SILVERLIGHT MIGRATION STRATEGIES

A Whitepaper by Wintellect, LLC

Authors: Noel Stieglitz

Version 1.0

Information in this document is subject to change without notice. Any example companies, organizations, products, people, and events depicted herein are fictitious. No association with any real company, organization, product, person or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Wintellect, LLC.

Wintellect may have patents, patent applications, trademarked, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Wintellect, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Other product and company names herein may be the trademarks of their respective owners.
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction and Motivation</td>
<td>4</td>
</tr>
<tr>
<td>Legacy Silverlight Technology Stack</td>
<td>5</td>
</tr>
<tr>
<td>Overview</td>
<td>5</td>
</tr>
<tr>
<td>Architecture and Languages</td>
<td>5</td>
</tr>
<tr>
<td>Core Presentation Framework</td>
<td>5</td>
</tr>
<tr>
<td>.NET Framework for Silverlight</td>
<td>5</td>
</tr>
<tr>
<td>XAML</td>
<td>5</td>
</tr>
<tr>
<td>Managed Programming Model (.NET for Silverlight)</td>
<td>5</td>
</tr>
<tr>
<td>Concepts, Patterns, and Practices</td>
<td>6</td>
</tr>
<tr>
<td>Templates and Custom Controls</td>
<td>6</td>
</tr>
<tr>
<td>Data Binding</td>
<td>7</td>
</tr>
<tr>
<td>Routed Events</td>
<td>8</td>
</tr>
<tr>
<td>Styles and Behaviors</td>
<td>8</td>
</tr>
<tr>
<td>MVVM</td>
<td>9</td>
</tr>
<tr>
<td>Prism</td>
<td>10</td>
</tr>
<tr>
<td>Dependency Injection and Service Locators</td>
<td>10</td>
</tr>
<tr>
<td>WCF RIA Services</td>
<td>11</td>
</tr>
<tr>
<td>Developer Tools</td>
<td>11</td>
</tr>
<tr>
<td>Visual Studio</td>
<td>11</td>
</tr>
<tr>
<td>Microsoft Expression Blend</td>
<td>11</td>
</tr>
<tr>
<td>Mitigating Risk Factors before and during Modernizing</td>
<td>11</td>
</tr>
<tr>
<td>MVVM Pattern Usage</td>
<td>11</td>
</tr>
<tr>
<td>Modularity</td>
<td>12</td>
</tr>
<tr>
<td>Portable Class Library</td>
<td>12</td>
</tr>
<tr>
<td>Dependency Injection, Following SRP</td>
<td>12</td>
</tr>
<tr>
<td>Service Code Quality and Technology Stack</td>
<td>12</td>
</tr>
<tr>
<td>Legacy Application Testing and Feature Documentation</td>
<td>12</td>
</tr>
<tr>
<td>A Path Forward</td>
<td>13</td>
</tr>
<tr>
<td>Web or Native</td>
<td>13</td>
</tr>
<tr>
<td>Scenario Recommendations</td>
<td>14</td>
</tr>
<tr>
<td>Mission Critical Enterprise or Core Business Application</td>
<td>14</td>
</tr>
</tbody>
</table>

---

Silverlight Migration Strategies Whitepaper  
http://www.wintellect.com/
AngularJS with RESTful Web API ................................................................. 14
Public Web Application ............................................................................... 16
JavaScript MV* Framework with Web API ............................................... 16
Non-Critical Enterprise or LOB .................................................................. 18
ASP.NET MVC ............................................................................................ 18
Standard or Advanced Media ...................................................................... 19
JavaScript MV* Framework/Windows Azure Media Services ......................... 19
Native Migration ......................................................................................... 20
Windows Presentation Foundation (WPF) ................................................. 20
Windows Store Apps .................................................................................. 21
New Concepts for Silverlight Developers .................................................. 21
Prism ........................................................................................................... 22
Web Migration ............................................................................................. 24
ASP.NET MVC ............................................................................................ 24
ASP.NET MVC Key Benefits ......................................................................... 24
MVC Architecture ........................................................................................ 24
Maintainability and Testability ..................................................................... 24
Standards Based HTML Output .................................................................... 25
Rich Client Web ........................................................................................... 25
AngularJS ..................................................................................................... 25
Knockout ...................................................................................................... 30
BreezeJS ....................................................................................................... 30
jQuery .......................................................................................................... 30
Modernizing Web Services .......................................................................... 31
ASP.NET Web API ......................................................................................... 31
APPENDIX A: Silverlight to AngularJS Technology Mapping ......................... 32
Appendix B: Relevant WintellectNOW Videos .............................................. 33
Resources ....................................................................................................... 34
INTRODUCTION AND MOTIVATION

Browser plugin based application architectures, such as Silverlight, are rapidly losing market and mind share to open source and modern proprietary technology options. The shortcomings of plugin based architecture, in terms of security, performance, stability, and other considerations are driving current development to open standards supported by a wider variety of browsers and platforms. Mobile web users on platforms which do not support Silverlight now make up nearly 18% of total global usage.  

Although Microsoft will provide support to Silverlight 5 through 10/12/21, industry support has waned considerably. For example, the Google Chrome development team plans to begin winding down support for the Netscape Plugin API (NPAPI) in 2014. NPAPI is currently a prerequisite for Silverlight (and other plugins) to run. Silverlight and a handful of other plugins will be whitelisted temporarily; however the goal is to completely remove support for NPAPI based plugins. NPAPI alternatives exist that could allow Silverlight to run. It’s likely that Chrome will continue to support Silverlight-based applications but this example illustrates the precarious nature of depending on Silverlight or any other non-standardized plug-in for critical or widely distributed web applications. 

