

PROGRAMMING INTERFACE FOR A COMPUTER PLATFORM

TECHNICAL FIELD

This invention relates to software and to development of such software. More particularly, this invention relates to a programming interface that facilitates use of a software platform by application programs and computer hardware.

BRIEF DESCRIPTION OF ACCOMPANYING COMPACT DISCS

Accompanying this specification is a set of three compact discs that stores a Software Development Kit (SDK) for the Microsoft® Windows® Code-Named "Longhorn" operating system. The SDK contains documentation for the Microsoft® Windows® Code-Named "Longhorn" operating system. Duplicate copies of each of these three compact discs also accompany this specification.

The first compact disc in the set of three compact discs (CD 1 of 3) includes a file folder named "lhsdk" that was created on October 22, 2003; it is 586 Mbytes in size, contains 9,692 sub-folders, and contains 44,292 sub-files. The second compact disc in the set of three compact discs (CD 2 of 3) includes a file folder named "ns" that was created on October 22, 2003; it is 605 Mbytes in size, contains 12,628 sub-folders, and contains 44,934 sub-files. The third compact disc in the set of three compact discs (CD 3 of 3) includes a file folder named "ns" that was created on October 22, 2003; it is 575 Mbytes in size, contains 9,881 sub-folders, and contains 43,630 sub-files. The files on each of these three compact discs can be executed on a Windows? -based computing device (e.g., IBM-PC, or equivalent) that executes a Windows? -brand operating system (e.g., Windows? NT, Windows? 98, Windows? 2000, Windows® XP, etc.). The files on each

compact disc in this set of three compact discs are hereby incorporated by reference.

Each compact disc in the set of three compact discs itself is a CD-R, and conforms to the ISO 9660 standard. The contents of each compact disc in the set of three compact discs is in compliance with the American Standard Code for Information Interchange (ASCII).

BACKGROUND

Very early on, computer software came to be categorized as “operating system” software or “application” software. Broadly speaking, an application is software meant to perform a specific task for the computer user such as solving a mathematical equation or supporting word processing. The operating system is the software that manages and controls the computer hardware. The goal of the operating system is to make the computer resources available to the application programmer while at the same time, hiding the complexity necessary to actually control the hardware.

The operating system makes the resources available via functions that are collectively known as the Application Program Interface or API. The term API is also used in reference to a single one of these functions. The functions are often grouped in terms of what resource or service they provide to the application programmer. Application software requests resources by calling individual API functions. API functions also serve as the means by which messages and information provided by the operating system are relayed back to the application software.

1 In addition to changes in hardware, another factor driving the evolution of
2 operating system software has been the desire to simplify and speed application
3 software development. Application software development can be a daunting task,
4 sometimes requiring years of developer time to create a sophisticated program
5 with millions of lines of code. For a popular operating system such as various
6 versions of the Microsoft Windows® operating system, application software
7 developers write thousands of different applications each year that utilize the
8 operating system. A coherent and usable operating system base is required to
9 support so many diverse application developers.

10 Often, development of application software can be made simpler by making
11 the operating system more complex. That is, if a function may be useful to several
12 different application programs, it may be better to write it once for inclusion in the
13 operating system, than requiring dozens of software developers to write it dozens
14 of times for inclusion in dozens of different applications. In this manner, if the
15 operating system supports a wide range of common functionality required by a
16 number of applications, significant savings in applications software development
17 costs and time can be achieved.

18 Regardless of where the line between operating system and application
19 software is drawn, it is clear that for a useful operating system, the API between
20 the operating system and the computer hardware and application software is as
21 important as efficient internal operation of the operating system itself.

22 When developing applications, developers use a variety of tools to generate
23 graphical items and other content. Additional tools are available to arrange
24 graphical items and other data to be displayed or rendered. These tools are
25 typically created by different entities or different tool developers. As a result, the

1 tools do not provide a consistent programming environment. Thus, a developer
2 using these different tools needs to learn how to utilize each of the tools and
3 attempt to make them communicate with one another. These activities can be
4 tedious and time consuming, taking time away from the actual development task at
5 hand.

6 Over the past few years, the universal adoption of the Internet, and
7 networking technology in general, has changed the landscape for computer
8 software developers. Traditionally, software developers focused on single-site
9 software applications for standalone desktop computers, or LAN-based computers
10 that were connected to a limited number of other computers via a local area
11 network (LAN). Such software applications were typically referred to as “shrink
12 wrapped” products because the software was marketed and sold in a shrink-
13 wrapped package. The applications utilized well-defined APIs to access the
14 underlying operating system of the computer.

15 As the Internet evolved and gained widespread acceptance, the industry
16 began to recognize the power of hosting applications at various sites on the World
17 Wide Web (or simply the “Web”). In the networked world, clients from anywhere
18 could submit requests to server-based applications hosted at diverse locations and
19 receive responses back in fractions of a second. These Web applications, however,
20 were typically developed using the same operating system platform that was
21 originally developed for standalone computing machines or locally networked
22 computers. Unfortunately, in some instances, these applications do not adequately
23 transfer to the distributed computing regime. The underlying platform was simply
24 not constructed with the idea of supporting limitless numbers of interconnected
25 computers.

1 To accommodate the shift to the distributed computing environment being
2 ushered in by the Internet, Microsoft Corporation developed a network software
3 platform known as the “.NET” Framework (read as “Dot Net”). Microsoft® .NET
4 is software for connecting people, information, systems, and devices. The
5 platform allows developers to create Web services that will execute over the
6 Internet. This dynamic shift was accompanied by a set of API functions for
7 Microsoft's .NET™ Framework.

8 As use of the .NET™ Framework has become increasingly common, ways
9 to increase the efficiency and/or performance of the platform have been identified.
10 The inventors have developed a unique set of API functions to allow for such
11 increased efficiency and/or performance.

12 13 **SUMMARY**

14 A programming interface, such as an API, provides functions for generating
15 applications, documents, media presentations and other content. These functions
16 allow developers to obtain services from an operating system, object model
17 service, or other system or service. In one embodiment, the functions allow a
18 developer to generate a graphical user interface.

19 20 **BRIEF DESCRIPTION OF THE DRAWINGS**

21 The same numbers are used throughout the drawings to reference like
22 features.

23 Fig. 1 illustrates a network architecture in which clients access Web
24 services over the Internet using conventional protocols.

1 Fig. 2 is a block diagram of a software architecture for a network platform,
2 which includes an application program interface (API).

3 Fig. 3 is a block diagram of the presentation subsystem supported by the
4 API, as well as function classes of the various API functions.

5 Fig. 4 is a block diagram of an exemplary computer that may execute all or
6 part of the software architecture.

7 Figs. 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 and 16 illustrate various example
8 implementations of a programming interface.

9 10 **DETAILED DESCRIPTION**

11 This disclosure addresses an application program interface (API) for a
12 network platform upon which developers can build Web applications and services.
13 More particularly, an exemplary API is described for operating systems that make
14 use of a network platform, such as the .NET? Framework created by Microsoft
15 Corporation. The .NET? Framework is a software platform for Web services and
16 Web applications implemented in the distributed computing environment. It
17 represents the next generation of Internet computing, using open communication
18 standards to communicate among loosely coupled Web services that are
19 collaborating to perform a particular task.

20 In the described implementation, the network platform utilizes XML
21 (extensible markup language), an open standard for describing data. XML is
22 managed by the World Wide Web Consortium (W3C). XML is used for defining
23 data elements on a Web page and business-to-business documents. XML uses a
24 similar tag structure as HTML; however, whereas HTML defines how elements
25 are displayed, XML defines what those elements contain. HTML uses predefined

1 tags, but XML allows tags to be defined by the developer of the page. Thus,
2 virtually any data items can be identified, allowing Web pages to function like
3 database records. Through the use of XML and other open protocols, such as
4 Simple Object Access Protocol (SOAP), the network platform allows integration
5 of a wide range of services that can be tailored to the needs of the user. Although
6 the embodiments described herein are described in conjunction with XML and
7 other open standards, such are not required for the operation of the claimed
8 invention. Other equally viable technologies will suffice to implement the
9 inventions described herein.

10 As used herein, the phrase application program interface or API includes
11 traditional interfaces that employ method or function calls, as well as remote calls
12 (e.g., a proxy, stub relationship) and SOAP/XML invocations.

13 It should be appreciated that in some of namespace descriptions below,
14 descriptions of certain classes, interfaces, enumerations and delegates are left
15 blank. More complete descriptions of these classes, interfaces, enumerations and
16 delegates can be found in the subject matter of the compact discs that store the
17 SDK referenced above.

18 19 EXEMPLARY NETWORK ENVIRONMENT

20 Fig. 1 shows a network environment 100 in which a network platform, such
21 as the .NET? Framework, may be implemented. The network environment 100
22 includes representative Web services 102(1), ..., 102(N), which provide services
23 that can be accessed over a network 104 (e.g., Internet). The Web services,
24 referenced generally as number 102, are programmable application components
25 that are reusable and interact programmatically over the network 104, typically

1 through industry standard Web protocols, such as XML, SOAP, WAP (wireless
2 application protocol), HTTP (hypertext transport protocol), and SMTP (simple
3 mail transfer protocol) although other means of interacting with the Web services
4 over the network may also be used, such as Remote Procedure Call (RPC) or
5 object broker type technology. A Web service can be self-describing and is often
6 defined in terms of formats and ordering of messages.

7 Web services 102 are accessible directly by other services (as represented
8 by communication link 106) or a software application, such as Web application
9 110 (as represented by communication links 112 and 114). Each Web service 102
10 is illustrated as including one or more servers that execute software to handle
11 requests for particular services. Such services often maintain databases that store
12 information to be served back to requesters. Web services may be configured to
13 perform any one of a variety of different services. Examples of Web services
14 include login verification, notification, database storage, stock quoting, location
15 directories, mapping, music, electronic wallet, calendar/scheduler, telephone
16 listings, news and information, games, ticketing, and so on. The Web services can
17 be combined with each other and with other applications to build intelligent
18 interactive experiences.

19 The network environment 100 also includes representative client devices
20 120(1), 120(2), 120(3), 120(4), ..., 120(M) that utilize the Web services 102 (as
21 represented by communication link 122) and/or the Web application 110 (as
22 represented by communication links 124, 126, and 128). The clients may
23 communicate with one another using standard protocols as well, as represented by
24 an exemplary XML link 130 between clients 120(3) and 120(4).
25

1 The client devices, referenced generally as number 120, can be
2 implemented many different ways. Examples of possible client implementations
3 include, without limitation, portable computers, stationary computers, tablet PCs,
4 televisions/set-top boxes, wireless communication devices, personal digital
5 assistants, gaming consoles, printers, photocopiers, and other smart devices.

