

# **Rich Internet Applications**

A comparison of two technologies that can be used for  
their development

By Geraint Huw Davies

A dissertation submitted in partial fulfilment of the requirements for the degree of Master of Science  
in Internet & Distributed Systems, Aberystwyth University.

Supervisor:

Christopher William Loftus, MBCS.

Aberystwyth University

September 2009

## Declarations

This work has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Signed \_\_\_\_\_ (**Geraint Huw Davies**)

Date\_\_\_\_\_

This work is being submitted in partial fulfilment of the requirements for the degree of Master of Science in Internet & Distributed Systems.

Signed \_\_\_\_\_ (**Geraint Huw Davies**)

Date\_\_\_\_\_

This work is the result of my own investigations, except where otherwise stated. Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.

Signed \_\_\_\_\_ (**Geraint Huw Davies**)

Date\_\_\_\_\_

I hereby give consent for my work, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organizations.

Signed \_\_\_\_\_ (**Geraint Huw Davies**)

Date\_\_\_\_\_

## Acknowledgements

I gratefully acknowledge and express my sincerest thanks for the support provided during both the taught programme and project phase of this MSc to my supervisor, Chris Loftus, who not only provided much needed advice and guidance but who also worked hard to help me maintain my focus during many external distractions. I hope I haven't let him down.

Likewise, my gratitude goes to all the staff of the Computer Science Department at Aberystwyth University for providing a thoroughly excellent Masters programme and a learning environment that I believe is second to none. I would like to acknowledge in particular: David Price, Dr. Fred Long, Neil Taylor, Dr. Lynda Thomas, Richard Shipman, Steven Kingston, Ann Robertson, David Sherratt and Sandy Spence for their tutoring and mentoring skills and their good humour. I also particularly wish to express my sincere gratitude and thanks to my tutor Dr. Edel Sherratt who, in addition to her academic responsibilities, provided much needed and timely pastoral support to this most immature of mature students. Many thanks also go to Daniel Gasienica, author of the Flex-based OpenZoom component who provided much needed assistance in its use and whose documentation provided a great grounding in the concepts that underpin multi-scale image technology. I would also like to thank my friend and colleague Michael Gulvin at BT, whose patient and informed teaching was a significant factor in helping me gain the confidence to undertake this programme – I hope this report goes some way in paying back the many favours I owe him.

Finally, my most heartfelt thanks go to my wife Llinos and daughter Angharad who showed great faith in me, provided much encouragement and oft needed emotional support and also made many personal sacrifices in order to support me during the two years of this MSc programme.

## Contents

<b>1. Abstract .....</b>	<b>6</b>
<b>2. Project Drivers &amp; Scope.....</b>	<b>8</b>
<b>3. Development Environment .....</b>	<b>9</b>
<b>4. Conventions Used .....</b>	<b>10</b>
4.1 Code Snippets .....	10
4.2 Terminology and Release / Version Numbers .....	10
4.3 Use of Footnotes & Bibliography .....	11
4.4 Italics, Highlighting and Quotation Marks .....	11
4.5 Glossary.....	13
<b>5. Project Overview.....</b>	<b>14</b>
5.1 Introduction .....	14
5.2 Top level approach.....	16
<b>6. Application Overview .....</b>	<b>17</b>
6.1 Description.....	17
6.2 Application Design .....	18
<b>7. Technology Overview.....</b>	<b>20</b>
7.1 Adobe Flex .....	20
7.2 Microsoft Silverlight.....	22
<b>8. Development Platforms .....</b>	<b>23</b>
8.1 Adobe Flex .....	23
8.2 Microsoft Silverlight.....	26
8.3 Comparison.....	30
<b>9. Learning Support Materials .....</b>	<b>32</b>
9.1 Introduction .....	32
9.2 Online Resources for Flex .....	32
9.3 Online Resources for Silverlight.....	37
9.4 Comparison.....	40
<b>10. Application Development.....</b>	<b>41</b>
10.1 Introduction .....	41
10.2 Initial Ideas.....	41
10.3 Flex Application.....	44
10.3.1 Visual Structure .....	44
10.3.2 Application Class.....	45
10.3.3 Containers.....	45
10.3.4 Application Startup.....	47

10.3.5	List Controls & Item Renderers .....	49
10.3.6	Binding .....	51
10.3.7	Animations.....	56
10.4	Silverlight Application .....	63
10.4.1	Visual Structure .....	63
10.4.2	Application Class.....	64
10.4.3	Containers.....	66
10.4.4	Application Startup.....	67
10.4.5	List Controls .....	71
10.4.6	Data Binding .....	72
10.4.7	Animations.....	75
10.5	Comparison.....	87
<b>11.</b>	<b>Multi-scale Images .....</b>	<b>89</b>
11.1	Introduction .....	89
11.2	Silverlight (Deep Zoom) .....	89
11.3	Flex (OpenZoom) .....	95
11.4	Comparison.....	98
<b>12.</b>	<b>User Experience .....</b>	<b>99</b>
12.1	Comparison.....	99
<b>13.</b>	<b>Conclusion.....</b>	<b>101</b>
<b>14.</b>	<b>Critical Evaluation .....</b>	<b>105</b>
<b>15.</b>	<b>Bibliography .....</b>	<b>108</b>

## 1. Abstract

Through this project I aim to identify and articulate the relative capabilities of two technologies (Adobe Flex and Microsoft Silverlight) in their support for the development of Rich Internet Applications.

Chapters 2 to 7 provide what is essentially background or supporting information for the key comparisons that I make in Chapters 8 to 12 (Note: Chapter 5 provides the key ideas that were encompassed in my original proposal together with supporting addenda). The later chapters each contain sub-sections in which I compare the two technologies with respect to the specific findings that I relate in those chapters. The overall conclusions which bring these all together are detailed in Chapter 13, with a critical evaluation of my work appearing in Chapter 14. Underpinning the investigation are two applications which I have developed in order to both assist my investigations and to use where necessary to demonstrate my conclusions and assertions.

Whilst not being able to address all of the aspects of the project that I had originally wished to, let alone all the facilities that the two technologies support, I believe that this project shows that there are enough differences between them such that for those parties who have yet to make any financial, intellectual or needs-driven commitment to either, careful consideration of their relative capabilities should be taken. Within this, I also contend that there is a philosophical difference between the two technologies, i.e. that Microsoft is aiming its Silverlight offering towards web designers, and Adobe's target for Flex is the application developer, and that this drive by both companies to poach a share of the market traditionally seen as the province of the other has resulted in some interesting differences between the implementations of their respective product platforms.

Through a demonstration of the results of my investigations I conclude that, all other external factors being equal, the Adobe Flex / Flex Builder 3 approach is overall more comprehensive, efficient and therefore I believe productive. However, with the exception of two very key issues (i.e. The lack of formal support by Adobe for an implementation of multi-scale image technology and problems with running Silverlight applications on Linux / Unix-based systems), I also conclude that the differences I have identified and have declared as important from my perspective could well be outweighed by other issues that are specifically pertinent to another individual, organisation or academic institution.

I also conclude that whilst I believe my arguments are sound in respect of the factors and criteria investigated, I also acknowledge that further work may identify other differences that may be influential. In addition, both Microsoft and Adobe announced beta release developments of their respective products during the course of this project and although I took the, what I believe was

justifiable, decision not to include them in my investigations (primarily for technical reasons), I must also acknowledge that it is possible that some of these developments may highlight a divergence (or desirable convergence) between the two technologies in one or more important measures.

## 2. Project Drivers & Scope

When embarking on any project, even one where the terms of reference have been explicitly defined by another party, the question “where should I start?” invariably springs to mind. The question that gets raised less often is “where should I stop?”

Whilst undertaking this MSc project, the second question popped into my head with increasing frequency as it became apparent that the functions and features provided in the two technologies under investigation are very many indeed. Even a subset of the capabilities provided in only one of the technologies could provide enough material for a large book – indeed they have already provided the material for many large books!

With this in mind, I think it is important to state upfront that this project is not intended to be a comprehensive source of information addressing all aspects of both Flex and Silverlight. Rather, it is intended to be a non-partisan comparison of some of the key facets of the two technologies with the focus on the issues and features that would be of interest to an audience of developers or academics who want to begin using either or both of them in their work.

On occasion, I may make passing reference to some area or other that I think might be of interest, but it would be impossible within the scope of this project to provide an in-depth commentary on, as an example, the differences between the internal structure of the Flex framework and how it compares to that provided in Silverlight / .Net. This is not to say that I think this type of analysis would be irrelevant or uninteresting, rather it reflects that a) I have neither the time nor the space within the bounds of this project to do it justice and b) there are many works that are widely available that make a far better job of it than I could.

So given my current position as an independent software application developer, my affinity to like-minded and similarly placed individuals within the Access development community and finally as a firm believer in the benefits of higher education, I aim to slant this investigation in a way that I hope is directly relevant to both “one man band” organisations like my own and also to those tasked with delivering courses and programmes in web application-related subjects. To this end, I will not be addressing issues or features that are applicable to large development teams or multi-developer environments.



### 3. Development Environment

To any developer, an understanding of the hardware and software environments that can be used to undertake any development is important, and the experience of others can sometimes influence your choice. The minimum specifications that are recommended by each supplier of the technologies can be found on the relevant web sites but I wanted to note the hardware specifications / software releases that I have used during the development phase of this project as a guide. I think it would be fair to note that at times I craved faster and bigger hardware, particularly when running both development environments simultaneously, but I can confirm that the following hardware and software combination has been usable when developing the sample applications that support this project:

#### **PC:**

- Dell Inspiron 5150 Pentium 4 3.06Ghz Single Core
- 2Gb RAM
- 120Gb Disk Drive
- 32Mb NVIDIA GeForce FX Go5200
- 15.4" Standard Aspect Screen with 1400 x 1050 resolution
- Microsoft Vista Ultimate – SP1

#### **Development Software:**

- Microsoft Visual Studio 2008 Standard Edition – SP1
- Microsoft Expression Blend 2 – SP1
- .Net Framework 3.5 – SP1
- Adobe Flex Builder 3 Professional Edition – Educational Licence
- Microsoft Deep Zoom Composer – November 2008 Edition
- OpenZoom SDK

#### **Development Browser:**

- Initially Internet Explorer 7, latterly Internet Explorer 8

#### **Browser Plug-ins:**

- Silverlight 2
- Flash Player 9 – Debug Version

## 4. Conventions Used

Within the limits of my creative and descriptive abilities, I have tried my best to make the following document as readable as possible. I have aimed to appropriately use diagrams, screen clips and code snippets to aid my explanations and hopefully also the reader's understanding. However, it may help if I describe some of my conventions at the start in order to minimise any confusion that might arise later.

### 4.1 Code Snippets

All code snippets have been copied from the source of the applications and are always placed in a single-cell table, like that seen below. When constructing the code – particularly XAML or MXML, I would sometimes put items on new lines if it aided my understanding of what was actually happening. However, when using the code to demonstrate a specific function or feature I have occasionally changed this approach in order to emphasise the point I wish to make, so the reader may observe what appear to be inconsistencies in code construction. At the very least, when a particular piece of code is fundamental to the point I am attempting to make, I have altered the font to add emphasis.

```
<mx:HSlider x="290" y="35" width="130" height="15"
  id="sldResize"
  minimum="{Number(txtResizeMin.text)}"
  maximum="{Number(txtResizeMax.text)}"
  labels=["Min', 'Max']"
  liveDragging="true"
  change="sldResize_Change(event)"/>
```

### 4.2 Terminology and Release / Version Numbers

Whilst reading through my first write-up of the project, I realised that I would sometimes refer to, for example, Adobe Flex at one point and Flex at another. One of the benefits of re-reads and re-writes is to identify such inconsistencies and eliminate them, but in case any slip through the net, the following can be considered synonymous:

- Adobe Flex, Flex, Flex 3
- Microsoft Silverlight, Silverlight, Silverlight2
- Visual Studio, Visual Studio 2008, VS, VS 2008
- Expression Blend, Blend, Blend 2
- ActionScript 3, ActionScript

An understanding of which releases I have used is also of particular importance as during the development of this project, both Microsoft and Adobe made announcements about the next releases of their products. Microsoft announced the beta releases of Silverlight 3 and Blend 3, and Adobe announced the beta release of Flex 4, together with the beta version of the corresponding

IDE – to be named Flash Builder 4 rather than Flex Builder 4. Whilst it could be argued that it would be beneficial to integrate a study of these new developments into this project, the complexities involved in running beta releases alongside full ones always tend to introduce problems and so I resisted the temptation. Therefore, to avoid confusion, any mention I make to Flex or Flex Builder can be assumed to be referring to the fully released versions, i.e. Flex 3 and Flex Builder 3 unless otherwise explicitly stated. Likewise, references to Silverlight and Blend should always be interpreted as Silverlight 2 and Expression Blend 2 unless otherwise explicitly stated.

In addition, the difference between *controls* and *components* could potentially cause confusion. In the Flex Builder 3 IDE, controls are a subset of components – controls being standard simple items such as buttons, sliders, checkboxes, etc., and components being things such as panels, charts, menu bars as well as items that have been custom built by the developer. I will attempt during the course of this report to be mindful of this subtle difference, but as there is a large degree of synonymy between the two terms, I hope that any reference I make incorrectly does not hinder understanding.

Finally, there is also a potential for misunderstanding with the terms used by both Microsoft and Adobe when referring to things that are associated with movement and change – words such as effect, behaviour, transformation, transition and animation spring to mind. Some of the available books and online resources help to clarify the differences between these, and I have tried to adhere to them if possible, but sometimes I found that it just got too confusing to be of any real benefit. To that end, please consider these words to be synonymous unless I make a specific point otherwise. In lieu of this, I use the word *animation* as a generic catch-all to reference a facility or function that provides a visual change in the user interface that occurs in relation to time.

#### **4.3 Use of Footnotes & Bibliography**

Where I make a reference to a specific portion of published text or webpage within this document, I have included the related details for the reference as a footnote to the page where the reference is made. The Bibliography lists the titles and associated details of the books that I have consulted from time-to-time during the project.

#### **4.4 Italics, Highlighting and Quotation Marks**

To minimise ambiguity and to differentiate them from normal words, I have italicised all technology-related words - i.e. words that have a specific meaning in relation to the code or concepts within the technologies being investigated.

As well as within the code snippets, boldening or highlighting is used within the main body of the report to provide emphasis.

In addition to being used to attribute ownership to a third party, quotation marks are sometimes also used to show the use of industry jargon, idioms or dare I say, even the occasional cliché.

Where they are obviously so, I have tended to type product names in standard text, e.g. Flex, Silverlight, Visual Studio, etc.

## 4.5 Glossary

I have tried to ensure that I explain the key acronyms as they occur during the report, but for those that are either a) less important or b) slipped through the net, the following table may be of use:

<b>Acronym</b>	<b>Expansion / Explanation</b>
.Net	Microsoft's framework which encompasses numerous facilities for the presentation, communication and workflow functions associated with computer-based applications
CLR	<b>Common Language Runtime</b> - a part of Microsoft's .Net framework which facilitates the use of many languages to develop the same functionality
IDE	<b>I</b> ntegrated <b>D</b> evelopment <b>E</b> nvironment - a generic term for an application that supports the development of computer applications
SDK	<b>S</b> oftware <b>D</b> evelopment <b>K</b> it - a generic term encompassing the libraries used to support the development of an application using a particular language or framework
Intellisense	A facility which uses the current context of the action being undertaken to try to "intelligently" predict and automatically complete what the user wishes to do – in a similar manner to "predictive texting" on a mobile 'phone. Although it is a Microsoft developed facility, it is rapidly becoming a generic term (in the same way that Hoover has) and I use it as such within this report.
SOAP	<b>S</b> ervice <b>O</b> riented <b>A</b> rchitecture <b>P</b> rotocol / <b>S</b> imple <b>O</b> bject <b>A</b> ccess <b>P</b> rotocol - An XML-based protocol for initiating method calls to a web service
SQL	<b>S</b> tructured <b>Q</b> uery <b>L</b> anguage - an industry-wide language supporting the transfer and manipulation of data to, within and from database systems
SQL Server	Microsoft's primary relational database product that employs SQL
SWF	Shockwave Flash - A file format supporting the rendering of video images in the Flash player browser plug-in
UI	<b>U</b> ser <b>I</b> nterface
WPF	<b>W</b> indows <b>P</b> resentation <b>F</b> oundation - A Microsoft developed framework that supports the management of visual components in many of its portfolio products
WSDL	<b>W</b> eb <b>S</b> ervices <b>D</b> efinition <b>L</b> anguage - An XML-based protocol for advertising the methods a web service supports
XML	<b>e</b> Xtensible <b>M</b> arkup <b>L</b> anguage - A flexible language that (amongst other things) supports documents readable by both humans and computer systems

## 5. Project Overview

### 5.1 Introduction

*“Rich Internet Applications (RIAs) offer a rich, engaging experience that improves user satisfaction and increases productivity. Using the broad reach of the Internet, RIAs can be deployed across browsers and desktops.” [1]*

Extending the above quote from Adobe, I think it is fair to state that the promise of Rich Internet Application technologies is to provide user experiences akin to those provided by applications which currently run on the desktop, but which can also harness the power of the Internet (and ubiquity of web browsers) to both enhance communication and provide access to services and resources not readily or easily available to traditional desktop applications.

Whilst it can be argued that the ability to provide enhanced user experiences via the Internet has existed for many years through the exploitation of technologies such as Java Applets, Ajax , DHTML and CSS, and more recently through the use of applications such as Google Gears, during the last 18 to 24 months new developments have been brought to market which promise to provide the application development community with new and more powerful ways of providing “desktop-like” products and services.