There are few cases when developing a new application in Silverlight would provide the most value to stakeholders. However, existing Silverlight applications have several possible migration paths based on a multitude of variables. Small intra-department Silverlight applications with a limited user base in a controlled deployment environment may not require any modernization for the next several years. However, in most scenarios, refactoring an existing Silverlight layer into a modern technology stack will likely provide the most value to stakeholders in terms of platform and browser support, user experience, maintainability, performance, security, and stability.

---

1 Mobile Phone Usage [http://bit.ly/1iGVUZD](http://bit.ly/1iGVUZD)
LEGACY SILVERLIGHT TECHNOLOGY STACK

OVERVIEW

Silverlight is a cross-browser, cross-platform technology intended to facilitate the construction of “Rich Internet Applications” (RIA). Silverlight currently runs on all major desktop browsers via a plugin, and more rarely as an Out of Browser application on the desktop. Although a similar framework is used to build native Windows Phone applications, Silverlight-based web applications are not available on any mobile device including Windows Phone. At the time of release, Silverlight excelled relative to competing frameworks at delivering media content, specifically DRM encoded video.

ARCHITECTURE AND LANGUAGES

CORE PRESENTATION FRAMEWORK

The core presentation framework includes XAML (described below) for layout, media support, DRM, vector graphic support, UI interactions, UI controls, and more.

.NET FRAMEWORK FOR SILVERLIGHT

The .NET Framework is a software framework that includes language interoperability between many languages, as well as the Base Class Library (BCL). The BCL is a set of libraries developers can use for common functionality such as database interactions, file I/O, and other core functionality. Silverlight uses a subset of the .NET Framework that includes the BCL, WCF, Data, and other core components. Most Silverlight applications use C# or VB.NET for the non-UI application logic.

XAML

Silverlight uses the Extensible Application Markup Language (XAML) to declaratively define rich user interfaces. XAML is an XML based markup language for generating rich object graphs that more easily allows the development and design teams to separate UI declaration from application logic. XAML can be created by a design team using Expression Blend or a development team using Visual Studio (the recent versions contain a pared-down version of the Expression Blend engine), and can also be written by hand.

MANAGED PROGRAMMING MODEL (.NET FOR SILVERLIGHT)
The Silverlight SDK includes a subset of the .NET Framework. It contains components and libraries such as a subset of the base class libraries (collections, etc.), data integration, the CLR, etc. Developers can use managed languages including C# and VB.NET to access and manipulate the presentation layer, implement business logic, and connect to external services like RIA Services (described below).

Silverlight includes several additional features like Isolated Storage, an asynchronous programming model, file management, and serialization support.

**CONCEPTS, PATTERNS, AND PRACTICES**

**TEMPLATES AND CUSTOM CONTROLS**

Silverlight templates enable heavy customization of Controls. Although styles can be used to customize properties of any Silverlight element (i.e. – the background color of a button), templates can completely replace almost every aspect of a control. Here is an example control template definition for a button:

Silverlight Migration Strategies Whitepaper

http://www.wintellect.com/
Developers can create custom controls that use control templates to build highly customizable user interactions.

DATA BINDING

Data binding in Silverlight is extremely powerful and flexible. The data binding mode can be configured as one way, two way, or one way to source. Developers can also hook into events when data changes via the INotifyPropertyChanged interface or via classes that derive from the base DependencyObject class.

DATA FORMATTING AND CONVERSION

The IValueConverter interface and the ValueConversionAttribute class provide a mechanism to apply custom formatting and conversion logic to a binding. For example a date converter may convert a DateTime object to a more readable string for displaying on the view.

DATA TEMPLATES

Data templates allow developers to specify how data is displayed. Data templates are defined in XAML:

```xml
<ControlTemplate TargetType="Button">
  <Border x:Name="RootElement">
    <!-- Create the SolidColorBrush for the Background as an object element and give it a name so it can be referred to elsewhere in the control template. -->
    <Border.Background>
      < SolidColorBrush x:Name="BorderBrush" Color="Black"/>
    </Border.Background>
    <!-- Create a border that has a different color by adding smaller grid. The background of this grid is specified by the button's Background property. -->
    <Grid Margin="4" Background="{TemplateBinding Background}">
      <!-- Use a ContentPresenter to display the Content of the Button. -->
      <ContentPresenter
        HorizontalAlignment="{TemplateBinding HorizontalContentAlignment}"
        VerticalAlignment="{TemplateBinding VerticalContentAlignment}"
        Margin="4,5,4,4" />
    </Grid>
  </Border>
</ControlTemplate>
```
Templates can be wired to a control via the **ItemTemplate** property:

```xml
<Grid.Resources>
  <DataTemplate x:Key="CBTemplate">
    <Grid>
      <Grid.ColumnDefinitions>
        <ColumnDefinition />
        <ColumnDefinition />
      </Grid.ColumnDefinitions>
      <Image Grid.Column="0" Width="50" Height="50" Source="{Binding Photo}" Stretch="Fill" />
      <TextBlock Grid.Column="1" Text="{Binding Title}" Margin="10" HorizontalAlignment="Left" FontSize="20" />
    </Grid>
  </DataTemplate>
</Grid.Resources>
```

**ROUTED EVENTS**

A routed event is an event that can “bubble” up the event from child elements to the successive parent elements in an object tree. This allows any object in the object tree to respond to an event further down the tree. Routed input events include:

- KeyDown
- KeyUp
- GotFocus
- LostFocus
- MouseLeftButtonDown
- MouseLeftButtonUp
- MouseRightButtonDown
- MouseRightButtonUp
- MouseMove
- MouseWheel
- BindingValidationError
- DragEnter
- DragLeave
- DragOver
- Drop

**STYLES AND BEHAVIORS**

A style is a group of property values that can be applied to a single or group of elements. Silverlight applications are stylized declaratively using XAML. Styles can be defined separately from controls and user interfaces for reuse and to standardize the look and feel of a Silverlight application. Styles can inherit from other styles to reduce
repetitive declarations. For consistency across applications, developers can create reusable themes, as used in the Silverlight Toolkit\(^3\).