6 The Web application 110 is an application designed to run on the network
7 platform and may utilize the Web services 102 when handling and servicing
8 requests from clients 120. The Web application 110 is composed of one or more
9 software applications 130 that run atop a programming framework 132, which are
10 executing on one or more servers 134 or other computer systems. Note that a
11 portion of Web application 110 may actually reside on one or more of clients 120.
12 Alternatively, Web application 110 may coordinate with other software on clients
13 120 to actually accomplish its tasks.

14 The programming framework 132 is the structure that supports the
15 applications and services developed by application developers. It permits multi-
16 language development and seamless integration by supporting multiple languages.
17 It supports open protocols, such as SOAP, and encapsulates the underlying
18 operating system and object model services. The framework provides a robust and
19 secure execution environment for the multiple programming languages and offers
20 secure, integrated class libraries.

21 The framework 132 is a multi-tiered architecture that includes an
22 application program interface (API) layer 142, a common language runtime (CLR)
23 layer 144, and an operating system/services layer 146. This layered architecture
24 allows updates and modifications to various layers without impacting other
25 portions of the framework. A common language specification (CLS) 140 allows

1 designers of various languages to write code that is able to access underlying
2 library functionality. The specification 140 functions as a contract between
3 language designers and library designers that can be used to promote language
4 interoperability. By adhering to the CLS, libraries written in one language can be
5 directly accessible to code modules written in other languages to achieve seamless
6 integration between code modules written in one language and code modules
7 written in another language. One exemplary detailed implementation of a CLS is
8 described in an ECMA standard created by participants in ECMA TC39/TG3.
9 The reader is directed to the ECMA web site at www.ecma.ch.

10 The API layer 142 presents groups of functions that the applications 130
11 can call to access the resources and services provided by layer 146. By exposing
12 the API functions for a network platform, application developers can create Web
13 applications for distributed computing systems that make full use of the network
14 resources and other Web services, without needing to understand the complex
15 interworkings of how those network resources actually operate or are made
16 available. Moreover, the Web applications can be written in any number of
17 programming languages, and translated into an intermediate language supported
18 by the common language runtime 144 and included as part of the common
19 language specification 140. In this way, the API layer 142 can provide methods
20 for a wide and diverse variety of applications.

21 Additionally, the framework 132 can be configured to support API calls
22 placed by remote applications executing remotely from the servers 134 that host
23 the framework. Representative applications 148(1) and 148(2) residing on clients
24 120(3) and 120(M), respectively, can use the API functions by making calls
25 directly, or indirectly, to the API layer 142 over the network 104.

1 The framework may also be implemented at the clients. Client 120(3)
2 represents the situation where a framework 150 is implemented at the client. This
3 framework may be identical to server-based framework 132, or modified for client
4 purposes. Alternatively, the client-based framework may be condensed in the
5 event that the client is a limited or dedicated function device, such as a cellular
6 phone, personal digital assistant, handheld computer, or other
7 communication/computing device.

8 9 DEVELOPERS' PROGRAMMING FRAMEWORK

10 Fig. 2 shows the programming framework 132 in more detail. The
11 common language specification (CLS) layer 140 supports applications written in a
12 variety of languages 130(1), 130(2), 130(3), 130(4), ..., 130(K). Such application
13 languages include Visual Basic, C++, C#, COBOL, Jscript, Perl, Eiffel, Python,
14 and so on. The common language specification 140 specifies a subset of features
15 or rules about features that, if followed, allow the various languages to
16 communicate. For example, some languages do not support a given type (e.g., an
17 "int*" type) that might otherwise be supported by the common language runtime
18 144. In this case, the common language specification 140 does not include the
19 type. On the other hand, types that are supported by all or most languages (e.g.,
20 the "int[]" type) is included in common language specification 140 so library
21 developers are free to use it and are assured that the languages can handle it. This
22 ability to communicate results in seamless integration between code modules
23 written in one language and code modules written in another language. Since
24 different languages are particularly well suited to particular tasks, the seamless
25 integration between languages allows a developer to select a particular language

1 for a particular code module with the ability to use that code module with modules
2 written in different languages. The common language runtime 144 allow seamless
3 multi-language development, with cross language inheritance, and provide a
4 robust and secure execution environment for the multiple programming languages.
5 For more information on the common language specification 140 and the common
6 language runtime 144, the reader is directed to co-pending applications entitled
7 “Method and System for Compiling Multiple Languages”, filed 6/21/2000 (serial
8 number 09/598,105) and “Unified Data Type System and Method” filed 7/10/2000
9 (serial number 09/613,289), which are incorporated by reference.

10 The framework 132 encapsulates the operating system 146(1) (e.g.,
11 Windows? -brand operating systems) and object model services 146(2) (e.g.,
12 Component Object Model (COM) or Distributed COM). The operating system
13 146(1) provides conventional functions, such as file management, notification,
14 event handling, user interfaces (e.g., windowing, menus, dialogs, etc.), security,
15 authentication, verification, processes and threads, memory management, and so
16 on. The object model services 146(2) provide interfacing with other objects to
17 perform various tasks. Calls made to the API layer 142 are handed to the common
18 language runtime layer 144 for local execution by the operating system 146(1)
19 and/or object model services 146(2).

20 The API 142 groups API functions into multiple namespaces. Namespaces
21 essentially define a collection of classes, interfaces, delegates, enumerations, and
22 structures, which are collectively called “types”, that provide a specific set of
23 related functionality. A class represents managed heap allocated data that has
24 reference assignment semantics. A delegate is an object oriented function pointer.
25 An enumeration is a special kind of value type that represents named constants. A

1 structure represents static allocated data that has value assignment semantics. An
2 interface defines a contract that other types can implement.

3 By using namespaces, a designer can organize a set of types into a
4 hierarchical namespace. The designer is able to create multiple groups from the
5 set of types, with each group containing at least one type that exposes logically
6 related functionality. In the exemplary implementation, the API 142 is organized
7 to include three root namespaces. It should be noted that although only three root
8 namespaces are illustrated in Fig. 2, additional root namespaces may also be
9 included in API 142. The three root namespaces illustrated in API 142 are: a first
10 namespace 200 for a presentation subsystem (which includes a namespace 202 for
11 a user interface shell), a second namespace 204 for web services, and a third
12 namespace 206 for a file system. Each group can then be assigned a name. For
13 instance, types in the presentation subsystem namespace 200 can be assigned the
14 name “Windows”, and types in the file system namespace 206 can be assigned
15 names “Storage”. The named groups can be organized under a single “global
16 root” namespace for system level APIs, such as an overall System namespace. By
17 selecting and prefixing a top level identifier, the types in each group can be easily
18 referenced by a hierarchical name that includes the selected top level identifier
19 prefixed to the name of the group containing the type. For instance, types in the
20 file system namespace 206 can be referenced using the hierarchical name
21 “System.Storage”. In this way, the individual namespaces 200, 204, and 206
22 become major branches off of the System namespace and can carry a designation
23 where the individual namespaces are prefixed with a designator, such as a
24 “System.” prefix.

1 The presentation subsystem namespace 200 pertains to programming and
2 content development. It supplies types that allow for the generation of
3 applications, documents, media presentations and other content. For example,
4 presentation subsystem namespace 200 provides a programming model that allows
5 developers to obtain services from the operating system 146(1) and/or object
6 model services 146(2).

7 The shell namespace 202 pertains to user interface functionality. It supplies
8 types that allow developers to embed user interface functionality in their
9 applications, and further allows developers to extend the user interface
10 functionality.

11 The web services namespace 204 pertains to an infrastructure for enabling
12 creation of a wide variety of applications, e.g. applications as simple as a chat
13 application that operates between two peers on an intranet, and/or as complex as a
14 scalable Web service for millions of users. The described infrastructure is
15 advantageously highly variable in that one need only use those parts that are
16 appropriate to the complexity of a particular solution. The infrastructure provides
17 a foundation for building message-based applications of various scale and
18 complexity. The infrastructure or framework provides APIs for basic messaging,
19 secure messaging, reliable messaging and transacted messaging. In the
20 embodiment described below, the associated APIs have been factored into a
21 hierarchy of namespaces in a manner that has been carefully crafted to balance
22 utility, usability, extensibility and versionability.

23 The file system namespace 206 pertains to storage. It supplies types that
24 allow for information storage and retrieval.

1 In addition to the framework 132, programming tools 210 are provided to
2 assist the developer in building Web services and/or applications. One example of
3 the programming tools 210 is Visual Studio? , a multi-language suite of
4 programming tools offered by Microsoft Corporation.

6 ROOT API NAMESPACES

7 Fig. 3 shows a portion of the presentation subsystem 200 in more detail. In
8 one embodiment, the namespaces are identified according to a hierarchical naming
9 convention in which strings of names are concatenated with periods. For instance,
10 the presentation subsystem namespace 200 is identified by the root name
11 “System.Windows”. Within the “System.Windows” namespace is another
12 namespace for various controls, identified as “System.Windows.Controls”, which
13 further identifies another namespace for primitives (not shown) known as
14 “System.Windows.Controls.Primitives”. With this naming convention in mind,
15 the following provides a general overview of selected namespaces of the API 142,
16 although other naming conventions could be used with equal effect.

17 As shown in Fig. 3, the presentation subsystem 200 includes multiple
18 namespaces. The namespaces shown in Fig. 3 represent a particular embodiment
19 of the presentation subsystem 200. Other embodiments of the presentation
20 subsystem 200 may include one or more additional namespaces or may omit one
21 or more of the namespaces shown in Fig. 3.

22 The presentation subsystem 200 is the root namespace for much of the
23 presentation functionality of the API 142. A controls namespace 310 includes
24 controls used to build a display of information, such as a user interface, and
25 classes that allow a user to interact with an application. Example controls include

1 “Button” that creates a button on the display, “RadioButton” that generates a
2 radio-style button on the display, “Menu” that creates a menu on the display,
3 “ToolBar” that creates a toolbar on the display, “Image” that generates an image
4 on the display and “TreeView” that creates a hierarchical view of information.

