child <locale> elements may be placed in the navigator.xml file or in a separate locales file. If the <locales> element is not present, Acrobat assumes the Navigator is localized for Acrobat's current locale. It is strongly recommended that the Navigator locale be explicitly specified. This allows Acrobat to distinguish between two Navigators that have the same Navigator ID but are localized for different locales.

Using the <locales> element and associated string tables provides localized text for strings directly accessed by Acrobat: the name, description, and category of the Navigator and the user-visible name of initial fields. A Navigator should provide name, description, and category strings for each locale supported by the Navigator SWF file. The Navigator may also be used to store additional localized strings.

The <locales> element contains zero or more child <locale> elements, and must contain the xmlns attribute. The <locale> element must include the following attributes.

- id —(Required) A string that specifies a locale according to Unicode Technical Standard #35. The canonical form of a locale ID uses a space as a separator but, as recommended, a processor should treat a hyphen (-) as equivalent to a space.
- strings—(Required) The location of the file that contains the strings defined for this locale, expressed as a URI relative to the file that contains this element (either navigator.xml or a locales file).

Strings file

The <name>, <description>, and <category> elements in the <navigator> element and the <displayName> element in a <field> element in navigator.xml may include sID attributes. The value of an sID attribute is an ID of a string in the strings file. When Acrobat loads a navigator (*.nav) file, it searches for the Navigator's name, description, and category that match Acrobat’s current locale. If no locale-specific value is found, it uses the element’s CharData as a default value, if there is one.

The navigator SWF file can also refer to strings in a localized strings file using the ActionScript API function INavigatorHost.getLocalizedString(). The behavior of this function is system-dependent when there is no value for an ID in the current locale and no default value.

A strings file is an XML file that associates string IDs with localized strings. String IDs are strings that are limited to the alphanumeric characters: [a-z, A-Z, 0-9]. The only supported format for the strings file is the Adobe Strings Format.

The following is an a example of a strings file.

```
<?xml version="1.0" encoding="UTF-8"?>
<asf version="1.0" xmlns="http://ns.adobe.com/asf" locale="fr_FR">
    <str name="sampleNavName">
        <val>nom français</val>
    </str>
    <str name="sampleNavDescription">
        <val>description français</val>
    </str>
    <str name="sampleNavCategory">
        <val>catégorie français</val>
    </str>
</asf>
```

Acrobat X supports a subset of ASF 1.0 for specifying localized strings. This subset includes only three elements: <asf>, <str>, and <val>. Other ASF 1.0 elements are ignored.

The <asf> element is the root element. It has zero or more <str> child elements. Child elements other than <str> are ignored. The <asf> element has the following attributes.

- version—(Required) The version of ASF used. It must be 1.0.
- xmlns—(Required) The ASF namespace. It must be "http://ns.adobe.com/asf"
- locale—(Required) A string that specifies a locale according to Unicode Technical Standard #35. This locale must match the locale specified by the id attribute of the <locale> element whose strings attribute refers to this strings file.

The <str> element has a name attribute whose value is a string ID. The <str> element must have one <val> child element. If a <str> element has more than one child <val> element, the text associated with the first child is used.

Resource files

The navigator SWF file can refer to other files. Any files other than types that are not allowed as PDF file attachments can be included and referred to by the Navigator, including images, video, and text files.

Relax NG Compact Schemas

This section lists the Relax NG Compact Schema for navigator.xml, the locales file, and the strings file.

Navigator.xml schema

The Relax NG Compact Schema for navigator.xml file is given below.

```
namespace xsd="http://www.w3c.org/2001/XMLSchema-datatypes"
default namespace pdf="http://ns.adobe.com/pdf/navigator/2007"
grammar {
start =
     element navigator { navigatorContent }
navigatorContent =
    attribute src { xsd:anyURI }&
    attribute id { xsd:anyURI } ? &
    attribute loadAs { "default" | "module" } ? &
    attribute version { versionNumber } ? &
    attribute apiVersion { versionNumber } &
    element name { localizableString } ? &
    element description { localizableString } ? &
    element category { localizableString } ? &
    element icon { attribute src { xsd:anyURI } } ? &
    element locales { localesContent } ? &
    element initialFields { initialFieldsContent } ? &
    element initialView { initialViewContent } ?
versionNumber =
    xsd:decimal { pattern="[0-9]+(\.[0-9]+){0,3}" }
localizableString =
    attribute sID { text } ? ,
    text
localesContent =
    attribute src { xsd:anyURI } |
    element locale { localeContent } *
localeContent =
    attribute id { localeID } &
    attribute strings { xsd:anyURI }
localeID =
    xsd:token { pattern="[a-zA-Z]{1,8}(_[a-zA-Z0-9]{1,8})*" }
initialFieldsContent =
    element field { fieldContent } *
fieldContent=
    attribute name { text },
    attribute type { "text" | "date" | "number" | "filename" | "desc" |
             "modDate" | "creationDate" | "size" } ? ,
    attribute visible { xsd:boolean } ? ,
    attribute editable { xsd:boolean } ? ,
    element displayName { localizableString } ?
initialViewContent =
    attribute splitDirection { "none" | "horizontal" | "vertical" } ? ,
    attribute splitPosition { xsd:int { minInclusive="0" maxInclusive="100" } } ?
}
```

Locales.xml schema

The Relax NG Compact Schema for the locales.xml file is given below.

```
namespace xsd="http://www.w3c.org/2001/XMLSchema-datatypes"
default namespace pdf="http://ns.adobe.com/pdf/navigator/2007"
grammar {
start =
    element locales { localesContent }
localesContent =
    attribute src { xsd:anyURI } |
    element locale { localeContent } *
localeContent =
    attribute id { localeID } &
    attribute strings { xsd:anyURI }
localeID =
    xsd:token { pattern="[a-zA-Z]{1,8}(_[a-zA-Z0-9]{1,8})*" }
}
```

Strings.asfx schema

The Relax NG Compact Schema for the strings.asfx file is given below. The strings schema is a subset of the ASF schema

```
namespace xsd="http://www.w3c.org/2001/XMLSchema-datatypes"
default namespace pdf="http://ns.adobe.com/asf"
grammar {
start =
     element asf { asfContent }
asfContent =
    attribute locale { localeID } &
    attribute version { "1.0" } &
    element str { strContent } *
localeID =
    xsd:token { pattern="[a-zA-Z]{1,8}(_[a-zA-Z0-9]{1,8})*" }
strContent =
    attribute name { text },
    element val { text }
}
```

Chapter 10: Universal Container Format

A Navigator file is a Universal Container Format (UCF) document. This section introduces the UCF specification and the conventions it uses.

Introduction

This section covers the purpose, scope, relationship to other specifications, and future directions for the UCF specification.

