Microsoft

더욱 강력해진 **UI** 개발,
Windows Presentation Foundation

David Chappell, Chappell & Associates

2007 년

Contents

애플리케이션에 대한 사용자 경험보다 더 중요한 것은 없다. 많은 소프트웨어 전문가들은 애플리케이션이 어떻게 동작하는지에 더 많은 관심을 가지고 있지만, 사용자들은 사용자 인터페이스에 대해 더 많은 관심을 가지고 있다. 애플리케이션의 인터페이스는 애플리케이션에서 사용자가 경험하는 대부분이며, 사용자들에게는 사용자 경험이 바로 애플리케이션의 모든 것이다. 더 좋은 인터페이스를 통해 더 좋은 경험을 제공하게 되면 생산성을 높이고, 충성도 높은 고객을 만들고, 웹 사이트에서의 판매를 늘리게 된다.

한 때는 문자 기반의 인터페이스에 만족했던 사용자들은 이제 그래픽 인터페이스에 익숙해졌다. 그리고 사용자 인터페이스에 대한 요구사항은 점차 더 확대되고 있다. 그래픽과 미디어가 점차 널리 사용되었고, 웹은 소프트웨어도 쉽게 사용하고 싶어하는 새로운 세대의 사용자를 만들어냈다. 사람들이 애플리케이션을 더 오래 사용할 수록, 애플리케이션의 인터페이스가 더 중요해졌다. 높아가는 기대치를 맞추기 위해, 사용자 인터페이스를 만드는데 사용된 기술도 진화되어야 한다.

Windows Presentation Foundation (WPF)의 목적은 윈도우에서도 이런 진화를 지원하는 것이다. .NET Framework 3.0 버전에 처음 발표된 이 기술은, .NET Framework 3.5 에서 더욱 발전하였다. 개발자와 디자이너는 WPF 를 사용해서 문서, 미디어, 2/3 차원 그래픽, 애니메이션 등을 통합하는 인터페이스를 만들 수 있다. .NET Framework 3.5 의 다른 구성요소와 마찬가지로, WPF 는 Windows Vista, Windows XP, Windows Server 2003, Windows Server 2008 에서 사용할 수 있다. 본 자료는 WPF 의 다양한 부분을 소개하고 있으며, WPF 가 어떤 문제점에 대해 어떤 해결책을 제시하고 있는 지를 설명하고 있다.

현재의 문제점

어떤 병원이 환자를 검사하고 모니터링하는 새 애플리케이션을 보유하고 싶어한다고 가정해보자. 이 새 애플리케이션의 사용자 인터페이스 요구사항은 **다음과 같을 것이다:**

환자에 대한 이미지와 텍스트를 디스플레이 할 것.

환자의 심장 박동률과 혈압과 같은 중요한 사인을 보여주는 2 차원 그래픽을 보여주고 업데이트.

환자 정보에 대해 3 차원 뷰와 오버레이(overlay)를 제공.

초음파와 다른 진단을 비디오로 보여주어서 외과의사와 간호원이 의견을 제시.

병원 스태프가 환자의 상태를 설명하는 문서의 의견을 읽고 추가.

이런 요구사항이 애매모호할 수는 있어도 불합리한 것은 아니다. 적시에 적절한 방식으로 정확한 정보를 제시하는 사용자 인터페이스는 굉장한 비즈니스 가치를 가질 수 있다. 앞에서 설명한 병원의 상황에서는 생명을 살릴 수도 있다. 온라인 상점 또는 다른 소비자 관련 애플리케이션과 같이 그다지 심각하지 않은 시나리오에서도, 강력한 사용자 경험을 제공한다면 그 기업을 경쟁자와 차별화시켜 주어서 매출과 브랜드 가치를 높이게 된다. 결론은, 많은 현대적 애플리케이션이 그래픽, 미디어, 문서와 다른 현대적 사용자 인터페이스 요소들을 통합하는 인터페이스를 제공해서 큰 혜택을 얻을 수 있다는 것이다.

WPF 이전에도 윈도우에서 이런 종류의 인터페이스를 구축하는 것이 가능했지만, 다음과 같은 두 가지 장벽 때문에 상당히 어려운 작업이었다: 그래픽, 이미지, 비디오 처리 작업에 매우 다양한 기술을 사용했었다. 이렇게 다양한 기술에 모두 능숙한 개발자는 찾기도 힘들고 비용도 너무 많이 들었으며 애플리케이션 유지보수 비용 역시 마찬가지였다.

사용자에게 이 모든 기능을 효과적으로 제공하는 인터페이스를 디자인하기는 쉽지 않다. 소프트웨어 개발자들은 일반적으로 인터페이스 디자인기술을 가지고 있지 않기 때문에 전문 디자이너가 필요하지만, 디자이너와 개발자들은 앞에서 설명한 완전한 기능을 제공하는 인터페이스를 함께 개발하는데 많은 어려움을 겪고 있다.

강력하고 현대적인 사용자 인터페이스가 반드시 이렇게 복잡해야 할 필요는 없다. 공용 프레임워크를 통해 개발자들에게 단일화된 개발 방식을 제공하는 동시에 디자이너가 중요한 역할을 하도록 지원해서 이 모든 문제를 해결할 수 있다. 앞으로 자세하게 설명하겠지만, 바로 이것이 WPF 의 가장 큰 목적이다.

문제 해결: Windows Presentation Foundation

WPF 는 다음과 같이 가장 중요한 3 가지 장점을 제공하고 있다:

-윈도우 기반의 사용자 인터페이스를 위한 통합 플랫폼

-개발자와 디자이너가 서로 협력할 수 있는 능력

-기존의 사용자 인터페이스 기술과의 연동

다음은 각 장점에 대한 설명이다.

윈도우 기반의 사용자 인터페이스를 위한 통합 플랫폼

WPF 이전에는 앞에서 설명한 것과 같은 윈도우 사용자 인터페이스를 만들려면, 다음의 표와 같이 다양한 기술을 사용해야만 했다.

	Windows Forms	PDF	Windows Forms/ GDI+	Windows Media Player	Direct3D	WPF
Graphical Interface, e.g., Forms and Controls	X					X
On-Screen Documents	X					X
Fixed-Format Documents		X				X
Images			X			X
Video and Audio				X		X
Two-Dimensional Graphics			X			X
Three-Dimensional Graphics					X	X

개발자는 윈도우 그래픽 사용자 인터페이스의 양식, 컨트롤과 다른 일반적인 부분 개발에 Windows Forms 를 가장 많이 사용할 것이다. 인터페이스가 문서를 보여주어야 한다면, Windows Forms 는 온스크린(on-screen) 문서를 사용하고, 고정 양식 문서는 Adobe 의 PDF 를 사용할 것이다.

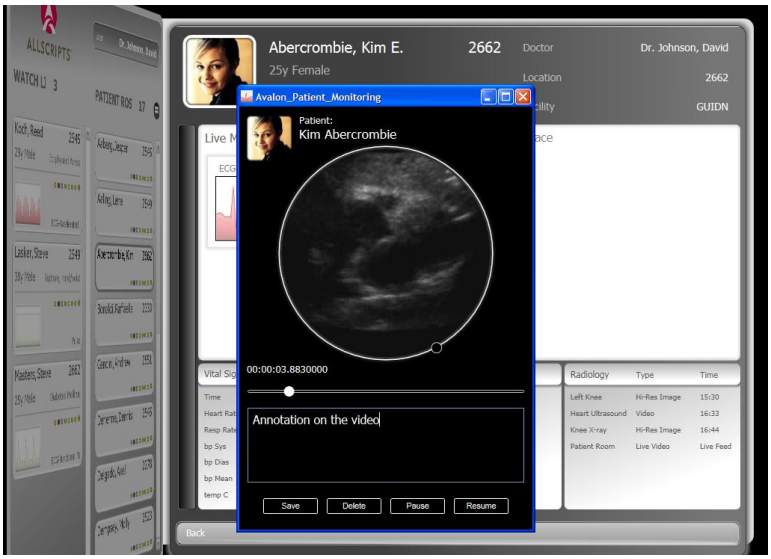
이미지와 2 차원 그래픽에 대해서는, Windows Forms 를 통해 접속할 수 있는 독특한 프로그래밍 모델인 GDI+를 사용할 것이다. 개발자는 Windows Media Player 를 사용해서 비디오와 오디오를 보여주고 3 차원 그래픽에 대해서는 윈도우 표준인 Direct3D 를 사용할 것이다.

기술진화 역사 때문에 이렇게 복잡하게 된 것일 뿐, 어느 누구도 이것이 제대로 된 것이라도 생각하지 않는다. 단일 통합 솔루션이 필요하며 WPF 가 바로 그 솔루션이다. WPF 를 사용해 애플리케이션을 개발하는 개발자들은 앞에서 설명한 모든 문제들을 해결할 수 있게 되었다. 무엇보다, 사용자 인터페이스를 만들 수 있는 하나의 통합 인프라가 있는데 다양한 독립 기술들을 사용해야 할 필요는 없다.

WPF 가 이 모든 것을 대체할 수 있는 것은 아니다. Windows Forms 애플리케이션은 앞으로도 계속 자신의 역할을 할 것이며, WPF 가 중심 기술이 된 후에도 일부 새 애플리케이션은 계속 Windows Forms 를 사용할 것이다. 또한 WPF 를 Windows Forms 와 함께 사용할 수도 있다. Windows Media Player 는 계속 독립적인 역할을 수행하게 될 것이며, PDF 문서도 계속 사용될 것이다. Direct3D 도 게임을 비롯한 다른 종류의 애플리케이션에 대해 중요한 기술로 사용될 것이다. 실제로, WPF 자신도 모든 렌더링에 대해 Direct3D 를 사용한다.

WPF 는 하나의 기술에서 광범위한 기능을 제공하기 때문에 현대적 사용자 인터페이스를 훨씬 쉽게 만들 수 있다. 이 통합 방식이 어떤 장점을 제공하는지는 다음의 WPF 기반의 의료 애플리케이션의 화면을 확인하면 쉽게 이해할 수 있다:

이 화면은 2 차원과 3 차원 그래픽과 함께 텍스트와 이미지를 보여주고 있다. 이 모든 인터페이스는 WPF 를 사용해서 만들어진 것이며, 개발자들은 GDI+ 또는 Direct3D 와 같은 특수 그래픽 기술을 사용하는 코드를 작성하지 않아도 된다. WPF 는 이외에도 아래와 같이 초음파 신호와 같은 비디오를 덧붙이거나 보여줄 수도 있다.



WPF 는 문서를 읽기 쉬운 방식으로 보여준다. 예를 들면, 외과의사는 의료 애플리케이션에서, 환자 치료에 대한 노트를 검색하고 관련된 주제에 대한 의료자료 검색을 접속할 수 있다. 비디오에서도, 외과의사는 다음 그림과 같이 주석을 추가할 수 있다.

문서는 읽기 쉬운 열 (columns)로 나타나고, 사용자는 문서를 스크롤하지 않고 한 번에 한 페이지만 나오게 할 수도 있다. 온스크린 가독성을 높이는 것이 WPF 의 중요한 목표다. 온스크린 문서가 매우 유용한 형식이지만 고정양식 문서가 더 적합한 경우도 있다. 고정양식 문서는 화면과 프린터 모두에서 같은 모습이기 때문에 어떤 상황에서도 일관된 외양을 제공한다. 마이크로소프트는 이런 형식의 문서를 정의하기 위해, XML Paper Specification (XPS)를 만들었다. WPF 는 개발자들이 XPS 문서를 만들고 사용할 수 있는 API 들을 제공하고 있다.

현대적 사용자 인터페이스를 만드는 것은 이전에 매우 다양했던 기술을 통합하는 정도가 아니라, 최신 그래픽 카드를 활용해야 하기 때문에, WPF 는 가능한 한 많은 작업부하를 그래픽 카드로 전환해서 GPU (graphics processing unit)의 모든 용량을 활용한다. 더 나아가, 현대적 인터페이스는 비트맵 그래픽에 따라 제한되어서는 안되기 때문에 WPF 는 벡터 그래픽만을 사용해서 이미지와 해상도가 화면에 따라 자동으로 조정된다. 작은 모니터와 큰 텔레비전을 위해 서로 다른 그래픽을 만들지 않아도, WPF 는 개발자를 위해 조정 작업을 처리해준다.