As a the owner of a small business focussed on providing Access database-based applications to schools and small businesses I, along with many of my fellow members of the UK Access User Group, have become increasingly concerned in recent times of both the ongoing attractiveness of standalone desktop applications and the continued development of, and support for, the Access product within Microsoft’s portfolio. Sitting within this context, we cannot ignore the increasing competition that Web / Internet-based applications can provide and that as new technologies appear, the traditionally perceived advantages provided by desktop application products such as Microsoft Access (rapid application development, richness of the user interface, ease of customisation, security, etc) will diminish or even disappear.

Adobe’s Flash/AIR and Microsoft’s Silverlight2/WPF are competing offerings from two of the largest players in the Rich Internet Application marketplace. Through the development and demonstration of a relatively simple application (i.e. an Image storage & retrieval application) I hope to be able to provide a certain degree of understanding about the key facilities that differentiate these two technologies.

Specifically, I aim to investigate and provide an understanding of the relative advantages and disadvantages of each of these technologies with respect to some, if not all, of the following areas:

- Development Platforms

---

[1] [http://www.adobe.com/resources/business/rich\\_internet\\_apps/](http://www.adobe.com/resources/business/rich_internet_apps/) (Last accessed 14/05/09)

- Learning support materials
- Application development issues
- User control types
- User experience / “look and feel”
- Accessibility issues
- Application deployment issues
- Local / Remote Database access
- Online / Offline capability
- Security issues
- Cross browser / cross platform support
- Web Services support
- Licensing / cost issues

Note: It could be argued that an alternative approach to this project might have been to consider a broader but shallower investigation, i.e. more technologies but less detail, however I feel that the context of my professional as well as academic circumstances make a narrower but more detailed investigation more relevant. Initially I planned to investigate the RIA technology offerings from the three largest players in the marketplace, i.e. to also include Sun’s JavaFX product, but the time constraints unfortunately made this impractical. When deciding which of the three technologies to abandon, I settled on JavaFX as the general impression from various web fora suggested that it was the least important of the three. There are no easily verifiable statistics available on usage or take-up that act as cast-iron justification for this decision but as exemplified by Yakov Fain of Farata Systems in the DevNexus Conference in 2009 [2], Adobe and Microsoft have offerings that, whilst relatively new, are considered to be more mature in their implementation and more significant in their potential impact. Mr Fain’s qualifying criteria for Rich Internet Application technologies are stated as being:

- Seamless deployment on the client
- High penetration of the runtime environment
- Web browser independence
- Fast client-server communication protocols
- Robust security

His contention is that as far as the three identified providers are concerned, Adobe already satisfies these, Microsoft **will** in 2010 and Sun **might** in 2010. As a representative of a company that is an Adobe Value Added Reseller, his assertions should not be treated as being totally non-partisan and one could also argue that his list of criteria does not include other important factors such as the relative sizes of the existing bases of development staff who are already familiar with each technology’s supported programming languages. However, an article by Sten Anderson in the

---

[2] [http://www.devnexus.com/presentations/PickingRightRIA\\_Atlanta.pdf](http://www.devnexus.com/presentations/PickingRightRIA_Atlanta.pdf) (Slide 4 - Last accessed 14/09/09)

online version of AjaxWorld magazine [3], whilst arguing that JavaFX has the potential to bring up-to-date RIA capability within reach of the very large Java Developer community, also concedes that *“...I don’t think JavaFX will win any ‘hearts and minds’...”*.

## 5.2 Top level approach

To support this project, I have exploited either free of charge or student / trial versions of the development packages recommended by the respective suppliers, together with relevant commercial or academic publications as appropriate.

Ignoring, for the moment, the work involved in bringing together this presentation of my findings, the project was split into two distinct phases:

- The primary objective of the first phase was to familiarise myself with the development platforms and languages provided (Flex Builder 3 / ActionScript / MXML, and Visual Studio 2008 / C# / XAML). Within this phase I also investigated the learning support tools and related documentation available, as well as building an understanding of some of the other issues such as support for security, richness of user controls, and web services support, etc.
- The second phase encompassed the development within each environment of an application that aimed to provide (as much as possible) the same or comparable functionality. Within this I aimed to not only build up an understanding of key issues such as development issues, but also to identify and address wherever possible some of the other measures covered in my proposal, e.g. user control types, user experience, deployment issues, cross browser support, etc. Within this I think it is important to note that the main driver for the development of the applications was to provide the knowledge and experience for the comparison (e.g. ease of use of the platform, richness of the development experience, etc), rather than to demonstrate any ability to write complex and well structured code. That’s to say that the application development was a means to an end, rather than being the principal deliverable.

Although my primary concern as a developer of databases for schools and small businesses is the capture and analysis of mainly text-based information, I felt that the promise provided by these technologies for the enhanced presentation and manipulation of graphic objects really called for the development of an application that exploited these types of functions. I therefore (initially) aimed to develop applications that combined database storage and retrieval functionality combined with graphic presentation features.

---

[3] <http://ajax.sys-con.com/node/768626> (Last accessed 13/09/09)

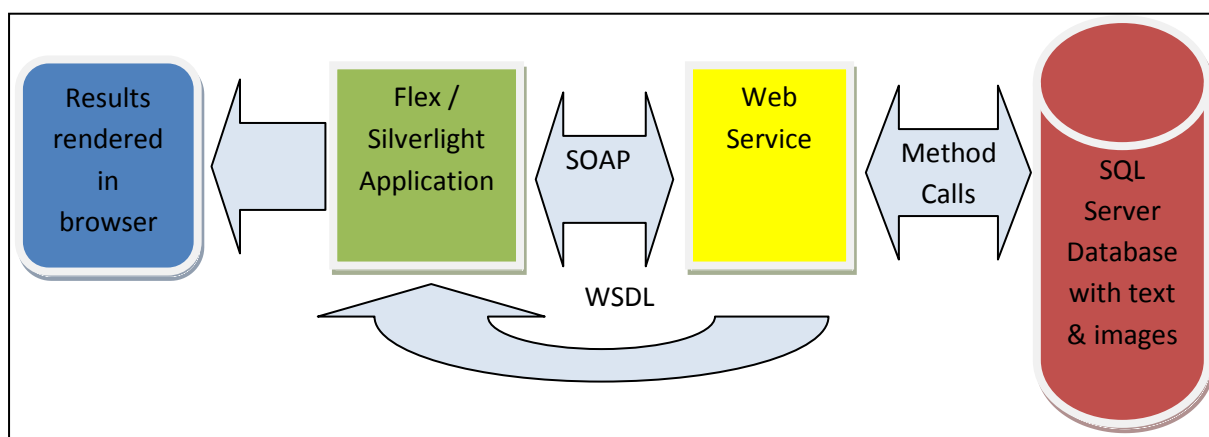


## 6. Application Overview

### 6.1 Description

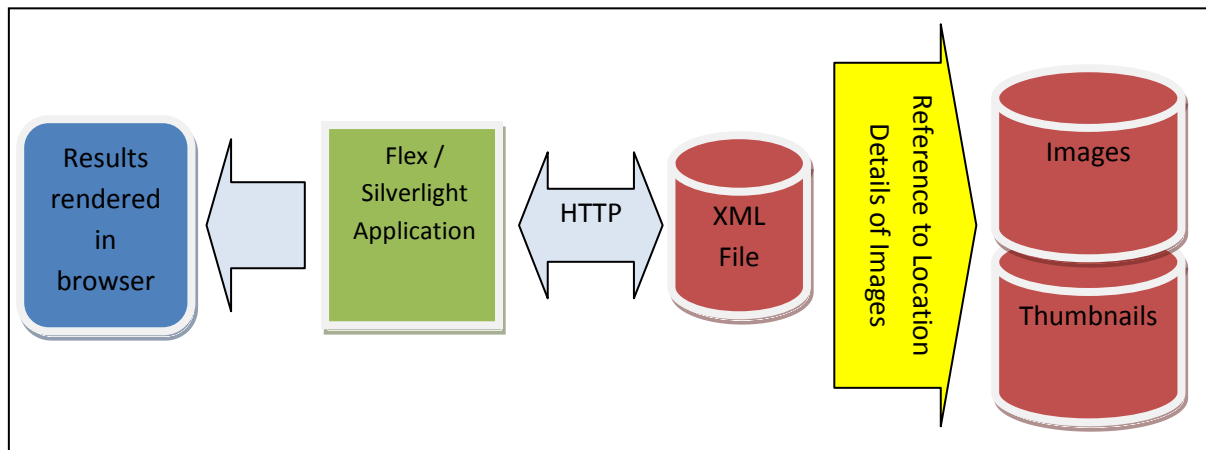
The principal design tenet of the applications that I have constructed to support this project was to provide a means of comparing the relative functions and facilities of the two platforms / frameworks – particularly with a view to the consumption and manipulation of images held in an external data source. However, within this, I also wanted to use the learning experience to act as base for the future use of either or both of these platforms for the development of real-world applications that I could deliver my customers.

To support this idea, my original plan was to provide a database of images and text that would act as the repository from which either application could access its content. I initially decided to create a simple Microsoft SQL Server database that would contain standard alphanumeric fields for the text I wanted to associate with each image, as well as a binary field for the storage of the images themselves - the plan being to have a back-end facility that would allow not only the downloading of the images and text information, but hopefully the uploading as well. I also intended to provide access to this data source via a SOAP / WSDL-based web service on the grounds that this would provide a supplier agnostic mechanism that it would be useful (and valuable) to experiment with in both of the consuming applications. The original idea for the design of the system can be seen in the following schematic:



Whilst the development of this back-end proved to be relatively simple, what became increasingly time-consuming was the construction of the code which would handle the transportation of the image files over SOAP. Numerous attempts at experimentation with coding proved fruitless, and whilst I was able to prove that the binary data held in the database could be rendered as an actual image in an application using a direct database connection, the results using SOAP / WSDL between the initial Flex application and the database produced many errors and no image – although text information was able to be retrieved by the application without a hitch.

I therefore, reluctantly, had to modify my original plan and use a simpler method for the consumption of images and associated text within the application. To that end, both applications have been constructed to exploit the support they both provide for accessing XML-based information and they now can each be represented with the schematic below:



An example of the elements within the XML file in the above schematic is as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<images>
  <image>
    <imageID>1</imageID>
    <imageArtist>Angharad</imageArtist>
    <imageName>Acanthus1</imageName>
    <imagePath>Acanthus1.jpg</imagePath>
    <imageDate>29/01/2009</imageDate>
    <imageComments>blah, blah, blah</imageComments>
  </image>
</images>
```

Each application uses a combination of thumbnail images to display within both data grid and list components and detailed (high resolution) images to be displayed in a dedicated image component when they have been selected in the data grid by the user. The information stored in the *<imagePath>* element in the XML file is used to provide a reference to the location of both the thumbnail and the detailed image.

Whilst this approach proved efficient in order to retrieve the actual images at runtime, it unfortunately suffers from a maintenance perspective – updates to the XML file and the contents of both of the folders containing the images need to be kept in step.

## 6.2 Application Design

To a large extent, the actual design of the application user interface grew from a desire to experiment with the standard facilities provided - both for the UI components and the animations that are supported by each technology. Both Adobe Flex and Microsoft Silverlight provide a broad

set of UI components and animations – some of which are common or overlap, and some which are unique to each platform. As this project represented a true “start at the ground floor” investigation, it seemed appropriate to start working with some of the basic or standard facilities and see where this took me, letting the development of my base of knowledge drive the “I wonder if it can do this?” investigations.

As a result of this the applications combine fairly simple facilities as well as some more complex ones with curiosity rather than any specific requirement providing the impetus. Having said that, each application also attempts to provide some degree of coverage of the standard range of controls and animations that can be implemented through each technology in order that suitable comparisons can be made. As well as standard objects such as buttons, textboxes and image controls, both applications implement data grid and list box components – both of which provide very useful methods for conveying information in a list format. In addition, both applications also implement slider components that provide visually appealing (or “tactile”) methods for altering relevant dimension or scale attributes. Behind the scenes, both applications also implement data binding techniques – a key methodology for linking both visual and non-visual components to the data source and also useful for linking components together. Finally, both applications attempt to demonstrate the key functions that are provided as standard for the visual manipulation of images. For Flex this includes *blur*, *dissolve*, *rotate* and *move* whereas for Silverlight this includes *scale*, *rotate*, *skew* and *translate*. Some of these functions achieve the same outcome but use different labels, e.g. a Flex *move* is equivalent to a Silverlight *translate*, but some are very different, e.g. Flex does not have a natural *skew* function, and Silverlight lacks an equivalent to Flex’s *blur* effect.

## 7. Technology Overview

### 7.1 Adobe Flex

*“Flex is a highly productive, free open source framework for building and maintaining expressive web applications that deploy consistently on all major browsers, desktops, and operating systems” [4]*

Flex is the technology framework provided by Adobe for developing and delivering in-browser RIAs and internet enabled desktop-based applications. Flex applications can be run in any browser that supports the Flash Player plug-in or (providing they have been developed to do so) run on any computer desktop (i.e. out of browser) that supports AIR (Adobe Integrated Runtime). Although the development of both application types can be undertaken using the open source (and therefore freely available) Adobe Flex SDK, the development of either type of application can also be achieved using the Eclipse-based Flex Builder 3 IDE.

**Note:** In order to minimise any confusion between Flex and Flash, I will attempt to maintain the following convention throughout the rest of this report: **Flex** refers to the application development framework that can be used to develop both in-browser applications (i.e. Flex applications) and internet-aware desktop applications (i.e. AIR applications); **Flash** is a catch-all term to that refers to both the browser plug-in and the applications that historically have been developed to run in it.

Both Flex and AIR applications support the use of a combination of MXML (**M**acromedia **eX**tensible **M**arkup **L**anguage) – a mark-up language that is principally used to define the visual attributes of the components within the application and ActionScript – an object oriented language based on the ECMAScript standard that is primarily used to define business logic and / or business objects. ActionScript 3 is the latest incarnation of the language and was introduced with Flash Player 9 – the first Flash runtime released by Adobe to support both Flex and AIR applications.

The MXML components within a Flex application are represented by XML-like tags or elements, e.g. `<mx:Button label="Dissolve"/>` which themselves correspond to ActionScript classes. When compiled, the MXML is parsed to generate the corresponding ActionScript classes which in turn are then compiled into SWF format bytecode and stored in a SWF file for later rendering in a browser using Flash Player or on the desktop using Adobe AIR. To all intents and purposes then, a compiled Flex or AIR application is a (Shockwave) Flash movie.

As well as being used to create separate classes within a Flex or AIR application, ActionScript can also be used “inline” within the main body of the MXML application (embedded within a *CDATA* block within an `<mx:Script>` element) as shown in the code snippet overleaf:

---

[4] <http://www.adobe.com/products/flex> (Last accessed 09/06/2009)

```

<mx:Script>
  <![CDATA[
    private function btnBlur_Click():void
    {
      if (sldBlur.value>=Number(txtBlurMax.text))
      {
        Alert.show("This image is already at the greatest level
of blur as defined in the MAX textbox", "Image Information");
      } else {
        effBlurBlur.play();
        sldBlur.value=Number(txtBlurMax.text);
      }
    }
  ]]>
</mx:Script>

```

Alternatively, separate ActionScript classes can be created and then called / referenced from the main application. This allows for the development of classes that represent business objects or for functions that execute more complex business logic. (Note: I have adopted the former (inline) approach within my Flex application).

In addition, Flex supports the support for inline CSS styling of components within a Flex / AIR application, as can be seen in the following code snippet:

```

<mx:Style>
  Application {
    backgroundColor: #000000;
    color: #cccccc;
  }

  Panel {
    borderStyle: solid;
    borderColor: #b7babc;
    borderThickness: 2;
    roundedBottomCorners: true;
    cornerRadius: 8;
    headerHeight: 22;
    backgroundAlpha: 1;
    highlightAlphas: 0.85, 0.1;
    headerColors: #000000, #999999;
    backgroundColor: #000000;
    shadowDistance: 4;
    shadowDirection: right;
    dropShadowColor: #666666;
    titleStyleName: "panelTitles";
  }