5 Certain controls are created by nesting and arranging multiple elements.
6 The controls have a logical model that hides the elements used to create the
7 controls, thereby simplifying the programming model. The controls can be styled
8 and themed by a developer or a user (e.g., by customizing the appearance and
9 behavior of user interface buttons). Some controls have addressable components
10 that allow an individual to adjust the style of individual controls. Additionally, the
11 controls can be sub-classed and extended by application developers and
12 component developers. The controls are rendered using vector graphics such that
13 they can be resized to fit the requirements of a particular interface or other display.
14 The controls are capable of utilizing animation to enhance, for example, the
15 interactive feel of a user interface and to show actions and reactions.

16 The controls namespace 310 includes one or more panels, which are
17 controls that measure and arrange their children (e.g., nested elements). For
18 example, a “DockPanel” panel arranges children by docking each child to the top,
19 left, bottom or right side of the display, and fills-in the remaining space with other
20 data. A particular panel may dock menus and toolbars to the top of the display, a
21 status bar to the bottom of the display, a folder list to the left side of the display,
22 and fills the rest of the space with a list of messages.

23 As mentioned above, System.Windows.Controls.Primitives is a namespace
24 that includes multiple controls that are components typically used by developers of
25 the controls in the System.Windows.Controls namespace and by developers

1 creating their own controls. Examples of these components include “Thumb and
2 RepeatButton”. “ScrollBar”, another component, is created using four repeat
3 buttons (one for “line up”, one for “line down”, one for “page up”, and one for
4 “page down”) and a “Thumb” for dragging the current view to another location in
5 the document. In another example, “ScrollViewer” is a control created using two
6 “ScrollBars” and one “ScrollArea” to provide a scrollable area.

7 The following list contains example classes exposed by the
8 System.Windows.Controls namespace. These classes allow a user to interact with,
9 for example, an application through various input and output capabilities as well
10 as additional display capabilities.

- 11 • AccessKey – AccessKey is a FrameworkElement element that wraps a
12 character, indicating that it is to receive keyboard cue decorations denoting
13 the character as a keyboard mnemonic. By default, the keyboard cue
14 decoration is an underline.
- 15 • Audio – Audio Element.
- 16 • Border – Draws a border, background, or both around another element.
- 17 • Button – Represents the standard button component that inherently reacts to
18 the Click event.
- 19 • Canvas – Defines an area within which a user can explicitly position child
20 elements by coordinates relative to the Canvas area.
- 21
- 22
- 23
- 24
- 25

- 1 • **CheckBox** – Use a **CheckBox** to give the user an option, such as true/false.
2 **CheckBox** allows the user to choose from a list of options. **CheckBox**
3 controls let the user pick a combination of options.
- 4 • **CheckedChangedEventArgs** – This **CheckedChangedEventArgs** class
5 contains additional information about the **CheckedChangedEvent** event.
- 6 • **CheckStateChangedEventArgs** – This **CheckStateChangedEventArgs** class
7 contains additional information about the **CheckStateChangedEvent** event.
- 8 • **ClickEventArgs** – Contains information about the **Click** event.
- 9 • **ColumnStyle** – Represents a changeable **ColumnStyle** object.
- 10 • **ColumnStyles** – Changeable pattern **ICollection** object that is a collection of
11 Changeable elements.
- 12 • **ComboBox** – **ComboBox** control.
- 13 • **ComboBoxItem** – Control that implements a selectable item inside a
14 **ComboBox**.
- 15 • **ContactPickerDialog** – Allows a user to select one or more contacts.
- 16 • **ContactPropertyRequest** – Allows an application to request information
17 about a contact property through a **ContactPickerDialog**. This class cannot
18 be inherited.
- 19 • **ContactPropertyRequest Collection** – Represents a collection of
20 **ContactPropertyRequest** objects.
- 21 • **ContactPropertyRequest Collection** – Represents a collection of
22 **ContactPropertyRequest** objects.
- 23 • **ContactPropertyRequest Collection** – Represents a collection of
24 **ContactPropertyRequest** objects.
- 25 • **ContactPropertyRequest Collection** – Represents a collection of
ContactPropertyRequest objects.

- 1 • **ContactSelection** – Information about a selected contact from Microsoft®
2 Windows® File System, code-named "WinFS" or Microsoft Active
3 Directory®.
- 4 • **ContactSelectionCollection** – Represents a collection of **ContactSelection**
5 objects.
- 6 • **ContactTextBox** – An edit control that supports picking contacts or
7 properties of contacts.
- 8 • **ContactTextBoxSelectionChangedEventArgs** – Arguments for the
9 **ContactTextBoxSelectionChanged** event.
- 10 • **ContactTextBoxTextChangedEventArgs** – Arguments for the
11 **ContactTextBoxTextChanged** event.
- 12 • **ContactTextBoxTextResolvedEventArgs** – Arguments for the
13 **ContactTextBoxTextResolved** event.
- 14 • **ContentChangedEventArgs** – The event arguments for
15 **ContentChangedEvent**.
- 16 • **ContentControl** – The base class for all controls with a single piece of
17 content.
- 18 • **ContentPresenter** – **ContentPresenter** is used within the style of a content
19 control to denote the place in the control's visual tree (chrome template)
20 where the content is to be added.
- 21 • **ContextMenu** – Control that defines a menu of choices for users to invoke.
- 22
- 23
- 24
- 25

- 1 • ContextMenuEventArgs – The data sent on a ContextMenuEvent.
- 2 • Control – Represents the base class for all user-interactive elements. This
- 3 class provides a base set of properties for its subclasses.
- 4
- 5 • Decorator – Base class for elements that apply effects onto or around a
- 6 single child element, such as Border.
- 7 • DockPanel – Defines an area within which you can arrange child elements
- 8 either horizontally or vertically, relative to each other.
- 9
- 10 • DragDeltaEventArgs – This DragDeltaEventArgs class contains additional
- 11 information about the DragDeltaEvent event.
- 12 • FixedPanel – FixedPanel is the root element used in fixed-format
- 13 documents to contain fixed pages for pagination. FixedPanel displays
- 14 paginated content one page at a time or as a scrollable stack of pages.
- 15 • FlowPanel – FlowPanel is used to break, wrap, and align content that
- 16 exceeds the length of a single line. FlowPanel provides line-breaking and
- 17 alignment properties that can be used when the flow of the container's
- 18 content, Text for example, is likely to exceed the length of a single line.
- 19
- 20 • Frame – An area that can load the contents of another markup tree.
- 21 • Generator – Generator is the object that generates a UI on behalf of an
- 22 ItemsControl, working under the supervision of a GeneratorFactory.
- 23 • GeneratorFactory – A GeneratorFactory is responsible for generating the UI
- 24 on behalf of an ItemsControl. It maintains the association between the
- 25

1 items in the control's ItemsCollection (flattened view) and the
2 corresponding UIElements. The control's item-container can ask the
3 factory for a Generator, which does the actual generation of UI.

- 4 • GridPanel – Defines a grid area consisting of columns and rows.
- 5
- 6 • HeaderItemsControl – The base class for all controls that contain multiple
7 items and have a header.
- 8 • HorizontalScrollBar – The Horizontal ScrollBar class.
- 9
- 10 • HorizontalSlider – The Horizontal Slider class.
- 11
- 12 • HyperLink – The HyperLink class implements navigation control. The
13 default presenter is TextPresenter.
- 14
- 15 • Image – Provides an easy way to include an image in a document or an
16 application.
- 17
- 18 • IncludeContactEventArgs – Arguments passed to handlers of the
19 ContactPickerDialog.IncludeContact event.
- 20
- 21 • ItemCollection – Maintains a collection of discrete items within a control.
22 Provides methods and properties that enable changing the collection
23 contents and obtaining data about the contents.
- 24 • ItemsChangedEventArgs – The ItemsChanged event is raised by a
25 GeneratorFactory to inform layouts that the items collection has changed.

- 1 • ItemsView – ItemsView provides a flattened view of an ItemCollection.
- 2 • KeyboardNavigation – KeyboardNavigation class provide methods for
- 3 logical (Tab) and directional (arrow) navigation between focusable
- 4 controls.
- 5
- 6 • ListBox – Control that implements a list of selectable items.
- 7 • ListItem – Control that implements a selectable item inside a ListBox.
- 8
- 9 • Menu – Control that defines a menu of choices for users to invoke.
- 10 • MenuItem – A child item of Menu. MenuItems can be selected to invoke
- 11 commands. MenuItems can be separators. MenuItems can be headers for
- 12 submenus. MenuItems can be checked or unchecked.
- 13
- 14 • PageViewer – Represents a document-viewing composite control that
- 15 contains a pagination control, a toolbar, and a page bar control.
- 16 • PaginationCompleteEventArgs – The event arguments for the
- 17 PaginationCompleteEvent.
- 18 • PaginationProgressEventArgs – The event arguments for the
- 19 PaginationProgressEvent.
- 20
- 21 • Pane – Provides a way to define window properties in a markup language
- 22 (e.g., "XAML") without launching a new window.
- 23 • Panel – Provides a base class for all Panel elements. In order to instantiate
- 24 a Panel element, use the derived concrete class.
- 25

- 1 • **RadioButton** – **RadioButton** implements an option button with two states:
2 true or false.
- 3 • **RadioButtonList** – This control serves as a grouping control for
4 **RadioButtons** and is the piece that handles **RadioButton** mutual exclusivity.
5 The **RadioButtonList** inherits from **Selector**. The **RadioButtonList** is
6 essentially a **Single SelectionMode Selector** and the concept of **Selection**
7 (from **Selector**) is keyed off of the **Checked** property of the **RadioButton** it
8 is grouping.
- 9 • **RowStyle** – Changeable pattern Changeable elements.
- 10 • **RowStyles** – Changeable pattern **ICollection** object that is a collection of
11 Changeable elements.
- 12 • **ScrollEventArgs** – The **ScrollEventArgs** describe a change
13 in scrolling state.
- 14 • **ScrollViewer** –
- 15 • **SelectedItemCollection** – A container for the selected items in a **Selector**.
- 16 • **SelectionChangedEventArgs** – The inputs to a selection changed event
17 handler.
- 18 • **SimpleText** – **SimpleText** is a lightweight, multi-line, single-format text
19 element intended for use in user interface (UI) scenarios. **SimpleText**
20 exposes several of the same formatting properties as **Text** and can often be
21 used for a performance gain at the cost of some versatility.
- 22
- 23
- 24
- 25