WPF 는 사용자 인터페이스를 만드는데 필요한 모든 기술을 한 인프라로 통합하여, 인터페이스를 만드는 사람들의 작업을 훨씬 간편하게 만들어준다. WPF 에서는 개발자나 디자이너가 한 환경만 배우면 되기 때문에 애플리케이션을 만들고 유지보수하는 비용이 더 적다. 그리고 그래픽, 비디오 등을 통합하는 인터페이스를 직관적으로 만들 수 있기 때문에, 사용자가 윈도우 애플리케이션을 보다 잘 사용해서 품질과 비즈니스 가치를 높일 수 있다.

개발자와 디자이너가 협업할 수 있는 능력

완전한 기능의 사용자 인터페이스를 만들 수 있는 통합 기술 인프라를 제공한다면 큰 도움이 된다. 그렇지만 일반 개발자가 이 인프라를 제대로 이해하고, 사용하기 쉬운 사용자 인터페이스를 만들어 줄 것으로 기대하는 것은 무리다. 특히 앞의 병원 예에서와 같은 종합적인 인터페이스는 소프트웨어 전문가들에게도 무리다. 많은 애플리케이션이 전문 인터페이스 전문가없이 개발되고 있지만 완벽한 사용자 인터페이스를 만들기 위해서는 그들의 도움이 반드시 필요하다.

그렇지만 디자이너와 개발자들이 어떻게 협업할 수 있을까? 서로 다른 두 성격의 협업은 많은 문제를 가지고 있다. 가장 일반적인 문제는, 디자이너는 그래픽 툴을 사용해서 애플리케이션이 보여주어야 하는 화면 레이아웃의 정적인 이미지를 만든다. 그 후에 이 이미지를 개발자에게 전달하고 개발자는 그 이미지가 동작하는 코드를 만들게 된다. 디자이너에게는 그리기 쉬운 인터페이스이지만 개발자에게는 구현하는 것이 거의 불가능에 가까울 수도 있다. 기술의 한계, 일정압박, 기술부족, 이해부족 또는 다른 의견 때문에 개발자는 디자이너의 비전을 그대로 구현하지 못할 수 있다. 이 두 독립된 인력이 인터페이스의 품질을 낮추지 않고도 협업할 수 있는 좋은 방법이 필요하다.

WPF 는 eXtensible Application Markup Language (XAML)를 통해 협업을 지원하고 있다. XAML 는 Button, TextBox, Label 과 같은 XML 요소들을 정의해서 사용자 인터페이스가 보이는 그대로 구현할 수 있게 해준다. XAML 요소들은 특성을 가지고 있어서 다양한 옵션을 설정할 수 있다. 예를 들면 다음의 간단한 XAML 코드는 “No”라는 단어를 가지고 잇는 붉은 색 버튼을 만든다:

```
<Button Background="Red“>
 No
</Button>
```

각각의 XAML 요소는 WPF 클래스 하나에 해당하며 그 요소의 특성들은 각각의 클래스의 대응 이벤트 또는 속성(property)을 가지고 있다. 예를 들면, 다음의 C# 코드로도 같은 붉은 색 버튼을 만들 수 있다:

```
Button btn = new Button();
btn.Background = Brushes.Red;
btn.Content = “No";
```

XAML 로 표현할 수 있는 모든 것을 코드로도 표현할 수 있다면 XAML 의 장점은 무엇일까? XML 기반의 기술 (description)을 만들고 생성하는 툴이 코드보다 그 작업을 훨씬 더 쉽게 할 수 있다는 것이다. XAML 은 사용자 인터페이스를 툴 친화적으로 기술 (description)할 수 있기 때문에, 개발자와 디자이너가 협력할 수 있는 더 좋은 방법이다. 다음의 그림은 협업 프로세스를 보여주고 있다.

디자이너는 Microsoft Expression Blend 와 같은 툴을 사용해서 사용자 인터페이스 모습과 동작을 지정할 수 있다. 이것은 WPF 인터페이스의 룩앤필을 정의하는 전용 툴로, 인터페이스에 대해 XAML 로 표현된 기술 (description)을 만든다. 앞의 예의 단순한 버튼도 실제 기술 (description)은 훨씬 더 복잡하다. 개발자는 Microsoft Visual Studio 와 같은 툴을 사용해서 이 XAML 기술 (description)을 연다. 디자이너가 만든 정적인 이미지에 따라 처음부터 인터페이스를 다시 만드는 대신에, 인터페이스 정의 자체가 전체적으로 활용된다. 개발자는 애플리케이션에서 필요한 다른 기능과 함께, 이벤트 처리자(handlers)와 같은 인터페이스 코드를 작성한다. 또한 애플리케이션 인터페이스에 전역으로(globally) 적용될 수 있는 형식을 만들고 다양한 상황에 대해 필요에 따라 사용자 지정할 수도 있다.

이렇게 개발자와 디자이너가 협업하면 디자이너가 만든 이미지에 따라 개발자가 인터페이스를 구현할 경우에 자주 발생되는 해석 문제를 줄이게 된다. 그리고 두 역할의 사람들이 동시에 작업을 해서 더 빨리 순환하고 더 좋은 피드백을 교환할 수 있게 한다. 그리고 두 환경은 동일한 빌드 시스템을 사용하기 때문에, WPF 애플리케이션이 두 개발 환경 사이트를 오갈 수 있다. 3 차원 인터페이스 요소를 만드는 Electric Rain 의 ZAM 3D 와 같은 전문 XAML 정의 인터페이스 툴도 사용할 수 있다.

더 좋은 사용자 인터페이스는 생산성을 높인다. WPF 가 활약하는 다양한 환경에서 진정으로 효과적인 인터페이스를 만들기 위해서는, 디자이너들이 일선에서 노력을 해주어야 한다. XAML 과 호환 툴 덕분에 디자이너는 제 역할을 할 수 있게 되었다.

기존의 사용자 인터페이스 기술과의 연동

.NET Framework 이 처음 발표된 후로, Windows Forms 를 사용한 애플리케이션이 많이 개발되어져 왔다. 또한 .NET 이전에는 Microsoft Foundation Classes (MFC)와 Win32 와 같은 기술을 통해 많은 애플리케이션이 개발되었었다. 여전히 3D 그래픽을 만드는데 DirectX 를 사용하는 애플리케이션들이

있다. WPF 가 이 모든 애플리케이션을 구현할 수 있더라도, 이미 구현된 애플리케이션과도 연동될 수 있어야 한다. 그렇기 때문에 WPF 의 중요한 목표 중 하나가 기존 사용자 인터페이스 기술과의 연동이다.

실제로 WPF 발표 이후에도, 일부 애플리케이션은 다른 인터페이스 기술을 계속 사용할 것이다. 예를 들면, Windows Forms 는 낮은 윈도우 버전과 같이 WPF 를 사용할 수 없는 시스템에 적합하다. 다른 이유 때문에 새 애플리케이션이 WPF 대신에 Windows Forms 를 선택할 수도 있다. LOB (line-of-business) 애플리케이션에서는 Windows Forms 가 효과적인 인터페이스 기술이 될 수 있지만, WPF 는 약간 다른 영역을 목표로 하고 있다. 의료 애플리케이션과 같이 방대한 사용자 인터페이스를 개발하는 개발자는 Windows Forms 보다 WPF 가 더 만족스러울 것이다. 그렇지만 일반적인 LOB 애플리케이션을 개발하는 사람들은 Windows Forms 를 선택할 수도 있다. Windows Forms 를 지원하는 기능들과 툴은 LOB 애플리케이션을 대상으로 하고 있는 반면에 WPF 는 약간 다른 애플리케이션에 초점이 맞춰져 있다.

또 다른 옵션은 한 애플리케이션에서 WPF 와 Windows Forms 를 혼합하여 사용하는 것이다. 각각의 기술이 서로 정의한 컨트롤이라는 사용자 인터페이스를 호스팅할 수 있기 때문에 혼합하여 사용할 수 있다. 예를 들면, Windows Forms 는 WPF 에는 없는 유용한 DataGridView 컨트롤을 제공하지만 WPF 는 3D 그래픽과 애니메이션과 같이 Windows Forms 에는 없는 많은 것들을 제공한다. WPF 와 Windows Forms 혼용에는 제약점도 있지만, 상당히 많은 윈도우 애플리케이션은 두 기술을 사용해서 사용자 인터페이스를 만들 수 있다.

WPF 는 MFC 또는 Win32 와도 사용될 수 있다. 다시 한 번 WPF 컨트롤은 기존의 Win32/MFC 코드 내에서 호스팅될 수 있으며, 기존의 Win32/MFC 컨트롤은 WPF 내에서 호스팅될 수 있다. Windows Forms 의 경우와 마찬가지로, 이 두 기술을 혼용하는 데에는 제약점이 따른다. 다시 한 번 더, 윈도우 애플리케이션은 WPF 와 Win32/MFC 를 함께 사용할 수 있다. WPF 를 사용하기 위해 애플리케이션의 기존의 모든 사용자 인터페이스 코드를 버릴 필요는 없다.

여기에서 Direct3D 라는 한 가지 인터페이스 기술을 더 설명할 필요가 있다. 마이크로소프트의 DirectX 기술 패밀리 중 하나인 이 API 는 그 동안 3 차원 그래픽을 만드는 윈도우 개발자의 주류 기술이었다. WPF 가 발표되었어도 Direct3D 사용을 중단할 필요가 없다. 앞에서 설명한 것과 같이, WPF 는 렌더링을 위해 Direct3D 를 사용한다. WPF 는 개발자들이 3D 그래픽을 만들 수 있게 해주기 때문에, 3D 작업을 하는 개발자는 둘 중 하나를 선택해야 하지만 그리 어려운 선택은 아닐 것이다. 게임, 3D 중심의 기술애플리케이션과 같은 3D 를 집중적으로 사용하는 애플리케이션의 경우에는 Direct3D 가 여전히 최고의 선택이다. 그렇지만 WPF 는 전문가가 아닌 더 많은 사람들이 3D 그래픽을 사용할 수 있게 해주며, 3D 그래픽을 2 차원 그래픽, 문서와 애플리케이션의 다른 부분과 자연스럽게 통합한다. WPF 와 Direct3D 는 각자만의 역할이 있으며 두 기술 모두 앞으로도 윈도우 플랫폼에서 많이 사용될 것이다.

Windows Presentation Foundation 사용하기

WPF 로 어떤 문제를 해결할 수 있는지를 아는 것이 좋지만, 어떻게 해결할 수 있는 지에 대해서도 어느 정도는 아는 것이 좋다. 여기에서는 WPF 기술을 살펴보고 윈도우 데스크톱 애플리케이션, 웹 브라우저, XPS 문서에서 서로 다르게 사용되는 방식을 설명하고 있다.

Windows Presentation Foundation 이라는 기술

WPF 가 사용자 인터페이스를 만들 수 있는 통합 인프라를 제공하지만, 문서, 이미지, 그래픽, 애니메이션 등 각 부분의 기술에 대해서는 구분해서 생각할 수 있다. 이 모든 기술은 WPF 의 기본 애플리케이션 모델을 따르고 있다.

애플리케이션 모델

.NET Framework 의 다른 부분과 같이, WPF 는 각 기능을 System.Windows 네임스페이스에 있는 네임스페이스 그룹으로 조직한다. WPF 가 어느 기능을 사용하던, XAML 페이지와 그 페이지에 연결된 코드로 된 모든 WPF 애플리케이션의 기본 구조는 상당히 비슷하다.

모든 애플리케이션은 WPF 의 표준 Application 클래스에서 상속을 받는다. 이 클래스는 모든 애플리케이션에서 많은 역할을 하는 공용 서비스를 제공한다. 서비스 중에는 전체 애플리케이션에 제공되어야 하는 상태를 보관하고, 애플리케이션을 시작하는 Run 또는 종료시키는 Shutdown 과 같은 표준 메서드를 제공하는 서비스 등이 포함되어 있다.

