Documents in WPF

Windows Presentation Foundation (WPF) offers a wide range of document features that enable the creation of high-fidelity content that is designed to be more easily accessed and read than in previous generations of Windows. In addition to enhanced capabilities and quality, WPF also provides integrated services for document display, packaging, and security. This topic provides an introduction to WPF document types and document packaging.

Types of Documents

WPF divides documents into two broad categories based on their intended use; these document categories are termed "fixed documents" and "flow documents."

Fixed documents are intended for applications that require a precise "what you see is what you get" (WYSIWYG) presentation, independent of the display or printer hardware used. Typical uses for fixed documents include desktop publishing, word processing, and form layout, where adherence to the original page design is critical. As part of its layout, a fixed document maintains the precise positional placement of content elements independent of the display or print device in use. For example, a fixed document page viewed on 96 dpi display will appear exactly the same when it is output to a 600 dpi laser printer as when it is output to a 4800 dpi phototypesetter. The page layout remains the same in all cases, while the document quality maximizes to the capabilities of each device.

By comparison, flow documents are designed to optimize viewing and readability and are best utilized when ease of reading is the primary document consumption scenario. Rather than being set to one predefined layout, flow documents dynamically adjust and reflow their content based on run-time variables such as window size, device resolution, and optional user preferences. A Web page is a simple example of a flow document where the page content is dynamically formatted to fit the current window. Flow documents optimize the viewing and reading experience for the user, based on the runtime environment. For example, the same flow document will dynamically reformat for optimal readability on either high-resolution 19-inch display or a small 2x3-inch PDA screen. In addition, flow documents have a number of built in features including search, viewing modes that optimize readability, and the ability to change the size and appearance of fonts. See Flow Document Overview for illustrations, examples, and in-depth information on flow documents.

Document Controls and Text Layout

The .NET Framework provides a set of pre-built controls that simplify using fixed documents, flow documents, and general text within your application. The display of fixed document content is supported using the DocumentViewer control. Display of flow document content is supported by three different controls: FlowDocumentReader, FlowDocumentPageViewer, and FlowDocumentScrollViewer which map to different user scenarios (see sections below). Other WPF controls provide simplified layout to support general text uses (see Text in the User Interface, below).

Fixed Document Control - DocumentViewer

The DocumentViewer control is designed to display FixedDocument content. The DocumentViewercontrol provides an intuitive user interface that provides built-in support for common operations including print output, copy to clipboard, zoom, and text search features. The control provides access to pages of content through a familiar scrolling mechanism. Like all WPF controls, DocumentViewer supports complete or partial restyling, which enables the control to be visually integrated into virtually any application or environment.

DocumentViewer is designed to display content in a read-only manner; editing or modification of content is not available and is not supported.

Flow Document Controls

Note: For more detailed information on flow document features and how to create them, see Flow Document Overview.

Display of flow document content is supported by three controls: FlowDocumentReader, FlowDocumentPageViewer, and FlowDocumentScrollViewer.

FlowDocumentReader

FlowDocumentReader includes features that enable the user to dynamically choose between various viewing modes, including a single-page (page-at-a-time) viewing mode, a two-page-at-a-time (book reading format) viewing mode, and a continuous scrolling (bottomless) viewing mode. For more information about these viewing modes, see FlowDocumentReaderViewingMode. If you do not need the ability to dynamically switch between different viewing modes, FlowDocumentPageViewer and FlowDocumentScrollViewer provide lighter-weight flow content viewers that are fixed in a particular viewing mode.

FlowDocumentPageViewer and FlowDocumentScrollViewer

FlowDocumentPageViewer shows content in page-at-a-time viewing mode, while FlowDocumentScrollViewer shows content in continuous scrolling mode. Both FlowDocumentPageViewer and FlowDocumentScrollViewer are fixed to a particular viewing mode. Compare to FlowDocumentReader, which includes features that enable the user to dynamically choose between various viewing modes (as provided by the FlowDocumentReaderViewingModeenumeration), at the cost of being more resource intensive than FlowDocumentPageViewer or FlowDocumentScrollViewer.

By default, a vertical scrollbar is always shown, and a horizontal scrollbar becomes visible if needed. The default UI for FlowDocumentScrollViewer does not include a toolbar; however, the IsToolBarVisible property can be used to enable a built-in toolbar.

Text in the User Interface

Besides adding text to documents, text can obviously be used in application UI such as forms. WPF includes multiple controls for drawing text to the screen. Each control is targeted to a different scenario and has its own list of features and limitations. In general, the TextBlock element should be used when limited text support is required, such as a brief sentence in a user interface (UI). Label can be used when minimal text support is required. For more information, see TextBlock Overview.