Purpose and scope

The UCF specification defines the Universal Container Format. UCF is a general-purpose container technology. It is based on the packaging principles of OCF, the OEBPS Container Format created by the International Digital Publishing Forum. The OCF specification describes a general-purpose container technology for encapsulating OEBPS publications. The OCF specification anticipates that the general-purpose container technology that it describes will ultimately be used in other bundling applications. However, the specification itself does not formally separate the generic technology from its use in the context of OEBPS. The goal of this specification is to specify a generic container format that can be used by many applications, including OCF.

As a general container format, UCF collects a related set of files into a single-file container. UCF can be used to collect files in various document and data formats and for various classes of applications. The single-file container makes it easy to move, manage, and randomly access the collection.

UCF defines rules for how to represent an abstract collection of files (the abstract container) into a physical representation within a Zip archive (the physical container). The rules for Zip containers build upon and are backward compatible with the Zip technology used by Open Document Format (ODF) 1.0.

UCF is the recommended single-file container technology for all Zip-based Adobe formats. It is designed to provide a set of lightweight constraints on the use of Zip. UCF includes a set of optional features, including digital signatures and encryption. If a UCF-based file format includes this optional functionality, it should follow the UCF specifications for use of these features. For example, not all UCF-based formats use digital signatures. However, if a format does include support for signatures, it should follow the UCF rules for signatures.

This specification borrows heavily from the OCF specification. Where possible, the same language is used, with the permission of the IDPF.

Definitions

ASCII American Standard Code for Information Interchange – a 7-bit character encoding based on the English alphabet (ANSI X3.4-1986). When used in this document, ASCII refers to the printable graphic characters in the range 33 (decimal) through 126 (decimal) and the non-printable space character 32 (decimal).

IRI Internationalized Resource Identifier; for a discussion and specification of IRIs, go to http://www.ietf.org/rfc/rfc3987.txt.

Mars An XML representation of PDF.

MIME Multipurpose Internet Mail Extensions; a discussion of the MIME format can be found at http://www.ietf.org/rfc/rfc2045.txt. MIME media types provide a standard methodology for specifying the content type of objects.

ODF Open Document Format; the specification for ODF can be found at http://www.oasis-open.org/committees/download.php/12572/OpenDocument-v1.0-os.pdf.

OEBPS Open eBook Publication Structure; the specification for OEBPS can be found at http://www.idpf.org/oebps/oebps1.2/index.htm.

Relax NG A schema language for XML; for further information, go to the Relax NG website: http://www.relaxng.org/.

RFC Literally Request For Comments, but more generally a document published by the Internet Engineering Task Force (IETF). For more information, see the RFC page on the IETF website: http://www.ietf.org/rfc.html.

Rootfile The top-level file of a rendition of a publication, either the root from which all other components can be found or the lone file encapsulating the rendition. The OEBPS root file is the OEBPS Package file. A PDF file that contains the PDF rendition can also be a root file.

UCF The Universal Container Format defined by this specification

UCF Container A container file that is compliant with the format defined in this specification.

UCF User Agent A combination of hardware and software that accepts documents or data packaged in a UCF Container and makes them available to consumers of the content.

XML Extensible Markup Language. For the specification for XML go to http://www.w3.org/TR/xml/.

Zip The industry standard for file bundling and compression. For information on the zip file format see http://www.pkware.com/support/zip-application-note.

Relationship to other specifications

This specification combines subsets and applications of other specifications. Together, these facilitate the construction, organization, presentation, and unambiguous interchange of electronic documents.

- The OEBPS Container Format specification (http://www.idpf.org/ocf/ocf1.0)
- Zip format (http://www.pkware.com/support/zip-application-note)
- The Extensible Markup Language (XML) 1.0 (Fourth Edition) specification (http://www.w3.org/TR/xml/)
- The Namespaces in XML 1.0 (Second Edition) specification (http://www.w3.org/TR/xml-names/)
- XML-Signature Syntax and Processing (http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/)
- XML Encryption Syntax and Processing (http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/)
- Extensible Metadata Platform (XMP) (http://www.adobe.com/products/xmp/)
- The Unicode Consortium. The Unicode Standard, Version 5.0.0, defined by: The Unicode Standard, Version 5.0 (Boston, MA, Addison-Wesley, 2007, ISBN 0-321-48091-0), as updated from time to time by the publication of new versions. (See http://www.unicode.org/unicode/standard/versions for the latest version and additional information about versions of the standard and of the Unicode Character Database).
- Particular MIME media types (http://www.iana.org/assignments/media-types/index.html)
- Open Document Format for Office Applications (Open Document) v1.0 (http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=office)

Conformance requirements

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, RECOMMENDED, MAY, and OPTIONAL in this document must be interpreted as described in http://www.ietf.org/rfc/rfc2119.txt.

Conforming Containers

The term Conforming UCF Abstract Container indicates a UCF Abstract Container (see "Abstract container vs. physical container" on page 29) that conforms to all the relevant conformance criteria defined in this specification. The term Conforming UCF Zip Container indicates a Zip archive that conforms to the relevant Zip container conformance criteria and that implements an instance of a Conforming UCF Abstract Container.

In addition to other conformance criteria defined in this specification, a Conforming UCF Abstract Container MUST meet the following conditions.

- All XML files defined by this specification MUST be well-formed (as defined in XML 1.0).
- All XML files defined by this specification MUST be compatible with the XML 1.0 specification (http://www.w3.org/TR/2006/REC-xml-20060816) and the Namespaces in the XML 1.0 specification (http://www.w3.org/TR/REC-xml-names).

These conditions do not apply to files in the container that are not defined by this specification (files other than container.xml, manifest.xml, metadata.xml, signatures.xml, encryption.xml, and rights.xml).

Conforming user agents

The term Conforming UCF User Agent indicates a UCF User Agent that supports all the mandatory features defined by this specification.

A UCF User Agent that does not support all the features defined in this specification MUST NOT claim to be a Conforming UCF User Agent and SHOULD provide readily available documentation of the subset of features it supports.

Future Directions

It is the intent of the contributors to this specification that subsequent versions of this specification continue in the directions established by the 1.0 release, as follows:

- Future versions of this specification are expected to improve alignment with OASIS/ODF and IDPF/OCF.
- Any required functionality not present in relevant official standards shall be defined in a manner consistent with its eventual submission to an appropriate standards body as an extension to existing standards.

UCF overview

UCF is designed as a general container technology. In particular, UCF is designed to be upwardly compatible with the container technology used in ODF 1.0 such that a future version of ODF may use UCF.

Abstract container vs. physical container

An *abstract container* defines a file system model for the contents of the container. The file system model MUST have a single common root directory for all the contents of the container. The special files REQUIRED by UCF MUST be included in the META-INF directory that is a direct child of the root directory.