Application 요소를 통한 XAML 또는 Application 클래스를 사용한 코드 중 하나로 Application 객체를 만들 수 있다. WPF 에서는 모두가 가능하지만, 간단한 설명을 위하여 XAML 옵션을 보여주고 있다. 다음은 간단한 XAML 예다:

```
<Application xmlns=
   "http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   StartupUri="Page1.xaml"
   x:Class="Example.SimpleApp">
```

```
…
</Application>
```

이 정의는 먼저 WPF 네임스페이스를 이 요소에 대한 기본값으로 지정하고, XAML 네임스페이스에 대한 접두사(prefix)를 정의한다. XAML 은 WPF 이외에도 사용되기 때문에, 두 네임스페이스가 같은 것이 아니다. StartupUri 특성을 사용해서 애플리케이션이 시작될 때에 로딩되고 나타나야 하는 XAML 페이지 이름을 알려준다. 마지막 특성인 Class 는 이 Application 과 관련된 코드가 있는 클래스를 찾는데 사용된다. 앞에서 설명한 것과 같이, WPF 애플리케이션은 일반적으로 XAML 과 C# 이나 Visual Basic 코드 모두를 가지고 있기 때문에 코드 비하인드 파일(code-behind) 은 이 클래스의 코드를 보관하기 위한 것이다. 열기 Application 태그 다음에 XAML 의 나머지가 나타나서 이 애플리케이션을 정의하는데, 앞의 예제 코드에서는 생략된 채로 Application 요소의 닫기 태그가 바로 나왔다.

모든 WPF 애플리케이션이 같은 루트 클래스에서 파생되더라도, 개발자가 선택할 여지는 여전히 많다. 이 중에서도 애플리케이션이 전통적인 대화식 인터페이스를 제공할 것인지 아니면 네비게이션식 인터페이스를 제공할 것인지에 대한 결정이 가장 큰 결정이다. 대화식 인터페이스는 모든 윈도우 사용자가 친숙한 버튼과 다른 요소들을 제공한다. 네비게이션 인터페이스는 브라우저 방식으로 동작한다. 예를 들면 새 대화를 위해 새 창을 열지 않고 새 페이지를 로딩한다. 이 같은 인터페이스는 각각의 페이지가 프로그래밍 언어로 만들어진 로직과 함께 XAML 로 정의된 사용자 인터페이스로 구성되어 있는 페이지 그룹으로 구현된다. HTML 로 정의된 브라우저 페이지와 마찬가지고, XAML 은 페이지들을 서로 연결하는 Hyperlink 요소를 제공한다. 사용자는 웹 기반의 애플리케이션 페이지에서 하듯이 페이지를 오가며 탐색한다. 그렇다고 혼란스러워할 필요는 없다. 이것도 여전히 윈도우 애플리케이션으로 브라우저 내에서 실행되지 않아도 된다.

애플리케이션이 어떤 인터페이스 형식을 사용하던, 한 두 개의 윈도우를 보여주게 되어 있다. WPF 는 창 디스플레이에 대해 몇 가지 선택권을 주고 있다. 간단한 Window 클래스는 창 보여주기, 숨기기와 닫기 등의 기본적인 창 기능을 제공하며 네비게이션 인터페이스를 사용하지 않을 WPF 애플리케이션에서 사용된다. 네비게이션 인터페이스를 가지는 애플리케이션에서 사용하는 NavigationWindow 는 기본 Window 클래스를 확장해서 네비게이션을 지원한다. 새 페이지로 애플리케이션이 이동할 수 있게 하는 Navigate 메서드, 사용자의 네비게이션 경로를 추적하는 저널 (journal) 그리고 네비게이션과 관련된 다양한 이벤트 등이 지원된다.

레이아웃과 컨트롤

WPF 애플리케이션은 인터페이스의 다양한 부분을 조직하기 위해 레이아웃 판넬(panels)을 사용한다. 각각의 판넬은 버튼과 텍스트 박스 등의 자식 컨트롤과 다른 판넬을 가질 수 있다. 다양한 판넬들이 서로 다른 레이아웃 옵션을 제공한다. 예를 들면, DockPanel 은 자식 요소들이 판넬 가장자리에 위치할 수 있게 하는 반면에 Grid 는 자식들이 격자에 정확하게 위치하게 만든다. 개발자는 그리드의 열과 행 수를 정의한 후에 자식이 있어야 하는 위치를 정확하게 지정한다. Canvas 는 개발자가 자식을 판넬의 경계 내 아무 곳이나 자유롭게 위치시킬 수 있게 해준다.

다른 모든 인터페이스 기술과 마찬가지로, WPF 는 많은 컨트롤을 제공하며 개발자들은 사용자 지정 컨트롤로 자유롭게 만들 수 있다. 표준으로는 Button, Label, TextBox, ListBox, Menu, Slider 와 다른 전통적인 사용자 인터페이스 디자인 요소들이 제공된다. SpellCheck, PasswordBox, Tablet PC 의 잉크와 연동되는 컨트롤 등 더 복잡한 컨트롤도 제공된다.

그래픽 인터페이스에서와 같이, 사용자가 마우스를 누르거나 키를 누르는 이벤트는 WPF 애플리케이션의 컨트롤로 포착해서 처리한다. 컨트롤과 다른 사용자 인터페이스 요소들은 XAML 을 사용해서 전부 지정할 수 있지만 이벤트는 코드로 작성되어야 한다. 예를 들면, 다음의 XAML 정의는 Canvas 의 간단한 버튼을 정의한 것이다:

```
<Canvas xmlns=
  "http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   x:Class="Example.CodeForCanvas">
 <Button Click="Button_Click">
  Click Here
 </Button>
</Canvas>
```

열기 Canvas 태그는 일반적인 WPF 와 XAML 네임스페이스를 정의하는 것부터 시작된다. 그리고는 이 XAML 과 관련된 코드를 .NET namespace Example 의 CodeForCanvas 라는 클래스에서 발견할 수 있다는 것을 지정한다. 다음에는 온스크린 텍스트로 “Click Here”를 지정하는 Button 자체의 정의가 온다. 열기 Button 태그의 Click 특성은 이 버튼이 Button_Click 이라는 이벤트를 사용해서 클릭 이벤트를 처리한다는 것을 알려준다. 이 메서드의 C# 코드는 다음과 같다:

```
namespace Example {
 public partial class CodeForCanvas : Canvas  {
  void Button_Click(object sender, RoutedEventArgs e)   {
   Button btn = e.Source as Button;
```

```
            btn.Background = Brushes.Purple;
        }
    }
}
```

네임스페이스와 클래스 명은 앞의 Canvas 태그에서 지정된 것과 일치한다. CodeForCanvas 클래스는 WPF 가 제공하는 기본 Canvas 클래스의 상속을 받으며, 부분(partial) 클래스로 정의된다. 부분 클래스는 .NET Framework 2.0 에서 추가된 기능으로 별도로 정의된 코드를 하나의 클래스로 결합할 수 있게 해준다. 이 예에서는, XAML 로 정의된 Canvas 는 부분 클래스를 사용해서 결합되는 부분 클래스를 만든다. 그 결과로 버튼이 있는 캔버스를 보여주고 그 이벤트를 모두 처리할 수 있는 완벽한 클래스가 만들어진다.

그 이벤트를 처리할 Button_Click 메서드는 CodeForCanvas 클래스에서 제공된다. 이벤트의 인자들(arguments)이 WPF 가 정의한 RoutedEventArgs 클래스를 사용해서 전송되기는 하지만, 이 메서드는 이벤트에 대해 일반적인 .NET Framework 규칙을 따른다. 이 클래스의 Source 속성은 그 이벤트를 만들어낸 Button 의 참조를 가지고 있으며, 메서드는 이것을 사용해서 버튼의 색상을 보라색으로 변경한다.

이 간단한 예제가 보여주듯이, WPF 사용자 인터페이스의 요소들은 시각적인 나무(visual tree)로 조직된다. 여기의 나무는 Button 자식 하나만 있는 Canvas 하나로 구성되어 있지만 실제 WPF 애플리케이션에서는, 이 나무는 훨씬 더 복잡하다. 온스크린 인터페이스를 실제로 만들려면, 이 나무가 렌더링되어야 한다. WPF 는 가능한 경우에는 언제나 하드웨어 렌더링을 통하기 때문에, 애플리케이션 컴퓨터에 설치된 그래픽 카드가 그 작업을 처리하게 만든다. 컴퓨터의 그래픽 하드웨어를 사용할 수 없다면, WPF 는 자신의 소프트웨어를 사용해서 인터페이스를 렌더링할 것이다. 그 결정은 런타임 시에 WPF 가 판단하기 때문에, 개발자들은 별다른 일을 할 필요가 없다.

렌더링을 하드웨어로 하던 소프트웨어로 하던, WPF 는 항상 유지 모드 (retained mode) 그래픽으로 알려진 방식을 사용한다. 애플리케이션 개발자는 XAML 과 코드를 결합해서 시각적 나무가 어떤 모습인지를 정의한다. 그 후에 WPF 가 이 나무에 정보를 보관한다. 예를 들면 사용자가 이 정보를 원하면 애플리케이션이 창의 모든 부분 또는 일부를 다시 그리지 않고, WPF 가 자체적으로 이것을 처리한다. 나무를 구성하는 요소들은 화면의 픽셀이 라니라 객체로 저장되기 때문에 WPF 는 이런 유형의 렌더링을 처리할 충분한 정보를 가지게 된다. 창과 창이 가진 컨트롤의 크기가 조정될 경우에도, WPF 는 자체적으로 모든 것을 다시 렌더링할 수 있다. WPF 는 그래픽 양식(선, 원 등)을 이해하고 픽셀 맵(maps) 대신에 벡터 그래픽을 사용하기 때문에, WPF 는 새 크기로 인터페이스를 다시 만들 수 있다.

스타일과 템플릿

사용자 인터페이스 요소의 외양을 결정하는 방법을 일단 정의한 후에, 그 외양을 다시 반복해서 정의하는 것이 많은 도움이 된다. Cascading Style Sheets (CSS)를 통해 HTML 페이지에서 이렇게 할 수 있다. WPF 는 스타일 (styles)과 비슷한 것을 제공한다. CSS 스타일시트의 인기가 말해주듯이 스타일을 정의할 수 있는 능력은 상당한 도움이 된다. 예를 들어 스타일시트는 디자이너가 인터페이스에 대해 동질적인 외양을 만들 수 있는 반면에 개발자는 이런 상세내용을 무시해도 되기 때문에, 디자이너와 개발자의 작업을 잘 분리시켜준다.

WPF 애플리케이션 개발자는 XAML 의 Style 요소를 사용해서 어떤 것이 어떤 모습이 되어야 하는지에 대해 한 두 영역을 정의한 후에 그 스타일을 반복해서 적용시킬 수 있다. 예를 들면, ButtonStyle 이라는 스타일은 다음과 같이 정의될 수 있다:

```
<Style x:Key="ButtonStyle">
  <Setter Property="Control.Background" Value="Red"/>
  <Setter Property="Control.FontSize" Value="16"/>
</Style>
```

이 스타일을 사용해서 정의된 모든 Button 은 붉은 색 배경과 16 폰트 크기를 가지게 된다. 다음은 Button 이 이 스타일을 사용하려 한다는 것을 명시할 수 있는 방법을 보여준다:

```
<Button Style="{StaticResource ButtonStyle}">
 Click Here
</Button>
```

이 코드의 “StaticResource”가 알려주듯이, WPF 스타일은 일반적으로 리소스로 정의되는데, 이 리소스는 애플리케이션 코드와 별도로 정의된 데이터다.

스타일은 앞의 코드에서 보여주는 것 이상의 기능을 한다. 예를 들면, 또 다른 스타일에서 파생된 스타일은 그 스타일의 설정을 상속받아 오버라이딩 할 수 있다. 또한 스타일은 반복 동작의 공용 부분을 지정하는 트리거 (triggers)를 정의할 수도 있다. 예를 들면, 스타일이 Button 위로 마우스를 움직이면 버튼의 배경을 노란색으로 바꾸게 지정할 수도 있다.