  .panelTitles {
    color: #ffffff;
    fontSize: 12;
    fontWeight: bold;
  }
</mx:Style>

```

## 7.2 Microsoft Silverlight

*“Microsoft Silverlight is a free runtime that powers rich application experiences and delivers high quality, interactive video across multiple platforms and browsers, using the .NET framework.”*[5]

Silverlight is widely regarded as being Microsoft’s response to Adobe’s dominance in the RIA marketplace – it has even been called by some commentators as Microsoft’s “Flash killer” and it is currently in its 2<sup>nd</sup> supported incarnation. Silverlight 1.0 was launched in September 2007 and provided support for JavaScript as a programming interface. Silverlight 2.0 (originally dubbed Silverlight 1.1) was launched in October 2008 and represents a significant upgrade to the first version. As well as adding support for many new UI controls, the programming aspect was significantly bolstered by the inclusion of support for .Net Framework CLR supported languages, e.g. C# and VB.Net, as well as support through the Dynamic Language Runtime for languages such as IronPython, IronRuby and managed JavaScript.

Like those developed using the Flex framework, Silverlight applications normally comprise of UI components that are instantiated using a declarative mark-up language combined with business logic / business object components that have been developed using an object oriented language. The mark-up language used to develop Silverlight applications is XAML (eXtensible Application Markup Language) and like MXML uses XML-like tags or elements to declare or define the components to be used, e.g. `<Button x:Name="btnGrow" Content="Grow"/>`.

Unlike Flex, which provides the ability to embed ActionScript inline within the markup, Silverlight uses the partial class / code behind model for the development of business logic and objects and these may be defined within the “code behind” for the class that contains the visual components. However, like Flex, other user-defined classes can be added to Silverlight applications to support more complex business objects or business logic.

When compiled, the a Silverlight application is packaged into a XAP file (which is equivalent a ZIP file with a different extension) that contains the necessary assemblies to run the application. When a Silverlight control is encountered in a web page, the Silverlight browser plug-in downloads this XAP file and executes it.

Note: The Silverlight product (or rather the collection of libraries of which it is comprised), is a subset of Microsoft’s Windows Presentation Foundation (WPF) framework. WPF could be considered to be analogous to the Adobe AIR product but actually offers far more functionality and interoperability which facilitates its use across more areas and with more products and services than

---

[5] <http://silverlight.net/default.aspx> (Last accessed 09/06/2009)

does Adobe AIR. Although it and Adobe AIR promise much for a developer to get his or her teeth into, due to time constraints I have not addressed any aspects of either within this project.

## 8. Development Platforms

### 8.1 Adobe Flex

The current principal development platform for Adobe Flex applications is Flex Builder 3. This comes in two editions - Standard and Professional, both of which are based on the Eclipse IDE. Adobe also provides a free of charge SDK (in both in-house and Open Source formats), which it claims can be used with “an IDE of your choice” [6] (Note: I am always wary of the universal availability implied by such claims, however at the time of writing, I can at least confirm the availability of a Flex SDK plug-in for Netbeans [7] In addition, a product that promises integration with Visual Studio is currently under development by SapphireSteel [8]). For academic users, Adobe provides the Flex Builder 3 Professional edition which can be obtained free of charge by registering your academic status (student or staff) with Adobe at <https://freeriatools.adobe.com/flex/>. In addition, the Adobe Professional version can also be obtained on a 60-day free trial. The main differences between the Standard and Professional editions of Flex Builder 3 are “*Flex Builder 3 Professional further adds powerful data visualization capabilities*” (i.e. charts), “*the new Advanced Datagrid, memory and performance profilers, and support for automated functional testing for developing business-critical applications*”[9]. Note: The Flex application that has been developed in support of this project was built using the Flex Builder 3 Professional Edition obtained under the Education licence arrangements.

The Flex Builder 3 IDE provides the developer with both visual (*Design*) and code (*Source*) views of the application within the same work area (as can be seen in the screen clips overleaf).

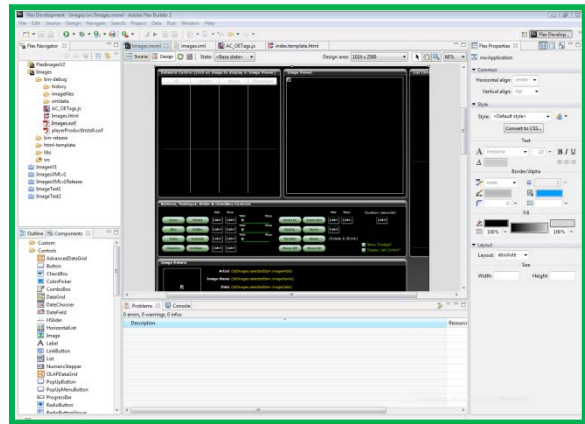
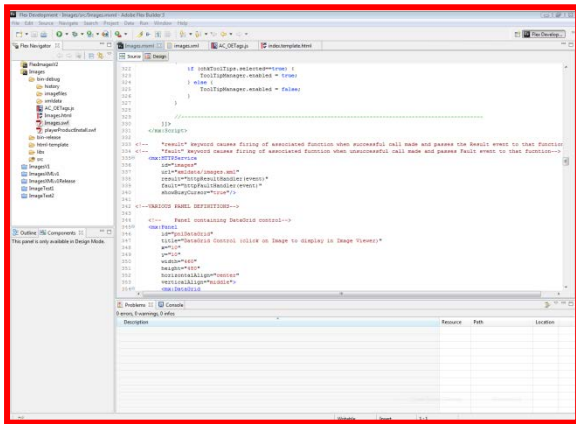
---

[6] <http://www.adobe.com/cfusion/entitlement/index.cfm?e=flex3sdk> (Last accessed 24/08/2009)

[7] <http://wiki.netbeans.org/FlexApplicationsWithNetBeans> (Last accessed 24/08/2009)

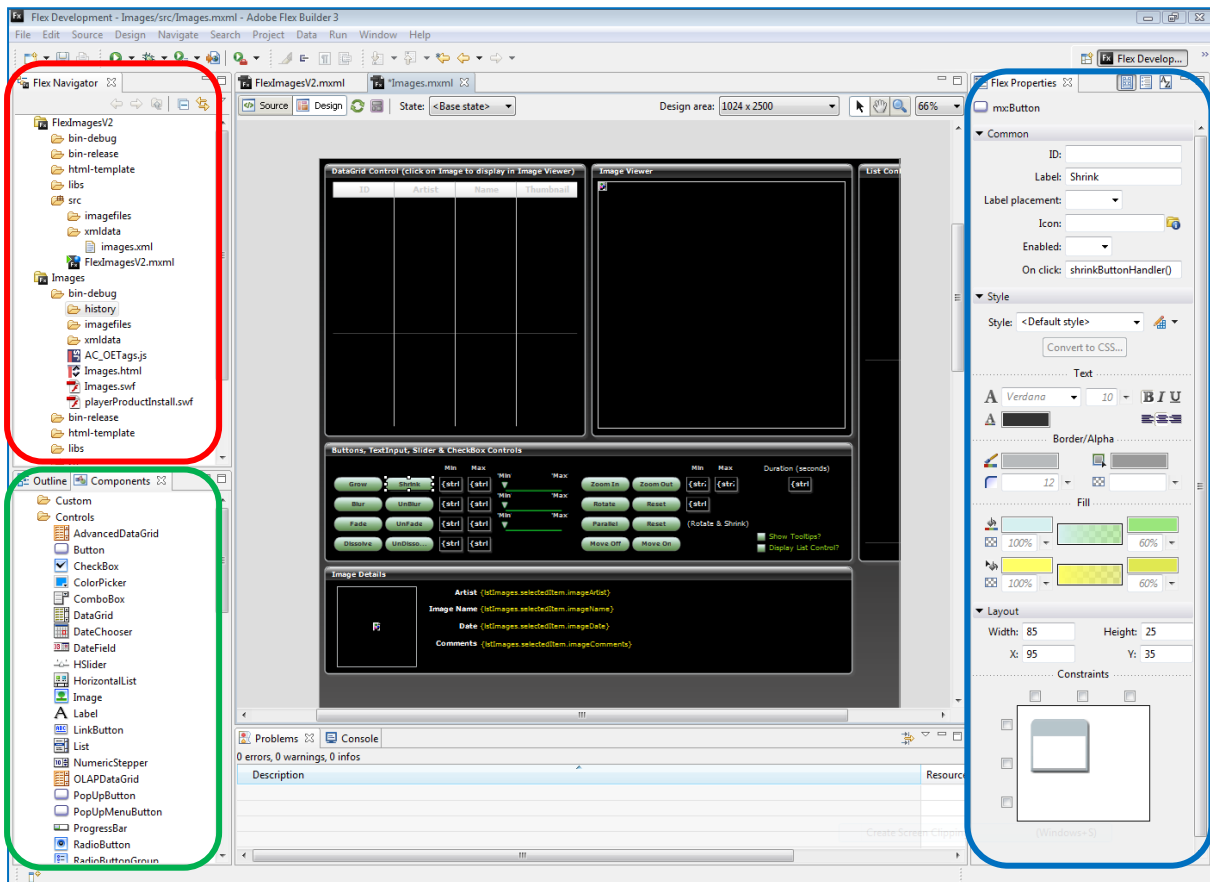
[8] <http://www.sapphiresteel.com/A-Brief-Guide-To-Amethyst> (Last accessed 24/08/2009)

[9] <http://www.adobe.com/products/flex/overview/> (Last accessed 24/08/2009)



Display of Flex application in Flex Builder 3 IDE in both **Source** and **Design** modes

The following screen clip demonstrates a typical layout of the Flex Builder 3 IDE when in *Design* mode. As with most IDEs, this layout can be customised to suit each developer's particular preferences.



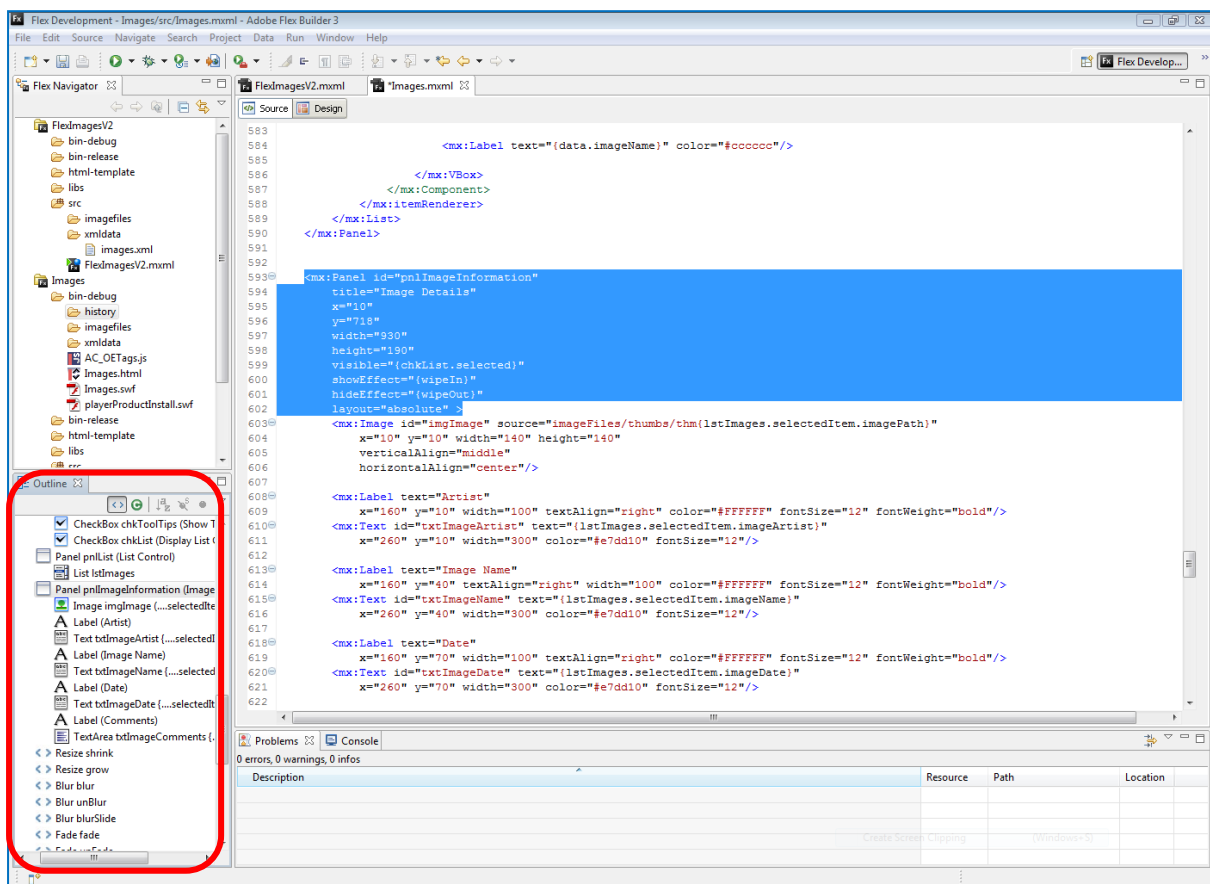
Detailed display of **Design** view mode of Flex Builder3 IDE



The *Flex Navigator* panel (bounded in **Red** in the previous screen clip) is available in both *Source* and *Design* views and allows the developer to identify and switch between the files and folders that comprise the Flex project currently under development.

When in *Design* view, the other panels that are frequently used are the *Components* panel (bounded in **Green**) and the *Flex Properties* panel (bounded in **Blue**). The *Components* panel provides access to both the standard and developer designed components within Flex (e.g. buttons, panels, datagrids, etc), each of which can be dragged and dropped into the design area. The *Flex Properties* panel is used to display and amend the properties of the currently selected component and provides an alternative (and sometimes quicker) method of altering the display attributes and behaviour of that component. (The attributes and behaviours of components can also be created and amended by directly working with the MXML / ActionScript code when in *Source* view).

When the IDE is switched to *Source* view (as shown in the screen clip below), the central part of the IDE screen displays the code for the application. In addition an *Outline* panel (bounded in **Red**) is displayed which identifies all of the components (and code blocks) used within the application. When any component or code block is selected in the *Outline* panel, the respective source code for that component or code block is highlighted.



**Detailed display of *Source* view mode of Flex Builder3 IDE**

As will be seen later, the approach provided by Adobe through Flex Builder 3 differs significantly to that provided by Microsoft. The environment in Flex Builder 3 is truly integrated in that it provides a single platform to support both the visual design and code development aspects of any given project. However, it must be said that the **Intellisense** facilities provided within the Flex Builder 3 IDE are not as complete as those provided with Visual Studio – especially when working with inline ActionScript code where it is necessary to enter the complete identifier of the control being referenced before the Intellisense / code completion routines kick-in. In addition, although the availability of both source and usable design views within the same platform is beneficial, it is a shame that they can't be viewed in a split configuration, i.e. both visible at the same time, as they can in Visual Studio.

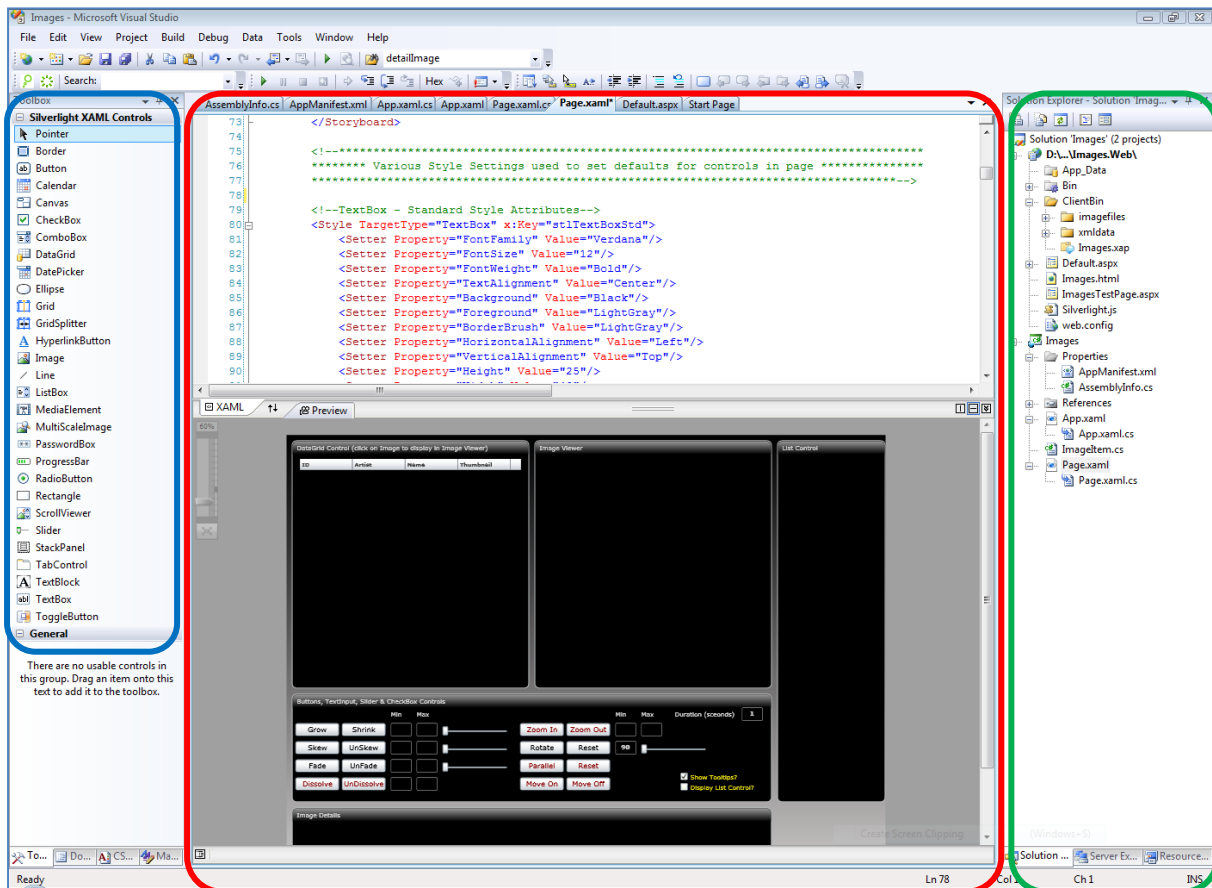
## 8.2 Microsoft Silverlight

On the assumption that the codebase for a Silverlight application will be written using one of the core .Net framework languages, e.g. C# or VB.Net, the principal platforms that are used for the development of Silverlight applications are Visual Studio 2008 and Expression Blend. (Note: For those interested in using their Python skills, there is a free Visual Studio shell-based IDE for IronPython created by CodePlex [10]).

As can be seen in the screen clip overleaf, Visual Studio provides as standard a split design / source view of a Silverlight application (bounded in **Red**) which provides at least a visual advantage over the comparable functionality provided in Flex Builder 3 (where it is necessary to use toggle buttons to flip between the two views). The *Solution Explorer* panel (bounded in **Green**) provides similar functionality to Flex Builder's *Flex Navigator* panel by displaying all of the objects that comprise the Silverlight application. The *Silverlight XAML Controls* panel (bounded in **Blue**) provides access to the controls that can be used to build the visual aspects of the application.

---

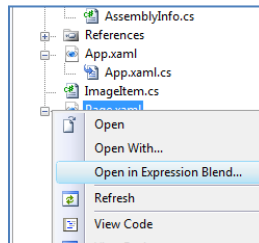
[10] <http://www.codeplex.com/IronPythonStudio> (Last accessed 18/09/09)



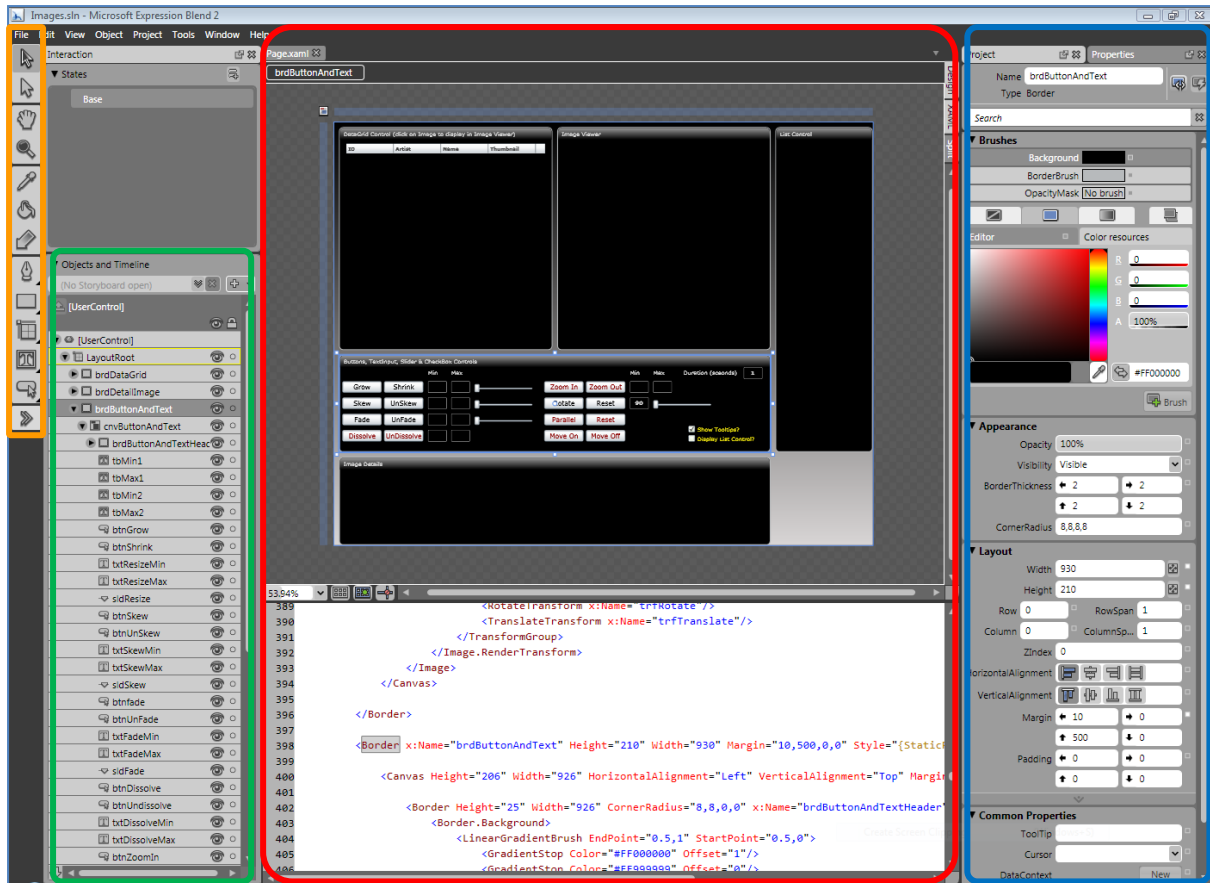
Typical view of a Silverlight application in Visual Studio

In addition to these panels, Visual Studio includes further panels to make other functionality accessible, e.g. A *Document Outline* panel which allows specific control elements to be selected and a *Server Explorer* panel for the management of connections to external data sources. As with Flex Builder 3, the Visual Studio IDE layout can be customised to suit an individual developer’s particular needs or preferences.

Visual Studio supports Intellisense for both XAML and managed code (and as mentioned previously, provides the functionality in a more comprehensive manner), but a significant difference between it and Flex Builder 3 is that it does not support the “drag & drop” of controls directly from the Silverlight XAML *Controls* panel on to the *Preview* area of visual design surface. Controls can be drag and dropped into the XAML area for manipulation and customisation after which the results will display in the *Preview* area, however to simulate the ability present within Flex Builder 3 which allows controls to be manipulated in a visual way, the Silverlight application needs to be opened in Expression Blend. This can be achieved by right-clicking on the *Page.XAML* object in the *Solution Explorer* panel and then selecting *Open with Expression Blend* (see screen clip overleaf):



After opening the application in Expression Blend, a display similar to the following screen clip is shown:

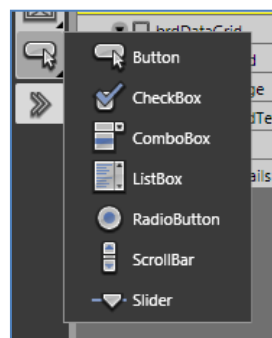


*Typical view of a Silverlight application opened in Expression Blend*

The key area of the Expression Blend interface is the work area in the centre (bounded in **Red**). This work area can provide a split configuration with both the *Design* and *XAML* views being displayed, or the developer / designer can select just the *Design* or just the *XAML* views (thereby mimicking the functionality of Visual Studio which is unfortunately lacking in Flex Builder 3). A component can be selected by either clicking on it in the *XAML* view or by selecting it from the list of components in the *Objects and Timeline* panel (bounded in **Green** in the screen clip). If a component is the outermost of a series of nested components, then it can also be selected by clicking on it in the *Design* view, but a particular annoyance with Expression Blend (in its current incarnation) is that components that form the inner elements of a nest cannot be selected this way. In my opinion,