A *physical container* holds the physical manifestation of an abstract container. This specification defines how an abstract container MUST be mapped to the following two physical container technologies.

- *File System Container*: The mapping of an abstract container to a file system within computer storage media on a specific platform, such as a hard disk on a computer or a data CD, MUST be a one-to-one mapping. Each directory and file within the abstract container is represented as a directory or file within the file system. "File names" on page 33 defines a set of restrictions on file system names intended to allow files to be easily stored in most modern file systems.
- *Zip container*: The mapping of an abstract container to a Zip archive is defined in "Zip container" on page 38.

If a user agent processed both types of physical container, the contents of an OCF container MUST be processed the same whether it uses a File System Container or a Zip container. In both cases, the UCF User Agent ultimately opens the root file, from which it can determine how to process the container.

Example of an OEBPS file

To illustrate the concepts from the previous section, assume we have a single OEBPS 1.0 document of Dickens' Great Expectations. It consists of an OEBPS 1.2 package file (Great Expectations.opf) and many HTML files, one for the cover page (such as cover.html) and one for each chapter (such as chapter01.html). The contents of the publication may be as follows.

```
OEBPS 1.2 Publication:
Great Expectations.opf
cover.html
chapters/
   chapter01.html
   chapter02.html
   … other HTML files for the remaining chapters …
```

The contents of the abstract container include all the assets from the Publication plus a few files defined by UCF in the META-INF directory. Note that the container.xml file is REQUIRED by OCF even though it is not required by UCF. See "META-INF" on page 34for descriptions of the files in the META-INF directory.

```
Abstract Container:
mimetype
META-INF/
  container.xml
  [manifest.xml]
  [metadata.xml]
  [signatures.xml]
  [encryption.xml]
  [rights.xml]
OEBPS/
  Great Expectations.opf
  cover.html
  chapters/
  chapter01.html
  chapter02.html
  … other HTML files for the remaining chapters …
```

When this abstract container is mapped to the following File System Container, the directory structure within the file system exactly matches the UCF Abstract Container directory structure shown above.

```
File System Container:
…some directory within the file system…/
mimetype
META-INF/
   container.xml
   [manifest.xml]
   [metadata.xml]
   [signatures.xml]
   [encryption.xml]
   [rights.xml]
OEBPS/
  Great Expectations.opf
   cover.html
   chapters/
  chapter01.html
  chapter02.html
  … other HTML files for the remaining chapters …
```

When the abstract container is stored in a Zip container, the contents of the Zip archive match the directory structure previously shown. When the Zip archive is created, the mimetype file must be the first file in the archive. [See "Container media type identification" on page 34.]

```
Zip container:
mimetype
META-INF/
  container.xml
  [manifest.xml]
  [metadata.xml]
  [signatures.xml]
  [encryption.xml]
  [rights.xml]
OEBPS/
  Great Expectations.opf
  cover.html
  chapters/
  chapter01.html
  chapter02.html
  … other HTML files for the remaining chapters …
```

The mimetype file contains the string *application/epub+zip*.

The corresponding META-INF/container.xml file may appear as follows.

```
<?xml version="1.0"?>
<container version="1.0"
  xmlns="urn:oasis:names:tc:opendocument:xmlns:container">
   <rootfiles>
      <rootfile full-path="OEBPS/Great Expectations.opf"
       media-type="application/oebps-package+xml" />
   </rootfiles>
</container>
```

Note: *The use of the specific namespace string urn:oasis:names:tc:opendocument:xmlns:container should be considered provisional until approved by an OASIS technical committee*

Example of a Mars file

A Mars file, unlike an OEBPS file, does not require the container.xml file to include a <rootfiles> element. The root file is always named backbone.xml. The example file does not include digital signatures, encrypted data, or DRM restrictions and therefore does not include the signatures.xml, encryption.xml, or rights.xml files.

```
Abstract Container:
mimetype
META-INF/
     container.xml
    [manifest.xml]
    [metadata.xml]
backbone.xml
page/0/
    info.xml
    pg.svg
    im_1.png
    f_1.fd
fonts/
f_1._sfnt
```

The corresponding META-INF/container.xml file may appear as follows.

```
<?xml version="1.0"?>
<container version="1.0"
    xmlns="urn:oasis:names:tc:opendocument:xmlns:container">
    <relationships xmlns:pdf="http://ns.adobe.com/pdf/2006">
    <relationship type="metadata" target="$path.xmp"/>
    <relationship type="pdf:annotation" target="$path.ann"/>
    </relationships>
 </container>
```

A Mars container.xml file includes relationship elements for meta data and annotations.

UCF container contents

This section provides the technical specification for the contents of UCF container.

File and directory structure

The virtual file system for the UCF Abstract Container MUST have a single common root directory for all the contents of the container.

The following file names in the root directory are reserved:

- mimetype
- META-INF

The mimetype file is discussed in "Zip container" on page 38. The META-INF/ directory contains the reserved files used by UCF. These reserved files are described in the following sections. All other files in the abstract container MAY be in any location descendant from the root directory except for mimetype at the root level or directly within the META-INF directory. A UCF-based format may include files in the META-INF directory as long as these files are in subdirectories in META-INF. The names of these subdirectories MUST NOT include the period character (.). This avoids conflicts with files specified by future versions of the UCF specification. Any file in the META-INF directory used by UCF will include a period character (.).

It is RECOMMENDED that the contents of individual documents or applications be stored in dedicated subdirectories. This minimizes potential file name collisions if multiple renditions are used or if multiple publications per container are supported in future versions of this Specification.

Relative IRIs for referencing other components

Files in the abstract container reference each other by using Relative IRI References (http://www.ietf.org/rfc/rfc3987.txt), whether the physical container is a File System Container or a Zip container. For example, if a file named chapter1.html references an image file named image1.jpg that is located in the same directory, chapter1.html may contain the following as part of its content:

```
<img src="image1.jpg" alt="…" …/>
```

For Relative IRI References, the Base IRI (see RFC 3986) is determined by the relevant language specifications for the given file formats. For example, the CSS specification defines how relative IRI references work in the context of CSS style sheets and property declarations.

Unlike many language specifications, the Base IRIs for all files in the META-INF/ directory use the root folder for the abstract container as the default Base IRI. For example, the META-INF/container.xml file has the following content:

```
<?xml version="1.0"?>
<container version="1.0"
    xmlns="urn:oasis:names:tc:opendocument:xmlns:container">
    <rootfiles>
        <rootfile full-path="OEBPS/Great Expectations.opf"
         media-type="application/oebps-package+xml" />
    </rootfiles>
</container>
```

The path OEBPS/Great Expectations.opf is relative to the root directory for the abstract container and not relative to the META-INF/ directory.