- 1 • **StyleSelector** – StyleSelector allows the app writer to provide custom style
2 selection logic. For example, with a class Bug as the Content, use a
3 particular style for Pri1 bugs and a different style for Pri2 bugs. An
4 application writer can override the SelectStyle method in a derived selector
5 class and assign an instance of this class to the StyleSelector property on
6 ContentPresenter class.
- 7 • **Text** – Represents a Text control that enables rendering of multiple formats
8 of Text. Text is best used within an application UI; more advanced text
9 scenarios benefit from the additional feature set of TextPanel. In most
10 cases where relatively simple text support is required, Text is the preferred
11 element because of its lightweight nature and range of features.
- 12 • **TextBox** – Represents the control that provides an editable region that
13 accepts text input.
- 14 • **TextChangedEventArgs** – The TextChangedEventArgs class represents a
15 type of RoutedEventArgs that are relevant to events raised by
16 TextRange.SetText().
- 17 • **TextPanel** – Formats, sizes, and draws text. TextPanel supports multiple
18 lines of text and multiple text formats.
- 19 • **ToolTip** – A control to display information when the user hovers over a
20 control.
- 21 • **ToolTipEventArgs** – The data sent on a ToolTipEvent.
- 22
- 23
- 24
- 25

- 1 • TransformDecorator – TransformDecorator contains a child and applies a
2 specified transform to it. TransformDecorator implements logic to measure
3 and arrange the child in its local (pre-transform) coordinates such that after
4 the transform, the child fits tightly within the decorator's space and uses
5 maximal area. The child therefore needs to have no knowledge that a
6 transform has been applied to it.
- 7 • UIElementCollection – A UIElementCollection is a ordered collection of
8 UIElements.
- 9 • ValueChangedEventArgs – This ValueChangedEventArgs class contains
10 additional information about the ValueChangedEvent event.
- 11 • VerticalScrollBar – The Vertical ScrollBar class.
- 12 • VerticalSlider – The Vertical Slider class.
- 13 • Video – Plays a streaming video or audio file in a specified rectangle within
14 the current user coordinate system.
- 15 • VisibleChangedEventArgs – The VisibleChangedEventArgs class contains
16 additional information about the VisibleChangedEvent event.
- 17
- 18
- 19
- 20

21 The System.Windows.Controls namespace also contains various
22 enumerations. The following list contains example enumerations associated with
23 the System.Windows.Controls namespace.

24

25

- 1 • **CharacterCase** – Specifies the case of characters in a TextBox control when
2 the text is typed.
- 3 • **CheckState** – Specifies the state of a control, such as a check box, that can
4 be checked, unchecked, or set to an indeterminate state.
- 5 • **ClickMode** – Specifies when the Click event should fire.
- 6 • **ContactControlPropertyPosition** – Controls the position and display of the
7 property of the contact.
- 8 • **ContactPickerDialogLayout** – Specifies how the ContactPickerDialog
9 should display selected properties.
- 10 • **ContactPropertyCategory** – Specifies which value to treat as the default in
11 the case where a property has multiple values for the user could choose
12 from. For example, if "Work" is specified as the preferred category when
13 requesting a phone number property from the ContactPickerDialog, and the
14 user selects a contact with both a work and home phone number, the work
15 phone number appears as the default selection. The user can then use the
16 UI to choose the home phone number instead.
- 17 • **ContactPropertyType** – Specifies a property of a contact that the
18 ContactPickerDialog can ask the user for.
- 19 • **ContactType** – Specifies which contact types to display in the
20 ContactPickerDialog.
- 21 • **Direction** – This enumeration is used by the GeneratorFactory and
22 Generator to specify the direction in which the generator produces UI.
- 23
24
25

- 1 • Dock – Specifies the Dock position of a child element within a DockPanel.
- 2 • GeneratorStatus – This enumeration is used by the GeneratorFactory to
- 3 indicate its status.
- 4
- 5 • KeyNavigationMode – The type of TabNavigation property specify how
- 6 the container will move the focus when Tab navigation occurs.
- 7 • MenuItemBehavior – Defines the different behaviors that a MenuItem
- 8 could have.
- 9 • MenuItemType – Defines the different placement types of MenuItems.
- 10
- 11 • Orientation – Slider orientation types.
- 12 • PageViewerFit – Selects how pages should be fit into the PageViewer's
- 13 Client area.
- 14 • PageViewerMode – Selects the current PageViewer mode Reflected in the
- 15 mode dropdown.
- 16
- 17 • ScrollerVisibility – ScrollerVisibilty defines the visiblity behavior of a
- 18 scrollbar.
- 19 • SelectionMode – Specifies the selection behavior for the ListBox.
- 20
- 21

22 “Position” is an example structure associated with the
23 System.Windows.Controls namespace. A user of the Generator describes
24 positions using this structure. For example: To start generating forward from the
25

1 beginning of the item list, specify position (-1, 0) and direction Forward. To start
2 generating backward from the end of the list, specify position (-1, 0) and direction
3 Backward. To generate the items after the element with index k, specify position
4 (k, 0) and direction Forward.

5
6
7 The following list contains example delegates associated with the
8 System.Windows.Controls namespace.

- 9 • **CheckedChangedEventHandler** – This delegate is used by handlers of the
10 **CheckedChangedEvent** event.
- 11 • **CheckStateChangedEventHandler** – This delegate is used by handlers of
12 the **CheckStateChangedEvent** event.
- 13
14 • **ClickEventHandler** – Represents the methods that handle the Click event.
- 15 • **ContactTextBoxSelectionChangedEventHandler** – A delegate handler for
16 the **ContactTextBoxSelectionChanged** event.
- 17 • **ContactTextBoxTextChangedEventHandler** – A delegate handler for the
18 **ContactTextBoxTextChanged** event.
- 19
20 • **ContactTextBoxTextResolvedEventHandler** – A delegate handler for the
21 **TextResolvedToContact** event.
- 22 • **ContentChangedDelegate** – Delegate for the **ContentChangedEvent**.
- 23
24 • **ContextMenuEventHandler** - The callback type for handling a
25 **ContextMenuEvent**.

- 1 • DragDeltaEventHandler – This delegate is used by handlers of the
2 DragDeltaEvent event.
- 3 • IncludeContactEventHandler – Handler for
4 ContactPickerDialog.IncludeContact event.
- 5 • ItemsChangedEventHandler – The delegate to use for handlers that receive
6 ItemsChangedEventArgs.
- 7 —
- 8 • OpenedEventHandler – Handler for ContactPickerDialog.Opened event.
- 9 • PaginationCompleteDelegate – Delegate for the PaginationCompleteEvent.
- 10 • PaginationProgressDelegate – Delegate for the PaginationProgressEvent.
- 11 • ScrollChangeEventHandler – This delegate is used by handlers of the
12 ScrollChangeEvent event.
- 13 • SelectionChangedEventHandler – The delegate type for handling a
14 selection changed event.
- 15 • TextChangedEventHandler – The delegate to use for handlers that receive
16 TextChangedEventArgs.
- 17 • ToolTipEventHandler – The callback type for handling a ToolTipEvent.
- 18 • ValueChangedEventHandler – This delegate is used by handlers of the
19 ValueChangeEvent event.
- 20 • VisibleChangedEventHandler – This delegate is used by handlers of the
21 VisibleChangeEvent event.
- 22 • VisibleChangedEventHandler – This delegate is used by handlers of the
23 VisibleChangeEvent event.
- 24 • VisibleChangedEventHandler – This delegate is used by handlers of the
25 VisibleChangeEvent event.

Another namespace, `System.Windows.Controls.Atoms`, is a sub-namespace of the `System.Windows.Controls` namespace. `System.Windows.Controls.Atoms` includes associated controls, event arguments and event handlers. The following list contains example classes associated with the `System.Windows.Controls.Atoms` namespace.

- `PageBar` – Represents a scrollable pagination control.
- `PageElement` – Renders a specific page of paginated content. The page to be rendered is specified by the `PageSource` property.
- `PageHoveredEventArgs` – `PageHoveredEventArgs` provides information about where the mouse pointer is hovering.
- `PageScrolledEventArgs` – The `PageScrolledEventArgs` contains info pertaining to the `PageScrolled` Event.
- `PageSelectedEventArgs` – The `PageSelectedEvent` is fired when a new row/column range selection is made.
- `PageSelector` – `PageSelector`: Allows the user to select a range of rows/columns of pages to be displayed.
- `PageSource` – Identifies the source of the content to be paginated. It also provides properties and methods for formatting paginated content.

1 The following list contains example delegates associated with the
2 `System.Windows.Controls.Atoms` namespace.

- 3 • `PageHoveredEventHandler` – This delegate is used by handlers of the
4 `PageHoveredEvent` event.
- 5 • `PageScrolledEventHandler` – This delegate is used by handlers of the
6 `PageScrolled` event.
- 7 • `PageSelectedEventHandler` – This delegate is used by handlers of the
8 `PageSelectedEvent` event.
- 9 • `PageSelectedEvent` event.

10
11
12 A `System.Windows.Controls.Primitives` namespace is another sub-
13 namespace of the `System.Windows.Controls` namespace. As mentioned above,
14 the `Primitives` sub-namespace includes controls that are intended to be used as
15 primitives by other more complex controls. The following list contains example
16 classes associated with the `System.Windows.Controls.Primitives` namespace.

- 17 • `ButtonBase` – When overridden in a derived class, defines the relevant
18 events and properties, and provides handlers for the relevant input events.
- 19 • `Popup` – A control that creates a fly-out window that contains content.
- 20 • `RangeBase` – Represents the base class for elements that have a specific
21 range. Examples of such elements are scroll bars and progress bars. This
22 class defines the relevant events and properties, and provides handlers for
23 the events.
- 24
- 25

- 1 • RepeatButton – RepeatButton control adds repeating semantics of when the
2 Click event occurs.
- 3 • ScrollArea – ScrollArea is the effective element for scrolling. It contains
4 content that it clips and provides properties to expose the content's offset
5 and extent. It also provides default input handling such that scrolling can
6 be driven programatically or via keyboard or mouse wheel.
- 7
- 8 • ScrollBar – The ScrollBar class.
- 9 • Selector – The base class for controls that select items from among their
10 children.
- 11
- 12 • Slider – The Slider class.
- 13 • Thumb – The thumb control enables basic drag-movement functionality for
14 scrollbars and window resizing widgets.
- 15

16
17 “IEnsureVisible” is an example interface associated with the
18 System.Windows.Controls.Primitives namespace. IEnsureVisible is implemented
19 on a visual to scroll/move a child visual into view.

20 The following list contains example enumerations associated with the
21 System.Windows.Controls.Primitives namespace.

- 22 • ArrowButtonStates –
- 23
- 24 • CloseModeType – Describes how a popup should behave to various mouse
25 events.