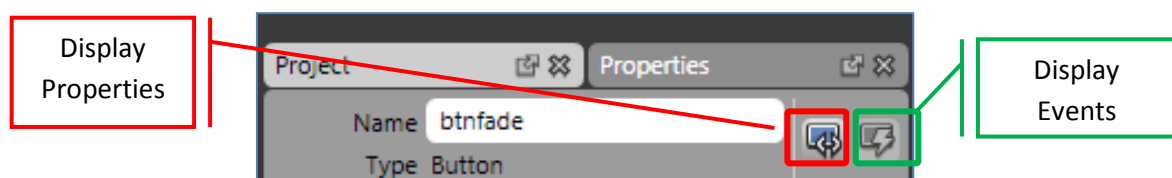
another significant productivity issue is the total lack of Intellisense support for the XAML view in Expression Blend – where the product scores highly for its support for the visual, it fails miserably in its support for the textual. (Note: As the name suggests, the *Objects and Timeline* panel also supports the construction of more complex timelines for animations, e.g. accelerations, decelerations and pauses, etc., but I have not used any of these within the current release of the Images application).

The *Toolbox* panel (bounded in Orange) is used to work with the visual components that are employed in the application. In addition to providing tools which allow for component selection, panning and zooming the display, working with colours and drawing lines (fixed or freeform), the *Toolbox* also includes the following components / controls: Rectangle, Ellipse, Line, Grid, Canvas, Stackpanel, Scrollviewer, Border, Textblock, Textbox, Password box, Button, Checkbox, Combobox, Listbox, Radio Button, Scrollbar and Slider. To expose the controls of a particular type in Expression Blend, you can either “right click” or “click & hold” the icon that represents that control type, e.g. expanding the controls of type *Button* in the *Toolbox* will provide the following:



*Expanded display of controls of type **Button** in Expression Blend*

The *Properties* panel (bounded in Blue in the large screen clip on the previous page) can be used to display the properties of a selected component and also any events that have been associated with it – the screen clip below highlights the buttons that are used to switch between these two views:



*Switching between **Properties** and **Events** views in Expression Blend*

Silverlight, like Flex, supports the use of custom components. These can be created in-house (usually comprising of a collection of standard components) or downloaded from various websites -

two such examples can be found at the websites for Telerik [11] and Netikatech [12]. Microsoft also supplies a **Silverlight Toolkit** which is *“a collection of Silverlight controls, components and utilities made available outside the normal Silverlight release cycle.”* [13] I will reference this site again in another context later in the report, but it is worth noting that more complex controls and components can be accessed through it.

### 8.3 Comparison

Differentiating the two technologies by comparing the standard development environment tools that are used to support them could be considered to be “putting the cart before the horse”, i.e. the application that is delivered is surely far more important than the tool used to develop it. However it must be remembered that the functions and facilities available in the respective IDEs have a direct impact on a developer’s productivity and therefore ultimately on the profitability of the work undertaken and there is no doubt in my mind that certain differences between the two development environments under investigation are significant. Visual Studio provides far better Intellisense and code completion functions and this can make a big difference for the application creator – especially when first starting to use the technology. However, the lack of support for the manipulation of visual components within Visual Studio, and the need to use the separate (and sometimes more complicated) Expression Blend program to achieve this is annoying at best and a severe hindrance to productivity at worst (and which is also particularly noticeable when working on a hardware platform that is several years older than the software running on it!). The lack of Intellisense functionality in Expression Blend (at least in Release 2) is also a severe deficiency. Flex Builder 3 certainly scores higher in the “integrated platform” stakes and the ability to easily manipulate visual elements from within the IDE is a big boon to productivity. Although it’s a shame that the IDE doesn’t support the same simultaneous design / source split view that is available in both Visual Studio and Expression Blend, my belief is that (notwithstanding the facilities provided in the respective technology frameworks) most application creators would find it more efficient to use than the Microsoft counterparts.

An observation that I think is worth making at this point in the report is that the view that Flex and Silverlight are targeting the same markets is only true on one level – namely that they both aim to facilitate the easy development of Rich Internet Applications. But looking at the product history of the two companies we can see that on another level they are approaching this from two different directions and I believe this explains some of the key differences in the development platforms. Adobe (and previously Macromedia) has traditionally supported the web designer / flash application community – a community used to working with visual objects, timelines and complex

---

[11] <http://www.telerik.com/products> (Last accessed 30/05/2009)

[12] <http://www.netikatech.com/products/toolkit> (Last accessed 30/05/2009)

[13] <http://www.codeplex.com/Silverlight> (Last accessed 24/08/2009)

animations, whereas Microsoft has traditionally targeted the application developer community. As a framework Flex supports animations, but Adobe has removed the complex issues of working with timelines in order that developers can now attempt to do (at a simple level) what designers used to be required for. Conversely, through the current Visual Studio / Expression Blend combination, Microsoft is now trying to bring the web designer community into its fold by creating a development environment where designers and developers can work in collaboration with common toolsets - and interestingly has included the timeline functionality that Adobe has removed. Indeed, a demonstration on Silverlight development held by the Visual Basic User Group in 2008 was delivered by a two-man designer & developer team and although they used a single PC, they switched between the Visual Studio and Expression Blend platforms as they took over their respective parts of the presentation. In short, as well as trying to expand existing markets, both companies are targeting the traditional market of the other and evolving their development / design product sets in ways that they believe will appeal to these markets, (a view also supported by Yakov Fain of Farata Systems [14]).

---

[14] [http://www.devnexus.com/presentations/PickingRightRIA\\_Atlanta.pdf](http://www.devnexus.com/presentations/PickingRightRIA_Atlanta.pdf) (Slide 16 - Last accessed 13/09/09)

## 9. Learning Support Materials

### 9.1 Introduction

Both technologies are comprehensively supported by the availability of many books and extensive online resources. The books that I have referenced throughout the period of my investigations are all listed in the Bibliography at the end of this document and I can certainly say that all have proved useful to some degree or other. However, to identify any as being better or worse than the rest would be both unfair and misleading as they all have particular strengths and weaknesses. I would also observe that with all of the technologies I have employed over the years, and contrary to the marketing hype found on their covers, I have never found a situation where “one book suits all”.

### 9.2 Online Resources for Flex

The principal online resource provided by Adobe to support Flex developers is the Adobe Developer Connection [15]. As well as acting as the primary gateway to the various Adobe supported fora for Flex developers (accessed via the **Community** tab [16]), through this portal Adobe also provides an excellent tutorial programme called **Learn Flex in a Week** [17] (see screen clip overleaf). The programme provides a series of approximately 45 online videos of varying duration together with approximately 25 exercises, (Note: the Adobe Media Player is required for the playback of the on line videos). As well as addressing the factors and issues associated with the design and development of Flex applications, the programme also covers topics related to the use of Adobe’s flagship development environment for Flex, i.e. Flex Builder 3, together with topics that cover integration with Adobe Creative Suite (CS3), PHP and Ajax.

---

[15] <http://www.adobe.com/devnet/flex/> (Last accessed 30/05/2009)

[16] <http://www.adobe.com/devnet/flex/?view=community> (Last accessed 21/09/09)

[17] <http://www.adobe.com/devnet/flex/videotraining/?sdid=DKOQK> (Last accessed 30/05/2009)



Adobe.com Sign in and join ADC | RSS | United States (Change)

ADOBE DEVELOPER CONNECTION

Search for... All Adobe.com Search

PRODUCTS TECHNOLOGIES

DEVELOPER RESOURCES

Home / Flex Developer Center /

## Flex in a Week

Learn Flex in a week by going through this video training course. To maximize your learning, we recommend that you view the videos and complete the exercises in the order in which they are listed. If you run into problems and have questions, please post your question to the [Flex in a Week](#) forum.


Note: Before you begin this training, be sure you have Flex Builder installed.

**Flex Builder 3**  
[Try](#) [Buy](#)

(Optional) Download a free copy of the O'Reilly book, [Getting Started with Flex 3](#) (PDF, 1.4 MB).

Want to view Flex in a Week videos in offline mode? Follow these steps:

1. Install [Adobe Media Player](#).
2. Click the My Favorites menu at the top.
3. Click Add RSS Feed at the bottom.
4. Copy and paste this URL: <http://sessions.adobe.com/FlexInAWeek/feed.xml>



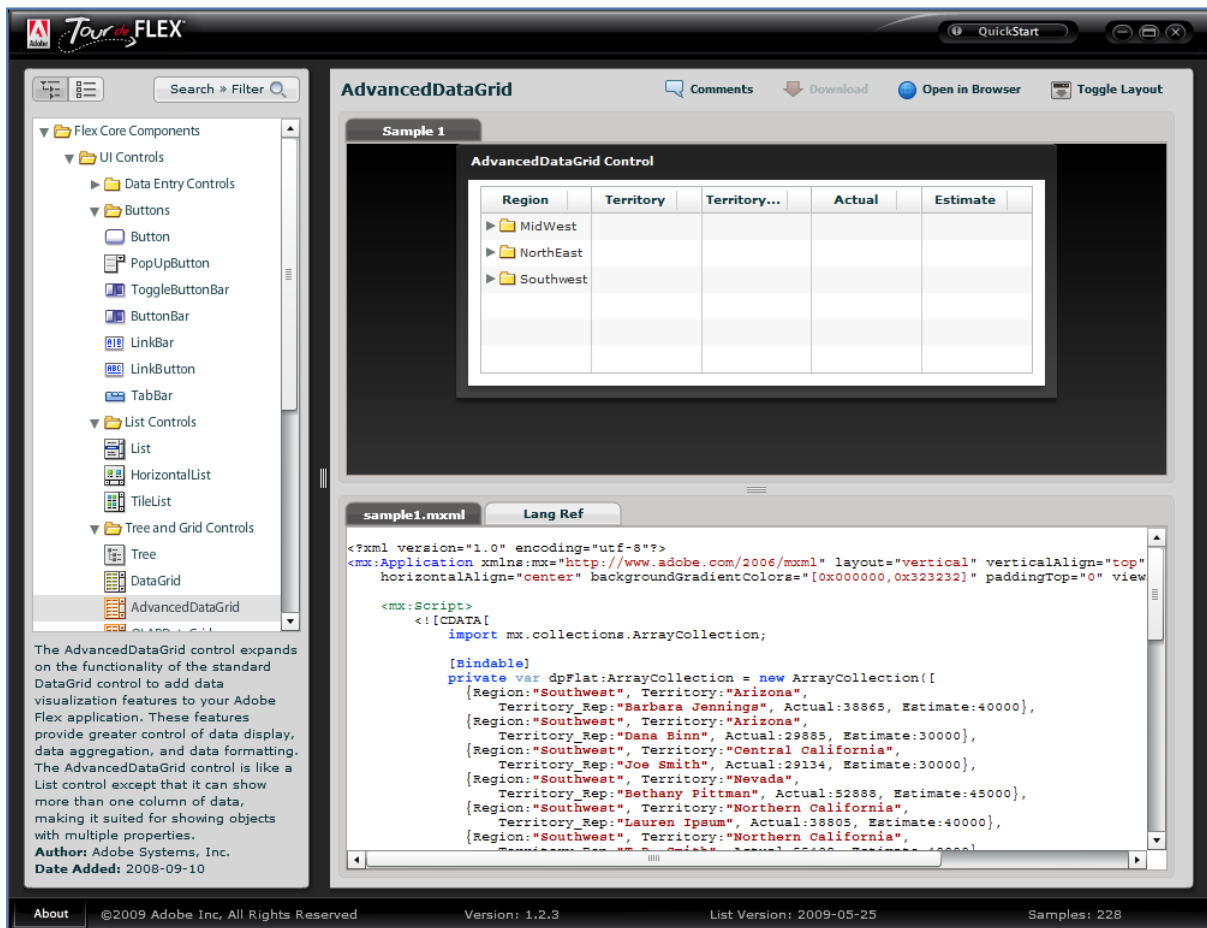
DAY 1: FLEX BASICS	DAY 2: COMPONENT DEVELOPMENT
<a href="#">Comparing Flash, Flex, Flash Player and Adobe AIR</a> (5:30) <a href="#">Introducing Flex Builder</a> (7:22) <a href="#">Creating a Flex Builder workspace and project</a> (7:04) <a href="#">Using pre-built Flex controls</a> (4:34) <a href="#">Understanding MXML</a> (4:07) <a href="#">Exercise 1: Creating a project and laying out controls</a> <a href="#">Binding data between controls</a> (5:48) <a href="#">Handling user events</a> (6:22) <a href="#">Introducing the event object</a> (10:40) <a href="#">Adding EventListeners with ActionScript</a> (5:21) <a href="#">Exercise 2: Binding data and handling a user event</a>	<a href="#">Displaying data in the DataGrid</a> (6:35) <a href="#">Working with containers</a> (12:12) <a href="#">Exercise 4: Working with containers</a> <a href="#">Creating custom MXML components</a> (8:42) <a href="#">Exercise 5: Creating custom MXML components</a> <a href="#">Implementing value object classes</a> (13:46) <a href="#">Creating custom events</a> (5:59) <a href="#">Creating custom event classes</a> (23:06) <a href="#">Exercise 6: Creating custom events and dispatching data</a> <a href="#">Customizing item renderers</a> (6:42) <a href="#">Exercise 7: Creating an item renderer</a>

[Create Screen Clipping](#)

*Home page of Adobe's **Learn Flex in a Week** online training programme*