If a Relative IRI Reference contains an absolute path (an IRI that has no schema or authority but begins with a slash (/)), the reference is resolved relative to the root directory of the abstract container. For example, in "Example of an OEBPS file" on page 30, the IRI /OEBPS/cover.html refers to the file OEBPS/cover.html no matter what file the IRI is found in.

File names

The term file name represents the name of any type of file, either a directory or an ordinary file in a directory in an abstract container. For a given directory in the abstract container, the term path name is the string holding all directory names in the full path concatenated with a slash character (/) separating the directory names. For a given file in the abstract container, the term path name is the string holding all directory names concatenated with a slash character (/) separating the directory names, followed by a slash character (/) and then the name of the file. File name restrictions are designed so that directory names and file names can be used without modification on most commonly used operating systems. This specification does not specify how a UCF User Agent that is unable to represent UCF conforming file names would compensate for this incompatibility.

The following restrictions apply to Conforming UCF Content.

- File names MUST be UTF-8 encoded with the restrictions below.
- When represented as UTF-8, file names MUST NOT exceed 255 bytes.
- When represented as UTF-8, the path name for any directory or file in the abstract container MUST NOT exceed 65535 bytes.

- File names MUST NOT use the following characters, which are not always supported across commonly used operating systems:
  - U+0022 " QUOTATION MARK
  - U+002A * ASTERISK
  - U+002E . FULL STOP, as the last character
  - U+002F / SOLIDUS
  - U+003A : COLON
  - U+003C < LESS-THAN SIGN
  - U+003E > GREATER-THAN SIGN
  - U+003F ? QUESTION MARK
  - U+005C \ REVERSE SOLIDUS
  - The C0 controls, U+0000 through U+001F and U+007F
- File names are case sensitive.
- Two file names in the same directory MUST NOT map to the same string following case normalization (http://www.unicode.org/reports/tr21/tr21-5.html). Two file names that differ only in case are not allowed in the same directory.
- Two file names in the same directory MUST NOT be canonically equivalent in the Unicode sense.

Note that some commercial Zip tools do not support the full Unicode range and may support only the ASCII range for file names. Content creators who want to use Zip tools that have these restrictions MAY find it is best to restrict their file names to the ASCII range. If the names of files cannot be preserved during the unzipping process, it is necessary to compensate for any name translation that took place when the files are referenced by URI from within the content.

Container media type identification

It is frequently necessary for applications to determine the media type of a file. This is usually accomplished by looking at the file extension, which gives applications a quick way to determine the type of the file without looking inside the file. UCF Container files SHOULD use an extension specific to its type of UCF Container.

Unfortunately, the identification of files through the use of file extensions is notoriously unreliable. As a result, it is desirable to have a more robust way of identifying files independent of their file names or extensions. One mechanism for doing this is to require the placement of specific information at specific file offsets. A processing agent can then check a fixed location to determine if the file is a specific type of UCF Container.

The method for doing this in Zip archives is to include an uncompressed, unencrypted file called mimetype as the first file in the Zip archive. The contents of this file are the media type of the file. UCF Containers MUST place the media type as an ASCII string in the mimetype file as the first file in the Zip archive. See “Zip container” on page 38 for more information about this mechanism.

META-INF

All valid UCF Containers MAY include a directory named META-INF at the root level of the container file system. This directory contains files that describe the contents, meta data, signatures, encryption, rights, and other information about the contained publication.

The semantics of the following files that MAY be present at the META-INF/ level are specified. All other files found at the META-INF/ level MUST be ignored by conforming UCF User Agents.

Container (META-INF/container.xml, Optional)

A UCF Container MAY include a file named container.xml in the META-INF directory at the root level of the container file system. If present, the container.xml file MAY identify the MIME type of, and path to, the root file for the container and any OPTIONAL alternative renditions included in the container. A UCF-based format MUST either require the container.xml file to identify the root file or specify a format-specific method for initiating processing of the container. The container.xml file MAY specify implicit relationships in the container, as described in "Relationships (Optional)" on page 36.

The container.xml file MUST NOT be encrypted.

The container.xml file contains XML that uses the urn:oasis:names:tc:opendocument:xmlns:container namespace for all its elements and attributes. The version="1.0" attribute MUST be included for all containers that conform to this version of the specification.

A RELAX NG UCF schema describing the <container> element that MUST be the root element in the container.xml file can be found in the "RELAX NG UCF schema" on page 39.

Rootfiles (Optional)

The <rootfiles> element MUST contain at least one <rootfile> element.

Each <rootfile> element specifies the root file of a single rendition of the contained publication. A root file often includes an enumeration of the other files needed by the rendition.

The following example shows a sample container.xml file for a UCF container inside of which is an OEBPS Publication with the root file "OEBPS/My%20Crazy%20Life.opf" (the OEBPS package file).

```
<?xml version="1.0"?>
<container version="1.0"
  xmlns="urn:oasis:names:tc:opendocument:xmlns:container">
    <rootfiles>
        <rootfile full-path="OEBPS/My%20Crazy%20Life.opf"
     media-type="application/oebps-package+xml" />
    </rootfiles>
</container>
```

The following example adds an alternative PDF version of the Publication.

```
<?xml version="1.0"?>
<container version="1.0"
  xmlns="urn:oasis:names:tc:opendocument:xmlns:container">
    <rootfiles>
      <rootfile full-path="OEBPS/My%20Crazy%20Life.opf"
           media-type="application/oebps-package+xml" />
      <rootfile full-path="PDF/My%20Crazy%20Life.pdf"
           media-type="application/pdf" />
    </rootfiles>
</container>
```

The values of the full-path attributes MUST contain a path component (as defined by RFC3986) that MUST take the form only of a path-rootless (as defined by RFC3986). The path components are relative to the root of the container in which they are used.

Conforming UCF User Agents MUST ignore unrecognized elements (and their contents) and unrecognized attributes in a container.xml file, including unrecognized elements and unrecognized attributes from other namespaces.

Conforming container.xml files MUST be valid according to the RELAX NG UCF schema with the <container> element as the root element after removing all elements (and child nodes of these elements) and attributes from other namespaces.

For example, the following container.xml file is conforming.

```
<?xml version="1.0"?>
<container version="1.0"
    xmlns="urn:oasis:names:tc:opendocument:xmlns:container"
    foo:xmlns=" … "
    foozle:xmlns=" … " />
    <foo:bar />
    <rootfiles foozle:identifier="bar">
    …
    </rootfiles>
</container>
```

But the following container.xml file is non-conforming due to the fact that the <foo> element is not namespace-qualified.

```
<?xml version="1.0"?>
<container version="1.0"
    xmlns="urn:oasis:names:tc:opendocument:xmlns:container">
    <foo />
    <rootfiles>
     …
    </rootfiles>
</container>
```

Likewise, the following container.xml file is also non-conforming due to the fact that the identifier attribute on the <rootfiles> element is not namespace-qualified.

```
<?xml version="1.0"?>
<container version="1.0"
    xmlns="urn:oasis:names:tc:opendocument:xmlns:container">
    <rootfiles identifier="bar">
     …
    </rootfiles>
</container>
```

Relationships (Optional)

The container.xml file MAY include information that identifies implicit relationships between files in a container. Usually, one file explicitly references another file. For example, an SVG page description may reference an image. At times, it is convenient to indirectly reference a file. For example, one file may have meta data associated with it stored in a second file. Using an indirect association makes it possible to add meta data by simply creating a meta data resource and not changing the original resource or any resource in the package.