또한 WPF 는 템플릿을 사용할 수도 있다. 템플릿은 스타일과 비슷하며 두 가지 종류를 사용할 수 있다:

데이터 템플릿: XAML 의 DataTemplate 요소를 사용해서 데이터가 보여지는 방법에 대해 특징 (characteristics) 그룹을 지정할 수 있다. 색상, 정렬 또는 다른 것이 데이터 템플릿에 한 번 정의되면 애플리케이션 사용자 인터페이스의 다른 곳에서도 사용될 수 있다.

컨트롤 템플릿: XAML 의 ControlTemplate 요소를 사용해서 컨트롤의 외관을 정의할 수 있다.

애플리케이션 개발자가 인터페이스의 외관을 정의할 수 있도록 직관적인 방법을 제공하는 것이 합리적이다. WPF 에서는, 스타일과 템플릿이 외관을 정의하는 주요 방법이다.

텍스트

대부분의 사용자 인터페이스는 최소한의 텍스트만 보여주며 나머지는 거의 보여주지 않는 인터페이스도 있다. 그렇지만 대부분의 사람들에게는 화면에서 텍스트를 읽는 것이 인쇄된 것을 읽는 것보다 좋을 리가 없다. 이미 우리는 책과 잡지의 고품질의 문자 출력과 배열에 익숙해져 있다. 온스크린 텍스트를 읽으면 인쇄된 것에 비해 가독성이 떨어진다.

WPF 는 온스크린 텍스트를 인쇄된 텍스트만큼 읽기 편하게 만들어 그 차이를 없애고 있다. 이 목적을 위해, WPF 는 산업표준의 OpenType 폰트를 지원해서 기존 폰트 라이브러리를 재사용할 수 있다. 또한 WPF 는 최근에 정의된 ClearType 기술도 지원한다. ClearType 은 최신 디스플레이 화면에서 각각의 픽셀을 구성하는 하위 요소들을 개별적으로 빛나게 하는 기법인 서브 픽셀 포지셔닝(sub-pixel positioning)을 통해, 텍스트를 사람의 눈에 더 부드럽게 보이도록 만든다. 또한 WPF 는 Glyphs 클래스를 통해 텍스트를 렌더링하는 저차원 지원도 한다. 뒤에서 설명하겠지만, 이 클래스는 XPS 문서에서 문자를 대표하는데 사용된다.

WPF 는 가독성을 더 높이기 위해, 문자 그룹을 하나의 연결 이미지로 교체하는 연자 (ligatures)와 같은 추가 기능을 제공한다. 예를 들면 “ffi”는 인쇄 페이지에서는 3 글자를 가지고 있는 하나의 연결 연자 (ligature)로 대체된다. 온스크린 텍스트에 이 기능을 추가해서 사람들이 연자라는 배경을 알지 못해도 더 편안하게 느끼게 한다.

문서

텍스트는 사용자 인터페이스 곳곳에 나타나기 때문에 가독성을 높이는 것이 좋다. 특히 문서와 같이 긴 내용을 읽는 경우에는 텍스트에 많은 신경을 쓰기 때문에, 온스크린 가독성을 개선하려면 문서가 보여지는 방법도 개선시켜야 한다. WPF 는 고정 문서와 플로우 문서라는 두 종류의 문서를 지원하여 문서의 표현 방법을 개선하고 있다:

고정 문서(Fixed documents)는 문서가 화면에서 렌더링되던 프린터에서 렌더링되던 완전히 똑 같은 모습이다. 법률 문서와 같이 문서가 항상 같은 모습이어야 하는 분야가 많기 때문에 고정 양식 문서가 중요하게 사용되는 분야가 많다. WPF 가 지원하는 고정 양식 문서는 XPS 를 통해 정의된다. 고정 문서의 내용은 XAML 의 FixedDocument 요소로 명시된다. 이 간단한 요소는 PageContent 요소 목록을 가지고 있으며, 이 요소들은 각각 고정 문서의 페이지 이름을 가지고 있는다. WPF 는 고정 문서를 나타내기 위해, DocumentViewer 컨트롤을 제공한다. 이 컨트롤은 XPS 문서를 읽기 전용으로 보여주어서, 읽는 사람이 문서의 앞 뒤로 이동하고, 특정 문자를 검색하는 작업을 할 수 있게 한다.

고정 문서는 화면과 인쇄에서 모두 사용되는 반면에, 플로우 문서는 화면 디스플레이 전용이다. 그 내용의 가독성을 가능한 한 높이기 위해, 플로우 문서는 창 크기와 다른 조건에 맞춰 텍스트와 그래픽이 보여지는 방법을 조정할 수 있다. 플로우 문서는 FlowDocument 라는 XAML 요소를 사용해서 정의된다. 다음은 간단한 예다:

```
<FlowDocument
  ColumnWidth="300"
  IsColumnWidthFlexible="True"
  IsHyphenationEnabled="True">
    <Paragraph FontSize="12">
     <Bold>Describing WPF</Bold>
    </Paragraph>
    <Paragraph FontSize="10">
     WPF is the user interface technology for the .NET
     Framework 3.5. It provides a unified foundation for modern
     user interfaces, including support for documents, two- and
     three-dimensional graphics, media, and more.
    </Paragraph>
</FlowDocument>
```

이 문서는 넓이가 300 미만인 열에 보여지도록 정의된다. 넓이는 장비에 상관없는 픽셀로 각각의 픽셀은 1 인치의 1/96 로 정의된다. 그렇지만 바로 다음 코드 라인에, 문서 작성자는 IsColumnWidthFlexible 속성을 참으로 설정해서 이 넓이가 변경될 수 있다는 것을 알려주고 있다. 이렇게 설정하면 WPF 는 이 문서를 보여주는데 사용될 열의 수와 넓이를 변경할 수 있게 된다. 사용자가 창의 넓이를 변경하면, WPF 는 열의 수와 넓이를 늘리거나 줄여서 문서의 텍스트를 보여준다.

다음으로, 문서는 IsHyphenationEnabled 속성을 참으로 설정해서 하이픈을 요청한다. 그 다음에는 이 문서가 가지고 있는 두 개의 문장이 나온다. 각 문장 안의 텍스트는 Paragraph 요소에 있는데, 각각 다른 폰트 크기로 설정된다. 첫 번째 문장의 텍스트는 볼드 체로 보여져야 한다는 것을 지정하고 있다.

WPF 는 몇 가지 FlowDocument 옵션을 통해 가독성을 높이고 있다. 예를 들면, IsOptimalParagraphEnabled 속성이 참으로 설정되면, WPF 는 한 문장 전체에 가능한 한 균등하게 빈 공간을 분산시킨다. 이렇게 되면 인쇄 출력물에서 종종 있는 너무 긴 빈 공간 때문에 가독성이 낮아지는 것을 막을 수 있다. 플로우 문서는 또한 일반 텍스트에 노트를 추가하거나 Tablet PC 에서 잉크를 추가할 수도 있다. 모든 주석은 문서의 어떤 내용에 주석이 연결되어 있는지를 알려주는 앵커(anchor)와 주석 자체의 내용을 가지고 있는 카고(cargo)로 구성되어 있다.

WPF 는 플로우 문서만을 보여주기 위해 다음과 같은 몇 가지 다른 컨트롤들을 제공한다:

FlowDocumentPageViewer: 한 번에 한 페이지를 볼 수 있다. 이 컨트롤은 사용자가 읽고 있는 문서의 크기를 조정할 수 있는 확대 컨트롤과 함께 앞 뒤 버튼을 제공한다.

FlowDocumentScrollViewer: 페이지 오른쪽에 스크롤바가 있는 전형적인 FlowDocument 스크롤 뷰를 제공한다.

FlowDocumentReader: FlowDocumentPageViewer 와 FlowDocumentScrollViewer 모두의 기능을 결합한다. 이 컨트롤은 사용자가 플로우 문서의 페이지 (한 번에 두 페이지 보기를 포함한) 뷰와 스크롤 뷰 사이를 전환할 수 있게 해준다.

점점 더 많은 정보를 디지털로 전달하면서, 화면에서 자료를 읽는 경험이 점점 더 중요해지고 있다. WPF 는 플로우 문서를 통해 정보를 사용자의 화면에 맞게 보여주어서, 윈도우 사용자의 경험을 향상시키고 있다.

이미지

이미지가 회사 로고, 여행 사진 또는 다른 어떤 것이던, 이미지는 사용자 인터페이스에 있어서 기본 요구사항이 되었다. WPF 에서는, Image 컨트롤을 사용해서 이미지를 나타낸다. 예를 들어, JPEG 파일을 보려면 다음의 XAML 코드를 사용한다:

```
<Image
 Width="200"
 Source="C:\Documents and Settings\All Users\Documents\
  My Pictures\Ava.jpg" />
```

이미지의 넓이는 200 으로 설정되었는데 이 단위는 장비에 독립적인 픽셀 단위다. 이미지를 가지고 있는 파일은 Source 특성을 사용해서 식별한다.

이미지 파일은 사용자가 적용한 핵심어와 등급과 같은 이미지에 대한 정보 (메타데이터)를 가지고 있을 수 있으며 WPF 애플리케이션은 이 정보를 읽고 기록할 수 있다. 또한 이미지는 회전하는 3 차원 객체 위에 칠하는 식으로 아주 재미있는 방식으로 표현될 수도 있다. 여기에서의 간단한 예는 가장 일반적인 예에 불과하며, WPF 는 훨씬 더 다양한 방식으로 이미지를 사용할 수 있다.

WPF 의 이미지 컨트롤은 JPEG, BMP, TIFF, GIF, PNG 를 포함한 다양한 형식으로 저장된 이미지를 보여줄 수 있다. 또한 Windows Vista 의 새로운 Windows Media Photo (HD Photo) 형식으로 저장된 이미지도 사용할 수 있다. 이미지가 어떤 형식이던, WPF 는 Windows Imaging Component (WIC)를 사용해서 이미지를 만들어낸다. WIC 는 앞에서 언급한 이미지 형식에 사용되는 코더/디코더(코덱)뿐 만 아니라 서드파티 코덱을 위한 프레임워크도 제공한다.

비디오와 오디오

네트워크와 프로세서 속도가 향상되면서, 비디오가 사람들이 사용하는 소프트웨어의 많은 부분을 차지하게 되었다. 사람들은 자신의 컴퓨터에서 음악과 오디오를 들으면서 많은 시간을 보내고 있다. WPF 는 MediaElement 컨트롤을 통해 비디오와 오디오에 대해서도 지원하고 있다.

다음은 이 컨트롤을 어떻게 사용할 수 있는 지에 대한 간단한 XAML 예다:

```
<MediaElement
 Source="C:\Documents and Settings\All Users\Documents\
  My Videos\Ruby.wmv" />
```

This control can play WMV, MPEG, and AVI video, along with various audio formats.

2 차원 그래픽

최근 20 년 동안, 윈도우에서 2 차원 그래픽을 만든 사람들은 Graphics Device Interface (GDI)와 GDI+를 사용해왔다. Windows Forms 애플리케이션 조차도 2D 그래픽을 사용자 인터페이스 기술에 통합될 수 없기 때문에 완전히 다른 네임스페이스를 통해 이 기능에 접속해야만 한다. 3 차원 그래픽의 경우에는 완전히 독립된 기술인 Direct3D 가 필요했기 때문에 더욱 어려웠었다. WPF 는 애플리케이션의 대부분 영역에서 이런 복잡성을 해소하고 있다. WPF 라이브러리를 사용해서 2D 와 3D 그래픽을 XAML 또는 코드로 직접 만들 수 있다. WPF 의 다른 기능과 같이, 사용되는 요소들은 애플리케이션의 시각적 나무의 또 다른 부분에 해당된다.

WPF 는 2D 그래픽에 대해 애플리케이션이 이미지를 만드는데 사용할 수 있는 다음 과 같은 셰이프 (shapes) 그룹을 정의한다:

Line: 두 지점 사이에 직선을 그린다.

Elllipse: 타원을 그린다.

Rectangle: 직사각형을 그린다.

Polygon: 연결된 직선 그룹으로 정의된 닫힌 셰이프를 그린다.

Polyline: 연결된 직선 그룹으로 정의된 열린 셰이프를 그린다.