- Part – The Part enumeration is used to indicate the semantic use of the controls that make up the scroll bar.
- PartStates – ScrollBar Part States.
- PlacementType – Describes where a popup should be placed on screen.
- SizeBoxStates –

A documents namespace 312 is a collection of semantic and formatting elements that are used to create richly formatted and semantically rich documents. In one embodiment, an “element” is a class that is primarily used in conjunction with a hierarchy of elements (referred to as a “tree”). These elements can be interactive (e.g., receiving user input via keyboard, mouse or other input device), can render images or objects, and can assist with the arrangement of other elements. Example elements include a “Block” element that implements a generic block, a “Body” element that represents content that includes the body of a table, a “Cell” element that contains tabular data within a table, a “Header” element that represents the content included in the header of a table, and a “PageBreak” element that is used to break content across multiple pages.

The following list contains example classes exposed by the System.Windows.Documents namespace.

- AdaptiveMetricsContext - AdaptiveMetricsContext provides the root element for adaptive-flow-format documents. Once a child panel is encapsulated in an AdaptiveMetricsContext element, the content of the panel is processed by the Reading Metrics Engine (RME). The size of the

child panel is used to calculate the number and size of any columns as well as optimum font sizes and line heights.

- **Block** - Implements a generic block element that does not induce default rendering behavior.
- **BlockElement** - Implements a base class for all Block elements.
- **Body** - Represents the content that comprises the body of a Table element.
- **Bold** - Implements a Bold element derived from Inline.
- **BreakRecord** - Stores information necessary to continue formatting paginated content across page breaks. Inherit from this class to provide pagination support. This is an abstract class.
- **Cell** - Cells contain tabular data within a Table. Cell elements are contained within a Row.
- **CellCollection** - Ordered collection of table cells.
- **Column** - The Column element is used to apportion the contents of a GridPanel or Table.
- **ColumnCollection** - A ColumnCollection is an ordered collection of Columns.
- **ColumnResult** - Represents a column's view-related information.

- 1 • **ContainerParagraphResult** – Provides access to calculated layout
2 parameters for a Paragraph object which contains only other Paragraph
3 objects.
- 4 • **ContentPosition** - Represents the position of content within a paragraph.
5 Inherit from this class to describe the position of associated content. This is
6 an abstract class.
- 7 • **Document** - The purpose of the Document class is to decouple the content
8 of a document from the UI "chrome" that surrounds it. "Decoupling"
9 means that you can author a document without thinking about (and without
10 committing to) its UI. The Document class holds document content,
11 typically a TextPanel or a FixedPanel and its children. A visual tree (by
12 default, a PageViewer) is associated with this element through the WPP
13 control styling mechanism.
- 14 • **DocumentPage** - Represents layout information for a control associated
15 with a page of a document subject to pagination. Inherit from this class to
16 implement to describe the layout information for these controls. This is an
17 abstract class.
- 18 • **DocumentPageParagraphResult** - Provides access to calculated layout
19 parameters for objects affected by pagination.
- 20 • **FindEngine** – Base class for find algorithms.
- 21 • **FindEngineFactory** – Find algorithms factory.
- 22
- 23
- 24
- 25

- 1 • FixedPage - Provides access to a single page of content within a fixed-
2 format layout document.
- 3 • Footer – Represents the content that comprises the footer of a Table
4 element.
- 5 • Header - Represents the content that comprises the header of a Table
6 element.
- 7
- 8 • Heading – Implements a block-level element that renders text as a heading.
- 9
- 10 • HyphenationDictionary - HyphenationDictionary represents a dictionary for
11 the purpose of providing hyphenation support within applications. It can
12 contain both an inline dictionary and a reference to an external dictionary.
13 The inline dictionary has higher priority and will be applied before entries
14 in the external dictionary.
- 15 • Hyphenator – The Hyphenator object maintains reference to hyphenation
16 data within a HyphenationDictionary and also performs hyphenation.
- 17 • Inline - Implements a generic Inline element that does not induce any
18 default rendering behavior.
- 19
- 20 • InlineElement - Implements a generic inline element as base class for all
21 inline elements.
- 22 • Italic – Implements an Italic element derived from Inline.
- 23 • LineBreak - Represents a markup element that forces a line break.
- 24
- 25 • LineResult - Provides access to calculated information of a line of text.

- 1 • List - Implements a List element. Lists are block-level elements designed
2 to be formatted with markers such as bullets or numbering.
- 3 • ListElementItem - Implements a ListElementItem, which supports markers
4 such as bullets or numbering.
- 5 • Note - Implements a Note element, which is analagous to the note element
6 in HTML.
- 7 • PageBreak - Represents a markup element used to break content across
8 various pages.
- 9 • PageDescriptor - Implements PageDescriptor, which stores information
10 necessary to create paginated layout.
- 11 • Paragraph - Implements a block-level element used to render text in a
12 paragraph. Rendering behavior is analagous to that of the paragraph
13 element in HTML.
- 14 • ParagraphResult - Provides access to calculated layout parameters for a
15 Paragraph object.
- 16 • Row – Defines a row within a GridPanel or Table element.
- 17 • RowCollection – RowCollection represents an ordered collection of Rows.
- 18 • RowGroup – Specifies property defaults for a group of rows in a Table or
19 GridPanel.
- 20 • Section - Implements a generic container element. Rendering behavior is
21 analagous to the div element in HTML.
- 22 •
- 23 •
- 24 •
- 25 •

- 1 • SmallCaps - Implements an inline SmallCaps element. SmallCaps are
2 typographic forms that render as small capital versions of letters for
3 emphasis, as in a title.
- 4 • Subscript - Represents an inline Subscript element. Subscript characters
5 are written immediately below, below and to the left, or below and to the
6 right of other characters.
- 7 • Superscript - Represents an inline Superscript element. Superscript
8 characters are typically letters or numbers and render immediately above,
9 above and to the left, or above and to the right of other characters.
- 10 • Table – Table is used to display complex data in tabular form using a
11 markup language (e.g., "XAML").
- 12 • TextArray - Base API for text access and manipulation.
- 13 • TextChangedEventArgs - The TextChangedEventArgs defines the event
14 arguments sent when a TextArray is changed.
- 15 • TextElement - TextElement provides TextRange facilities for the TextTree.
16 It is an immutable, continuous TextRange with fixed endpoints. It provides
17 ContentElement Input, Focus and Eventing support. It also provides
18 DependencyObject property support.
- 19 • TextNavigator - This can enumerate text content. Implements a movable
20 TextPosition. It can move by text run or be positioned at a know location
21 in text.
- 22
- 23
- 24
- 25

- 1 • **TextParagraphResult** - Provides access to calculated layout parameters for
2 text, including floated objects and figures.
- 3 • **TextPosition** - This is an object representing a certain position in a
4 **TextArray**. A compact object representing a position in text automatically
5 maintains position when text changes. Comparison operations are only
6 applicable to positions within same **TextArray** (same Context) **TextPosition**
7 can be static or movable. **IsChangeable** property tells the kind of position.
- 8 • **TextRange** - **TextRange** is an abstract class providing generic association of
9 zero or more subranges with properties. Subrange manipulation is defined
10 on derived classes.
- 11 • **TextRangeMovable** – **TextRangeMovable** is an abstract class for movable
12 **TextRanges**. It adds the ability to move the start and end points based on
13 **TextUnits**.
- 14 • **TextTreeChangedEventArgs** - The **TextChangedEventArgs** defines the
15 event arguments sent when a **TextArray** is changed.
- 16 • **TextTreeDumper** - **TreeDumper** is a tree test class that is public due to
17 packaging issues.
- 18 • **TextTreeNavigator** - This is an object representing a certain moveable
19 position in a **TextTree**. It is a specific implementation of **TextNavigator** for
20 use only in the **TextTree**.
- 21
22
23
24
25

- 1 • **TextTreePosition** - This is an object representing a certain immutable
2 position in a **TextTree**. It is a specific implementation of **TextPosition** for
3 use only in the **TextTree**.
- 4 • **TextTreeRange** - Provides **TextRange** facilities for the **TextTree**. It is a
5 mutable, continuous **TextRange** with movable endpoints.
- 6 • **TextTreeRangeContentEnumerator** - Enumerator on object children directly
7 under a **TextTreeRange**.
- 8 • **TextUnit** - Extensible unit of text navigation.
- 9 • **TextUnits** - Commonly used text units for **TextPosition** and **TextRange**.
- 10 • **Typography** - Provides access to a rich set of **OpenType** typography
11 properties.
- 12 • **UIElementParagraphResult** - The **ParagraphResult** for a paragraph which is
13 composed entirely of a **UIElement**. Used for **Floaters**, **Figures** and
14 embedded block level **UIElements**.
- 15 • **Underline** - Implements an **Underline** element derived from **InlineElement**.
- 16
- 17
- 18
- 19
- 20

21 The following list contains example interfaces associated with the
22 **System.Windows.Documents** namespace.

- 23 • **IDocumentContentHost** - Implement this interface on a content host so that
24 children of that host can notify the host when content is changing.
- 25

- **IDocumentFormatter** - Implement this interface on an element to provide support for document features such as pagination.
- **ITextDocumentResult** - Implement this interface to maintain column information for a document.
- **ITextParagraphResult** - Implement this interface to provide text and positioning information for text paragraphs.

The following list contains example enumerations associated with the `System.Windows.Documents` namespace.

- **ElementEdge** - This identifies the edge of an object where a `TextPosition` is located.
- **FindAdvancedOptions** - The advanced search options used by `FindAlgorithm` (search initialization) and `TextRangeMovable/TextSelection` (simplified search execution) classes.
- **FindOptions** - The simplified search options used by `TextBox.Find` methods.
- **LogicalDirection** - `LogicalDirection` defines a logical direction for movement in text. It is also used to determine where a `TextPosition` will move when content is inserted at the `TextPosition`.
- **TextArrayRunType** - This identifies the run where a `TextPosition` is located, taking `LogicalDiretion` into account.

- `TextChangeOptions` - Possible text changes for `CanChangeText`.
- `TextMoveOptions` - This controls the movement of `TextNavigator` by specifying conditions to halt navigation.

The following list contains example delegates associated with the `System.Windows.Documents` namespace.

- `ObjectCloneDelegate` - Callback method to provide a clone or copy of a `DependencyObject` when a portion of a `TextArray` is being copied or moved.
- `TextChangedEventHandler` - The `TextChangedEventHandler` delegate is called with `TextChangedEventArgs` every time content is added to or removed from the `TextTree`.