Beyond convenience, indirect associations enable container files to be digitally signed while still permitting certain kinds of data, such as meta data or annotations, to be added. For example, a document with no annotations can be signed. Annotations can then be added without invalidating the signature.

UCF provides a generic mechanism for establishing a relationship between two container files. A relationship specifies a relationship type and a mapping from a set of source names to target names. For any given file, this relationship can be used to determine the related file. However, UCF does not require that the related file exist.

The root <container> element MAY contain a child <relationships> element. This element MAY contain one or more child <relationship> elements that describe specific relationships. Each <relationship> element includes the attributes type and target that specify a relationship type and a pattern that maps source to target names. The type is a qualified name. UCF defines one relationship type, meta data. UCF-based formats can add other types by specifying a namespace. The target pattern is a string that may include the following variables that are substituted when a relationship is resolved.

- variable: *path* definition: full path to the resource, including the file name Example: /a/b/c.d
- variable: *dir* definition: folder path without the file name Example: /a/b
- variable: *filename* definition: file name plus its extension Example: c.d
- variable: *basename* definition: file name without its extension Example: c
- variable: *ext* definition: file name extension Example: d

These variables are specified by enclosing their name in braces and preceding the opening brace with a dollar sign ($). Braces may be omitted if the first character after the variable name is not a letter. A target is resolved by copying the ordinary text in the pattern and replacing the variables with their values.

Here is an example.

```
<?xml version="1.0"?>
<container version="1.0"
    xmlns="urn:oasis:names:tc:opendocument:xmlns:container">
    <relationships xmlns:pdf="http://ns.adobe.com/pdf/2006">
        <relationship type="metadata" target="$path.xmp"/>
        <relationship type="pdf:annotation" target="$path.ann"/>
    </relationships>
</container>
```

The first relationship specifies that any file may have associated meta data. This meta data is found in a file that has the same name as the original with .xmp added as an extension. For example, /page/0/pg.svg may have meta data in /page/0/pg.svg.xmp. The second relationship specifies that any file may have associated annotations. The annotations are found in a file that has .ann added as an extension.

Manifest (META-INF/manifest.xml, Optional)

An OPTIONAL file with the reserved name manifest.xml in the META-INF directory at the root level of the container file system may appear in a valid UCF container. If present, the file's content MUST be as defined in the ODF 1.0 manifest schema.

The manifest.xml file, if present, MUST NOT be encrypted.

Metadata (META-INF/metadata.xml, Optional)

An OPTIONAL file with the reserved name metadata.xml in the META-INF directory at the root level of the container file system may appear in a valid UCF container. This file, if present, MUST be used for container-level meta data. In version 1.0 of OCF, no such container-level meta data is specified.

If the META-INF/metadata.xml file exists, its contents MUST be valid XML with namespace-qualified elements to avoid collision with future versions of OCF that MAY specify a particular grammar and namespace for elements and attributes in this file.

Adobe-defined formats based on UCF MUST use XMP to specify meta data.

Digital signatures (META-INF/signatures.xml, Optional)

An OPTIONAL file named signatures.xml in the META-INF directory at the root level of the container file system holds digital signatures of the container and its contents. The contents of this file are not specified in UCF 1.0. However, a future revision of UCF will define the format for this file. See "Digital signatures" on page 45 for the likely definition.

Encryption (META-INF/encryption.xml, Optional)

An OPTIONAL file named encryption.xml in the META-INF directory at the root level of the container file system holds all encryption information about the contents of the container. The content of this file is not specified in UCF 1.0. However, a future revision of UCF will define the format for this file. See "Encryption" on page 46 for the likely definition.

Rights management (META-INF/rights.xml, Optional)

An OPTIONAL file with the reserved name rights.xml in the META-INF directory at the root level of the container file system may appear in a valid UCF container. This location is reserved for digital rights management (DRM) information for trusted exchange of Publications among rights holders, intermediaries, and users. In version 1.0 of UCF, there is not a REQUIRED format for DRM information, but a future version of this specification MAY specify a particular format for DRM information.

If the META-INF/rights.xml file exists, it MUST be a well-formed XML document that uses and conforms to XML Namespaces that it uses, and its contents SHOULD be valid XML with namespace-qualified elements to avoid collision with future versions of UCF that MAY specify a particular format for this file.

The rights.xml file MUST NOT be encrypted.

When the rights.xml file is not present, the UCF container provides no information that indicates that any part of the container is rights governed.

Zip container

UCF's Zip container supports the Zip format as specified by the application note at http://www.pkware.com/support/zip-application-note, but with the following constraints and clarifications:

- Conforming UCF Zip containers MUST NOT use the features in the Zip application note that allow Zip files to be split across multiple storage media. Conforming UCF User Agents MUST treat any UCF files that specify that the Zip file is split across multiple storage media as being in error.
- Conforming UCF Zip containers MUST only include uncompressed files or Flate-compressed files in the Zip archive. Conforming UCF User Agents MUST treat any UCF Containers that use compression techniques other than Flate as being in error.
- Conforming UCF Zip containers MAY use the Zip64 extensions and SHOULD only use those extensions when the content requires them. Conforming UCF User Agents MUST support the Zip64 extensions.
- Conforming UCF Zip containers MUST NOT use the encryption features defined by the Zip format. Instead, encryption MUST be done using the features described in "Encryption" on page 46. Conforming UCF User Agents MUST treat any other UCF Zip containers that use Zip encryption features as being in error.

- It is not required that conforming UCF User Agents preserve information from a UCF Zip container through load and save operations that do not map to a corresponding representation in the UCF Abstract Container. In particular, a conforming UCF User Agent does not have to preserve CRC values, comment fields, or fields that hold file system information corresponding to a particular operating system, such as the fields named External file attributes and Extra field.
- Conforming UCF Zip containers MUST encode file system names using UTF-8.

Note the following details about particular fields in the Zip archive.