Path: 임의의 경로로 설명된 셰이프를 그린다. 이 셰이프는 열리거나 닫힐 수 있으며 경로의 선은 직선이거나 곡선이 될 수 있다. Path 를 사용해서 선,

타원, 직사각형, 폴리곤, 폴리라인 등을 모두 그릴 수 있기 때문에, 실제로 다른 모든 셰이프들은 편의를 위해서만 존재한다.

이 클래스들을 사용해서 간단한 그래픽을 쉽게 만들 수 있다. 예를 들면, 다음의 XAML 은 붉은 타원을 그린다:

```
<Ellipse Width=”30” Height=”10” Fill="Red" />
```

셰이프 채우기는 브러시(brush)를 사용한다. 앞의 예는 굵은 색 브러시(solid color brush) 기본값을 사용하지만, WPF 는 다른 옵션을 여러 개 제공한다. 예를 들면, 수평으로 붉은 색에서 노란색으로 변하는 색으로 채워진 직사각형을 다음과 같이 정의할 수 있다:

```
<Rectangle Width=”30” Height=”10”
 Fill="HorizontalGradient Red Yellow" />
```

수직 그레디언트 (vertical gradient), 방사형 그레디언트 (radial gradient) 등의 브러시, 이미지, 비트맵 등을 색칠하는 브러시 등도 사용할 수 있다. 여기에서는 설명하고 있지 않지만, 셰이프는 펜을 사용해서 외곽의 색상, 넓이와 스타일을 지정할 수도 있다.

모든 것이 공용 인프라를 사용하기 때문에 사용자 인터페이스의 서로 다른 부분을 쉽게 결합할 수 있다는 것이 WPF 의 핵심이다. 애플리케이션은 Rectangle 내에서 Image 를 보여주고, Button 내에 Ellipse 를 놓을 수 있다. 그렇기 때문에, 2D 그래픽과 3D 그래픽과 인터페이스의 다른 부분을 쉽게 결합할 수 있다.

WPF 는 셰이프와 함께, 2 차원 그래픽과의 작업을 위한 또 다른 클래스 그룹을 제공한다. Geometries 라고 부르는 이 클래스들은 여러 가지 면에서 셰이프와 비슷하다. Line, Rectangle, Ellipse, Path 와 같은 선택권을 제공하는 셰이프와 마찬가지로, geometries 는 LineGeometry, RectangleGeometry, EllipseGeometry, PathGeometry 등의 옵션을 제공한다. 두 클래스 간의 가장 중요한 차이는 셰이프는 일반적으로 시각적인 이미지를 그리는데 사용되는 반면에, geometries 는 리전(Region)을 정의하는데 더 많이 사용된다는 것이다. 예를 들어 사각 이미지를 원 안에 들어가도록 잘라야 한다면 EllipseGeometry 를 사용해서 원의 경계를 지정할 수 있다. 마찬가지로, 애플리케이션이 마우스 클릭을 알아채는 영역과 같이 hit-testing 리전을 정의하고 싶다면, 그 리전에 대해 geometry 를 지정하면 된다.

마지막으로 여기에서 설명된 모든 것은 비주얼 레이어 (visual layer)라고 하는 더 낮은 수준 위에서 구현된다는 것을 알아야 한다. 이 레이어를 직접 사용해서 그래픽, 이미지와 텍스트를 만들 수도 있다. 단순하면서도 고성능의 그래픽을 만드는 상황에서는 레이어를 직접 사용하는 것이 좋지만, 거의 대부분의 애플리케이션은 WPF 가 제공하는 셰이프와 고차원의 추상표현(abstraction)을 사용하게 될 것이다.

3 차원 그래픽

2 차원 그래픽은 윈도우 인터페이스의 일반적인 부분이기 때문에 WPF 는 이 분야에 대해 상당히 많은 기술을 제공하고 있다. 3 차원 그래픽은 이것에 비해 많이 사용되지는 않지만, 더 좋은 데이터 시각화, 3D 차트, 제품 렌더링 등을 통해 훨씬 높은 가치를 제공한다. 3D 작업은 전통적으로 게임 개발자와 다른 전문 그룹에 한정된 독특한 기술력을 필요로 해왔다. WPF 는 표준 환경의 3D 그래픽 부분을 지원해서 다른 개발자들도 3D 를 사용할 수 있게 하고 있다.

WPF 가 없다면, 윈도우에서의 3D 개발은 Direct3D API 를 사용하는 것이 보통이다. WPF 는 3D 그래픽에 대해 Direct3D 를 사용하지만 개발자들은 매우 간단한 환경만을 보게 된다. WPF 보다는 Direct3D 를 사용하는 것이 더 적합한 경우가 있지만, 마이크로소프트는 윈도우 인터페이스에 대한 3D 개발에서도 WPF 를 사용하도록 권장하고 있다.

애플리케이션은 Viewport3D 컨트롤을 사용해서 WPF 에서 3D 그래픽을 보여준다. 이 컨트롤은 윈도우를 3 차원의 세계로 바꾼다. Viewport3D 컨트롤은 WPF 인터페이스 중 어디에서나 3D 그래픽을 보여줄 수 있게 한다.

3D 장면을 만들기 위해, 개발자는 한 개 이상의 모델을 기술 (description)한 후에, 이 모델이 보여지고 조명이 비춰지는 지를 지정한다. 이 모든 것은 XAML, 코드 또는 두 가지를 혼합해서 지정할 수 있다. WPF 에서는 모델을 기술 (description)하기 위해, 모델의 셰이프를 정의할 수 있는 GeometryModel3D 클래스가 제공된다. 모델이 정의되면, 그 외양은 서로 다른 종류의 material 을 적용해서 바꿀 수 있다. 예를 들면, SpecularMaterial 클래스는 표면이 빛나게 하지만 DiffuseMaterial 클래스는 빛나게 하지 않는다.

WPF 가 사용하는 material 과 상관없이, 모델은 다양한 방식으로 조명을 받을 수 있다. DirectionalLight 는 특정 방향에서 빛을 주는 반면에 AmbientLight 는 그 장면의 모든 것에 균등한 조명을 준다. 마지막으로 모델을 보는 방법을 정의하기 위해, 개발자는 카메라를 지정한다. 예를 들면, PerspectiveCamera 는 모델이 보이는 거리와 관점을 지정할 수 있는 반면에, OrthographicCamera 는 거리만을 지정해서 카메라에서 멀리 있는 객체들도 더 작게 보이지 않는다.

XAML 또는 코드로 복잡한 3D 장면을 직접 만드는 것은 어려운 일이다. 3D 를 사용하는 대다수의 WPF 애플리케이션 개발자들은 그래픽 툴을 사용해서 필요한 정의를 할 것이 분명하다. 그 정의 작업을 어떻게 하던 상관없이 표준 사용자 인터페이스에서 3D 그래픽을 사용할 수 있는 능력은 사용자의 경험을 크게 향상시켜줄 것이 분명하다.

변형과 효과

WPF 는 셰이프와 다른 요소들을 정의할 수 있는 능력과 함께, 요소들을 회전시키고 크기를 변경하여 변형시킬 수 있는 능력을 제공한다. XAML 에서는, RotateTransform 과 ScaleTransform 을 사용해서 변형시킬 수 있다. 모든 사용자 인터페이스 요소를 이렇게 변형시킬 수 있다. 다음은 간단한 예다:

```
</Button>
 <Button Content=”Click Here”>
 <Button.RenderTransform>
  <RotateTransform Angle=“45” />
 </Button.RenderTransform>
</Button>
```

RotateTransform 요소는 버튼을 45 도 회전시킨다. 이처럼 버튼을 회전시키는 것이 중요한 것이 아니라 WPF 의 디자인의 범용성을 보여주기 위한 것이다. 사용자 인터페이스의 다양한 부분이 서로 다른 기본 기술을 사용하지 않기 때문에, 다양한 방식으로 결합될 수 있다.

또한 WPF 에는 몇 가지 사전 정의된 효과가 포함되어 있다. 이 효과들은 변형과 마찬가지로 Buttons, ComboBoxes 등의 다양한 사용자 인터페이스 부분에 적용될 수 있다. 인터페이스 요소가 흐릿하게 보이게 만드는 번짐 효과, 요소가 불타오르는 것 같은 외곽 발광 효과, 인터페이스 요소 뒤에 그림자를 추가하는 그림자 효과 등을 적용할 수 있다.

애니메이션

인터페이스의 요소가 움직이는 애니메이션 효과를 낼 수 있는 능력은 많은 도움이 된다. 예를 들면 버튼을 클릭하면 아래로 내려가게 만든다면 사용자에게 더 좋은 경험을 제공할 수 있다. 더 복잡한 애니메이션은 사용자의 관심을 유도하고 매력적으로 보이게 해서 사용자 경험을 향상시킨다. WPF 는 애니메이션을 지원하고 있다.

변형에서와 같이, 애니메이션은 버튼, 셰이프, 이미지 등을 포함한 인터페이스의 다양한 부분에 적용될 수 있다. 객체의 속성들의 값을 시간 경과에 따라 변경해서 애니메이션 효과를 얻을 수 있다. 예를 들면, Ellipse 는 2 초 후에 Height 속성 값을 점차 줄여서 천천히 납작하게 만들 수 있다.

관련 애니메이션 그룹을 정의하는 것도 많은 도움이 된다. WPF 는 Storyboard 클래스를 제공하는데, 이 클래스는 각각 한 개 이상의 애니메이션을 가지고 있는 타임라인(timelines)을 가질 수 있다. 다양한 종류의 타임라인을 사용해서 순차로 또는 병렬로 애니메이션을 실행할 수 있다. 다음은 Ellipse 를 납작하게 만드는 것을 보여주는 간단한 XAML 예다:

```
<Ellipse Width="100" Height="50" Fill="Blue"
 Name="EllipseForSquashing">
 …
 <Storyboard>
  <DoubleAnimation
   Storyboard.TargetName="EllipseForSquashing"
   Storyboard.TargetProperty="Height"
    From="50" To="25" Duration="0:0:2" />
  </Storyboard>
  …
</Ellipse>
```

이 예는 Ellipse 정의로 시작되지만, Name 속성도 사용해서 Ellipse 가 뒤에서 참조될 수 있는 식별자 (identifier)를 할당한다. 몇 가지 상세내용이 빠졌지만 XAML 로 애니메이션을 정의하려면 Storyboard 요소가 있어야 한다. Ellipse 의 Height 속성은 double 형이어서 Storyboard 는 DoubleAnimation 요소를 가진다. 이 요소는 애니메이션 효과를 내는 Ellipse 이름, 변경될 속성과 어떻게 변경될 것인지를 지정한다. 여기에서는 Height 의 값은 2 초 동안 50 에서 25 로 변경된다.

애니메이션은 이것보다 훨씬 더 복잡할 수 있다. 마우스 클릭과 같은 이벤트로 실행되고, 중단되고, 재실행되고, 지정한 횟수만큼 반복될 수 있다. 애니메이션의 목적은 더 좋은 피드백과 더 많은 기능을 제공하며 전반적으로 사용하기 쉬운 사용자 인터페이스를 만드는 것이다.

데이터 바인딩

대부분의 사용자 인터페이스는 특정 유형의 데이터를 보여준다. 데이터 바인딩을 사용해서 데이터를 더 쉽게 보여준다면 인터페이스 개발자의 부담이 크게 줄어든다. 데이터 바인딩은 WPF 컨트롤이 보여주는 것과 외부에 존재하는 데이터를 직접 연결할 수 있게 해준다. 예를 들면, WPF TextBox 컨트롤의 Text 속성 값은 애플리케이션의 비즈니스 로직의 일부인 Employee 객체의 Name 속성에 바인딩될 수 있다. 어느 한 속성을 변경하면 다른 속성에도 반영된다. 사용자가 TextBox 값을 업데이트하면, Employee 객체의 Name 속성도 변경된다. 그 반대의 경우도 마찬가지다.