A shapes namespace 314 is a collection of vector graphics elements used to create images and objects. The use of vector graphics elements allows the elements to be easily resized to fit the requirements of a particular interface or display device. The following list contains example classes exposed by the `System.Windows.Shapes` namespace.

- `Ellipse` – Draws an ellipse.
- `Glyphs` - Represents a glyph shape in a markup language such as "XAML".
Glyphs are used to represent fonts.
- `Line` – Draws a straight line between two points.

- 1 • Path – Draws a series of connected lines and curves.
- 2 • Polygon – Draws a polygon (a connected series of lines that forms a closed
- 3 shape).
- 4 • Polyline – Draws a series of connected straight lines.
- 5 • Rectangle – Draws a rectangle.
- 6 • Shape – An abstract class that provides base functionality for shape
- 7 elements, such as ellipse, polygon and rectangle.
- 8
- 9

10 The System.Windows.Controls, System.Windows.Documents and
11 System.Windows.Shapes namespaces provide an integrated system for developing
12 applications and related components. This integrated system provides a common
13 programming model for all three namespaces, thereby simplifying development of
14 application programs. This interoperability among all three namespaces allows
15 developers to learn a single programming architecture that is applied to any of the
16 features provided by three namespaces. For example, a common markup language
17 is used across all three namespaces. This common markup language provides for
18 the mapping of classes and properties specified in XML markup to an instantiated
19 tree of objects.

20 Additionally, a consistent programming model and consistent services are
21 used across the three namespaces. For example, a consistent event system is used
22 to initiate and process various events. A common property system is used to style
23 various properties, bind data to a property, or animate a property, regardless of
24 whether the property is associated with the “Controls”, “Documents”, or “Shapes”
25

1 namespace. Additionally, the same input paradigms and layout handling is
2 common across all three namespaces. For example, various controls from the
3 System.Windows.Controls namespace can be nested in the middle of a document's
4 content defined using the System.Windows.Documents namespace.

5 Example source files will include a set of windows and panes (also referred
6 to as "pages") that are declaratively defined using "Controls", "Documents" and
7 "Shapes". Interaction logic is also provided for the windows and panes. The
8 interaction logic identifies program code that is executed in response to a
9 particular user action or in response to occurrence of an event or activity. The
10 interaction logic is defined, for example, using a Common Language Runtime
11 (CLR) language. A CLR is a runtime environment that handles execution of
12 program code (e.g., .NET program code) and provides various services such as
13 security-related services and memory-related services. Example CLR languages
14 include C# and visual basic. Source files may also include other stand-alone
15 programming language files, such as C# or visual basic files.

16 Although this discussion refers to the integration of the "Controls",
17 "Documents" and "Shapes" namespaces, this integration may be applied to any or
18 all of the namespaces and sub-namespaces discussed herein.

19 A data namespace 316 includes classes and interfaces used to bind
20 properties of elements to data sources, data source classes, and data-specific
21 implementations of data collections and views. These classes and interfaces are
22 also used to handle exceptions in data entry and allow runtime creation of a user
23 interface based on information in various data sources. Data can be displayed in
24 textual form or can be utilized to change the formatting of the display, such as
25 displaying dollar amounts in red if they are negative. Example classes include a

1 “Bind” class that represents a binding declaration object that manages bindings
2 between a dynamic property user interface and source data, and an
3 “XmlDataSource” class that serves as a data source for data binding to XML
4 content nodes.

5 A media namespace 318 provides various media classes. Application
6 developers as well as component developers may use these classes to develop
7 various presentation functionality. Example classes in media namespace 318
8 include an “ImageEffect” class that permits certain imaging effects (e.g., blur and
9 grayscale), and a “Brush” class that provides a mechanism for filling an area using
10 solid colors, gradients, images, video, and the like.

11 The media namespace 318 includes a sub-namespace
12 System.Windows.Media.Animation that includes services that allow a developer
13 to animate properties and coordinate a set of animations with a set of timelines.
14 An animation is an object that changes a value over a period of time. Animation
15 effects include moving an object on the display, and changing the size, shape, or
16 color of an object. Multiple animation classes are provided to implement various
17 animation effects. Effects can be achieved by associating an animation with an
18 element’s property value. For example, to create a rectangle that fades in and out
19 of view, one or more animations are associated with the opacity property of the
20 rectangle.

21 The media namespace 318 also includes a sub-namespace
22 System.Windows.Media.TextFormatting that provides various text services. For
23 example, a “TextFormatter” text engine provides services for breaking text lines
24 and formatting text presented on a display. “TextFormatter” is capable of handling
25

1 different text character formats and paragraph styles as well as handling
2 international text layout.

3 A design namespace 320 provides classes that enable the editing of forms
4 and text, formatting data and cross-process data sharing. These classes provide an
5 extensible framework for editing documents, applications, and other content.

6 An input namespace 322 includes an input manager that coordinates inputs
7 received by the system. The input namespace 322 also includes classes that help
8 manage and provide control for different input devices, such as a keyboard or a
9 mouse.

10 A navigation namespace 324 provides a set of classes and services that
11 allow the building of applications with navigation paradigms, such as a browser
12 application. These classes and services permit the development of applications
13 with customized navigation experiences. For example, when purchasing a product
14 or service from an online merchant, clicking a “Back” button causes the
15 application to display a different page that asks the user if they want to cancel or
16 change their order. In another example, activating a “Refresh” button causes an
17 application to retrieve new data instead of first reloading the application followed
18 by retrieving the new data. The navigation namespace 324 also includes page
19 functions that provide a mechanism for generating a hierarchy of questions that are
20 presented to a user.

21 An automation namespace 326 provides a set of classes that support
22 accessibility and user interface automation.

23 A serialization namespace 328 provides a parser that can load or save a
24 hierarchy of objects (e.g., elements) from or to an XML file or a file with a binary
25

1 representation. This process also sets properties associated with the objects and
2 associates event handlers.

3 An interop namespace 330 provides a set of classes that enable
4 interoperability with other operating systems or computing platforms.

5 A forms.interop namespace 332 provides an element that allows an
6 application to host a form control operation.

7 Another namespace, System.IO.CompoundFile (not shown in Fig. 3)
8 provides services to utilize a compound file in which various document
9 distributable files are stored. These services allow for the encryption and
10 compression of content. The services also support the storage of multiple
11 renditions of the same content, such as a re-flowable document and a fixed-format
12 document.

13 14 EXEMPLARY COMPUTING SYSTEM AND ENVIRONMENT

15 Fig. 4 illustrates an example of a suitable computing environment 400
16 within which the programming framework 132 may be implemented (either fully
17 or partially). The computing environment 400 may be utilized in the computer
18 and network architectures described herein.

19 The exemplary computing environment 400 is only one example of a
20 computing environment and is not intended to suggest any limitation as to the
21 scope of use or functionality of the computer and network architectures. Neither
22 should the computing environment 400 be interpreted as having any dependency
23 or requirement relating to any one or combination of components illustrated in the
24 exemplary computing environment 400.
25

1 The framework 132 may be implemented with numerous other general
2 purpose or special purpose computing system environments or configurations.
3 Examples of well known computing systems, environments, and/or configurations
4 that may be suitable for use include, but are not limited to, personal computers,
5 server computers, multiprocessor systems, microprocessor-based systems, network
6 PCs, minicomputers, mainframe computers, distributed computing environments
7 that include any of the above systems or devices, and so on. Compact or subset
8 versions of the framework may also be implemented in clients of limited
9 resources, such as cellular phones, personal digital assistants, handheld computers,
10 or other communication/computing devices.

11 The framework 132 may be described in the general context of computer-
12 executable instructions, such as program modules, being executed by one or more
13 computers or other devices. Generally, program modules include routines,
14 programs, objects, components, data structures, etc. that perform particular tasks
15 or implement particular abstract data types. The framework 132 may also be
16 practiced in distributed computing environments where tasks are performed by
17 remote processing devices that are linked through a communications network. In
18 a distributed computing environment, program modules may be located in both
19 local and remote computer storage media including memory storage devices.

20 The computing environment 400 includes a general-purpose computing
21 device in the form of a computer 402. The components of computer 402 can
22 include, by are not limited to, one or more processors or processing units 404, a
23 system memory 406, and a system bus 408 that couples various system
24 components including the processor 404 to the system memory 406.

1 The system bus 408 represents one or more of several possible types of bus
2 structures, including a memory bus or memory controller, a peripheral bus, an
3 accelerated graphics port, and a processor or local bus using any of a variety of
4 bus architectures. By way of example, such architectures can include an Industry
5 Standard Architecture (ISA) bus, a Micro Channel Architecture (MCA) bus, an
6 Enhanced ISA (EISA) bus, a Video Electronics Standards Association (VESA)
7 local bus, and a Peripheral Component Interconnects (PCI) bus also known as a
8 Mezzanine bus.

9 Computer 402 typically includes a variety of computer readable media.
10 Such media can be any available media that is accessible by computer 402 and
11 includes both volatile and non-volatile media, removable and non-removable
12 media.

13 The system memory 406 includes computer readable media in the form of
14 volatile memory, such as random access memory (RAM) 410, and/or non-volatile
15 memory, such as read only memory (ROM) 412. A basic input/output system
16 (BIOS) 414, containing the basic routines that help to transfer information
17 between elements within computer 402, such as during start-up, is stored in ROM
18 412. RAM 410 typically contains data and/or program modules that are
19 immediately accessible to and/or presently operated on by the processing unit 404.

20 Computer 402 may also include other removable/non-removable,
21 volatile/non-volatile computer storage media. By way of example, Fig. 4
22 illustrates a hard disk drive 416 for reading from and writing to a non-removable,
23 non-volatile magnetic media (not shown), a magnetic disk drive 418 for reading
24 from and writing to a removable, non-volatile magnetic disk 420 (e.g., a “floppy
25 disk”), and an optical disk drive 422 for reading from and/or writing to a

1 removable, non-volatile optical disk 424 such as a CD-ROM, DVD-ROM, or other
2 optical media. The hard disk drive 416, magnetic disk drive 418, and optical disk
3 drive 422 are each connected to the system bus 408 by one or more data media
4 interfaces 426. Alternatively, the hard disk drive 416, magnetic disk drive 418,
5 and optical disk drive 422 can be connected to the system bus 408 by one or more
6 interfaces (not shown).