- On the local file header table, conforming UCF Zip containers MUST set the version needed to extract fields to the values 10, 20, or 45 to match the maximum version level needed by the given file, such as 20 if Deflate is needed and 45 if Zip64 is needed. Conforming UCF User Agents MUST treat any other values as being in error.
- On the local file header table, conforming UCF Zip containers MUST set the compression method field to the values 0 or 8. Conforming UCF User Agents MUST treat any other values as being in error.
- Conforming UCF User Agents MUST treat UCF Zip containers with an Archive decryption header or an Archive extra data record as being in error.

The first file in the Zip container MUST be a file with the ASCII name of mimetype, which holds the MIME type for the Zip container (application/epub+zip as an ASCII string; no padding, white-space, or case change). The file MUST be neither compressed nor encrypted, and there MUST NOT be an extra field in its Zip header. If this is done, the Zip container offers convenient magic number support as described in RFC 2048, and the following holds true:

- The bytes for PK are at the beginning of the file.
- he bytes for mimetype are at position 30.
- The actual MIME type (the ASCII string application/epub+zip) begins at position 38.

RELAX NG UCF schema

Here is a RELAX NG schema that defines the grammar of a <container> element.

```
<?xml version="1.0" encoding="UTF-8"?>
<choice xmlns="http://relaxng.org/ns/structure/1.0">
  <element name="container">
    <attribute name="version">
      <value>1.0</value>
    </attribute>
    <attribute name="xmlns">
      <value>urn:oasis:names:tc:opendocument:xmlns:container</value>
    </attribute>
    <optional>
      <element name="rootfiles">
        <oneOrMore>
          <element name="rootfile">
            <attribute name="full-path">
              <text/>
            </attribute>
            <attribute name="media-type">
              <text/>
            </attribute>
          </element>
        </oneOrMore>
      </element>
```

```
    </optional>
    <optional>
      <element name="relationships">
        <oneOrMore>
          <element name="relationship">
            <attribute name="type">
              <text/>
            </attribute>
            <attribute name="target">
              <text/>
            </attribute>
          </element>
        </oneOrMore>
      </element>
    </optional>
  </element>
  <element name="signatures">
    <attribute name="xmlns">
      <value>urn:oasis:names:tc:opendocument:xmlns:container</value>
    </attribute>
    <oneOrMore>
      <element name="Signature" ns="http://www.w3.org/2001/04/xmldsig#">
        <externalRef
          href="http://www.w3.org/Signature/2002/07/xmldsig-core-schema.rng"/>
      </element>
    </oneOrMore>
  </element>
  <element name="encryption">
    <attribute name="xmlns">
      <value>urn:oasis:names:tc:opendocument:xmlns:container</value>
    </attribute>
    <oneOrMore>
      <choice>
        <element name="EncryptedData" ns="http://www.w3.org/2001/04/xmlenc#">
          <externalRef
            href="http://www.w3.org/Encryption/2002/07/xenc-schema.rng"/>
        </element>
        <element name="EncryptedKey" ns="http://www.w3.org/2001/04/xmlenc#">
          <externalRef
            href="http://www.w3.org/Encryption/2002/07/xenc-schema.rng"/>
        </element>
      </choice>
    </oneOrMore>
  </element>
</choice>
```

Example

The following example demonstrates the use of the UCF format to contain a signed and encrypted OEBPS publication with an alternative PDF rendition in a Zip container.

Ordered list of files in the Zip container:

```
mimetype
META-INF/container.xml
META-INF/signatures.xml
META-INF/encryption.xml
OEBPS/As You Like It.opf
OEBPS/book.html
OEBPS/images/cover.png
PDF/As You Like It.pdf
```

The mimetype file:

```
application/epub+zip
```

The META-INF/container.xml file:

```
<?xml version="1.0"?>
<container version="1.0"
  xmlns="urn:oasis:names:tc:opendocument:xmlns:container">
  <rootfiles>
    <rootfile full-path="OEBPS/As You Like It.opf"
    media-type="application/oebps-package+xml" />
    <rootfile full-path="OEBPS/As You Like It.pdf"
      media-type="application/pdf" />
  </rootfiles>
</container>
```

The META-INF/signatures.xml file:

```
<?xml version="1.0"?>
<signatures xmlns="urn:oasis:names:tc:opendocument:xmlns:container">
  <Signature Id="AsYouLikeItSignature"
    xmlns="http://www.w3.org/2000/09/xmldsig#">
    <!-- SignedInfo is the information that is actually signed. -->
    <!-- In this case -->
    <!-- the SHA1 algorithm is used to sign the canonical form -->
    <!-- of the XML documents enumerated in the Object element below -->
    <SignedInfo>
      <CanonicalizationMethod
         Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
      <SignatureMethod
        Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
      <Reference URI="#AsYouLikeIt">
        <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</DigestValue>
      </Reference>
    </SignedInfo>
    <!-- The signed value of the digest above using the DSA algorithm -->
    <SignatureValue>MC0CFFrVLtRlk= … </SignatureValue>
    <!-- The key to use to validate the signature -->
    <KeyInfo>
      <KeyValue>
        <DSAKeyValue>
          <P> … </P>
          <Q> … </Q>
          <G> … </G>
          <Y> … </Y>
        </DSAKeyValue>
      </KeyValue>
    </KeyInfo>
```

```
    <!-- The list documents to sign. Note that the canonical form of XML -->
    <!-- documents is signed while the binary form of the other documents -->
    <!-- is used -->
    <Object>
      <Manifest Id="AsYouLikeIt">
        <Reference URI="OEBPS/As You Like It.opf">
          <Transforms>
            <Transform
             Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
          </Transforms>
        </Reference>
        <Reference URI="OEBPS/book.html">
          <Transforms>
            <Transform
             Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
          </Transforms>
        </Reference>
        <Reference URI="OEBPS/images/cover.png"/>
        <Reference URI="PDF/As You Like It.pdf"/>
      </Manifest>
    </Object>
  </Signature>
</signatures>
```