In addition, Adobe provides an application called **TourDeFlex** [18] (which, as an AIR application, is downloaded to and runs on the desktop), that provides an extremely useful portal for investigating the potential use of many Flex components. As can be seen in the screen clip overleaf, TourDeFlex provides examples of the practical use of many of these, together with the MXML and / or ActionScript3 code that is used to generate them. The application also provides links to the associated MXML / ActionScript3 language reference. TourDeFlex also provides an area where skilled developers can demonstrate any advanced functions and components that they have developed – again supported by the provision of the associated code, and which in some respects provides similar functionality to the **Silverlight Toolkit** which I will describe later.

[18] <http://www.adobe.com/devnet/flex/tourdeflex/> (Last accessed 30/05/2009)

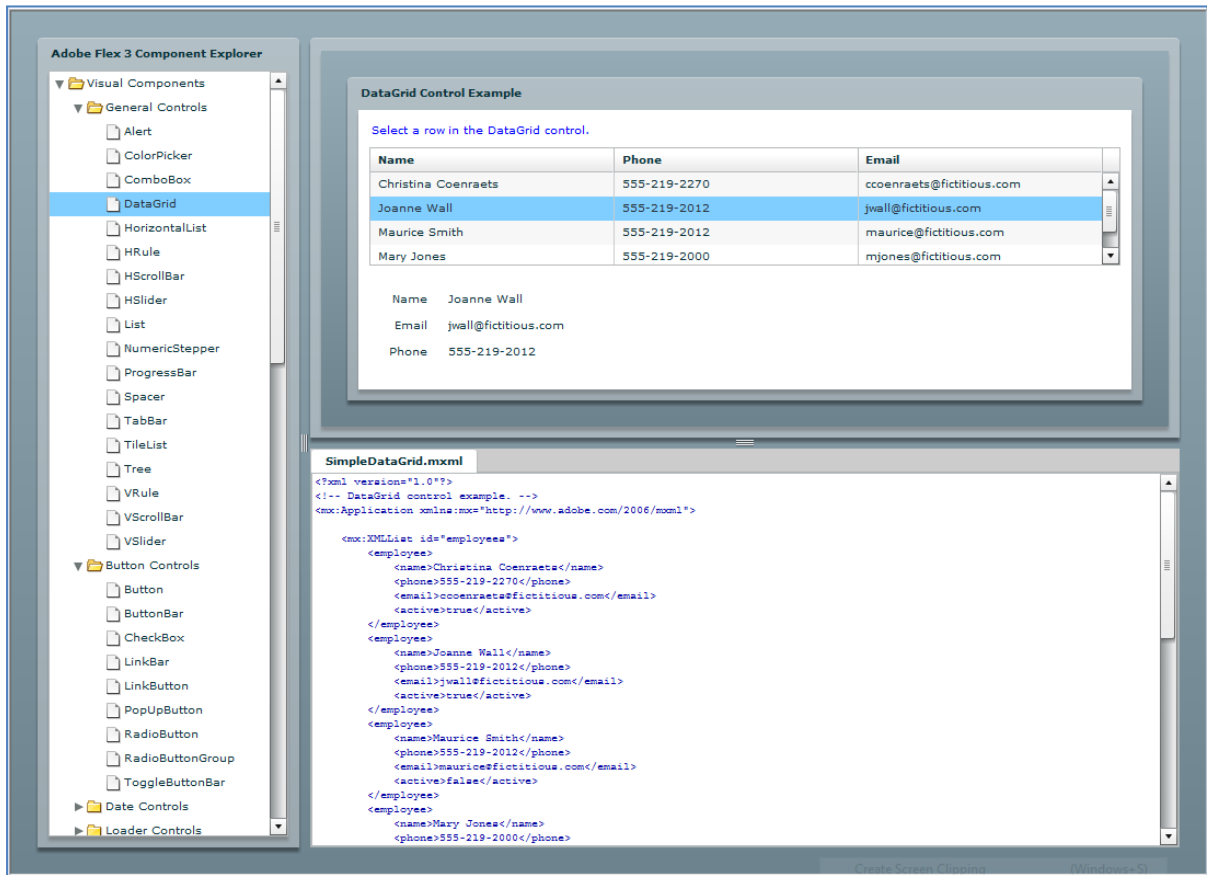


Using Adobe's *TourDeFlex* application to view a sample DataGrid control with associated code.

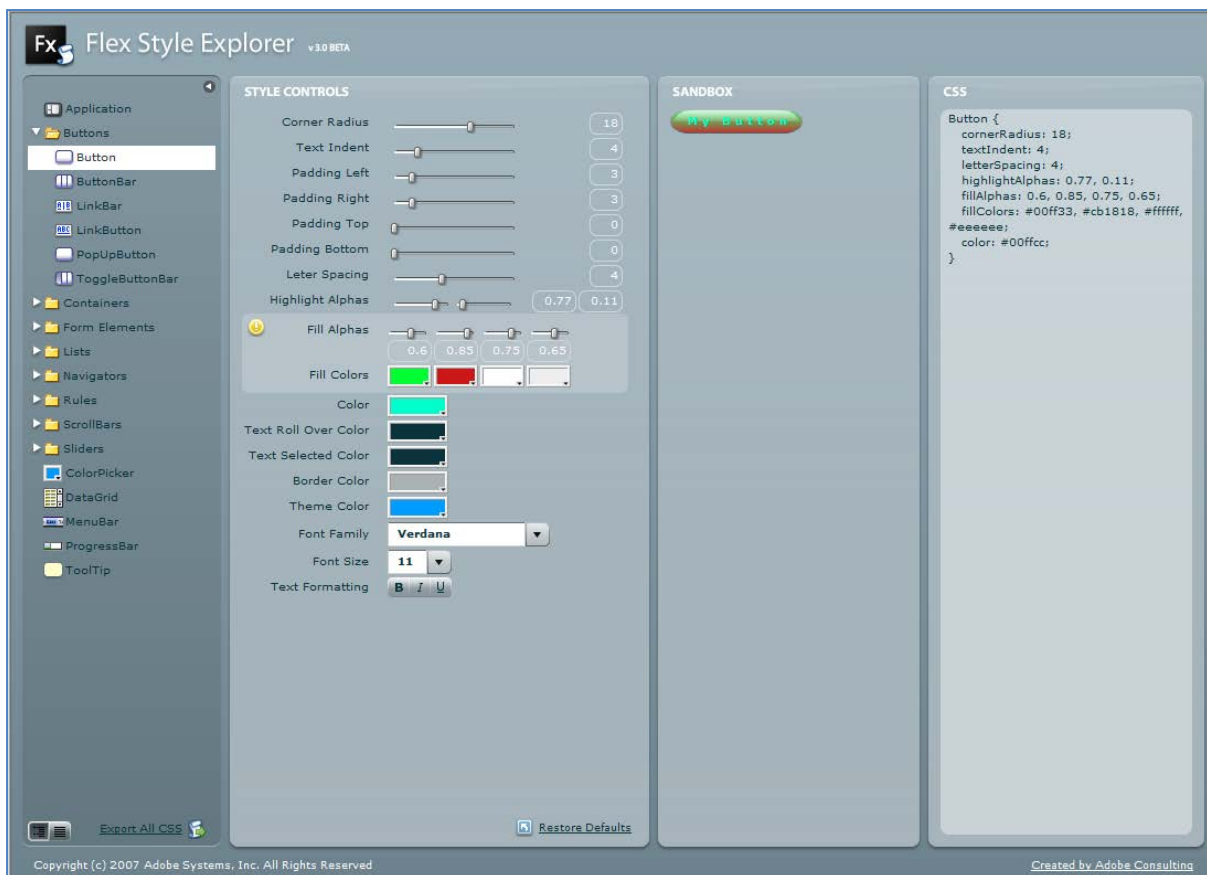
Two other very useful applications provided by Adobe (this time via Flex applications accessed through a browser), are the **Flex Component Explorer** [19] and the **Flex Style Explorer** [20], (screen clips for both of which can be seen overleaf). Flex Component Explorer provides both visual examples of all the key components and animations that are supported in Flex together with sample code that can be used to generate them. Flex Style Explorer goes one step further in allowing a developer to customise the appearance of a component and receive immediate visual feedback on the changes they have instigated. In addition, the CSS code that corresponds to these changes is provided in the **CSS** panel from which it can be copied into the developer's own application.

[19] <http://examples.adobe.com/flex3/componentexplorer/explorer.html> (Last accessed 30/05/2009)

[20] <http://examples.adobe.com/flex3/consulting/styleexplorer/Flex3StyleExplorer.html#> (Last accessed 30/05/2009)

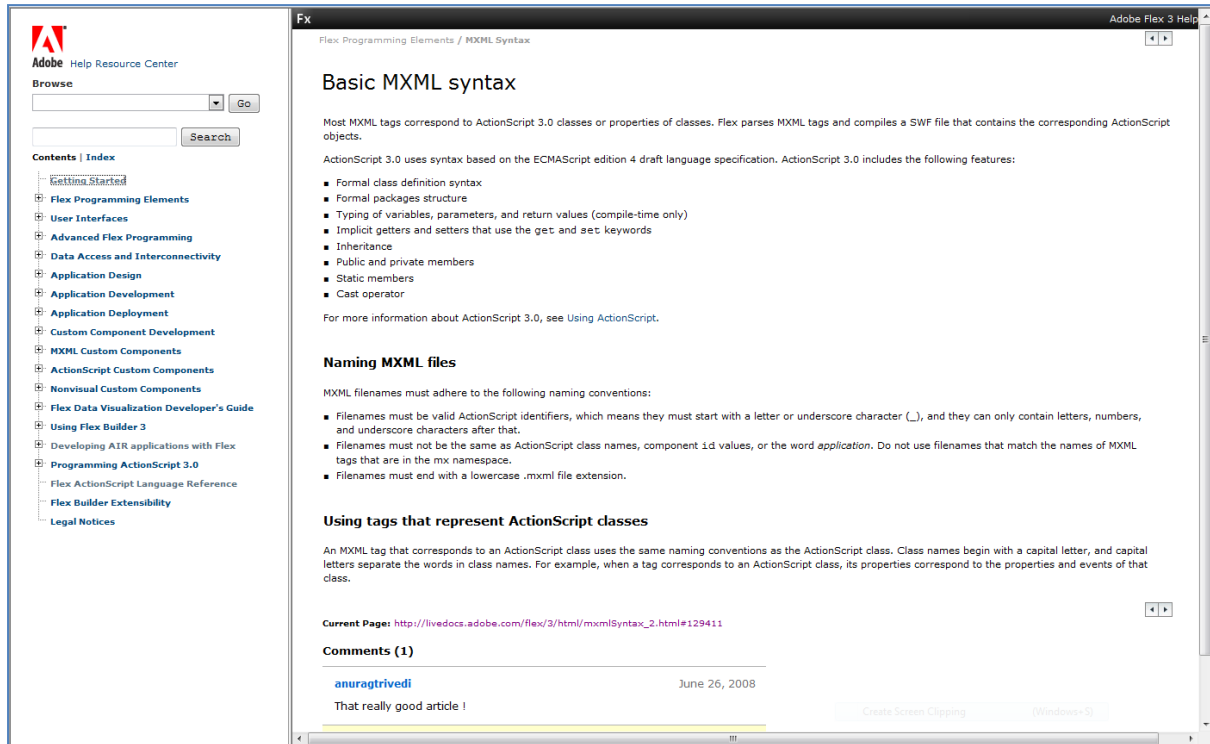


Using Adobe's **Flex Component Explorer** application to view sample MXML code for a DataGrid



Using Adobe's **Flex Style Explorer** application to generate CSS code for a button style

Finally, Adobe also provides an online resource called **Live Docs** [21] (shown in the screen clip below) which contains a comprehensive set of links to many other resources designed to support the development of Flex and AIR applications - see screen clip below. These resources include Language References, MXML Custom Components, information about the Flex Builder 3 development environment, etc.



*View of Adobe's Live Docs application*

[21] <http://livedocs.adobe.com/flex/3/html/index.html> (Last accessed 30/05/2009)

### 9.3 Online Resources for Silverlight

Microsoft's primary online portal for the Silverlight product can be found at [Silverlight.net](http://Silverlight.net) [22], a screen clip of which can be seen below.



*Microsoft's principal portal for Silverlight*

As the name suggests, the **Get Started** page is the place to start when first embarking on the Silverlight learning path. This page contains instructions and links on how to install the packages that will be needed for application development and also contains many links to various blogs, tutorials and online videos which are available to support a developer's learning activities.

A particular criticism however, has to be that the learning trails provided via the **Learn** tab, whilst comprehensive, do appear to be disjointed. Unlike Adobe's Learn Flex in a Week offering, which provides a step-by-step learning trail where the learning undertaken in one video or exercise logically precedes that provided in later videos or exercises, the video tutorials provided by Microsoft are less well defined in their groupings. This is not to say that they are not valuable resources, and the amount of information provided in approximately 75 video tutorials is no doubt extensive, but I remain to be convinced that starting at Video#1 and working one's way methodically through the series returns the same benefits that are provided by Adobe's approach. This trait is particularly noticeable on the page [23] that provides links to various screencasts. In the screen clip below, the screencast that one would expect to be first in any introduction to a technology (i.e.

[22] <http://silverlight.net/> (Last accessed 30/05/2009)

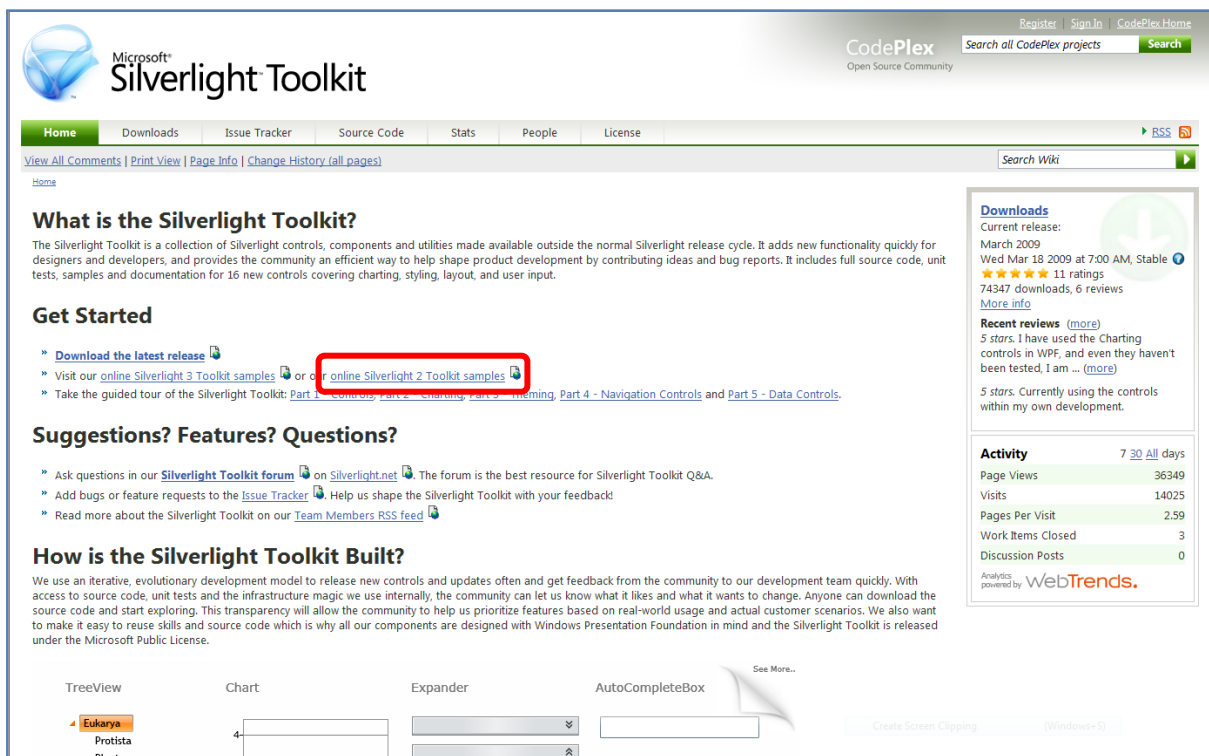
[23] <http://silverlight.net/Learn/videocat.aspx?cat=10> (Last accessed 31/05/2009)

Hello World) comes after screencasts about **Dynamically Loading Assemblies/Code** and **Using Custom Types in XAML** – surely topics aimed at people more experienced in using this particular technology.



*Demonstration of the illogical order of Silverlight screencasts*

In addition to the standard controls provided for Silverlight applications, Microsoft also supplies additional and more complex controls that represent between release work in progress for the Silverlight product. These controls can be downloaded from the **Silverlight Toolkit** site [24].



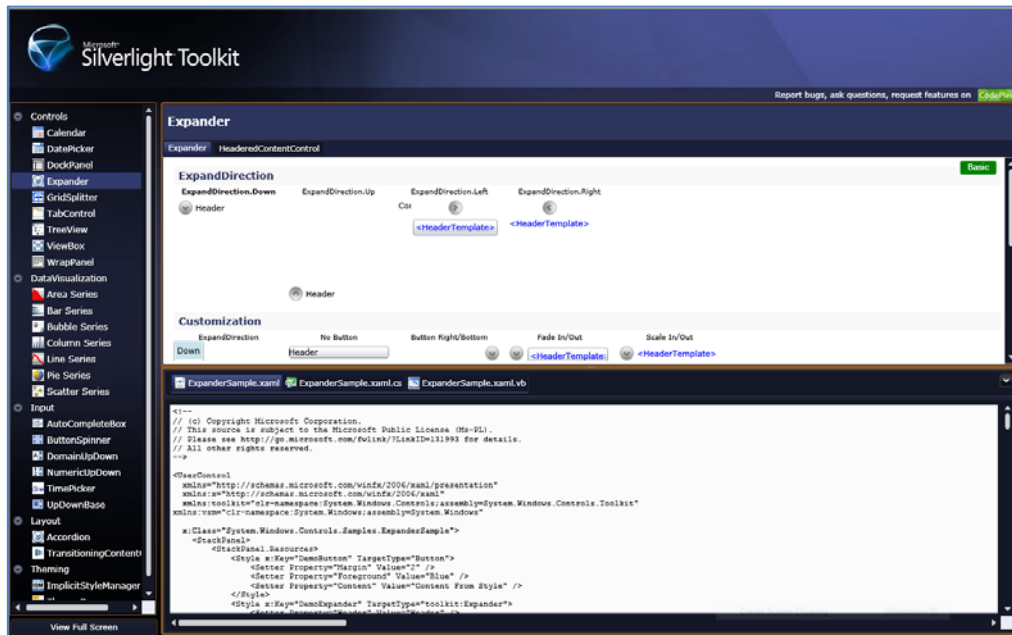
*View of Microsoft's Silverlight Toolkit main page*

This page also contains a link [25] (bounded in **Red**) that enables the developer to view an online facility that demonstrates both the visual capabilities of these advanced controls, together with the supporting XAML, C# and VB.Net code. As can be seen in the screen clip overleaf, although

[24] <http://silverlight.codeplex.com/> (Last accessed 31/05/2009)

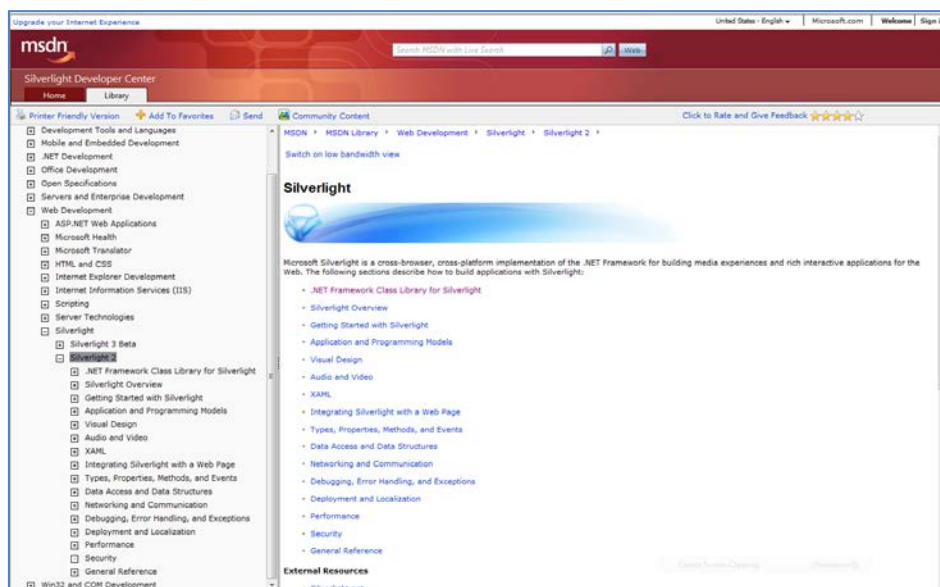
[25] <http://silverlight.net/samples/sl2/toolkitcontrolsamples/run/default.html> (Last accessed 31/05/2009)

this provides similar functionality to that provided by Adobe’s TourDeFlex application, it is to be noted that this facility is only available for the advanced controls that are provided as part of the Silverlight Toolkit, and not for standard controls.



View of Microsoft’s online **Silverlight Toolkit** for advanced Silverlight controls

In addition, Microsoft provides access (both online and through the help pages of Visual Studio) to the **Microsoft Developer Network (MSDN)** [26]. This resource provides extensive support for all technical aspects of Silverlight application development, including support for the Visual Studio IDE and Expression Studio suite of tools.



View of **MSDN** main page for Silverlight

[26] <http://msdn.microsoft.com/en-us/library/default.aspx> (Last accessed 31/05/2009)

## 9.4 Comparison

As befits organisations who are principal players in the online application marketplace, the online resources provided by both Adobe and Microsoft to support their respective product sets are extensive. The use of different delivery styles adopted by both suppliers, e.g. screencasts and videos, exercises, blogs, articles, help pages, etc., allow each individual to choose to receive the required information in a way that supports their particular learning style or is most suitable for their current position in the learning process, i.e. videos and exercises are more likely to be used by individuals who are new to a technology, whereas language references such as those provided via Adobe's Live Docs and Microsoft's MSDN are more likely to be used once an individual has gone beyond the basics. However, of the two competing offerings, I believe that the approach adopted by Adobe is both better structured and more productive. The logical flow of the Learn Flex in a Week programme is far better than Microsoft's haphazard approach, and the facilities provided by the Tour de Flex and Flex Style Explorer facilitate both rapid code generation and better understanding in ways that I believe Microsoft would be well advised to emulate. Comparing Microsoft's MSDN and Adobe's Live Docs is more difficult – MSDN is exceedingly comprehensive but I occasionally found myself going around in a circle before finding the answer to a particular issue, whereas Live Docs is better structured but less comprehensive. At the risk of being dubbed as indecisive, I would suggest that the honours are shared between these two particular facilities.



## 10. Application Development

### 10.1 Introduction

The applications that I have built in support of this MSc project aim to demonstrate some of the key facilities available within the two products for the retrieval, display and animation of images. I started my work in Flex (for no other reason than I had to start with one or the other of the technologies), and to a large extent the application grew from a desire to see what could be done – to understand the facilities described in a publication and to weave them into something that looked reasonable. In some respects, and particularly in organisations where there is a strong separation of responsibilities, this approach is not realistic – I am not an experienced designer of web sites, and to combine both design and development responsibilities in one individual is nearly always going to result in something that does not meet today’s professional standards by one measure or another. Having said that, one of my objectives in embarking on this study was to investigate what could be realistically achieved by one person – particularly one who has traditionally been both designer and developer of applications using Microsoft’s Access RDBMS package – a package that is used by many “one man band” organisations like my own, and one that frequently requires the presence of database design, code development and UI design skills in a single head.

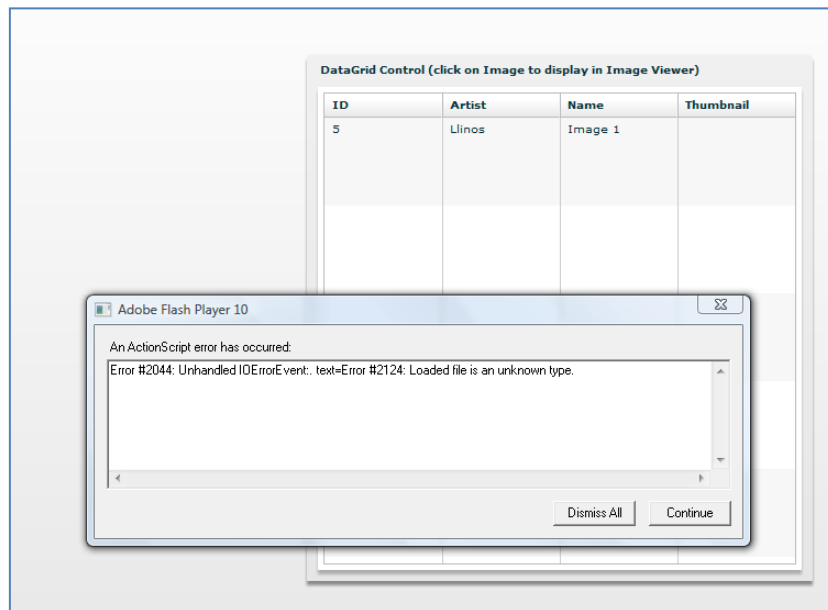
Note: I originally intended that this chapter would contain all of the technology-based information that I feel pertinent to the final design of the two versions of the applications, but given the nature of the technology known as Multi-scale Images, I have decided to present my findings on this as a separate chapter which immediately follows this one (although I do make passing references to the technology within this chapter when appropriate).

### 10.2 Initial Ideas

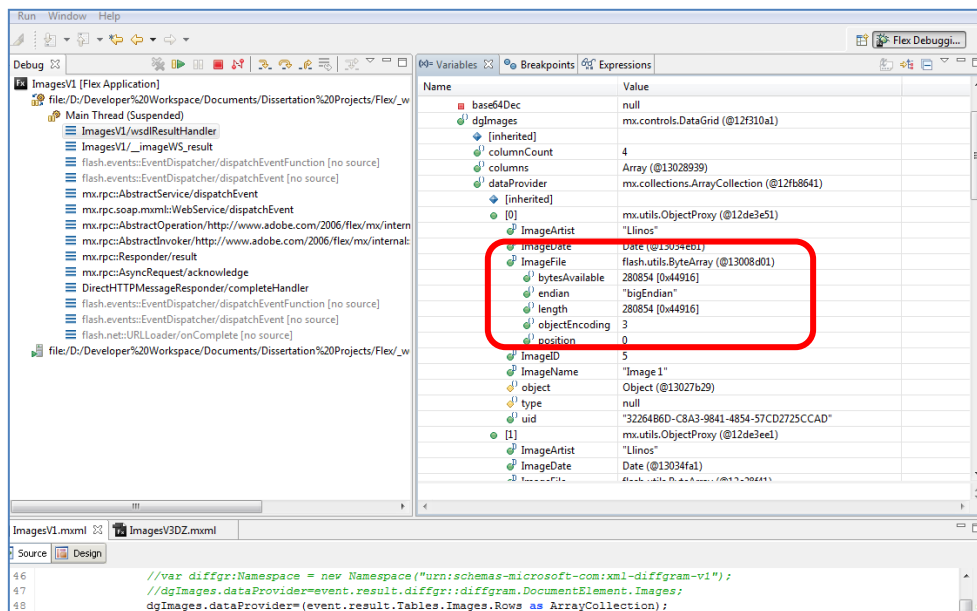
As discussed in Chapter 6, my original intention was to use a back-end database as a repository of the images and text to be consumed by the application, and to provide access from the applications to this database via a SOAP / WSDL-based web service. I had two reasons for initially adopting this approach: firstly, as part of my initial proposal I had envisaged investigating the support provided in each technology to support the consumption of web services – a facility that is growing in importance and popularity in web applications; secondly, I wanted to minimise the amount of time spent developing a repository of data for each application and believed that a reusable back-end would facilitate this.

After scouting around to find the code needed to insert JPG images into a Binary field in SQL Server database, I then constructed the method needed within this back-end to invoke a SQL Select statement and return the results as a collection to the consuming application. All seemed to be going well when I was able to bind the members of this collection to a data grid control in my Flex application, and see the contents of text fields appear. However, the images steadfastly refused to

display and the application threw up numerous errors indicating that the images were represented by an incompatible data type (see screen clip below). In short, I experienced severe difficulties in rendering the image objects which were being transported across the SOAP connection using Base64 Encoding.



Screen clip of error message posted during SOAP transfer from SQL Database to Flex application. Note the successful rendering of text information



Screen clip of Flex debug panel indicating that Image is in correct format (ByteArray) after the SOAP transfer