**MVVM**

The Model-View-ViewModel (MVVM) pattern is used almost ubiquitously in Silverlight applications. MVVM provides an abstraction of the view from the ViewModel and business model (e.g. business domain) and was created specifically to take advantage of the data-binding paradigm.

The view, (declaratively defined in XAML in Silverlight applications), has no knowledge of the ViewModel or business model.

The ViewModel is responsible for translating and exposing the business model into a subset of data needed for the view as well as providing presentation logic. This clean separation between the view, the ViewModel, and business model allow for easy testing, loose coupling, and powerful data binding. It also allows for a streamlined workflow where designers can focus exclusively on the view while developers focus on the ViewModel and business model. MVVM is a more specialized pattern of Martin Fowler’s Presentation Model pattern.

**PRISM**

Prism provides guidance and a code library to Silverlight developers with the overarching goal of modularity, reusability, and long-term maintainability. While complexity is somewhat high, and the learning curve steep, Prism does help guide developers into a modular and layered architecture. For large composite applications that have many screens and interact with many back end systems, Prism helps provide the guidance needed to keep these system manageable.

**EVENT AGGREGATION**

The Event Aggregator provides an eventing pattern to enable communication between loosely coupled components. The service provides a publication/subscription model that does not require a direct reference between components.

**MODULARITY**

Prism provides support for modularity that includes module registration, lifetime management, and runtime loading. Each module implements the `IModule` interface and provides a hook for initialization. Inter-module communication is achieved through the event aggregation service. Modules can be partitioned along domain boundaries (i.e. Customers, Products, etc.) or application layers (Shared Services, Business logic, UI module, etc.).

Behaviors add reusable visual functionality to controls such as drag and drop, driving animation, etc.

**DEPENDENCY INJECTION AND SERVICE LOCATORS**

Dependency Injection (DI) containers and the Service Locator library enable loose coupling between dependencies and enable granular testing. Popular options include the Unity IoC container and the Microsoft Extensibility Framework (MEF).
WCF RIA SERVICES

Microsoft created a complimentary services framework, named RIA Services, to simplify services development for Silverlight applications. It provides tooling and a framework to build and coordinate application logic between the presentation tier and middle tier. RIA Services provides a tool that generates client side code to consume domain services. It also enables LINQ querying over the service boundary. RIA Services is a subset of, and builds upon WCF technology. Service messages are usually sent over the wire wrapped in a SOAP envelope, but the framework also supports REST and limited ODATA. RIA Services use a subset of the WCF services bindings, including HTTP/HTTPS for transport, and binary or text for transport. WS* specifications and other bindings are not available to RIA services.

DEVELOPER TOOLS

VISUAL STUDIO

Visual Studio is Microsoft’s flagship IDE that supports many languages, development paradigms, and frameworks. It has Silverlight designer support. Visual Studio integrates with Version Control Systems (VCSs), the NuGet package manager, and supports several deployment scenarios in addition to other functionality. Visual Studio is an exceptionally powerful tool for any Microsoft-centric developer.

MICROSOFT EXPRESSION BLEND

Microsoft Blend for Visual Studio (Microsoft Expression Blend) is a user interface design tool with features for developing web and desktop applications. The WYSIWYG editor can create XAML based interfaces for WPF and Silverlight. Blend for Visual Studio includes support for creating Windows Store app and Windows Phone app UIs.

MITIGATING RISK FACTORS BEFORE AND DURING MODERNIZING

Modernizing an application can present different challenges depending on the general quality, test coverage, and architecture of the legacy code base. A few factors can significantly change the scope of modernization depending on the target platform.

MVVM PATTERN AND SOLID PRINCIPLES USAGE

If the legacy Silverlight application strictly uses the MVVM pattern, migrating to a WPF or Native Windows Store Apps platform will deliver maximum code reuse. Following the SOLID\(^4\) principles also tends to lead to a higher

percent of code reuse. Migrating to the web (either rich client SPA or MVC) is not impacted by pattern usage or architecture in the legacy Silverlight application.

### MODULARITY

Well architected legacy Silverlight applications that are built modularly (either using PRISM, MEF, or some other mechanism) may be migrated to the web one module at a time. This technique may reduce risk and provide a smoother transition for users of legacy applications already in production. This technique works well when migrating from Silverlight to Angular.

### PORTABLE CLASS LIBRARY

Using a portable class library for shared client side logic increases device support across Windows platforms including Windows Desktop, Windows Phone, Windows RT, Xbox One, etc.

### DEPENDENCY INJECTION, FOLLOWING SRP

Dependency Injection and loose coupling in the business layer improves code reuse along with following the Single Responsibility Principle (SRP). The Single Responsibility Principle states that every class should have a single responsibility, where a responsibility is defined as “a reason to change”.

### SERVICE CODE QUALITY AND TECHNOLOGY STACK

Code quality on the server impacts the ability to reuse legacy RPC style web services and the ability to transition to RESTful services.

### LEGACY APPLICATION TESTING AND FEATURE DOCUMENTATION

Full unit and integration testing on all layers of the legacy application result in a higher quality modernized application. The tests serve to document functionality which may otherwise be misinterpreted when modernizing to a new technology stack.
A PATH FORWARD

WEB OR NATIVE

Several factors impact the decision to modernize to a native or web application. Native applications may make sense if one or more of the following are true:

- The application is a desktop application that requires extensive data entry.
- Users of the application may operate largely in offline scenarios that heavily interact with local storage.
- The application requires COM interoperability.
- The current development team’s skillset and preferences are geared toward native development.
- Rich touch interactions are required to use the application.