The META-INF/encryption.xml file:

```
<?xml version="1.0"?>
<encryption xmlns="urn:oasis:names:tc:opendocument:xmlns:container"
  xmlns:enc="http://www.w3.org/2001/04/xmlenc#"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <-- The RSA encrypted AES-128 symmetric key used to encrypt the data -->
  <enc:EncryptedKey Id="EK">
     <enc:EncryptionMethod
       Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
    <ds:KeyInfo>
      <ds:KeyName>John Smith</ds:KeyName>
    </ds:KeyInfo>
    <enc:CipherData>
      <enc:CipherValue>xyzabc … </enc:CipherValue>
    </enc:CipherData>
  </enc:EncryptedKey>
  <!-- Each EncryptedData block identifies a single document
        that has been encrypted using the AES-128 algorithm.
        The data remains stored in its encrypted form in the
        original file within the container.                -->
   <enc:EncryptedData Id="ED1">
    <enc:EncryptionMethod
     Algorithm="http://www.w3.org/2001/04/xmlenc#kw-aes128"/>
    <ds:KeyInfo>
      <ds:RetrievalMethod URI="#EK"
       Type="http://www.w3.org/2001/04/xmlenc#EncryptedKey"/>
    </ds:KeyInfo>
    <enc:CipherData>
      <enc:CipherReference URI="OEBPS/book.html"/>
    </enc:CipherData>
  </enc:EncryptedData>
  <enc:EncryptedData Id="ED2">
```

```
    <enc:EncryptionMethod
     Algorithm="http://www.w3.org/2001/04/xmlenc#kw-aes128"/>
    <ds:KeyInfo>
      <ds:RetrievalMethod URI="#EK"
       Type="http://www.w3.org/2001/04/xmlenc#EncryptedKey"/>
    </ds:KeyInfo>
    <enc:CipherData>
      <enc:CipherReference URI="OEBPS/images/cover.png"/>
    </enc:CipherData>
  </enc:EncryptedData>
  <enc:EncryptedData Id="ED3">
    <enc:EncryptionMethod
     Algorithm="http://www.w3.org/2001/04/xmlenc#kw-aes128"/>
    <enc:KeyInfo>
      <enc:RetrievalMethod URI="#EK"
       Type="http://www.w3.org/2001/04/xmlenc#EncryptedKey"/>
    </enc:KeyInfo>
    <enc:CipherData>
      <enc:CipherReference URI="PDF/As You Like It.pdf"/>
    </enc:CipherData>
  </enc:EncryptedData>
</encryption>
```

The OEBPS/As You Like It.opf file:

```
<?xml version="1.0"?>
<!DOCTYPE package PUBLIC "+//ISBN 0-9673008-1-9//DTD OEB 1.2 Package//EN"
 "http://openebook.org/dtds/oeb-1.2/oebpkg12.dtd"><package unique-identifier="Package-ID">
  <metadata>
    <dc-metadata xmlns:dc="http://purl.org/dc/elements/1.0"
      xmlns:oebpackage="http://openebook.org/namespaces/oeb-package/1.0">
      <dc:Identifier id="Package-ID">
        ebook:guid-6B2DF0030656ED9D8
      </dc:Identifier>
      <dc:Title>As You Like It</dc:Title>
      <dc:Creator role="aut">William Shakespeare</dc:Creator>
      <dc:Identifier>0-7410-1455-6</dc:Identifier>
      <dc:Subject></dc:Subject>
      <dc:Type></dc:Type>
      <dc:Date event="publication">3/24/2000</dc:Date>
      <dc:Date event="copyright">1/1/9999</dc:Date>
      <dc:Identifier scheme="ISBN">0-7410-1455-6</dc:Identifier>
      <dc:Publisher>Project Gutenberg</dc:Publisher>
      <dc:Language></dc:Language>
    </dc-metadata>
  </metadata>
  <manifest>
    <item id="4915" href="book.html" media-type="text/x-oeb1-document"/>
    <item id="7184" href="images/cover.png" media-type="image/png" />
  </manifest>
  <spine>
    <itemref idref="4915"/>
  </spine>
</package>
```

The OEBPS/book.html file would be binary and encrypted. Its decrypted contents would look something like this:

```
<?xml version="1.0" ?>
<!DOCTYPE html PUBLIC
  "+//ISBN 0-9673008-1-9//DTD OEB 1.2 Document//EN"
  "http://openebook.org/dtds/oeb-1.2/oebdoc12.dtd">
<html>
  <head>
   …
  </head>
  <body>
   …
     <img src="images/cover.png"
      alt="Cover image: a picture of the Bard of Avon" />
   …
  </body>
</html>
```

The OEBPS/images/cover.png file contains the encrypted binary bytes of the cover.png file.

The OEBPS/As You Like It.pdf file contains the encrypted binary bytes of the PDF file.

Comparison of UCF and OCF

As described in the introduction, UCF is OCF without the OEBPS dependencies. There are only a few significant differences.

- The OCF specification specifies that the media type of the container must be application/epub+zip, while UCF specifies that UCF-based formats should choose an appropriate media type. OCF implicitly encourages the use of +zip to identify Zip-based formats.
- OCF requires all XML documents in a container to be compatible with XML 1.1. UCF requires all XML documents defined by this specification to be compatible with XML 1.0. (The use of XML 1.0 follows the generally recommended practice to use XML 1.0 rather than XML 1.1 unless XML 1.1 features are required.)
- OCF forbids certain characters in names. In addition to those characters, UCF also forbids characters corresponding to the non-printing ASCII codes. While the OCF specification lists forbidden characters by ASCII names, the UCF specification lists Unicode code points.
- OCF requires that two file names in a container be unique after case normalization. UCF also requires that names be unique after character normalization using Unicode normalization form C (canonical decomposition followed by canonical composition).
- OCF requires the container.xml file which, in turn, requires specification of a root file. UCF does not require the container.xml file. The rationale is that a UCF-based format can specify how to begin processing the container. For example, Mars always uses the backbone.xml file. This file eliminates the need to find, read, and process an extra file when opening a Mars file, which can have a significant cost when viewing Mars files on the Web.
- UCF adds file relationships, providing an application-independent method for storing and finding meta data associated with container files.
- OCF specifies that the metadata.xml file must not be encrypted, while UCF allows this. Even when a publication is protected with encryption (usually to support digital rights management), the IDPF wants reading systems to be able to provide users with useful meta data. Mars (and possibly other UCF formats as well) has more general requirements and must leave this as an option for the container creator.

- OCF encourages but does not require each publication (in UCF, a publication is generalized to a document or an application) to reside in its own directory in the container. This makes it easier to contain multiple renditions of the publication in the container.
- The UCF specification has deferred the definition of encryption and digital signature features to a future revision. This provides implementers more time to evaluate the proposed definition.

Digital signatures

Support of digital signatures in UCF is deferred until a future revision of the specification. However, it is likely that the specification of this feature will match the OCF specification, which follows.

An OPTIONAL signatures.xml file in the META-INF directory at the root level of the container file system holds digital signatures of the container and its contents. This file is an XML document whose root element is <signatures>. The <signatures> element contains child elements of type <Signature> as defined by XML-Signature Syntax and Processing (http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/). Signatures can be applied to the publication and any alternative renditions as a whole or to parts of the publication and renditions. XML Signature can specify the signing of any kind of data, not just XML.

The signatures.xml file MUST NOT be encrypted.

When the signatures.xml file is not present, the UCF container provides no information indicating that any part of the container is digitally signed at the container level. It is, however, possible that digital signing exists in any optional alternative contained renditions.

A RELAX NG UCF schema describing the <signature> element that MUST be the root element in the signatures.xml file can be found in the "RELAX NG UCF schema" on page 39.