Document Packaging

The System.IO.Packaging APIs provide an efficient means to organize application data, document content, and related resources in a single container that is simple to access, portable, and easy to distribute. A ZIP file is an example of a Package type capable of holding multiple objects as a single unit. The packaging APIs provide a default ZipPackage implementation designed using an Open Packaging Conventions standard with XML and ZIP file architecture. The WPF packaging APIs make it simple to create packages, and to store and access objects within them. An object stored in a Package is referred to as a PackagePart ("part"). Packages can also include signed digital certificates that can be used to identify the originator of a part and to validate that the contents of a package have not been modified. Packages also include a PackageRelationship feature that allows additional information to be added to a package or associated with specific parts without actually modifying the content of existing parts. Package services also support Microsoft Windows Rights Management (RM).

The WPF Package architecture serves as the foundation for a number of key technologies:

- XPS documents conforming to the XML Paper Specification (XPS).

- Microsoft Office "12" open XML format documents (.docx).
- Custom storage formats for your own application design.

Based on the packaging APIs, an XpsDocument is specifically designed for storing WPF fixed content documents. An XpsDocument is a self-contained document that can be opened in a viewer, displayed in a DocumentViewer control, routed to a print spool, or output directly to an XPS-compatible printer.

The following sections provide additional information on the Package and XpsDocument APIs provided with WPF.

Package Components

The WPF packaging APIs allow application data and documents to be organized into a single portable unit. A ZIP file is one of the most common types of packages and is the default package type provided with WPF. Package itself is an abstract class from which ZipPackage is implemented using an open standard XML and ZIP file architecture. The Open method uses ZipPackage to create and use ZIP files by default. A package can contain three basic types of items:

PackagePart	Application content, data, documents, and resource files.
PackageDigitalSignature	[X.509 Certificate] for identification, authentication and validation.
PackageRelationship	Added information related to the package or a specific part.

PackageParts

A PackagePart ("part") is an abstract class that refers to an object stored in a Package. In a ZIP file, the package parts correspond to the individual files stored within the ZIP file. ZipPackagePartprovides the default implementation for serializable objects stored in a ZipPackage. Like a file system, parts contained in the package are stored in hierarchical directory or "folder-style" organization. Using the WPF packaging APIs, applications can write, store, and read multiple PackagePart objects using a single ZIP file container.

PackageDigitalSignatures

For security, a PackageDigitalSignature ("digital signature") can be associated with parts within a package. A PackageDigitalSignature incorporates a [509] that provides two features:

1. Identifies and authenticates the originator of the part.
2. Validates that the part has not been modified.

The digital signature does not preclude a part from being modified, but a validation check against the digital signature will fail if the part is altered in any way. The application can then take appropriate action—for example, block opening the part or notify the user that the part has been modified and is not secure.

PackageRelationships

A PackageRelationship ("relationship") provides a mechanism for associating additional information with the package or a part within the package. A relationship is a package-level facility that can associate additional information with a part without modifying the actual part content. Inserting new data directly into the part content of is usually not practical in many cases:

- The actual type of the part and its content schema is not known.
- Even if known, the content schema might not provide a means for adding new information.
- The part might be digitally signed or encrypted, precluding any modification.

Package relationships provide a discoverable means for adding and associating additional information with individual parts or with the entire package. Package relationships are used for two primary functions:

1. Defining dependency relationships from one part to another part.
2. Defining information relationships that add notes or other data related to the part.

A PackageRelationship provides a quick, discoverable means to define dependencies and add other information associated with a part of the package or the package as a whole.

Dependency Relationships

Dependency relationships are used to describe dependencies that one part makes to other parts. For example, a package might contain an HTML part that includes one or more <img> image tags. The image tags refer to images that are located either as other parts internal to the package or external to the package (such as accessible over the Internet). Creating a PackageRelationshipassociated with HTML file makes discovering and accessing the dependent resources quick and easy. A browser or viewer application can directly access the part relationships and immediately begin assembling the dependent resources without knowing the schema or parsing the document.

Information Relationships

Similar to a note or annotation, a PackageRelationship can also be used to store other types of information to be associated with a part without having to actually modify the part content itself.

XPS Documents

XML Paper Specification (XPS) document is a package that contains one or more fixed-documents along with all the resources and information required for rendering. XPS is also the native Windows Vista print spool file format. An XpsDocument is stored in standard ZIP dataset, and can include a combination of XML and binary components, such as image and font files. PackageRelationshipsare used to define the dependencies between the content and the resources required to fully render the document.
The XpsDocument design provides a single, high-fidelity document solution that supports multiple uses:

- Reading, writing, and storing fixed-document content and resources as a single, portable, and easy-to-distribute file.
- Displaying documents with the XPS Viewer application.
- Outputting documents in the native print spool output format of Windows Vista.
- Routing documents directly to an XPS-compatible printer.

See also

- FixedDocument
- FlowDocument
- XpsDocument
- ZipPackage
- ZipPackagePart
- PackageRelationship
- DocumentViewer
- Text
- Flow Document Overview
- Printing Overview
- Document Serialization and Storage