두 객체의 속성 간에 이런 연결을 하려면 WPF 의 Binding 클래스를 사용해야 한다. 다음은 약간 간단해진 XAML 의 예다:

```
<TextBox …>
 <TextBox.Text>
  <Binding Path=”Name” />
 </TextBox.Text>
</TextBox>
```

이 예에서는, Binding 요소의 Path 특성을 사용해서 TextBox 의 Text 속성이 바인딩되어야 하는 속성을 찾는다. Path 특성은 이 속성이 C# 또는 Visual Basic 와 같은 개발언어로 정의된 Common Language Runtime (CLR) 객체의 일부인 경우 (런타임에서 지정될)에 사용된다. CLR 객체와 함께, WPF 의 데이터 바인딩은 Binding 의 XPath 속성을 사용해서 XML 데이터에 직접 연결된다. 이 옵션은 지정된 데이터를 참조하는 XML 문서에서 한 개 이상의 노드를 선택하는 XPath 질의를 만든다. 그리고 .NET Framework 3.5 의 WPF 에서도 Language Integrated Query (LINQ)를 사용해서 XML 로 정의된 데이터에 접속할 수 있다.

더 복잡한 데이터 바인딩 옵션도 사용할 수 있다. 예를 들면, 리스트 바인딩은 ListBox 컨트롤의 내용에 표준 IEnumerable 인터페이스를 구현하는 CLR 객체의 데이터를 이식한다. 필요한 경우에는 사용자에게 제시하기 전에 데이터를 필터링하거나 저장할 수도 있다. 어떤 데이터 바인딩 옵션을 사용하던, 사용자 인터페이스에서 데이터를 쉽게 보여줄 수 있도록 지원하고 있다.

사용자 인터페이스 자동화

물론, WPF 인터페이스를 가장 많이 사용하는 사용자는 사람이다. 다른 소프트웨어가 사용자 인터페이스를 조작해야 하는 경우가 있으며 WPF 의 사용자 인터페이스 자동화를 통해 이것을 지원한다.

예를 들면 어떤 개발자가 인터페이스에 대해 자동화된 테스트 스크립트를 만들고 싶어한다고 하자. UI 자동화가 제공하는 프로그래밍 접속 기능을 사용해서, 사람이 하듯이 인터페이스를 조작하는 스크립트를 만들 수 있다. 또한 UI 자동화는 인터페이스의 다양한 요소들을 큰 소리로 읽는 사용 보조 툴을 만드는데도 도움이 된다. UI 자동화는 이 요소들이 있는 나무를 프로그래밍으로 훑어갈 수 있도록 지원하기 때문에 이런 사용 보조 툴을 만들 수 있는 것이다.

이 목적을 위해, WPF 는 UI 자동화 나무를 만든다. 이 나무는 AutomationElement 객체들로 구성되며, 이 객체들은 각각 인터페이스의 무엇인가를 대표한다. 나무의 루트는 Desktop 이며 열려있는 각각의 애플리케이션이 이 루트의 자식이 된다. 나무는 애플리케이션들 각각으로 분기되는데, 모든 WPF 컨트롤은 하나의 (경우에 따라 하나 이상의) AutomationElement 객체로 표현된다. 프로그래밍으로 완벽하게 인터페이스에 접속할 수 있도록, 각각 구분된 AutomationElement 가 사용자가 조작할 모든 것들을 대신한다. 예를 들어, 여러 버튼을 가진 컨트롤이 자신과 각각이 AutomationElement 객체인 버튼들을 가질 수 있다. UI 자동화 나무를 이런 수준까지 상세하게 구현하게 되면, 클라이언트 애플리케이션 (테스트 스크립트, 사용 보조 툴, 기타)은 사람이 하듯이 각각의 인터페이스 컴포넌트를 조작할 수 있게 된다.

UI 자동화는 WPF 의 핵심은 아니다. 대부분의 일반 사용자들은 이것을 사용할 일이 없을 것이다. 그렇지만 소프트웨어 테스트와 장애인과 같은 사람들은 다른 무엇보다도 이 기능이 큰 도움이 된다.

애드인 인터페이스

많은 개발자들은 자신이 만드는 애플리케이션에 다른 사람들이 기능을 추가하게 만들고 싶을 것이다. 이것을 위해, .NET Framework 3.5 는 독립된 애드인 개발을 지원하고 있다. 개발자들은 새로운 System.AddIn 네임스페이스를 사용해, 메신저에 확장 기능을 제공하고, 뉴스 리더에 광고를 올리는 등의 작업을 위해 애드인을 사용할 수 있다. .NET Framework 3.5 에서 호스트 애플리케이션의 인터페이스와 연동되고 확장하는 애드인 사용자 인터페이스를 만들 수 있게 되었다.

Windows Presentation Foundation 의 적용

WPF 는 방대한 기술을 제공하고 있다. 이 모든 것을 사람의 조작을 지원하기 위한 것으로, 독립형 WPF 애플리케이션, XAML Browser Applications (XBAP 응용 프로그램들), XPS 문서의 3 가지 방식으로 사용된다. 여기에서는 이 각각의 사용 방식에 대해 설명하고 있다.

독립형 WPF 애플리케이션

WPF 를 사용하는 가장 일반적인 방식은 독립형 애플리케이션이다. 윈도우에서 실행되는 거의 모든 .NET Framework 애플리케이션은 WPF 인터페이스를 가질 수 있다. 이 애플리케이션은 윈도우에서만 실행되거나 인터넷 또는 다른 네트워크를 통해 다른 소프트웨어와 통신할 수 있다. 반드시 그래야 하는 것은 아니지만, WPF 애플리케이션은 일반적으로 Windows Communication Foundation (WCF)를 사용해 통신을 한다.

독립형 WPF 애플리케이션은 다른 윈도우 애플리케이션과 마찬가지로, 로칼 디스크 또는 네트워크 서버에서 설치된다. .NET Framework 의 ClickOnce 기능을 사용해서 설치될 수도 있다. ClickOnce 는 Internet Explorer 사용자가 WPF 애플리케이션을 포함한 윈도우 애플리케이션을 쉽게 다운로드해서 설치할 수 있게 해주고, 구성이 변경될 때에는 자동으로 업데이트한다.

XAML Browser Applications: XBAP 응용 프로그램들

독립형 WPF 애플리케이션이 대부분의 기능을 제공하지만, 항상 최상의 선택일 수는 없다. 윈도우 애플리케이션이 아닌 웹 브라우저에서 클라이언트를 실행하는 것이 더 합리적인 경우도 있다. 웹 브라우저 클라이언트들이 기업환경의 현대적 사용자 인터페이스를 제공할 수 있도록, WPF 는 XBAP 응용 프로그램들을 지원한다.

개발자들은 XBAP 응용 프로그램들을 통해 브라우저 애플리케이션에서 WPF 의 기능 대부분을 사용할 수 있다. 개발자들은 공용 프로그래밍 모델을 통해 독립형 애플리케이션과 브라우저 애플리케이션에 동일한 코드를 사용할 수 있게 되었다. XBAP 응용 프로그램들은 Internet Explorer 또는 Firefox 에서 실행될 수 있으며 ASP.NET, JavaServer Pages (JSP), 또는 다른 웹 애플리케이션을 사용해서 개발된 웹 애플리케이션의 클라이언트 역할도 할 수 있다. XBAP 는 이런 웹 애플리케이션과 HTTP 또는 SOAP 를 통해 통신을 한다. 어떤 서버 플랫폼을 사용하던, XBAP 는 항상 ClickOnce 를 통해 로딩된다. 로딩 과정 중에는 어떤 다이얼로그나 프롬프트도 나타나지 않으며, XBAP 는 그냥 웹 페이지처럼 로딩된다. 그렇기 때문에 XBAP 응용 프로그램들은 시작 메뉴나 추가/삭제 프로그램에는 나타나지 않는다.

꼭 그렇게 해야 하는 것은 아니지만, XBAP 응용 프로그램들은 일반적으로 사용자에게 네비게이션 인터페이스를 제공하기 때문에, 애플리케이션은 웹 클라이언트처럼 동작한다. Internet Explorer 7 에서, XBAP 응용 프로그램들은 브라우저 자체의 앞으로 뒤로 버튼을 사용하며, 사용자가 접속하는 XAML 페이지는 브라우저의 검색기록에 나타난다. Internet Explorer 6 에서, XBAP 는 자체 앞으로 뒤로 버튼을 제공하며 자체 검색기록을 유지한다.

XBAP 는 웹에서 로딩되고 브라우저 내에서 실행되기 때문에, .NET Framework 코드 접속 보안에 따라 부분 신뢰만 얻는다. 따라서 독립형 WPF 애플리케이션은 XBAP 에서 할 수 없는 많은 일을 한다. 예를 들면, 인터넷에서 전개된 XBAP 는 독립형 윈도우를 만들거나 애플리케이션이 정의한 다이얼로그를 보여주거나 제한된 격리저장 영역 이상의 파일 시스템에 접속하지 않는다. 또한 Windows Forms, MFC, Win32 직접 호출 코드를 사용하지도 않으며 비관리형 코드를 사용할 수도 없다. XBAP 응용 프로그램들을 기업용 애플리케이션을 브라우저에서 호스팅하는 WPF 클라이언트라는 좁은 의미의 솔루션으로 생각하는 것이 맞지만 그 경우에는 매우 큰 역할을 한다.

XPS 문서

WPF 환경에서는 XPS 문서인 고정 양식 문서는 사용자 인터페이스에서 중요한 역할을 한다. 앞에서 설명한 것과 같이, WPF 는 XPS 문서를 보여줄 수 있는 DocumentViewer 컨트롤을 제공한다. 이 컨트롤을 WPF 에서 제공하는 것이 맞지만 XPS 를 WPF 의 일부로 생각해야 할 필요가 있을까 하는 의문이 든다. XPS 스펙은 고정 양식 문서를 정의할 수 있는 매우 상세한 방법을 제공하며 문서 자체를 다양한 방식으로 사용할 수 있다. WPF 의 다른 모든 기능은 사용자 인터페이스 개발에만 초점을 맞추고 있다. 이런 큰 그림을 생각해보면, XPS 를 WPF 에 포함시키는 것이 맞다.

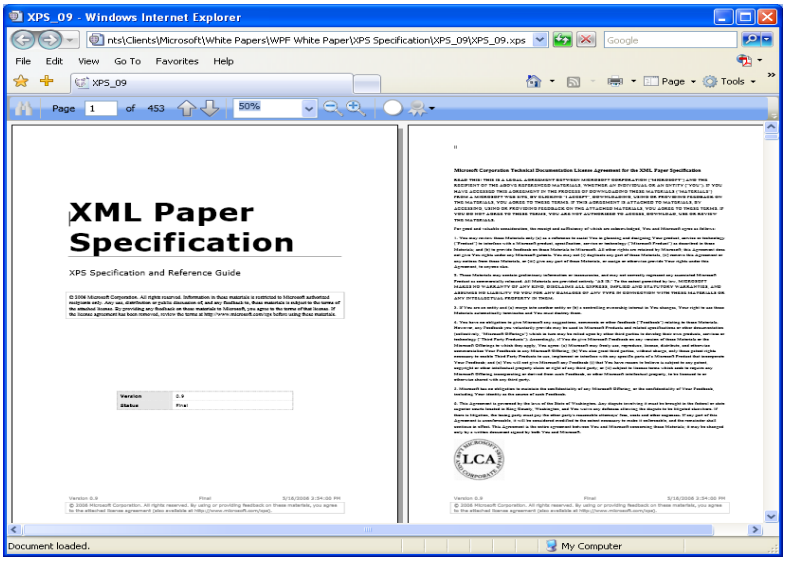
한 가지 큰 이유는, XPS 문서는 XAML 을 사용해서 정의된다는 것이다. 레이아웃에 대해 Canvas 요소가, 텍스트 표현에 Glyphs 요소가, 2 차원 그래픽 작성에 Path 요소 등의 경우와 같이 XAML 의 일부 하위 셋만이 사용되지만 모든 XPS 문서는 실제로는 XAML 문서다. 이것을 감안하면 XPS 를 WPF 의 일부로 보는 것이 맞다.