Web applications have many advantages over native or desktop applications, and are likely the best choice for the majority of scenarios. Web applications provide the widest reach in terms of browser and device support. With the recent device revolution, the desktop is becoming less important relative to mobile and tablet form factors. Although native applications can still provide a richer user interface, modern web frameworks and libraries continue to narrow the gap in browser application capabilities relative to native desktop.
SCENARIO RECOMMENDATIONS

MISSION CRITICAL ENTERPRISE OR CORE BUSINESS APPLICATION

Mission Critical Enterprise

Critical Core Business Functionality
Integrates with other applications and data stores
Continued evolution
Application lifetime could exceed several years

Priorities include maintainability, low total cost of ownership, and modularity.

AngularJS RESTful Web API

Context and Priority

A large, modular, connected application providing critical core business functionality. The application continues to grow and evolve with changing business needs. The application lifetime could exceed several years. Maintainability, total cost of ownership, long term sustainability, and modularity are important considerations.

Recommended Migration Path(s)

ANGULARJS WITH RESTFUL WEB API

Scenario Specific Benefits

- Rich UI
- Modular
- Widest device and modern browser support
- Minimal deployment and client support effort
- Improved scalability over non RESTful applications

ALTERNATIVE(S)

ASP.NET MVC


**Scenario Specific Benefits**

- Mature Framework
- Rapid initial development
- Ample third party control frameworks available
- Modular
- Wide device and browser (including legacy browser) support
- Can be used in conjunction with AngularJS
- Minimal deployment and client support effort

**WPF**

**Scenario Specific Benefits**

- Mature Framework
- Access to hardware (often required for medical, point of sale, and other applications)
- Excels at Microsoft Office integration scenarios
- Access to local resources and COM interop
- Maximum code and skill reuse
- Excels in offline scenarios
- Extremely rich UI
PUBLIC WEB APPLICATION

Public Web

Disperate users with a wide range of devices including mobile and tablet.

Priorities include performance, scalability, browser/device support. Unique, and rich UI are important differentiators.

AngularJS
RESTful Web API

Context and Priority

A public web application possibly available in a global context. Maximizing browser, device, and form factor support is of the upmost importance. A unique and rich UI, performance, security, and ease of deployment are important.

Recommended Migration Path(s)

RICH CLIENT WEB FRAMEWORK WITH WEB API

Scenario Specific Benefits

- Rich UI
- Reduced roundtrips to the server for better perceived performance
- Widest device and modern browser support
- REST based Web API provides maximum scalability

ALTERNATIVE(S)

ASP.NET MVC

http://www.wintellect.com/
Scenario Specific Benefits

- Mature Framework
- Ample third party control frameworks available
- Wide device and browser (including legacy browser) support
- Can be used in conjunction with AngularJS
## Non-critical Enterprise or LOB

<table>
<thead>
<tr>
<th>Medium range application lifetime</th>
<th>Priorities include rapid development and deployment, low cost initial development and support.</th>
<th>ASP.NET MVC or WPF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mostly isolated from other applications</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data oriented</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Recommended Migration Path(s)

#### ASP.NET MVC

**Scenario Specific Benefits**

- Relatively low learning curve
- Modular Architecture
- High Testability

#### ALTERNATIVE(S)

**WPF**

**Scenario Specific Benefits**

- Capitalize on existing Silverlight/XAML skillset
- Native is often preferred in cases that require
  - Drag and drop support
  - Microsoft Office integration
  - Hardware device support

---

http://www.wintellect.com/
## STANDARD OR ADVANCED MEDIA

### Standard or Advanced Media

| Video and media streaming, possibly with DRM and/or DVR | Priorities include the ability to play video and handle advanced media scenarios such as DRM and DVR functionality | Continue with Silverlight in the near term or consider HTML5 |

### Recommended Migration Path(s)

**RICH CLIENT WEB FRAMEWORK/WINDOWS AZURE MEDIA SERVICES**

**Scenario Specific Benefits**

- Wide and growing modern browser support
- DRM is part of the HTML5 Spec and already has support in IE11 and Chrome

### ALTERNATIVE(S)

**WINDOWS STORE APP**

- Smooth and adaptive streaming support
- Comprehensive DRM solution

### DELAYED MIGRATION

- Low cost
- Allows time for HTML5 DRM and DVR implementation and browser support to mature
Once the decision has been made to modernize with a native solution, there is still the choice between Windows Presentation Foundation (WPF) and Windows Store App.

**WINDOWS PRESENTATION FOUNDATION (WPF)**

Windows Presentation Foundation (WPF) is the recommended choice for native desktop development. WPF is very similar to Silverlight and in most cases will provide the fastest and least expensive migration path. Silverlight is a subset of WPF with some solutions multi-targeting both WPF and Silverlight.

WPF provides both the most flexible security model and the most flexible deployment model.

WPF is a mature and powerful technology that will run on all supported Windows desktop operating systems. It has a multitude of third party controls kits and frameworks, as well as established patterns and practices.

WPF uses the same languages and tools as Silverlight. WPF employs XAML for UI declaration and .NET for application logic. WPF follows similar patterns used in a Silverlight application, like MVVM, Prism, data binding, routed events, and more.

Unlike Silverlight, WPF applications have access to the full .NET runtime and are not hosted within the browser (note: WPF actually has XBAP support to run a browser, and Silverlight includes OOB support to run outside of the browser, however these features are rarely used).

A handful of controls are available in Silverlight but not available in WPF. These controls are listed in the table below. Keep in mind that any third party controls may need to be replaced with the WPF equivalent (if available), or possibly replaced with custom controls.