When a UCF agent creates a signature of data in a container, it SHOULD add the new signature as the last child <Signature> element of the <signatures> element in the signatures.xml file.

Each <Signature> element in the signatures.xml file identifies by IRI the data to which the signature applies, using the XML Signature <Manifest> element and its <Reference> sub-elements. Individual contained files MAY be signed separately or together. Separately signing each file creates a digest value for the resource that can be validated independently. This approach MAY make a <Signature> element larger. If files are signed together, the set of signed files can be listed in a single XML Signature <Manifest> element and referenced by one or more <Signature> elements.

Any or all files in the container can be signed in their entirety with the exception of the signatures.xml file because that file contains the computed signature information. Whether and how the signatures.xml file SHOULD be signed depends on the objective of the signer.

- If the signer wants to allow signatures to be added or removed from the container without invalidating the signer's signature, the signatures.xml file SHOULD NOT be signed.
- If the signer wants any addition or removal of a signature to invalidate the signer's signature, the Enveloped Signature transform (defined in Section 6.6.4 of XML-Signature Syntax and Processing) can be used to sign the entire preexisting signature file excluding the <Signature> being created. This transform would sign all previous signatures, and it would become invalid if a subsequent signature was added to the package.
- If the signer wants the removal of an existing signature to invalidate the signer's signature but also wants to allow the addition of signatures, an XPath transform can be used to sign just the existing signatures. (This is only a suggestion. The particular XPath transform is not a part of UCF specification.)

Although XML-Signature does not associate any semantics with a signature, an agent MAY include semantic information, such as by adding information to the <Signature> element that describes the signature. XML Signature describes how additional information can be added to a signature, such as by using the <SignatureProperties> element.

The following XML expression shows the content of an example signatures.xml file, and is based on the examples found in Section 2 of XML-Signature Syntax and Processing. It contains one signature, and the signature applies to two resources, OEBFPS/book.html and OEBFPS/images/cover.jpeg, in the container.

```
<signatures>
  <Signature Id="MyFirstSignature" xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
      <CanonicalizationMethod
       Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
      <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
      <Reference URI="#Manifest1">
        <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</DigestValue>
      </Reference>
    </SignedInfo>
    <SignatureValue>MC0CFFrVLtRlk= … </SignatureValue>
    <KeyInfo>
      <KeyValue>
        <DSAKeyValue>
          <P> … </P><Q> … </Q><G> … </G><Y> … </Y>
        </DSAKeyValue>
      </KeyValue>
    </KeyInfo>
    <Object>
      <Manifest Id="Manifest1">
        <Reference URI="OEBFPS/book.xml">
          <Transforms>
            <Transform
             Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
          </Transforms>
        </Reference>
        <Reference URI="OEBFPS/images/cover.jpeg"/>
      </Manifest>
    </Object>
  </Signature>
</signatures>
```

Encryption

Support of encryption in UCF has been deferred until a future revision of the specification. With one exception, it is likely that the specification of this feature will match the OCF specification, which follows this paragraph. The exception is that it will be possible to specify that a particular algorithm does not require Flate compression of data before encryption.

An OPTIONAL encryption.xml file in the META-INF directory at the root level of the container file system holds all encryption information about the contents of the container. This file is an XML document whose root element is <encryption>. The <encryption> element contains child elements of type <EncryptedKey> and <EncryptedData> as defined by XML Encryption Syntax and Processing. Each <EncryptedKey> element describes how one or more container files are encrypted. Consequently, if any resource in the container is encrypted, the encryption.xml file MUST be present to indicate that the resource is encrypted and to provide information about how it is encrypted.

An <EncryptedKey> element describes each encryption key used in the container, while an <EncryptedData> element describes each encrypted file. Each <EncryptedData> element refers to an <EncryptedKey> element, as described in XML Encryption.

A RELAX NG UCF schema describing the <encryption> element that MUST be the root element in the encryption.xml file can be found in the "RELAX NG UCF schema" on page 39.

When the encryption.xml file is not present, the UCF container provides no information that indicates that any part of the container is encrypted.

UCF encrypts individual files independently, trading off some security for improved performance, allowing the container contents to be incrementally decrypted. Encryption in this way still exposes the directory structure and file naming of the whole package.

UCF uses XML Encryption to provide a framework for encryption, allowing a variety of algorithms to be used. XML Encryption specifies a process for encrypting arbitrary data and representing the result in XML. Even though a UCF container MAY contain non-XML data, XML Encryption can be used to encrypt all data in a UCF container. UCF encryption supports encryption only of whole files. The encryption.xml file, if present, MUST NOT be encrypted.

Encrypted data replaces unencrypted data in a UCF container. For example, if an image named photo.jpeg is encrypted, the contents of the photo.jpeg resource SHOULD be replaced by its encrypted contents. When stored in a Zip container, files MUST be compressed before they are encrypted; Flate compression MUST be used. In the Zip local file header and central directory, encrypted files SHOULD be listed as stored rather than Flate-compressed.

The following files MUST never be encrypted regardless of whether default or specific encryption is requested.

- mimetype
- META-INF/container.xml
- META-INF/manifest.xml
- META-INF/metadata.xml
- META-INF/signatures.xml
- META-INF/encryption.xml
- META-INF/rights.xml

Signed resources MAY subsequently be encrypted by using the Decryption Transform for XML Signature. This feature enables an application such as a UCF agent to distinguish data that was encrypted before signing from data that was encrypted after signing. Only data that was encrypted after signing MUST be decrypted before computing the digest used to validate the signature.

In the following example, adapted from Section 2.2.1 of XML Encryption Syntax and Processing, the resource image.jpeg is encrypted using a symmetric key algorithm (AES), and the symmetric key is further encrypted using an asymmetric key algorithm (RSA) with a key of John Smith.

```
<encryption
 xmlns ="urn:oasis:names:tc:opendocument:xmlns:container"
xmlns:enc="http://www.w3.org/2001/04/xmlenc#" xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <enc:EncryptedKey Id="EK">
      <enc:EncryptionMethod
       Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
      <ds:KeyInfo>
        <ds:KeyName>John Smith</ds:KeyName>
      </ds:KeyInfo>
      <enc:CipherData>
        <enc:CipherValue>xyzabc</enc:CipherValue>
      </enc:CipherData>
    </enc:EncryptedKey>
    <enc:EncryptedData Id="ED1">
      <enc:EncryptionMethod                  Algorithm="http://www.w3.org/2001/04/xmlenc#kw-
aes128"/>
      <ds:KeyInfo>
        <ds:RetrievalMethod URI="#EK"
Type="http://www.w3.org/2001/04/xmlenc#EncryptedKey"/>
      </ds:KeyInfo>
    <enc:CipherData>
      <enc:CipherReference URI="image.jpeg"/>
    </enc:CipherData>
  </enc:EncryptedData>
</encryption>
```