XPS 의 가장 중요한 애플리케이션 중 하나는 온스크린 사용자 인터페이스에 대한 것이 아니다. Windows Vista 발표와 함께 제공된, XPS 는 윈도우의 네이티브 인쇄 형식이 되고 있다. XPS 는 페이지 설명 언어 역할을 하기 때문에 XPS 문서는 XPS 를 인식하는 프린터에서 바로 렌더링될 수 있다. 그렇기 때문에 XAML 이라는 단일 설명 형식을 화면과 프린터에서 모두 사용할 수 있다. 또한 윈도우의 GDI 를 이용한 인쇄 메커니즘도 향상시켜 투명도와 흐림과 같은 복잡한 그래픽 효과를 더 잘 인쇄할 수 있게 해준다.

XAML 과 함께, XPS 문서는 다양한 이미지 형식 (JPEG, PNG, TIFF, HD Photo), 폰트 데이터, 문서 구조에 대한 정보 등의 이진형 데이터를 가질 수 있다. 필요하다면, XPS 문서는 W3C XML Signature 정의와 X.509 인증서를 사용해서 디지털 서명을 할 수도 있다. XPS 문서가 무엇을 가지고 있던, 모든 문서는 Open Packaging Conventions (OPC)로 정의한 형식으로 저장된다. OPC 는 XML 문서 (XPS 또는 XAML 문서가 아닌) 의 다양한 부분이 어떻게 연결되는지, 표준 ZIP 형식으로 어떻게 저장되는지 등을 지정한다. Microsoft Office 2007 도 XML 형식에 대해 OPC 를 사용하여 두 종류의 문서 간의 동질성을 유지한다.

WPF 애플리케이션 사용자들은 앞에서 설명한 것과 같이 WPF 의 DocumentViewer 컨트롤을 통해 XPS 문서를 본다. 마이크로소프트도 다음 그림과 같이 DocumentViewer 컨트롤을 사용한 XPS 뷰어 애플리케이션을 제공하고 있다. 이 애플리케이션은 사용자들이 페이지 단위로 문서를 검색하고, 텍스트를

검색하는 등의 일을 할 수 있게 해준다. XPS 문서는 윈도우 전용이 아니기 때문에 마이크로소프트는 애플 매킨토시와 같은 다른 플랫폼에도 XPS 뷰어를 제공할 계획이다.



개발자가 XPS 문서를 처리할 수 있도록, WPF 는 문서를 만들고, 로딩하고, 조작할 수 있는 API 들을 제공하고 있다. WPF 애플리케이션은 OPC 차원에서 문서를 처리할 수 있기 때문에 XPS 문서, Office 2007 문서 등을 특별한 방법을 사용하지 않고도 이용한다. Windows Workflow Foundation 을 사용해서 만든 애플리케이션도 이 API 들을 사용해서 XPS 문서를 사용하는 워크플로우를 만들 수 있다.

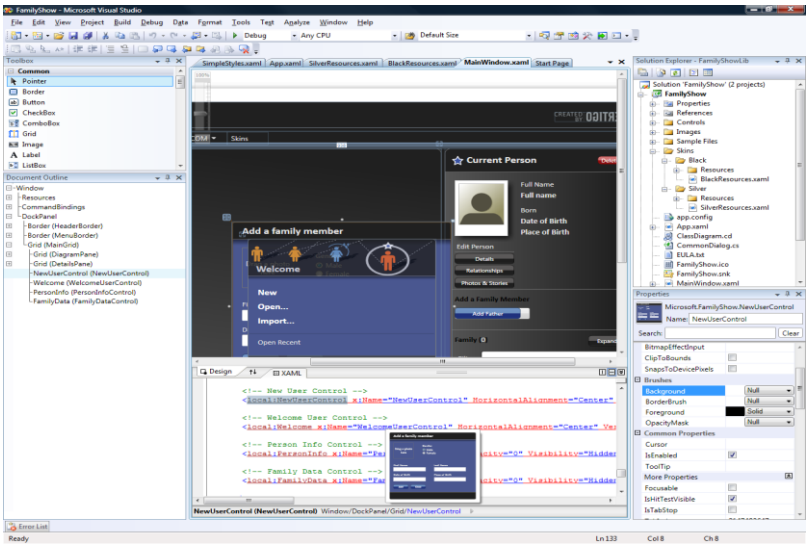
WPF 애플리케이션은 고정 양식 문서를 보여주고 처리할 수 있기 때문에 Windows Vista 사용자들은 화면에서 보는 것을 인쇄에서도 똑 같이 경험할 수 있다. 사람들이 사용자 인터페이스에 대해 요구하는 것 중 이것이 가장 중요한 것은 아니겠지만, XPS 의 범용적인 사용은 WPF 가 지원하는 광범위한 기술을 지원한다는 것을 반증하고 있다.

Windows Presentation Foundation 의 툴

WPF 는 개발자에게 많은 기능을 제공하고 있다. 원래 강력한 성능의 기술이라도 해도 좋은 툴이 곁들여진다면 더욱 높은 성능을 제공할 수 있다. WPF 의 경우, 마이크로소프트는 개발자를 위한 툴과 디자이너를 위한 툴을 각각 제공하고 있다. 여기에서는 두 툴에 대해 간단한 소개를 하고 있다.

개발자: Visual Studio 의 WPF Designer

소프트웨어 개발자를 위한 마이크로소프트의 최고의 툴인 Visual Studio 2008 는 WPF Designer 를 제공하고 있다. 개발자들은 이 툴을 사용해서 WPF 인터페이스를 그래픽으로 만들 수 있다. 다음의 그림은 그 예를 보여주고 있다.



앞의 그림에서와 같이, 개발자들은 WPF Designer 를 사용해서 인터페이스를 그래픽으로 또는 XAML 로 만들 수 있다. 물론, 개발자가 원한다면 C# 또는 VB 코드를 사용할 수도 있다. Visual Studio 에서 사용자 인터페이스를 개발하듯이, 왼쪽 상단의 팔레트에 사용할 수 있는 컨트롤들이 보이며, 개발자들은 이 컨트롤을 디자인 면에 드랙앤드롭해서 인터페이스를 만들게 된다. 우측 하단의 속성창에서 각각의 컨트롤의 외양과 동작 속성을 설정할 수 있다.

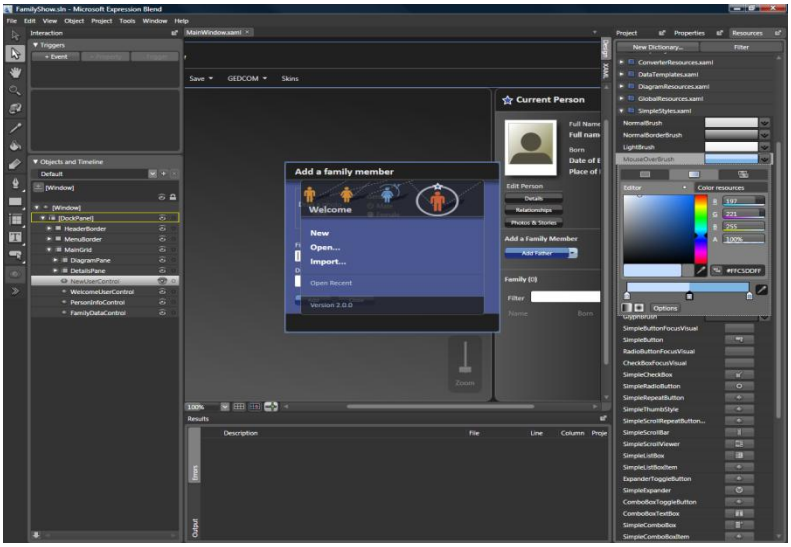
WPF Designer 는 여러 면에서 Windows Forms 의 그래픽 디자이너와 비슷하다. 그렇지만 WPF 와 Windows Forms 가 약간 다른 문제점을 해결하고 있기 때문에 각각의 개발툴도 조금씩 다르다는 것을 알아야 한다. Windows Forms 디자이너는 LOB 애플리케이션의 개발자를 풍부한 기능으로 지원하는 반면에, WPF 비주얼 디자이너는 WPF 개발자의 생산성을 높이려는 목적을 가지고 있다. WPF 는 LOB 개발자들이 필요로 하는 모든 것을 제공하고 있지 않듯이, WPF 비주얼 디자이너는 Windows Forms 디자이너가 제공하는 모든 기능을 제공하지는 않는다.

이 툴은 WPF 개발자를 위한 매우 다양한 기능을 제공한다. XAML 용 Intellisense 를 통해 XAML 편집기에서 문장을 완성하거나, Live Thumbnaills 를 통해 현재의 시각 요소가 어떤 모습이 될 것인지를 실시간으로 확인할 수 있다. 예를 들면, 앞의 예에서는 화면의 중심 하단의 작은 푸른색 박스는 Add a family member 요소가 화면에서 어떻게 보일 것인지를 알려준다. 또한 디자이너는 WPF 애플리케이션을 디버깅할 수도 있다.

그래픽 디자인 툴은 현대적 개발자 역량에 있어서 중요한 부분이다. WPF Designer 는 이 인터페이스 기술을 사용해서 사용자 인터페이스를 개발하는 개발자들의 부담을 크게 덜어주고 있다. 그렇지만 개발자들만이 훌륭한 사용자 인터페이스를 만드는 유일한 사람들이 아니기 때문에 Visual Studio 에서 제공하는 툴만으로는 충분하지 않다. 디자이너들이 인터페이스 개발 과정에 적극적으로 참여할 수 있도록, 마이크로소프트는 다음에서 설명하고 있는 Expression Blend 를 제공하고 있다.

디자이너: Expression Blend

WPF 의 주 목적은 디자이너가 사용자 인터페이스 개발의 주역이 되게 하는 것이다. XAML 에서도 이것이 가능하지만 디자이너를 지원하는 툴이 제공되어야 한다. 이 목적을 위해, 마이크로소프트는 Expression Blend 를 개발하였다. 다음의 화면은 Visual Studio 의 예가 이 툴에서 어떤 모습인지를 보여주고 있다.



이 예에서와 같이, Expression Blend 는 소프트웨어 개발환경보다는 디자인 툴에 가깝기 때문에 디자이너가 친숙한 방식으로 작업을 할 수 있다. Expression Blend 는 WPF 애플리케이션 인터페이스 개발에만 사용된다. 예를 들면 제일 좌측의 아이콘은 일반적으로 사용되는 WPF 컨트롤을 디자이너가 친숙한 방식으로 처리할 수 있게 해준다. 이 툴은 그래픽으로 애니메이션을 만들고 변형을 시키고 효과를 추가하는 작업도 할 수 있다. 디자이너 작업 결과는 XAML 파일로 생성되어 Visual Studio 에서 추가 작업을 할 수 있다.

Expression Blend 는 마이크로소프트 Expression 제품군의 4 가지 멤버 중 하나다. 다른 제품군은 표준 웹 인터페이스를 만드는 Expression Web, 벡터와 비트맵 이미지를 만드는 Expression Design, 이미지와 같은 디지털 자산을 관리하는 Expression Media 다. 4 가지 중에서, Expression Blend 만이 WPF 애플리케이션의 사용자 인터페이스를 만든다. 디자이너는 다른 툴을 사용해서 사용자 인터페이스의 부분을 만들 수 있다. 예를 들면 Expression Design 로 만들어진 인터페이스 GIF 를 사용할 수도 있지만, 이 툴들은 WPF 전용은 아니다.