<table>
<thead>
<tr>
<th>Control Name</th>
<th>Replacement Options in WPF</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoCompleteBox (Silverlight SDK)</td>
<td>AutoCompleteBox (WPF Toolkit) or AutoSelectToolbox (Extended WPF Toolkit)</td>
</tr>
<tr>
<td>DataPager (Silverlight SDK)</td>
<td>Custom Control or 3rd Party</td>
</tr>
<tr>
<td>DescriptionViewer (Silverlight SDK)</td>
<td>Validation.ErrorTemplate</td>
</tr>
<tr>
<td>HyperlinkButton (Runtime)</td>
<td>Hyperlink (System.Windows.Document) or Button with custom control template</td>
</tr>
<tr>
<td>MultiScaleImage (Runtime)</td>
<td>Custom Control or 3rd Party</td>
</tr>
<tr>
<td>NumericUpDown (Silverlight Toolkit)</td>
<td>NumericUpDown (WPF Toolkit)</td>
</tr>
<tr>
<td>ValidationSummary (Runtime)</td>
<td>ValidationSummary (Runtime)</td>
</tr>
<tr>
<td>WebBrowserBrush (Runtime)</td>
<td>If required, launch an independent browser instance rather using WebBrowser</td>
</tr>
</tbody>
</table>
WINDOWS STORE APPS

Windows Store Apps provide a consistent deployment model, a powerful security model and rich touch/gesture support.

Migrating to a Windows Store application has the advantage of using familiar technology, namely XAML and .NET. Windows Store apps can still allow for significant code reuse when ported from an existing Silverlight application. Many libraries available for Silverlight, for example Prism, are also available for Windows store apps. Windows Store apps can share an even larger amount of code when the legacy application used portable class libraries (PCL).

NEW CONCEPTS FOR SILVERLIGHT DEVELOPERS

LOOK, FEEL, AND BEHAVIOR

Windows Store applications include a unique look and feel as well as behavior. Microsoft provides strict UX guidance for Windows Store apps as well as an application certification kit to ensure certain minimum requirements for all applications available within the app store.

The UX Guidance ensures applications have a consistent navigation behavior, animations, consistent touch interactions, scaling behavior, and commanding.

The certification kit verifies applications follow the UX Guidance, is touch enabled, is responsive even on low powered devices, and ensures usability and consistency.

While deploying from the application store is not a requirement (in which case certification is not required), it is still considered a best practice to follow Microsoft’s guidance and certification process.

DEPLOYMENT OPTIONS

Currently there are two ways to deploy a Windows Store app:

1.) The Windows App Store. The Windows app store is the preferred choice for Windows applications for public consumption. It provides automatic updates, a secured sandbox to operate within, and a certification process to ensure application quality.

2.) Side Loading. Side loading allows developers and IT departments to deploy applications without using the Windows App Store. This can be accomplished using PowerShell scripts, via System Center, or using InTune⁵. The application should be signed with a trusted root certificate. Side loading is a good choice if the target application is an internal LOB or enterprise application. A major downside to side loading is that updates can be more complicated (i.e. – no auto updates from the store). Side loading has other

⁵ Sideload Windows Store Apps with Intune http://bit.ly/1o2PdQw
complexities and costs that should be considered\(^6\). While not as unfettered as a traditional desktop application, side loaded applications do not required the application certification process.

### APPLICATION LIFECYCLE

Windows Store Apps need to handle application activation, resuming, and suspension. This is referred to as the application lifecycle\(^7\). Developers need to consider whether to resume or activate an application based on the time which has passed since the user last used the application. Whether the application should start fresh or resume depends on the type of application and scenario. For example an application with a shopping cart would likely resume while a weather application with stale data may start fresh. On suspension, the application should save app data. If an application has crashed, it should start fresh.

### TILES AND NOTIFICATION

Tiles are an applications representation on the Windows Start Screen. Tiles can change in response to notifications ("live tiles"). Toast notifications provide app specific notifications. Toast notifications can be periodic or push\(^8\). Windows Push Notification Service (WNS) allows developers to send toast notifications and update tiles accordingly. Windows Store apps can also be configured to poll a pre-configured URL to obtain tile information and will poll immediately upon installation to create a custom initial tile.

### TOUCH AND INPUT

Touch is one of the core value propositions in Window Store applications. Microsoft’s guidance gives users a consistent touch experience across all applications so they are easier to learn\(^9\).

- Tap for primary action
- Slide to pan
- Swipe to select, command and move
- Turn to rotate
- Pinch and stretch to zoom
- Swipe from edge for app commands (up from bottom of screen or down from top)
- Swipe from edge for system commands (right or left from edge)

---


\(^8\) WinRT by Example [http://bit.ly/1exI1P](http://bit.ly/1exI1P)

While often not as important as it is for desktop applications, Prism for the Windows Runtime (WinRT) provides guidance to developers for building Windows Store business apps using XAML, C#, WinRT, and development best practices. The Prism library and AdventureWorks sample application exhibit how to:

- Implement MVVM
- Manage application lifecycle
- Implement validation
- Manage application data
- Implement controls and pages
- Use touch, search, and tile notifications
- Implement localization and accessibility

Future releases of Prism will include more implementations using the portable class library (PCL). PCL support will enable more code reuse across various Windows based operating systems like Windows Phone, Windows RT, etc.

---

WEB MIGRATION

ASP.NET MVC

ASP.NET MVC is a great choice for large-scale enterprise web applications. It is a mature MVC (described below) based framework that provides a clear separation of concerns with an overarching goal of high testability. Unlike ASP.NET Web Forms, ASP.NET MVC embraces the latest web standards and the stateless nature of the web. ASP.NET MVC allows mobile first “responsive” web development.

ASP.NET MVC can be enhanced with the rich client libraries and frameworks listed below.