```
<?xml version="1.0" encoding="utf-8" ?>
- <DataSet xmlns="http://tempuri.com/">
- <xs:schema id="NewDataSet" xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
- <xs:element name="NewDataSet" msdata:IsDataSet="true" msdata:UseCurrentLocale="true">
- <xs:complexType>
- <xs:choice minOccurs="0" maxOccurs="unbounded">
- <xs:element name="Images">
- <xs:complexType>
- <xs:sequence>
- <xs:element name="ImageID" type="xs:int" minOccurs="0" />
- <xs:element name="ImageName" type="xs:string" minOccurs="0" />
- <xs:element name="ImageArtist" type="xs:string" minOccurs="0" />
- <xs:element name="ImageDate" type="xs:dateTime" minOccurs="0" />
- <xs:element name="ImageFile" type="xs:base64Binary" minOccurs="0" />
- </xs:sequence>
- </xs:complexType>
- </xs:element>
- </xs:choice>
- </xs:complexType>
- </xs:element>
- </xs:schema>
- <diffgr:diffgram xmlns:msdata="urn:schemas-microsoft-com:xml-msdata" xmlns:diffgr="urn:schemas-microsoft-com:xml-diffgram-v1">
- <NewDataSet xmlns="">
- <Images diffgr:id="Images1" msdata:rowOrder="0">
- <ImageID>5</ImageID>
- <ImageName>Image 1</ImageName>
- <ImageArtist>Llinos</ImageArtist>
- <ImageDate>2009-01-29T00:00:00+00:00</ImageDate>
- <ImageFile>Qk0WSQAAAAAADYAAAAoAAAAQAEAAcWBAABABgAAAAAAAAAAAAAFWAAEhCAAAAAAAAAAAAAA7f3/m7JGc6zGeM
- </Images>
- <Images diffgr:id="Images2" msdata:rowOrder="1">
- <ImageID>6</ImageID>
- <ImageName>Image 2</ImageName>
- <ImageArtist>Llinos</ImageArtist>
- <ImageDate>2009-01-29T00:00:00+00:00</ImageDate>
- <ImageFile>
- </ImageFile>
- </Images>
- </NewDataSet>
- </diffgr:diffgram>
- </DataSet>
- </>
```

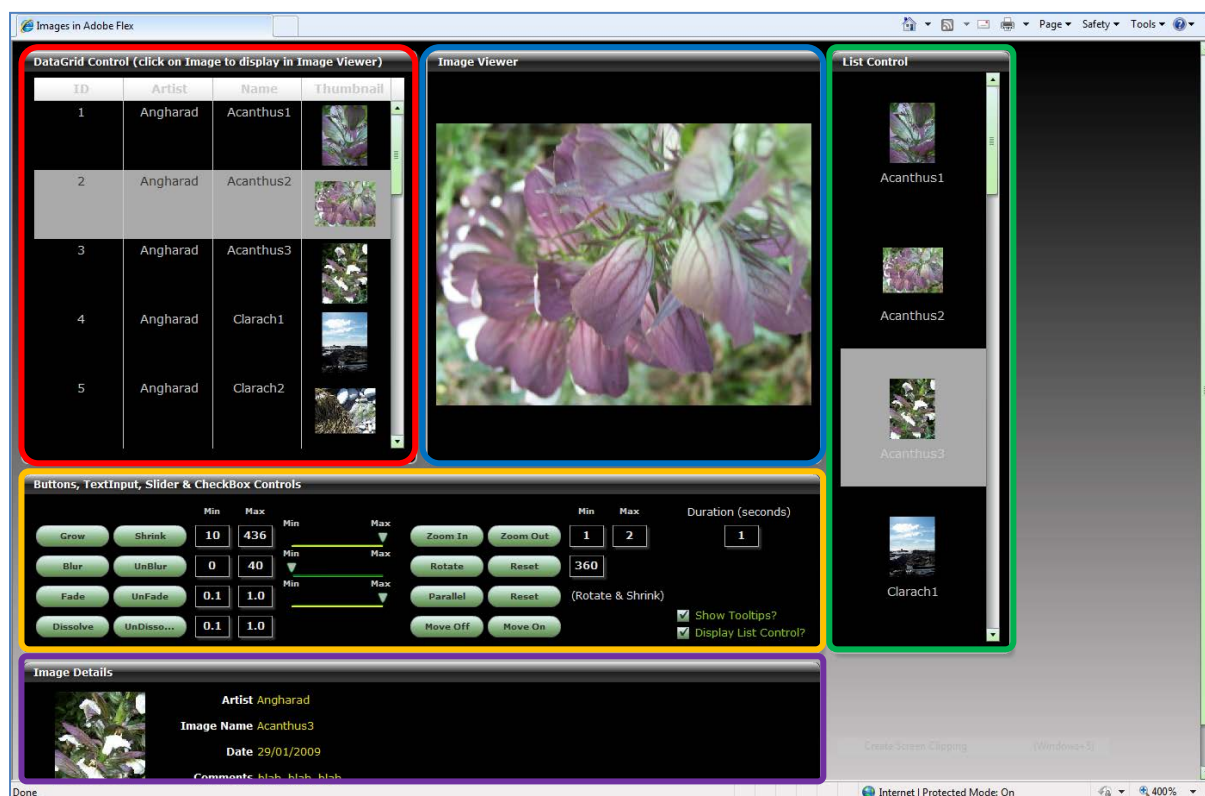
*Sample of the SOAP message showing the declarations of data types (bounded in Red) and encapsulation of actual data (bounded in Green)*

At this point I took the decision to cut my losses and resort to a simpler method for getting the data into the application. I therefore decided to work with a combination of an XML file (which contains elements supporting the fields and data for the text information) and separate folders / directories containing the images – one containing the smaller thumbnails and one containing the larger but more detailed (higher resolution) equivalents. This meant that, at least temporarily, I would be abandoning the web services approach and adopting a more traditional method for data retrieval, but as the overall objective was to investigate as many aspects of each technology as possible it made sense to minimise the time spent chasing solutions to only one aspect of the project, and maximise the time spent on investigating other functions and facilities. In addition, by adopting this approach, I would at least be able to compare the methods supported in each technology for retrieving XML-based data so the idea didn't represent a complete climb down.

Another deliberate choice from the beginning was to build each application as a discrete whole, rather than embed them within a traditional web page. This is probably not the normal approach that would be adopted in the real world, but again offered the benefit of eliminating time spent on using standard technologies to provide the plumbing, and therefore maximising the time available to look at the main facilities provided within each of the technologies.

## 10.3 Flex Application

The screen clip below shows the current Flex release of the application [27]. Note: Both the Flex and the Silverlight applications that have been developed for this project are called **Images**.



*Full view of Images application delivered via Adobe Flex*

### 10.3.1 Visual Structure

The application displayed above comprises many components and animation effects (some of which cannot be demonstrated via static displays), but the overall display is split into 5 key areas:

- A **DataGrid Control** area (bounded in **Red**) displaying a list of all images stored in the XML file together with a thumbnail of each image.
- An **Image Viewer** area (bounded in **Blue**) which is used to display the detailed version (higher resolution) of whichever image has been selected by the user from those presented in the DataGrid area.
- A **List Control** area (bounded in **Green**) which provides a vertical display of the image thumbnails and their name.
- A **Control** area (bounded in **Orange**) that contains a set of buttons, textboxes, sliders and checkboxes that can be used to control the application.
- An **Image Details** area (bounded in **Purple**) that is used to display the data about an image in an alternative form when that image is selected by the user from those presented in the List Control area.

(Note: The display of the List Control and Image Details areas is turned on or off by the use of the **Display List Control?** checkbox provided in the Control area.

[27] <http://www.acanthus-advance.com/Flex/Images.html> (Last accessed 01/06/2009)

### 10.3.2 Application Class

As discussed previously, the visual components of a Flex application are (usually) constructed using the MXML mark-up language and the top level element within the MXML file is the `<mx:Application>` element. This element represents an instance of the *Application* class and is a container object that defines the main application page and in which all other page content is placed. Other page content can include controls and components (standard or user defined), animations and ActionScript code (if defined within a *CDATA* block within an `<mx:Script>` element). As it represents the whole application, there is **only one** `<mx:Application>` element per Flex application. Although other MXML files may be constructed to support custom components that need to be used within the main application page, these files cannot contain an `<mx:Application>` element. The code snippet below shows the declaration of the `<mx:Application>` element in the Images application. In this can be seen various attribute declarations that will apply across the whole application, together with the assignment of functions to both the *initialize* and *creationComplete* events which I will reference again later.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  layout="absolute"
  initialize="initApp()"
  creationComplete="createApp()"
  verticalScrollPolicy="auto"
  horizontalScrollPolicy="auto"
  backgroundGradientAlphas="[1.0, 1.0]"
  backgroundGradientColors="[#000000, #D9D6D6]">
```