7 The disk drives and their associated computer-readable media provide non-
8 volatile storage of computer readable instructions, data structures, program
9 modules, and other data for computer 402. Although the example illustrates a hard
10 disk 416, a removable magnetic disk 420, and a removable optical disk 424, it is to
11 be appreciated that other types of computer readable media which can store data
12 that is accessible by a computer, such as magnetic cassettes or other magnetic
13 storage devices, flash memory cards, CD-ROM, digital versatile disks (DVD) or
14 other optical storage, random access memories (RAM), read only memories
15 (ROM), electrically erasable programmable read-only memory (EEPROM), and
16 the like, can also be utilized to implement the exemplary computing system and
17 environment.

18 Any number of program modules can be stored on the hard disk 416,
19 magnetic disk 420, optical disk 424, ROM 412, and/or RAM 410, including by
20 way of example, an operating system 426, one or more application programs 428,
21 other program modules 430, and program data 432. Each of the operating system
22 426, one or more application programs 428, other program modules 430, and
23 program data 432 (or some combination thereof) may include elements of the
24 programming framework 132.

1 A user can enter commands and information into computer 402 via input
2 devices such as a keyboard 434 and a pointing device 436 (e.g., a “mouse”).
3 Other input devices 438 (not shown specifically) may include a microphone,
4 joystick, game pad, satellite dish, serial port, scanner, and/or the like. These and
5 other input devices are connected to the processing unit 404 via input/output
6 interfaces 440 that are coupled to the system bus 408, but may be connected by
7 other interface and bus structures, such as a parallel port, game port, or a universal
8 serial bus (USB).

9 A monitor 442 or other type of display device can also be connected to the
10 system bus 408 via an interface, such as a video adapter 444. In addition to the
11 monitor 442, other output peripheral devices can include components such as
12 speakers (not shown) and a printer 446 which can be connected to computer 402
13 via the input/output interfaces 440.

14 Computer 402 can operate in a networked environment using logical
15 connections to one or more remote computers, such as a remote computing device
16 448. By way of example, the remote computing device 448 can be a personal
17 computer, portable computer, a server, a router, a network computer, a peer device
18 or other common network node, and so on. The remote computing device 448 is
19 illustrated as a portable computer that can include many or all of the elements and
20 features described herein relative to computer 402.

21 Logical connections between computer 402 and the remote computer 448
22 are depicted as a local area network (LAN) 450 and a general wide area network
23 (WAN) 452. Such networking environments are commonplace in offices,
24 enterprise-wide computer networks, intranets, and the Internet.

1 When implemented in a LAN networking environment, the computer 402 is
2 connected to a local network 450 via a network interface or adapter 454. When
3 implemented in a WAN networking environment, the computer 402 typically
4 includes a modem 456 or other means for establishing communications over the
5 wide network 452. The modem 456, which can be internal or external to computer
6 402, can be connected to the system bus 408 via the input/output interfaces 440 or
7 other appropriate mechanisms. It is to be appreciated that the illustrated network
8 connections are exemplary and that other means of establishing communication
9 link(s) between the computers 402 and 448 can be employed.

10 In a networked environment, such as that illustrated with computing
11 environment 400, program modules depicted relative to the computer 402, or
12 portions thereof, may be stored in a remote memory storage device. By way of
13 example, remote application programs 458 reside on a memory device of remote
14 computer 448. For purposes of illustration, application programs and other
15 executable program components such as the operating system are illustrated herein
16 as discrete blocks, although it is recognized that such programs and components
17 reside at various times in different storage components of the computing device
18 402, and are executed by the data processor(s) of the computer.

19 An implementation of the framework 132, and particularly, the API 142 or
20 calls made to the API 142, may be stored on or transmitted across some form of
21 computer readable media. Computer readable media can be any available media
22 that can be accessed by a computer. By way of example, and not limitation,
23 computer readable media may comprise “computer storage media” and
24 “communications media.” “Computer storage media” include volatile and non-
25 volatile, removable and non-removable media implemented in any method or

1 technology for storage of information such as computer readable instructions, data
2 structures, program modules, or other data. Computer storage media includes, but
3 is not limited to, RAM, ROM, EEPROM, flash memory or other memory
4 technology, CD-ROM, digital versatile disks (DVD) or other optical storage,
5 magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage
6 devices, or any other medium which can be used to store the desired information
7 and which can be accessed by a computer.

8 “Communication media” typically embodies computer readable
9 instructions, data structures, program modules, or other data in a modulated data
10 signal, such as carrier wave or other transport mechanism. Communication media
11 also includes any information delivery media. The term “modulated data signal”
12 means a signal that has one or more of its characteristics set or changed in such a
13 manner as to encode information in the signal. By way of example, and not
14 limitation, communication media includes wired media such as a wired network or
15 direct-wired connection, and wireless media such as acoustic, RF, infrared, and
16 other wireless media. Combinations of any of the above are also included within
17 the scope of computer readable media.

18 Alternatively, portions of the framework may be implemented in hardware
19 or a combination of hardware, software, and/or firmware. For example, one or
20 more application specific integrated circuits (ASICs) or programmable logic
21 devices (PLDs) could be designed or programmed to implement one or more
22 portions of the framework.

23 A programming interface (or more simply, interface) may be viewed as any
24 mechanism, process, protocol for enabling one or more segment(s) of code to
25 communicate with or access the functionality provided by one or more other

1 segment(s) of code. Alternatively, a programming interface may be viewed as one
2 or more mechanism(s), method(s), function call(s), module(s), object(s), etc. of a
3 component of a system capable of communicative coupling to one or more
4 mechanism(s), method(s), function call(s), module(s), etc. of other component(s).
5 The term “segment of code” in the preceding sentence is intended to include one
6 or more instructions or lines of code, and includes, e.g., code modules, objects,
7 subroutines, functions, and so on, regardless of the terminology applied or whether
8 the code segments are separately compiled, or whether the code segments are
9 provided as source, intermediate, or object code, whether the code segments are
10 utilized in a runtime system or process, or whether they are located on the same or
11 different machines or distributed across multiple machines, or whether the
12 functionality represented by the segments of code are implemented wholly in
13 software, wholly in hardware, or a combination of hardware and software.

14 Notionally, a programming interface may be viewed generically, as shown
15 in Fig. 5 or Fig. 6. Fig. 5 illustrates an interface Interface1 as a conduit through
16 which first and second code segments communicate. Fig. 6 illustrates an interface
17 as comprising interface objects I1 and I2 (which may or may not be part of the
18 first and second code segments), which enable first and second code segments of a
19 system to communicate via medium M. In the view of Fig. 6, one may consider
20 interface objects I1 and I2 as separate interfaces of the same system and one may
21 also consider that objects I1 and I2 plus medium M comprise the interface.
22 Although Figs. 5 and 6 show bi-directional flow and interfaces on each side of the
23 flow, certain implementations may only have information flow in one direction (or
24 no information flow as described below) or may only have an interface object on
25 one side. By way of example, and not limitation, terms such as application

1 programming or program interface (API), entry point, method, function,
2 subroutine, remote procedure call, and component object model (COM) interface,
3 are encompassed within the definition of programming interface.

4 Aspects of such a programming interface may include the method whereby
5 the first code segment transmits information (where “information” is used in its
6 broadest sense and includes data, commands, requests, etc.) to the second code
7 segment; the method whereby the second code segment receives the information;
8 and the structure, sequence, syntax, organization, schema, timing and content of
9 the information. In this regard, the underlying transport medium itself may be
10 unimportant to the operation of the interface, whether the medium be wired or
11 wireless, or a combination of both, as long as the information is transported in the
12 manner defined by the interface. In certain situations, information may not be
13 passed in one or both directions in the conventional sense, as the information
14 transfer may be either via another mechanism (e.g. information placed in a buffer,
15 file, etc. separate from information flow between the code segments) or non-
16 existent, as when one code segment simply accesses functionality performed by a
17 second code segment. Any or all of these aspects may be important in a given
18 situation, e.g., depending on whether the code segments are part of a system in a
19 loosely coupled or tightly coupled configuration, and so this list should be
20 considered illustrative and non-limiting.

21 This notion of a programming interface is known to those skilled in the art
22 and is clear from the foregoing detailed description of the invention. There are,
23 however, other ways to implement a programming interface, and, unless expressly
24 excluded, these too are intended to be encompassed by the claims set forth at the
25 end of this specification. Such other ways may appear to be more sophisticated or

1 complex than the simplistic view of Figs. 5 and 6, but they nonetheless perform a
2 similar function to accomplish the same overall result. We will now briefly
3 describe some illustrative alternative implementations of a programming interface.

4 5 A. FACTORING

6 A communication from one code segment to another may be accomplished
7 indirectly by breaking the communication into multiple discrete communications.
8 This is depicted schematically in Figs. 7 and 8. As shown, some interfaces can be
9 described in terms of divisible sets of functionality. Thus, the interface
10 functionality of Figs. 5 and 6 may be factored to achieve the same result, just as
11 one may mathematically provide 24, or 2 times 2 times 3 times 2. Accordingly, as
12 illustrated in Fig. 7, the function provided by interface Interface1 may be
13 subdivided to convert the communications of the interface into multiple interfaces
14 Interface1A, Interface 1B, Interface 1C, etc. while achieving the same result. As
15 illustrated in Fig. 8, the function provided by interface I1 may be subdivided into
16 multiple interfaces I1a, I1b, I1c, etc. while achieving the same result. Similarly,
17 interface I2 of the second code segment which receives information from the first
18 code segment may be factored into multiple interfaces I2a, I2b, I2c, etc. When
19 factoring, the number of interfaces included with the 1st code segment need not
20 match the number of interfaces included with the 2nd code segment. In either of the
21 cases of Figs. 7 and 8, the functional spirit of interfaces Interface1 and I1 remain
22 the same as with Figs. 5 and 6, respectively. The factoring of interfaces may also
23 follow associative, commutative, and other mathematical properties such that the
24 factoring may be difficult to recognize. For instance, ordering of operations may
25 be unimportant, and consequently, a function carried out by an interface may be

1 carried out well in advance of reaching the interface, by another piece of code or
2 interface, or performed by a separate component of the system. Moreover, one of
3 ordinary skill in the programming arts can appreciate that there are a variety of
4 ways of making different function calls that achieve the same result.