ASP.NET MVC KEY BENEFITS

ASP.NET MVC is a mature and stable framework. It is well suited for enterprise applications that require a wide reach across devices and platforms (including table/mobile devices). ASP.NET MVC integrates well with rich client libraries and frameworks to provide richer dynamic UIs where required, and falling back to a relatively simple UI when speed to market trumps rich usability. The modular architecture provides a framework for long term maintainability and the ability to easily add new features, modules, and functionality. C#, used on the back end is an exceptionally powerful general purpose language. While HTML and JavaScript are ubiquitous in all major web development efforts on the front end, ASP.NET MVC uses a view engine, Razor, to enrich view declaration.

MVC ARCHITECTURE

The ASP.NET MVC framework implements the MVC design pattern. The MVC pattern improves separation of concern, by keeping business logic separate from view logic. This allows for better testability and a great parallel workflow for view designers and middle tier developers. The framework encourages building loosely coupled components as well as using dependency injection. It’s easy to apply domain driven design (DDD) in an ASP.NET MVC application because the model is completely separate from the view and controller.

ASP.NET MVC employs “convention over configuration” which decreases the number of decisions developers need to make to implement boilerplate logic. Convention over configuration simply means that developers can follow a specified pattern instead of writing configuration for standardized behavior. The naming conventions also allow for faster debugging because developers can easily URLs to controllers and actions based on the routing rules.

The default ASP.NET view engine is Razor. Razor provides the ability to define views in HTML with control flow and other helper methods written in C# or VB.NET. Razor is easy to learn and was designed to be expressive, fluid, readable, and compact.

MAINTAINABILITY AND TESTABILITY
Much like the MVVM pattern enables TDD and general testability for Silverlight applications, the MVC makes it easier to apply TDD and general testability to ASP.NET MVC applications. Visual Studio provides project templates that stub out Unit and Integration tests.

### Standards Based HTML Output

ASP.NET MVC provides outstanding control over the generated HTML. Developers are free to use any HTML based controls without it being specifically for the framework (as with Web Forms). MVC integrates with many client side libraries like jQuery, Angular, etc.

### Rich Client Web

Rich client web frameworks and libraries have gained explosive growth over the past few years. These JavaScript frameworks provide a rich user experience, add structure and reduce complexity relative to traditionally unstructured JavaScript heavy applications. These libraries are used to build enterprise grade web applications. Rich client web applications are often referred to as “Single Page Applications” (SPAs).

The numerous JavaScript libraries and frameworks vary in scope, intended purpose, industry support, and maturity. Some frameworks, such as AngularJS and EmberJS respectively provide a comprehensive solution. EmberJS and AngularJS are said to be “strongly opinionated” because they push a prescribed architecture that guides developers into solving problems in a particular manner. BackboneJS takes a different approach, providing minimum guidance allowing developers to roll their own solution.

Other frameworks intend to solve only a subset of problems. For example, BreezeJS is a JavaScript library for advanced client side data management. It includes LINQ like querying, caching, batch saving, and other data related features. BreezeJS is not considered a comprehensive framework because it intends to solve only the problem of front end data management. It does not try to solve the problem of structuring the application, data binding, testing, or other considerations. BreezeJS can provide value even in comprehensive frameworks such as AngularJS.

The realm of rich client web offerings is a rapidly evolving space. New libraries are constantly on the horizon. It is imperative to find frameworks that have broad industry usage to help mitigate risk.

Although covering all of the rich client JavaScript libraries is out of the scope of this paper, the following section provides a summary of the frameworks that provided the most value for Wintellect’s clients’ projects and internal applications.

### AngularJS

13 Getting Started with BreezeJS: [http://bit.ly/1gAPeNX](http://bit.ly/1gAPeNX)
AngularJS is a comprehensive open source JavaScript framework developed by employees at Google. It is an excellent choice for many development scenarios, including large enterprise web applications that require rich usability. Angular can be used to build the entire presentation layer, or can be included as a mini “Single Page Application” (SPA) within an ASP.NET MVC application.

Wintellect has had success implementing large scale complex enterprise applications that rely on AngularJS. The framework simplifies the building of dynamic, structured rich client web applications.

**KEY BENEFITS**

AngularJS allows developers to build rich, expressive, dynamic web applications. The framework includes MVC/MVVM pattern usage, dependency injection, a mechanism to build reusable components, data binding, standard patterns for server interactions, and much more. Features include:

- An MV* pattern to separate application logic, models, and view declaration
- Data binding that feels intuitive to Silverlight developers
- A service that abstracts away the complexities of asynchronous server interaction
- Dependency Injection
- The ability to build reusable UI components via Directives
- The ability to build reusable business components via Services
- Extensibility in almost every part of the framework
- Wide industry and community support with continued active development
- URL Routing
- Some limited touch and gesture based capabilities
- Localization/Globalization support
- Very high testability: Unit and Integration Test support including mocking functionality
- Broad desktop and mobile browser support

Despite the broad feature set, Angular’s client side JavaScript file is less than 36KB\(^{14}\) when compressed and minified. Angular is released under the MIT license with the source code publicly available on GitHub. See Appendix A for an architectural mapping between Silverlight and Angular components and patterns.

Angular and Silverlight’s architecture allow a “module at a time” migration path that may reduce the risk of migrating existing production applications.

**ARCHITECTURE**

Angular provides several mechanisms that help development teams build modular, reusable, testable, and maintainable applications. The major components are described below.

**SCOPE**

The scope is a fundamental part of Angular used throughout the framework. Scope serves as the execution context for expressions. Scopes provide both the functions and properties that the view needs for binding. The scope serves a similar to the view-model in a MVVM application.

```javascript
angular.module('crmApp.controllers', [])
 .controller('CustomerListController', ['$scope', '$location', 'custDataService',
   function($scope, $location, custDataService) {
     $scope.search();

     $scope.search = function () {
       custDataService.getAllCustomers($scope.criteria, $scope.sort).then(function (data) {
         $scope.customers = data.results;
         $scope.totalItems = data.inlineCount;
         $scope.totalPages = data.inlineCount / 5;

         console.log('Error calling api: ' + error.message);
       });

       $scope setPage = function (pageNumber) {
         $scope.currentPage = pageNumber;
         $scope.search();
       }
   });
```

**CONTROLLERS**

The controller is a function that is used to provide state and behavior, via the scope, to the view. A well designed Angular application will have little business logic defined in a controller. The controller can have services, which encapsulate business logic, injected into them. In the latest versions of Angular, the use of a convention called “controller as” collapses the role of the $scope into the controller itself, causing it to function much more like a self-contained viewmodel.