### 10.3.3 Containers

MXML supports several different types of containers (visual objects that can contain children objects), including Box, VBox & HBox (Vertical Box & Horizontal Box), Canvas, Tile and Panel. As each of the areas defined in my application was to contain other controls, it was necessary to use one of these container types to define them. I chose to use the panel container component (`<mx:Panel>`) as it also provides a title bar area which I wanted to exploit. The code snippet below shows the declaration of the panel container which is used to define the DataGrid Control area that can be seen in the top left portion of the Images application:

```
<!-- Panel containing DataGrid control-->
<mx:Panel
  id="pnlDataGrid"
  title="DataGrid Control (click on Image to display in Image
Viewer)"
  x="10"
  y="10"
  width="460"
  height="480"
  horizontalAlign="center"
  verticalAlign="middle">
```

Some of the attributes of the panel container include:

- *id* – a unique name that allows the object to be referenced or for data binding
- *x* and *y* - co-ordinates representing the distance right and down in pixels from the top-left corner of the panel's parent container
- *height* and *width* – the dimensions in pixels of the panel
- *title* – the text that is to be displayed in the title area

Once a container object such as a panel has been placed in an application, other components (including other containers) can be placed within it. An example of this approach, using the panel container that is used to define the Image Viewer area, can be seen in the following code snippet:

```
<!-- Panel containing Image control-->
<mx:Panel
  id="pnlDetailImage"
  x="480"
  y="10"
  width="460"
  height="480"
  title="Image Viewer"
  verticalAlign="middle"
  horizontalAlign="center"
  paddingTop="5" tooltip="Use the button and slider controls in the
Panel below to manipulate this image">
  <mx:VBox
    width="436"
    height="436"
    backgroundColor="#000000"
    horizontalScrollPolicy="auto"
    verticalScrollPolicy="auto"
    horizontalAlign="left"
    verticalAlign="top"
    verticalGap="0">
    <mx:Image
      id="imgDetailImage"
      source="imageFiles/{dgImages.selectedItem.imagePath}"
      scaleContent="true"
      maxHeight="{Number(txtResizeMax.text)}"
      maxWidth="{Number(txtResizeMax.text)}"
      rollOverEffect="{effGlowGlowOn}"
      rollOutEffect="{effGlowGlowOff}"
      horizontalAlign="center"
      verticalAlign="middle"
      showBusyCursor="true" themeColor="#B4B4B4" />
    </mx:VBox>
  </mx:Panel>
```

From this we can see that the Image Viewer area comprises of a panel container (*id="pnlDetailImage"*) which then contains a *<mx:VBox>* component which itself contains an *<mx:Image>* control (*id="imgDetailImage"*). There are also additional attributes identified in this code, some of which will be covered later.

### 10.3.4 Application Startup

As mentioned previously MXML is *principally* used to declare the visual elements that comprise an application, but it is sometimes used to declare non-visual functions or components as well. A prime example of this can be seen in the use of MXML to declare an *HTTPService* object which is used to send data from the server to the application at runtime – this is done by invoking the *send()* method of that *HTTPService* object. In this application, the MXML-declared *HTTPService* object is used in combination with ActionScript code to achieve such a transfer in the following way:

Firstly, the *HTTPService* object is declared and as such is instantiated during the creation of the application:

```
<mx:HTTPService
    id="httpImageData"
    url="xmldata/images.xml"
    result="httpResultHandler(event)"
    fault="httpFaultHandler(event)"
    showBusyCursor="true" />
```

The *HTTPService* object is given an identifier through the *id* attribute such that it can be referenced and also a value is assigned to the *url* attribute which provides the relative location on the server where the data is held – in this instance, it is an XML file called **images.xml** which is in the **xmldata** folder. The *HTTPService* object also supports events which can be declared using the *result* and *fault* attributes – the values assigned here are the ActionScript functions that will be called in the event of either a successful or failed send action.

Although an Application object that defines each Flex application has only two native events, it has many inherited events – some of which are applicable during the creation lifecycle of the application, e.g.

- preinitialise
- initialise
- creationComplete
- updateComplete
- applicationComplete

As can be seen in the following code snippet, the Images application uses two of these events, and they are declared using attributes of the *<mx:Application>* element:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute"
    initialize="initApp()"
    creationComplete="createApp()"
    verticalScrollPolicy="auto"
    horizontalScrollPolicy="auto"
    backgroundGradientAlphas="[1.0, 1.0]"
    backgroundGradientColors="[#000000, #D9D6D6]" />
```

When the *creationComplete* event for this application fires, the *createApp()* ActionScript function is run which in turn causes the *send()* method of the *HTTPService* object (to which I assigned the *id* of *httpImageData*) to be invoked - remember that the *HTTPService* object has been instantiated whilst the application is being created and is therefore able accommodate this:

```
private function createApp():void
{
    httpImageData.send();
}
```

Finally, after the *HTTPService* object's *send()* method is invoked, one of its two events (a successful result or unsuccessful fault event) will fire, so I needed to define functions that would handle either eventuality:

```
//Variable used to store data retrieved and make it BINDABLE for use in
the grid controls
[Bindable]
private var arcImageData:ArrayCollection;

//If successful fetch of data, place it in an array collection for later
use
private function httpResultHandler(event:ResultEvent) :void
{
    arcImageData=event.result.images.image;
}

//If unsuccessful fetch of data, display a message
private function httpFaultHandler(event:FaultEvent):void
{
    Alert.show("There has been a problem loading the data", "Data Load
Error");
}
```

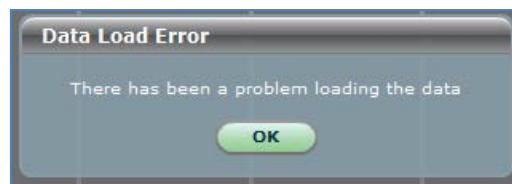
Note: An *ArrayCollection* is similar to an *Array*, but provides methods that allow it to be sorted and is therefore a very useful object to use with a *DataGrid* Control.

Recalling the sample from the xml file presented earlier (and re-produced below) we can see that the assignment of the data retrieved via the result event (*arcImageData=event.result.images.image*) matches the xml file structure:

```
<?xml version="1.0" encoding="utf-8"?>
<images>
  <image>
    <imageID>1</imageID>
    <imageArtist>Angharad</imageArtist>
    <imageName>Acanthus1</imageName>
    <imagePath>Acanthus1.jpg</imagePath>
    <imageDate>29/01/2009</imageDate>
    <imageComments>blah, blah, blah</imageComments>
  </image>
</images>
```



In the event that send action is successful, the *httpResultHandler* function is run and a series of xml *<image>* elements (as defined within the images.xml file) is passed via the *ResultEvent* and placed in an *ArrayCollection* variable called *arclImageData*. As this *ArrayCollection* has been declared with the *[Bindable]* metatag, it can then be bound to other objects / components in the application – as we'll see shortly. However, in the event that there is a problem in completing the send action, e.g. the images.xml file cannot be found, the *httpFaultHandler* function is run. In this simple example, the information held in the *FaultEvent* that is passed as an argument to the *httpFaultHandler* function is not used, but instead a simple error message is displayed using an the ActionScript *Alert* class's *show()* method as demonstrated in the screen clip below:



Graphical display of Flex Alert

Note: As mentioned previously, Flex applications invariably contain both MXML markup and ActionScript code. Separate ActionScript classes can be defined if needed – particularly if code re-use is important, if the application becomes complex / needs defined Business Objects or if a clearer separation of function is required, but it is also possible to use ActionScript code within the MXML, providing it is placed within a *CDATA* block within *<mx:Script>* element tags. Although it is not clear in the above code snippets, the ActionScript code within the Images application has been managed using this second method.

### 10.3.5 List Controls & Item Renderers

No matter how it is stored, displaying data in list format is a fundamental function required of most user interfaces and Flex provides several standard components to achieve this, namely: *DataGrid*, *List*, *HorizontalList*, and *TileList*. (Note: The Professional Edition of the Flex Builder 3 IDE also contains the *Advanced DataGrid* which adds further functionality to the standard *DataGrid* component such as the ability to include other components such as *TreeLists* within it).

Within the Images application I wanted to demonstrate the use of lists and how data selected within a list can be used to trigger the display of other (related) data and so I elected to use a standard *DataGrid* and *List* component to achieve this.

The standard *DataGrid* is a powerful component and provides facilities to allow the user to sort, resize and move columns – all of which are already built into the component without the need to add additional code or declarations (Note: These facilities are on by default, but they can be switched off by respectively assigning a Boolean *false* value to the *sortable*, *resizable* and *draggable*

attributes). I will look further into the data binding issues when working with list components in the next section, but the code snippet below shows the declaration of the `<mx:DataGrid>` element in my implementation.

```
<!-- Panel containing DataGrid control-->
<mx:Panel
.
.
  <mx:DataGrid id="dgImages" dataProvider="{imageData}"
    x="20" y="10" width="430" height="430"
    rowHeight="80" columnWidth="80" styleName="listStyles">
    <mx:columns>
      <mx:DataGridColumn headerText="ID" dataField="imageID"/>
      <mx:DataGridColumn headerText="Artist" dataField="imageArtist"/>
      <mx:DataGridColumn headerText="Name" dataField="imageName"/>
      <mx:DataGridColumn headerText="Thumbnail">
        <mx:itemRenderer>
          <mx:Component>
            <mx:VBox
              horizontalScrollPolicy="off"
              verticalScrollPolicy="off"
              horizontalAlign="center"
              verticalAlign="middle">
              <mx:Image
                source="imagefiles/thumbs/thm{data.imagePath}"
                width="70" height="70"
                horizontalAlign="center" verticalAlign="middle"/>
            </mx:VBox>
          </mx:Component>
        </mx:itemRenderer>
      </mx:DataGridColumn>
    </mx:columns>
  </mx:DataGrid>
</mx:Panel>
```

From this we can see that within the declaration of the `<mx:DataGrid>` element we can assign values for standard dimension attributes such as *width*, *height*, *x* & *y*, etc., and also set dimensions for the width and height of the columns and rows that it will contain. Declaring the first 3 columns that will hold only text information is simply achieved with the inclusion of three `<mx:DataGridColumn>` elements but the declaration of the column to be used to contain the thumbnail images is slightly more complex – and, at least initially for me, quite difficult to get to work.

The first requirement was to place the `<mx:Image>` element within an `<mx:itemRenderer>` / `<mx:Component>` element pair – this provides the functionality for the rendering of the image. In order to ensure that I also had control of where the image would appear within the confines of the column / row space assigned to it, I also had to add an `<mx:VBox>` element which possessed the relevant attributes for horizontal and vertical alignment. By including the `<mx:VBox>`, I had to

construct the *source* attribute of the image to include a *data* property in order that the value contained in the XML *<imagePath>* element would be passed to the source (concatenated with the string value). My initial (and in hindsight very naive) attempts were based on the premise that I could source the data for the column of images in a way that was similar to a text column but these proved to be fruitless and it was only after further experimentation and research that I found the correct method which is encompassed in the above code.

### 10.3.6 Binding

Binding the data that has been transferred from a server to a component or set of components, or using a value held in one component as a parameter for another are key functions within many applications. Flex provides methods of achieving such binding in a fairly simple and intuitive way.

In order to make variables or objects that have been declared in ActionScript as being bindable, (irrespective of whether the code is declared inline or in a separate class), they must be preceded with the *[Bindable]* metatag. Once this is done, then the values held by any particular variable or object can be bound to other application items (controls, components, events, effects, etc) within MXML. To signify to any particular MXML component that it is to use a bound value, the name of the bound object (or one of its attributes) or variable is placed within opening and closing curly braces, i.e. {}.

The first example that is demonstrated within the Images application is binding a DataGrid component to the data that has been transferred from the server. Previously, I demonstrated how the use of the *ResultEvent* of the *HTTPService* object allowed the contents of the images.xml file to be placed in an *ArrayCollection* object called *arcImageData*. In the code snippet below, you can see how this *ArrayCollection* object is assigned as the value for the *dataProvider* attribute of the *<mx:DataGrid>* control (Note the {} notation).

```
<mx:DataGrid
  id="dgImages"
  dataProvider="{arcImageData}"
  x="20" y="10"
  .
  .
  .
  </mx:columns>
</mx:DataGrid>
```

This was able to be achieved because when the *arcImageData ArrayCollection* was declared, it was tagged with the *[Bindable]* metatag (see code snippet overleaf), allowing it to be bound to other components in the application:

**[Bindable]****private var arcImageData:ArrayCollection;**

Now that the `<mx:DataGrid>` component has a data source (in the form of an `ArrayCollection` object), we need to associate individual parts of that source to the relevant columns in the `<mx:DataGrid>`. In the code snippet below we can see that the first three columns that have been defined for the `<mx:DataGrid>` have associations to the relevant fields in the `arcImageData` `ArrayCollection` – these fields map to the respective child elements of the `<image>` element of the XML file that was used to populate `arcImageData`.