인터페이스 기술 선택하기

현재 사용자 인터페이스를 매우 다양한 방식으로 만들 수 있다. 특정 애플리케이션에 어느 것이 적합한지를 결정하기는 쉽지 않다. 옵션을 제대로 선택하기 위해서는, 다음 3 가지 분야로 구분하는 것이 도움이 된다:

윈도우 애플리케이션 인터페이스;

표준 웹 인터페이스;

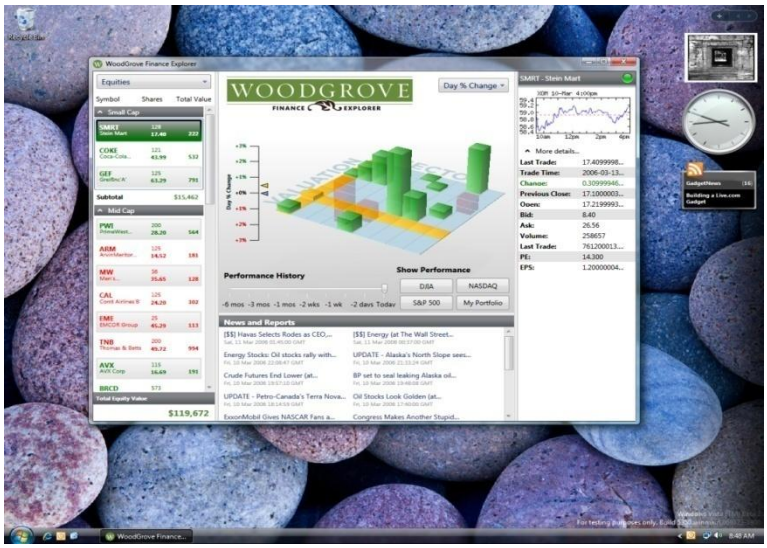
Rich Internet applications (RIAs).

이것들을 한쪽 끝에는 윈도우 애플리케이션, 다른 끝에는 웹 인터페이스 그리고 중간에 RIAs 가 있는 연속선상으로 볼 수도 있다. 여기에서는 각각의 옵션에 대해 설명하고 있다.

윈도우 애플리케이션의 인터페이스: WPF 와 Windows Forms

애플리케이션이 윈도우에서만 실행된다면, 사용자 인터페이스 기술을 선택하기가 비교적 쉽다. 더 현대적이고, 많은 인터페이스를 제시해야 하는 애플리케이션은 WPF 를 사용하는 반면에, LOB 애플리케이션은 Windows Forms 를 사용해야 한다. 마이크로소프트는 이런 인터페이스 유형을 설명하기 위해 “Connected Desktop Experiences”이라는 말을 사용해서 애플리케이션이 인터넷에 필요에 따라 접속하는 온라인/오프라인 능력을 요약하고 있다.

WPF 를 어떻게 사용해서 이런 유형의 애플리케이션을 만들 것인지에 대한 예는 앞에서 이미 설명했었지만, 다음은 WPF 에서 만들어진 윈도우 애플리케이션의 멋진 사용자 인터페이스 그림이다.

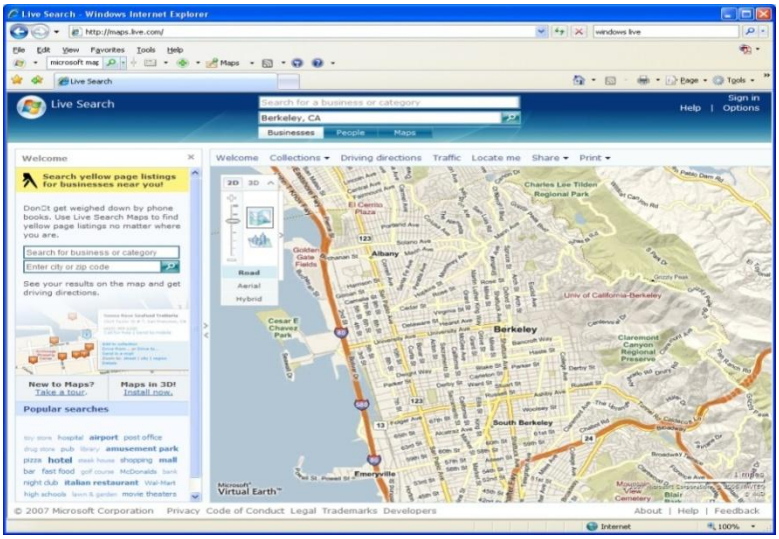


이 경우, Windows Vista 에서 실행되는 WPF 애플리케이션은 웹을 통해 원격 데이터베이스에서 조회해온 회계 데이터를 보여주고 있다. 이 예는 다양한 종류의 텍스트 디스플레이, 2D 와 3D 그래픽 등을 사용하고 있다. WPF 이전 기술을 사용했다면 이런 인터페이슬 만들기 힘들었겠지만 WPF 를 사용하면 훨씬 쉬워진다.

표준 웹 인터페이스: ASP.NET 과 ASP.NET AJAX

윈도우 애플리케이션뿐만 아니라 웹 애플리케이션도 많은 도움이 된다. 웹 사용자 인터페이스를 만들 수 있는 표준 기술은 JavaScript 가 사용된 HTML 이다. Asynchronous JavaScript and XML (AJAX)로 알려진 더 강력한 방식이 점점 더 유명해지고 있다. 애플리케이션은 AJAX 를 활용해 응답성은 크게 향상되면서도 클라이언트 시스템에는 브라우저만 있으면 된다.

마이크로소프트는 웹 인터페이스를 만들기 위해 ASP.NET 을, 그 인터페이스를 보여주기 위해 Internet Explorer 를 제공하고 있다. 또한 마이크로소프트는 Internet Explorer 와 다른 웹 브라우저에서 AJAX 형식의 인터페이스를 보여주는 ASP.NET 애플리케이션을 만들 수 있는 패키지 기능인 ASP.NET AJAX 를 제공하고 있다. 다음의 화면은 AJAX 를 이용한 표준 웹 인터페이스를 제공하는 애플리케이션을 보여주고 있다.



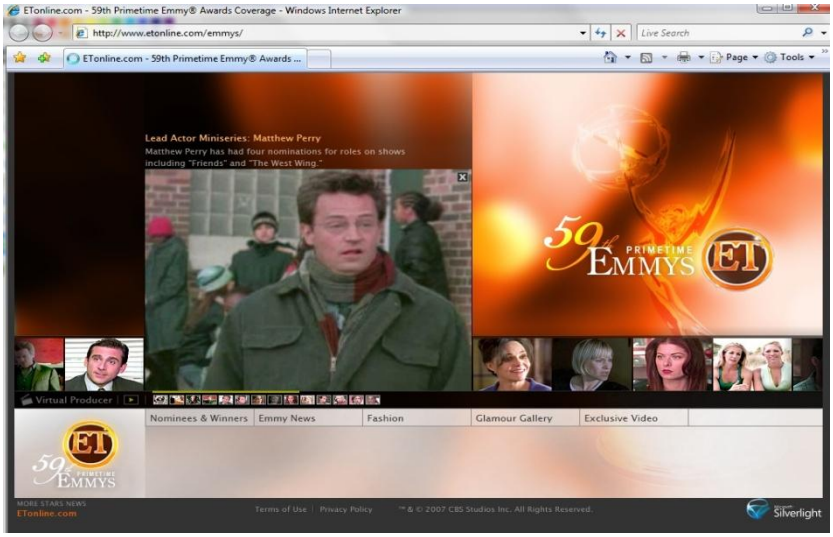
이 예는 마이크로소프트의 맵 서비스인 maps.live.com 을 보여주고 있다. 이 서비스는 AJAX 를 사용해서 응답성높은 인터페이스에서 대량의 데이터를 사용자에게 제공하고 있다. 특별히 다운로드받을 필요가 없으며 Internet Explorer 또는 Firefox 만 있으면 된다.

Rich Internet Applications: Silverlight

WPF 는 윈도우 애플리케이션이 현대적, 완벽한 기능의 사용자 인터페이스를 제공할 수 있게 해준다. 표준 웹 인터페이스로 유용하지만 간단한 인터페이스를 제공하는 일반 브라우저는 AJAX 로 응답성이 더 높아진다.

WPF 와 같은 데스크톱 중심의 기술과 달리, RIAs 는 브라우저 기반의 인터페이스를 제공한다. 표준 웹 기술과 달리, RIAs 는 애니메이션, 사운드, 비디오 등을 포함하는 더 강력한 방법을 제공한다. 이런 유형의 인터페이스를 지원하기 위해, 마이크로소프트는 Silverlight 를 제공한다.

Silverlight 의 목적은 다양한 플랫폼에서 WPF 의 하부 기능들을 제공하는 것이다. 실제로 이 기술의 코드명은 WPF/Everywhere 였었다. 이 목적을 위해, Silverlight 는 2D 그래픽, 이미지, 사운드, 비디오, 애니메이션, 텍스트를 지원하지만 3D 그래픽과 같은 첨단 기능은 지원하지 않는다. 윈도우, 매킨토시, 리눅스 (노벨)에서 실행되며 Internet Explorer, Firefox, Netscape 를 포함한 다양한 웹 브라우저를 지원한다. 개발자들은 가장 유명한 브라우저 호스팅 로직인 JavaScript 를 사용해서 코드를 작성할 수 있다. 그리고 Silverlight 는 WPF 기반이기 때문에, .NET Framework 와 XAML 을 활용하고 있다. 그렇기 때문에 개발자와 디자이너는 같은 지식과 툴을 활용하여 WPF 로 데스크톱 애플리케이션을, Silverlight 로 RIAs 를 개발할 수 있다. 다음은 Silverlight 로 만들어진 RIA 의 예다:



표준 웹 인터페이스와 달리, Silverlight 인터페이스는 사용자가 플러그인을 다운로드 받아야 한다. 그렇지만 다운로드 크기가 작고 한 번만 다운로드 받으면 되기 때문에 큰 부담이 되지 않는다. Silverlight 플러그인이 설치되면, 앞의 연예 사이트 같은 인터페이스가 가능해진다. AJAX 를 Silverlight 와 같은 RIA 기술과 결합시킬 수도 있는데, 이것은 사용자 인터페이스를 연속선상으로 생각해서 특정 애플리케이션에 가장 적합한 옵션을 선택하는 좋은 예다.

WPF 환경에서 Silverlight 를 설명하게 되면 의문 하나가 든다. Silverlight 는 WPF 의 XBAP 응용 프로그램들과 어떻게 다른가? 두 기술 모두 브라우저에서 실행되며 오디오, 비디오와 다른 현대적 인터페이스 기능을 지원한다. 대답은 아주 간단하다. XBAP 응용 프로그램들은 WPF (.NET Framework)가 설치되어있는 윈도우 전용 애플리케이션이며, 기업 애플리케이션에서 특히 유용하게 사용되어서 애플리케이션 개발자들은 그 애플리케이션이 윈도우에서만 실행되고 .NET Framework 이 설치되어 있다는 것을 알면 된다. 반대로 Silverlight 는 플랫폼 독립적으로, 매우 다양한 클라이언트에서 사용될

수 있다. 간단히 말해서, XBAP 응용 프로그램들은 브라우저에서 실행되는 윈도우 애플리케이션을 개발하지만, Silverlight 는 가장 유명한 시스템 모두에 대해 RIAs 를 개발하는 목적이다.

결론

사용자 인터페이스는 근본적으로 대부분의 애플리케이션에서 중요한 부분이다. 이런 인터페이스를 가능한 한 효과적으로 개발한다면 그것을 사용하는 사람과 기업에 많은 장점을 가져다 줄 수 있다. WPF 의 주 목적은 개발자가 이런 장점을 구현하게 해주며 윈도우 애플리케이션을 만들거나 사용하는 사람들에게는 WPF 가 대단한 선물이다.

WPF 는 현대적 사용자 인터페이스를 위한 통합 플랫폼을 제공하고, 디자이너가 인터페이스 개발에 더 적극적으로 참여할 수 있게 하고, 초기 인터페이스 기술과 연동될 수 있게 해서, 윈도우 사용자 인터페이스 경험을 크게 향상시키고 있다. WPF 가 대체하는 일부 기술은 지난 20 년 동안 윈도우 사용자 인터페이스의 인프라로 사용되어 왔다. WPF 의 목표는 향후 20 년 동안의 인프라를 제공하는 것이다.

참조 자료

Windows Presentation Foundation:

http://msdn2.microsoft.com/en-us/netframework/aa663326.aspx

Microsoft Expression:

http://www.microsoft.com/expression

Microsoft Silverlight:

http://silverlight.net

Electric Rain ZAM 3D:

http://www.erain.com/Products/ZAM3D/DefaultPDC.asp

저자 소개

David Chappell 은 Chappell & Associates (www.davidchappell.com)의 대표로, 강연, 저술과 컨설팅을 통해 전세계 IT 전문가들이 기업용 소프트웨어를 보다 잘 이해하고 사용하도록 돕고 있다.