**SERVICES**
Services provide a way mechanism to keep objects in an Angular application for the entire lifetime of the application. Services are lazy loaded singletons. Services generally provide methods related to a single domain, for example, the built-in $http service provides a way to communicate with web services. Angular includes several services out of the box, and all applications of size may define their own services.

**VIEW**

Angular views are defined declaratively in HTML and Angular specific markup. The Angular specific markup includes custom directives (described below) and expressions `{{ customer.length }}`, which bind to the scope. The declarative nature of the presentation layer will be very familiar to XAML developers.

```
<div data-ng-controller="CustomerListController">
  <div class="row">
    <div class="col-md-3">
      <pagination on-select-page="setPage(page)" items-per-page="5" direction="" class="panel panel-default">
        <label>Customer Count: </label>{{customers.length}}
        <div class="panel-heading">Filter Results</div>
        <div class="panel-body">
          <form role="form" class="form-inline">
            <div class="form-group">
              <input data-ng-model="criteria.customerID" type="text" />
            </div>
            <div class="form-group">
              <input data-ng-model="criteria.minFreightCost" type="text" />
            </div>
            <div class="form-group">
              <button class="btn" data-ng-click="search(criteria)"/>
            </div>
          </form>
        </div>
      </pagination>
    </div>
  </div>
</div>
```

**MODULE**

The module is a way to isolate blocks of functionality in an Angular application. The benefits include testability, code reuse, and the ability to load different pieces of an Angular application in different orders. An Angular application can contain one or more modules.

**ROUTE PROVIDER**
Angular matches URLs to specific controllers and views using the route configuration defined with the $routeProvider service. Below is an example:

```javascript
var app = angular.module('crmApp', [
  'ngRoute',
  'crmApp.services',
  'crmApp.controllers',
  'breeze.angular',
  'breeze.directives',
  'ui.bootstrap'
]);

config([,
  '$routeProvider', function($routeProvider) {
    $routeProvider
      .when('/customer', {
        templateUrl: 'partials/CustomerList.html',
        controller: 'CustomerListController'
      })
      .otherwise({ redirectTo: '/' });
  }
]);
```

The built in $routeProvider and $route services are relatively inflexible compared to some other frameworks, however, thanks to Angular’s extensibility, there are other routing options. The most popular is the Angular UI-Router project. It provides state based routing, nested route capability, as well as other features.

**FILTERS**

Filters give Angular developers a way to format data that is displayed to a user. For example, a number could be displayed as currency using the built in currency filter. There are several built in filters, and like most things in Angular, developers can create their own when needed. Silverlight developers will find filters to be similar to Value Converters.

**DIRECTIVES**

Directives provide a way to “extend the vocabulary of HTML”. Directives can define new HTML elements, HTML attributes, CSS classes, or much less commonly, comments. Directives are used in almost every single Angular
view. Directives can be used to build custom UI components. It’s also where Angular developers can add logic to manipulate the DOM.

**KNOCKOUT**

Knockout is a JavaScript data binding library that uses the MVVM pattern. Because of the MVVM pattern usage, many Silverlight developers find Knockout to have a lower learning curve. Knockout includes declarative binding, dependency tracking and templating. It is not a comprehensive framework like Angular. A good case for Knockout is to add limited dynamic behavior to an MVC application. Wintellect does not recommend building a full-fledged rich client SPA application with Knockout.

**BREEZEJS**

Breeze is a JavaScript library that helps manage data in rich client web applications. Breeze is especially useful for enterprise applications that often have complex object graphs or large amounts of data. Breeze projects the same functionality of WCF RIA Services into a JavaScript application. Breeze has integration points with other popular frameworks including Angular on the front end, and Web API and Entity Framework on the service tier.

**JQUERY**

By some measures, JQuery is the most widely used JavaScript library in existence\(^\text{15}\). It excels at HTML document traversal and manipulation, event handling, and much more in a way that works with most desktop and mobile browsers. Its purpose is to normalize access to the DOM across disparate browsers. Much of the functionality of JQuery is replicated in Angular’s jqLite library. Angular applications do not necessarily depend on JQuery; however selecting nearly any other library or framework for a rich client JavaScript application will likely require the use of JQuery.

---

MODERNIZING WEB SERVICES

Silverlight applications often use WCF or WCF RIA Services as the server communication mechanism. WCF and WCF RIA Services embrace an RPC style of communication and abstract away the underlying transport mechanism from the service logic. While it is possible to use RPC style services from many client technologies, upgrading to a modernized service architecture will provide long term value in most cases.

ASP.NET WEB API

ASP.NET Web API is a framework geared toward building RESTful JSON based services. RESTful JSON based services are best suited for access from mobile devices, tablets, and JavaScript frameworks/SPA applications. JSON is relatively lightweight compared to SOAP. Unlike WCF and RIA Services, a RESTful architecture embraces HTTP and will only work over HTTP. See the table below for a line by line comparison.