6 B. REDEFINITION

7 In some cases, it may be possible to ignore, add or redefine certain aspects
8 (e.g., parameters) of a programming interface while still accomplishing the
9 intended result. This is illustrated in Figs. 9 and 10. For example, assume interface
10 Interface1 of Fig. 5 includes a function call *Square(input, precision, output)*, a call
11 that includes three parameters, *input*, *precision* and *output*, and which is issued
12 from the 1st Code Segment to the 2nd Code Segment., If the middle parameter
13 *precision* is of no concern in a given scenario, as shown in Fig. 9, it could just as
14 well be ignored or even replaced with a *meaningless* (in this situation) parameter.
15 One may also add an *additional* parameter of no concern. In either event, the
16 functionality of square can be achieved, so long as output is returned after input is
17 squared by the second code segment. *Precision* may very well be a meaningful
18 parameter to some downstream or other portion of the computing system;
19 however, once it is recognized that *precision* is not necessary for the narrow
20 purpose of calculating the square, it may be replaced or ignored. For example,
21 instead of passing a valid *precision* value, a meaningless value such as a birth date
22 could be passed without adversely affecting the result. Similarly, as shown in Fig.
23 10, interface I1 is replaced by interface I1', redefined to ignore or add parameters
24 to the interface. Interface I2 may similarly be redefined as interface I2', redefined
25 to ignore unnecessary parameters, or parameters that may be processed elsewhere.

1 The point here is that in some cases a programming interface may include aspects,
2 such as parameters, that are not needed for some purpose, and so they may be
3 ignored or redefined, or processed elsewhere for other purposes.

4 5 C. INLINE CODING

6 It may also be feasible to merge some or all of the functionality of two
7 separate code modules such that the “interface” between them changes form. For
8 example, the functionality of Figs. 5 and 6 may be converted to the functionality
9 of Figs. 11 and 12, respectively. In Fig. 11, the previous 1st and 2nd Code Segments
10 of Fig. 5 are merged into a module containing both of them. In this case, the code
11 segments may still be communicating with each other but the interface may be
12 adapted to a form which is more suitable to the single module. Thus, for example,
13 formal Call and Return statements may no longer be necessary, but similar
14 processing or response(s) pursuant to interface Interface1 may still be in effect.
15 Similarly, shown in Fig. 12, part (or all) of interface I2 from Fig. 6 may be written
16 inline into interface I1 to form interface I1”. As illustrated, interface I2 is divided
17 into I2a and I2b, and interface portion I2a has been coded in-line with interface I1
18 to form interface I1”. For a concrete example, consider that the interface I1 from
19 Fig. 6 performs a function call *square (input, output)*, which is received by
20 interface I2, which after processing the value passed with *input* (to square it) by
21 the second code segment, passes back the squared result with *output*. In such a
22 case, the processing performed by the second code segment (squaring *input*) can
23 be performed by the first code segment without a call to the interface.

24 25 D. DIVORCE

1 A communication from one code segment to another may be accomplished
2 indirectly by breaking the communication into multiple discrete communications.
3 This is depicted schematically in Figs. 13 and 14. As shown in Fig. 13, one or
4 more piece(s) of middleware (Divorce Interface(s), since they divorce
5 functionality and / or interface functions from the original interface) are provided
6 to convert the communications on the first interface, Interface1, to conform them
7 to a different interface, in this case interfaces Interface2A, Interface2B and
8 Interface2C. This might be done, e.g., where there is an installed base of
9 applications designed to communicate with, say, an operating system in
10 accordance with an Interface1 protocol, but then the operating system is changed
11 to use a different interface, in this case interfaces Interface2A, Interface2B and
12 Interface2C. The point is that the original interface used by the 2nd Code Segment
13 is changed such that it is no longer compatible with the interface used by the 1st
14 Code Segment, and so an intermediary is used to make the old and new interfaces
15 compatible. Similarly, as shown in Fig. 14, a third code segment can be introduced
16 with divorce interface DI1 to receive the communications from interface I1 and
17 with divorce interface DI2 to transmit the interface functionality to, for example,
18 interfaces I2a and I2b, redesigned to work with DI2, but to provide the same
19 functional result. Similarly, DI1 and DI2 may work together to translate the
20 functionality of interfaces I1 and I2 of Fig. 6 to a new operating system, while
21 providing the same or similar functional result.

22 23 E. REWRITING

24 Yet another possible variant is to dynamically rewrite the code to replace
25 the interface functionality with something else but which achieves the same

1 overall result. For example, there may be a system in which a code segment
2 presented in an intermediate language (e.g. Microsoft IL, Java ByteCode, etc.) is
3 provided to a Just-in-Time (JIT) compiler or interpreter in an execution
4 environment (such as that provided by the .Net framework, the Java runtime
5 environment, or other similar runtime type environments). The JIT compiler may
6 be written so as to dynamically convert the communications from the 1st Code
7 Segment to the 2nd Code Segment, i.e., to conform them to a different interface as
8 may be required by the 2nd Code Segment (either the original or a different 2nd
9 Code Segment). This is depicted in Figs. 15 and 16. As can be seen in Fig. 15, this
10 approach is similar to the Divorce scenario described above. It might be done, e.g.,
11 where an installed base of applications are designed to communicate with an
12 operating system in accordance with an Interface 1 protocol, but then the operating
13 system is changed to use a different interface. The JIT Compiler could be used to
14 conform the communications on the fly from the installed-base applications to the
15 new interface of the operating system. As depicted in Fig. 16, this approach of
16 dynamically rewriting the interface(s) may be applied to dynamically factor, or
17 otherwise alter the interface(s) as well.

18 It is also noted that the above-described scenarios for achieving the same or
19 similar result as an interface via alternative embodiments may also be combined in
20 various ways, serially and/or in parallel, or with other intervening code. Thus, the
21 alternative embodiments presented above are not mutually exclusive and may be
22 mixed, matched and combined to produce the same or equivalent scenarios to the
23 generic scenarios presented in Figs. 5 and 6. It is also noted that, as with most
24 programming constructs, there are other similar ways of achieving the same or
25 similar functionality of an interface which may not be described herein, but

1 nonetheless are represented by the spirit and scope of the invention, i.e., it is noted
2 that it is at least partly the functionality represented by, and the advantageous
3 results enabled by, an interface that underlie the value of an interface.

4 5 **Conclusion**

6 Although the invention has been described in language specific to structural
7 features and/or methodological acts, it is to be understood that the invention
8 defined in the appended claims is not necessarily limited to the specific features or
9 acts described. Rather, the specific features and acts are disclosed as exemplary
10 forms of implementing the claimed invention.

CLAIMS:

1. A programming interface embodied on one or more computer readable media, comprising:
a first group of services related to generating graphical objects;
a second group of services related to formatting content; and
a third group of services related to creating components of the graphical objects;
wherein the first group of services, the second group of services and the third group of services share a common programming model.
2. A programming interface as recited in claim 1, wherein the first group of services, the second group of services and the third group of services utilize a common markup language.
3. A programming interface as recited in claim 1, wherein the first group of services, the second group of services and the third group of services share a common event system.
4. A programming interface as recited in claim 1, wherein the first group of services, the second group of services and the third group of services share a common property definition system.
5. A programming interface as recited in claim 1, wherein the first group of services, the second group of services and the third group of services share a common input paradigm.
6. A programming interface as recited in claim 1, wherein the first group of services, the second group of services and the third group of services share a common system for nesting elements associated with a particular group of services within elements associated with another group of services.

7. A programming interface as recited in claim 1, wherein the first group of services includes a service that determines an appearance of the graphical objects.
8. A programming interface as recited in claim 1, wherein the first group of services includes a service that determines a behavior of the graphical objects.
9. A programming interface as recited in claim 1, wherein the first group of services includes a service that determines an arrangement of the graphical objects.
10. A programming interface as recited in claim 1, wherein the first group of services includes a plurality of nested elements that define the graphical objects.
11. A programming interface as recited in claim 1, wherein the graphical objects are comprised of one or more elements defined by vector graphics.
12. A programming interface as recited in claim 1, wherein the first group of services can define window properties in a markup language without launching a new window.
13. A programming interface as recited in claim 1, wherein the first group of services generate a user interface containing a plurality of graphical objects.
14. A programming interface as recited in claim 1, wherein the second group of services arrange the graphical objects.
15. A software architecture comprising the programming interface as recited in claim 1.

16. An application program interface embodied on one or more computer readable media, comprising:

a first group of services related to generating graphical objects;

a second group of services related to formatting content; and

a third group of services related to creating components of the graphical objects, wherein the first group of services, the second group of services and the third group of services share a common programming model.

17. An application program interface as recited in claim 17, wherein the first group of services, the second group of services and the third group of services utilize a common markup language.

18. An application program interface as recited in claim 17, wherein the third group of services includes services to generate geometric shapes.

19. An application program interface as recited in claim 17, wherein the second group of services includes arranging a plurality of data elements.

20. An application program interface as recited in claim 17, wherein the first group of services includes:

a service that determines an appearance of a graphical object; and

a service that determines a behavior of the graphical object.

21. An application program interface as recited in claim 17, wherein the first group of services includes a service that defines window properties in a markup language without launching a new window.

22. A computer system including one or more microprocessors and one or more software programs, the one or more software programs utilizing a programming interface to request services from an operating system, the programming interface including separate commands to request services consisting of the following groups of services:

a first group of services for generating graphical objects; and

a second group of services for creating components of the graphical objects, wherein the first group of services and the second group of services share a common programming model.

23. A computer system as recited in claim 23, wherein the first group of services includes:

a service for defining an appearance of the graphical objects; and
a service for defining an arrangement of the graphical objects.

24. A computer system as recited in claim 23, wherein the second group of services includes services to generate a plurality of geometric shapes.

25. A method comprising:

calling one or more first functions to facilitate generating graphical objects;
and

calling one or more second functions to facilitate formatting content, wherein the first functions and the second functions share a common programming model.

26. A method as recited in claim 26, further including calling one or more third functions to facilitate creating components of the graphical objects.

27. A method as recited in claim 26, further including calling one or more third functions to facilitate generating geometric shapes contained in the graphical objects.

28. A method as recited in claim 26, wherein the first functions facilitate:
defining window properties in a markup language without launching a new window; and
generating a user interface containing a plurality of graphical objects.

29. A system comprising:

means for exposing a first set of functions that enable generating graphical objects; and

means for exposing a second set of functions that enable creating components of the graphical objects, wherein the components of the graphical objects include a plurality of geometric shapes, and wherein the first set of functions and the second set of functions share a common programming model.

30. A system as recited in claim 30, wherein the second set of functions further enable arrangement of the geometric shapes on a page to be rendered.

31. A system as recited in claim 30, further comprising means for exposing a third set of functions that enable formatting content for display.

32. A system as recited in claim 30, wherein the first set of functions and the second set of functions utilize a common markup language.

33. A system as recited in claim 30, wherein the first set of functions and the second set of functions share a common event system and a common property definition system.