<table>
<thead>
<tr>
<th>WCF</th>
<th>ASP.NET Web API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enables building services that support multiple transport protocols (HTTP, TCP, UDP, and custom transports) and allows switching between them.</td>
<td>HTTP only. First-class programming model for HTTP. More suitable for access from various browsers, mobile devices etc enabling wide reach.</td>
</tr>
<tr>
<td>Enables building services that support multiple encodings (Text, MTOM, and Binary) of the same message type and allows switching between them.</td>
<td>Enables building Web APIs that support wide variety of media types including XML, JSON etc.</td>
</tr>
<tr>
<td>Supports building services with WS-* standards like Reliable Messaging, Transactions, Message Security.</td>
<td>Uses basic protocol and formats such as HTTP, WebSockets, SSL, JQuery, JSON, and XML. There is no support for higher level protocols such as Reliable Messaging or Transactions.</td>
</tr>
<tr>
<td>Supports Request-Reply, One Way, and Duplex message exchange patterns.</td>
<td>HTTP is request/response but additional patterns can be supported through SignalR and WebSockets integration.</td>
</tr>
<tr>
<td>WCF SOAP services can be described in WSDL allowing automated tools to generate client proxies even for services with complex schemas.</td>
<td>There is a variety of ways to describe a Web API ranging from auto-generated HTML help page describing snippets to structured metadata for OData integrated APIs.</td>
</tr>
<tr>
<td>Ships with the .NET framework.</td>
<td>Ships with .NET framework but is open-source and is also available out-of-band as independent download.</td>
</tr>
</tbody>
</table>

### APPENDIX A: SILVERLIGHT TO ANGULARJS TECHNOLOGY MAPPING

<table>
<thead>
<tr>
<th>Task</th>
<th>Silverlight</th>
<th>AngularJS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declarative UI Definition</td>
<td>XAML</td>
<td>HTML</td>
</tr>
<tr>
<td>Reusable UI Components</td>
<td>Control Templates or CustomControl</td>
<td>Custom Directives (Element)</td>
</tr>
<tr>
<td>Animations</td>
<td>System.Windows.Media.Animation</td>
<td>CSS3 Animations and/or JavaScript via ngAnimate</td>
</tr>
<tr>
<td>Styles and Theming</td>
<td>XAML/Resources</td>
<td>CSS3 and LESS</td>
</tr>
<tr>
<td>Style Inheritance</td>
<td>Implicit Styles and/or BasedOn</td>
<td>LESS Mixins</td>
</tr>
<tr>
<td>Customizing Value Display</td>
<td>IValueConverter</td>
<td>Filters</td>
</tr>
<tr>
<td>Binding Models to Views</td>
<td>ViewModel and DataContext</td>
<td>Controller and Scope</td>
</tr>
<tr>
<td>Reusable Services</td>
<td>Prism Service Locator</td>
<td>Custom Services/Service Factory</td>
</tr>
<tr>
<td>Bubbling Events</td>
<td>Routed Events</td>
<td>$emit and $broadcast</td>
</tr>
<tr>
<td>Application Modularity</td>
<td>MEF and/or Prism Module Catalog</td>
<td>Angular Modules</td>
</tr>
<tr>
<td>Preferred Web Services</td>
<td>WCF or WCF RIA Services</td>
<td>RESTful services (accessed via the Angular $http/$resource Services)</td>
</tr>
<tr>
<td>Navigation and Routing</td>
<td>UriMapper</td>
<td>Routing</td>
</tr>
<tr>
<td>Inter-Component Communication</td>
<td>Prism Event Aggregator</td>
<td>Shared Services</td>
</tr>
<tr>
<td>Video</td>
<td>MediaElement</td>
<td>HTML5 Video Element</td>
</tr>
<tr>
<td>Out of Browser Support</td>
<td>Yes</td>
<td>Yes, via Chrome Apps</td>
</tr>
<tr>
<td>Encapsulating UI Logic</td>
<td>Behaviors</td>
<td>Custom Directives (Attribute)</td>
</tr>
<tr>
<td>View and Logic Separation</td>
<td>Control Templates</td>
<td>CSS/LESS</td>
</tr>
<tr>
<td>Cross Domain Calls</td>
<td>Access-Policy</td>
<td>CORS</td>
</tr>
<tr>
<td>One way data binding</td>
<td>BindingMode.One</td>
<td>Ng-Bind</td>
</tr>
<tr>
<td>Two way data binding</td>
<td>BindingMode.TwoWay</td>
<td>Ng-Model</td>
</tr>
<tr>
<td>One way to source binding</td>
<td>BindingMode.OneTime</td>
<td>Ng-Model</td>
</tr>
</tbody>
</table>
## APPENDIX B: RELEVANT WINETLECTNOW VIDEOS

<table>
<thead>
<tr>
<th>Video Title</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mastering the ASP.NET Web API</td>
<td><a href="http://bit.ly/1kPTkN">http://bit.ly/1kPTkN</a></td>
</tr>
<tr>
<td>Spicing up ASP.NET MVC Apps with KnockoutJS</td>
<td><a href="http://bit.ly/1nWU4Tu">http://bit.ly/1nWU4Tu</a></td>
</tr>
</tbody>
</table>
RESOURCES

Migrating Silverlight or WPF XAML to Windows Store app: http://bit.ly/1dWZ8Ek


Chrome to drop NPAPI: http://bit.ly/1guUjz2

Silverlight Architecture: http://bit.ly/1kPVvEr

Mobile Web Share: http://bit.ly/1fWA82h

MVVM: http://bit.ly/1ijezZ4

MVVM: http://bit.ly/1cWLjYh

MVVM http://bit.ly/PyEX6Y

Extended WPF Toolkit: http://bit.ly/1qARLi4


Routed Events: http://bit.ly/1iH9193


Windows Azure Media Services: http://bit.ly/1goBiCn

WCF and ASP.NET Web API: http://bit.ly/1iH97xo

Silverlight Animations: http://bit.ly/1fWAK1o

EmberJS: http://bit.ly/PyFg1E


BackboneJS: http://bit.ly/1dX0m2b