```
<mx:DataGrid
  id="dgImages"
  dataProvider="{arcImageData}"
  x="20" y="10" width="430" height="430"
  rowHeight="80" columnWidth="80" styleName="listStyles">
  <mx:columns>
    <mx:DataGridColumn headerText="ID" dataField="imageID" />
    <mx:DataGridColumn headerText="Artist" dataField="imageArtist" />
    <mx:DataGridColumn headerText="Name" dataField="imageName" />
    <mx:DataGridColumn headerText="Thumbnail">
      <mx:itemRenderer>
        <mx:Component>
          <mx:VBox
            horizontalScrollPolicy="off" verticalScrollPolicy="off"
            horizontalAlign="center" verticalAlign="middle">
            <mx:Image
              source="imagefiles/thumbs/thm{data.imagePath}"
              width="70" height="70"
              horizontalAlign="center" verticalAlign="middle" />
          </mx:VBox>
        </mx:Component>
      </mx:itemRenderer>
    </mx:DataGridColumn>
  </mx:columns>
</mx:DataGrid>
</mx:Panel>
```

A second binding method is also demonstrated in the above code snippet. If you note the value assigned to the `source` attribute of the `<mx:Image>` control, you will note that it comprises of some fixed text (`imagefiles/thumbs/thm`) and a binding (`{data.imagePath}`). The `source` attribute defines the URL of the image that needs to be rendered within the `<mx:Image>` control. At this point, the only information available to the application is that which has been transferred from the `images.xml` file stored on the server, i.e. a series of elements containing just text. However, within that XML file, each `<image>` element contains a child element called `<imagePath>` which contains the file name of the image, e.g. `Graduation1.jpg` or `Acanthus1.jpg`. (Note: Remember that the images for the application are stored in two folders on the server – there is an `imagefiles` folder which contains the high resolution images, and within this there is a `thumbs` folder which contains

the low resolution thumbnail copies of each image. The thumbnail images all have the same name as their high resolution counterparts but with **thm** appended to beginning of each file name). Therefore, the binding method for the image source that we can see above effectively picks up the filename of each image from the *imagePath* field in the *arclImageData ArrayCollection*, e.g. *Acanthus1.jpg*, and concatenates it with the fixed text to provide a URL, e.g. *imagefiles/thumbs/thmAcanthus1.jpg*, for each image in the collection.

To summarise then, we have seen two different ways to use binding in this one example:

- Binding a whole `<mx:DataGrid>` control to an *ArrayCollection* object i.e. the value for the *dataProvider* attribute is bound to an *ArrayCollection*
- Binding the data available in the fields in each record / row of that *ArrayCollection* to the both the standard DataGrid columns **and** the value for the *source* attribute (URL) for an `<mx:Image>` control in order that the image it represents can also be downloaded from the server, i.e. part of the *source* attribute is bound to text contained in one of the fields of the *dataProvider*.

Another example of data binding in Flex, and one that is simpler than those described above, is between a variable that holds a value that can later be used as the value to be assigned to an attribute of an MXML component. The (cut down) code snippet below shows the declaration of two private string variables (which as they are declared using inline ActionScript code are tagged with the *[Bindable]* metatag) which are then assigned a value within the *initApp()* function.

```
[Bindable] private var strResizeMin:String;  
[Bindable] private var strResizeMax:String;  
. . .  
private function initApp():void  
{  
    strResizeMin="10";  
    strResizeMax="436";  
    .  
    .  
    .  
}
```

As discussed previously, the Images application contains a panel container which contains a series of button, textbox, slider and checkbox controls that allow the user to amend some of the effects / behaviours. The MXML declarations for some of these can be seen in the code snippet overleaf:

```

.
.
.
<mx:TextInput id="txtResizeMin" text="{strResizeMin}"
    x="190" y="35"
    width="40" height="25" />
<mx:TextInput id="txtResizeMax" text="{strResizeMax}"
    x="240" y="35"
    width="40" height="25" />

<mx:HSlider id="sldResize"
    x="290" y="35" width="130" height="15"
    minimum="{Number(txtResizeMin.text)}"
    maximum="{Number(txtResizeMax.text)}"
    labels=["Min', 'Max']"
    liveDragging="true"
    change="sldResize_Change(event)" />
.
.
.

```

What can initially be seen here, is that the values for the *text* attributes of the two `<mx:TextInput>` controls are bound to the values that were assigned to the private string variables declared previously. Secondly, it can be seen that the values for the *minimum* and *maximum* attributes of the `<mx:HSlider>` control are then bound to the values assigned to the *text* attributes of these two `<mx:TextInput>` controls (with the string values cast as numbers). However, it can also be seen that we have not declared these *text* attributes or the `<mx:TextInput>` controls to which they belong to be *[Bindable]*. This is an important point – unlike variables or objects that are defined and assigned values in ActionScript, binding between MXML components occurs automatically. Providing the component that you want to bind to has been explicitly assigned an identifier, its intrinsic value can be bound to the attribute of another MXML component.

Binding does not just work with alphanumeric values. In the code snippet overleaf we can see that panel containers have an attribute called *visible* that accepts Boolean values. In this case, the value for this panel's *visible* attribute is provided by the value of a checkbox's *selected* property – which is also Boolean. When the checkbox with the *id* of *chkList* is checked (Boolean *True*), the panel will become visible and it will become invisible when the checkbox is unchecked (Boolean *False*). What is also of interest here is that this happens without the declaration of any event handler code – there is an automatic and real-time change to the value assigned to the panel's *visible* attribute whenever the checkbox value is changed:

```

<mx:Panel id="pnlImageInformation"
  title="Image Details"
  x="10"
  y="718"
  width="930"
  height="190"
  visible="{chkList.selected}"
  showEffect="{effWipeRightWipeIn}"
  hideEffect="{effWipeRightWipeOut}"
  layout="absolute" >

```

I will be covering animations in greater detail later, but for the moment you will also be able to see in the code snippets above and below how binding can be used to control animations. The panel component has *hideEffect* and *showEffect* attributes whose value can be bound to an effect declared elsewhere (see code snippet below). Combining the *{chkList.selected}* binding together with the bindings assigned to the *hideEffect* and *showEffect* attributes, I have provided a facility that when the user wishes to hide (or display) the Image Details panel, the panel does not simply switch between visible and invisible, it does so using an animation. For example, by checking the *chkList* checkbox the panel's *visible* attribute is set a value true. Because it has been declared, Flex then initiates the effect that has been assigned (and bound) to the panel's *showEffect* attribute - the effect identified as *effWipeWipeIn* (which is an *<mx:WipeRight>* effect).

```

<mx:WipeRight id="effWipeRightWipeIn"
  duration="{Number(txtDuration.text)*1000}"/>

```

Finally, (and again thanks to binding) it will perform this effect over a period of time that is currently held as the value in the *txtDuration* textbox – if the user changes this value, it will either lengthen or shorten the time that this effect takes to complete. (Note that I have also provided a corresponding binding connection between the panel's *hideEffect* attribute and an effect called *effWipeRightWipeOut* which performs the opposite action).

As a final example of binding, the code snippet below shows how binding can also be used to create a relationship with an entire component, rather than just one particular attribute. Again using an effect to demonstrate this, we can see that the value for the *target* attribute of an *<mx:Zoom>* effect is defined as *imgDetailImage* – this is the *id* of the whole *<mx:Image>* control (rather than just one of its attributes) that is used to render the high resolution version of whichever image has been selected in the DataGrid Control area of the application.

```

<mx:Zoom id="effZoomZoomOut"
  target="{imgDetailImage}"
  zoomHeightTo="{Number(txtZoomMin.text)}"
  zoomWidthTo="{Number(txtZoomMin.text)}"
  duration="{Number(txtDuration.text)*1000}"/>

```

### 10.3.7 Animations

The heritage of Flex is through Flash, and Flash has for many years provided facilities to generate animations in the UI. Flash developers have been accustomed to using a “timeline” facility to generate such animations on a frame-by-frame basis, but “...one of Macromedia’s most important motivations in creating Flex was to free developers with a coding background from having to work with the timeline at all” [28]. As a result of this development, animating parts of a Flex application is a relatively simple activity, but before demonstrating the animations that have been utilised in the Images application I will cover some of the terminologies that are used.

In Flex, an **effect** is an ActionScript class that manages the changes (primarily visual, but also audible) to a part of the application with respect to time. Effects apply to both **behaviours** - which are pre-built animations that can be applied to components (both inbuilt and user-defined) and **transitions** - which enable you to apply changes to and between view states.

Flex animations also rely on the concept of **triggers**. A trigger is not an event; rather it is a special member of a component that can have an effect assigned to it – i.e. a trigger causes an instance of an effect class to *play*. An event is something that is responded to with an event handler, e.g. a button’s *click* event can be associated with developer defined ActionScript code – this code may or may not invoke the playing of an effect. (Note: As the Images application does not exploit view states, and therefore does not invoke transitions, the rest of this section will be slanted towards behaviours, i.e. a combination of the effects and triggers that are applied to cause animation of components).

Effects that are provided as standard in Flex include:

- Fade (a change in the opacity of a component)
- Dissolve (similar to fade, but uses a colour overlay over the component)
- Resize (changes the size of the component using pixels as a unit of measure)
- Move (changes the location of the component)
- Rotate (rotates a component about a fixed point)
- Zoom (similar to resize but uses relative values)
- Wipe (hides or displays a component starting at top, bottom, left or right)
- Glow (a change in the border of a component to provide / remove highlighting)
- Blur (a change in the “focus” of a component – like moving the focus ring on a camera or telescope)

In addition, Flex provides the ability to achieve more complex effects with the `<mx:Sequence>` and `<mx:Parallel>` tags. As their names imply, these respectively allow two or more effects to be invoked serially or simultaneously.

---

[28] Adobe Flex 3 Bible – David Gassner. ISBN(13) 978-0-470-28764-4. Wiley Publishing - 2008. p401.



We have already seen (in the above section on Binding), how to create an animation by implementing a behaviour using an effect / trigger combination. In that case, the clicking of a checkbox caused the flip of a Boolean value that controlled the *visible* attribute of another component which in turn invoked either a *showEffect* or *hideEffect* trigger which in turn caused the playing of either of two `<mx:WipeRight>` effects . (Note: In the current incarnation of the Images application, the two effects (*effWipeWipeOut* and *effWipeWipeIn*) do exactly the same thing, but by declaring two separate effects with different *id* attributes, I have the flexibility of using different effects in the future).

Another example in the Images application of this approach can be found in the panel that contains the detailed image. As can be seen in the code snippet below, the `<mx:Image>` component has been declared with the *rollOutEffect* and *rollOverEffect* triggers (which respectively respond to the *mouseout* and *mouseover* events), and have been assigned (bound) to the effects identified as *effGlowGlowOn* and *effGlowGlowOff*:

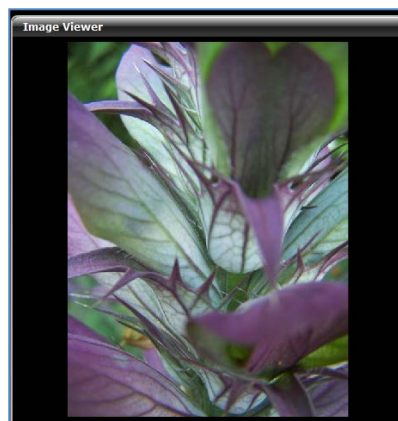
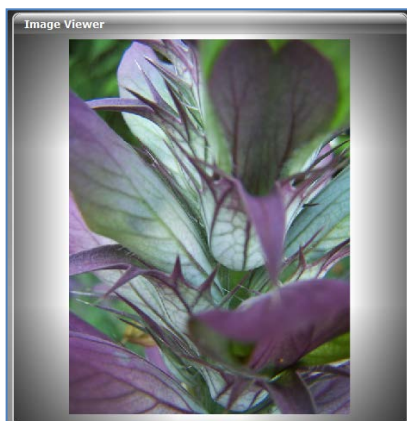
```
<mx:Image
  id="imgDetailImage"
  source="imageFiles/{dgImages.selectedItem.imagePath}"
  scaleContent="true" maxHeight="{Number(txtResizeMax.text)}"
  maxWidth="{Number(txtResizeMax.text)}"
  rollOverEffect="{effGlowGlowOn}"
  rollOutEffect="{effGlowGlowOff}"
  horizontalAlign="center" verticalAlign="middle"
  showBusyCursor="true" themeColor="#B4B4B4" />
```

Even though the above triggers have had effects assigned to them, they will not invoke the animation in themselves – the actual animation that is desired is defined by declaring an instance of the relevant effect class and assigning the necessary attributes to it. The code snippet below shows the declaration of both the *effGlowGlowOn* and *effGlowGlowOff* effects – both of which are instances of the Glow effect class. Amongst other attributes, the Glow effect class includes attributes that define the extent of the start and end blurs (in both the X & Y planes) together with the desired opacity – the *alphaFrom* & *alphaTo* attributes define this, with the value of 0 being totally transparent and 1 being totally opaque.

```
<mx:Glow id="effGlowGlowOn"
  color="ffffff"
  blurXFrom="0" blurXTo="300"
  blurYFrom="0" blurYTo="300"
  alphaFrom="0.1" alphaTo="1.0" />

<mx:Glow id="effGlowGlowOff"
  color="ffffff"
  blurXFrom="300" blurXTo="0"
  blurYFrom="300" blurYTo="0"
  alphaFrom="1.0" alphaTo="0.1" />
```

The visual outcome of these effects can be seen in the two screen clips below:



Demonstration of the visual outcome of the **effGlowGlowOn** (left) and **effGlowGlowOff** effects in the Flex version of the Images application

This method for using triggers and effects allows me to achieve the desired behaviour within a single visual component, i.e. the `<mx:Image>` control contains the triggers that cause the invocation of the `<mx:Glow>` effects on the self same `<mx:Image>` component. However, in some parts of the Images application, I wanted get Flex to implement an animation after the user clicked on a button, i.e. the animation would be instigated on one component after the firing of an event that was associated with another component, rather than in response to the invocation of a trigger.

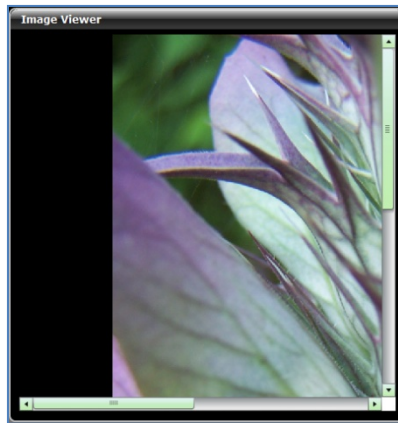
Flex allows this approach to animation to be achieved in one of two ways. Firstly, by using a combination of the relevant effect class's `play()` method together with its `target` attribute. An example of this approach can be seen in the code snippets below:

```
<mx:Button id="btnZoomIn" label="Zoom In" click="effZoomZoomIn.play()"
  x="440" y="35" width="85" height="25" />
<mx:Button id="btnZoomOut" label="Zoom Out" click="effZoomZoomOut.play()"
  x="530" y="35" width="85" height="25" />
```

Above, we have the MXML declaration of two button components, and we can see that the `play()` method of an instance of the `<mx:Zoom>` effect class has been assigned to each. Below are the declarations of the two `<mx:Zoom>` effects, and in each we can see the use of the `target` attribute to bind the respective effects to the `<mx:Image>` component called `imgDetailImage`.

```
<mx:Zoom id="effZoomZoomIn"
  target="{imgDetailImage}"
  zoomHeightTo="{Number(txtZoomMax.text)}"
  zoomWidthTo="{Number(txtZoomMax.text)}"
  duration="{Number(txtDuration.text)*1000}" />
<mx:Zoom id="effZoomZoomOut"
  target="{imgDetailImage}"
  zoomHeightTo="{Number(txtZoomMin.text)}"
  zoomWidthTo="{Number(txtZoomMin.text)}"
  duration="{Number(txtDuration.text)*1000}" />
```

The end point of the *effZoomZoomIn* effect that is achieved using the above approach can be seen below:



Sample image display at endpoint of the *effZoomZoomIn* effect.

The second approach that can be exploited to use an event of one component to invoke an animation on another is to use a combination of MXML and ActionScript code. This approach is particularly useful if you want to perform some other activity in addition to invoking the effect, e.g. to check for a condition or to perform some function on another component or set of components. Within the Images application, I have used this approach to both instigate an animation on the *imgDetailImage* `<mx:Image>` component while at the same time amending the properties of the slider components. The following code snippets demonstrate this:

Firstly, I have defined an `<mx:Button>` component which declares a *click* event to which is assigned an ActionScript function:

```
<mx:Button id="btnFade" label="Fade" click="btnFade_Click()"
  x="5" y="105" width="85" height="25"/>
```

The ActionScript function is then defined as follows:

```
//Respond to Button press to FADE
private function btnFade_Click():void
{
    if (sldFade.value<=Number(txtFadeMin.text))
    {
        Alert.show("This image is already at the greatest level of
fade as defined in the MIN textbox","Image Information");
    } else {
        effFadeFade.play();
        sldFade.value=Number(txtFadeMin.text);
    }
}
```

Here we can see that the function first checks to see if the *imgDetailImage* component is already at its greatest level of fade by checking to see whether the minimum value for the associated slider component (*sldFade*) is equal to or less than the value in the *txtFadeMin* textbox that is associated with the “Fade” button. If it is, then the application posts an *Alert* to the user, otherwise it invokes the *play()* method of an effect object called *effFadeFade*. It also sets the value for the slider component (*sldFade*) to be equal to the value contained in the *txtFadeMin* textbox.

The following code snippet shows the declaration of the *effFadeFade* object, which is an instance of an `<mx:Fade>` effect class:

```
<mx:Fade id="effFadeFade"
  target="{imgDetailImage}"
  alphaFrom="{sldFade.value}" alphaTo="{Number(txtFadeMin.text)}"
  duration="{Number(txtDuration.text)*1000}"/>
```

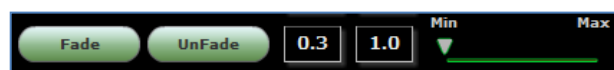
The *effFadeFade* instance of the `<mx:Fade>` effect class fades the image starting from the value that is represented by the current position of its associated slider control down to the value held in the *txtFadeMin* textbox. It does this over a duration (in seconds) that is represented in the *txtDuration* textbox. The end point of the fade can be seen in the following screen clip (left) together with the before and after views of the associated slider component (right):



Slider before “Fade” button pressed



Slider after “Fade” button pressed



It is also possible to invoke an ActionScript function from an event that is declared within an instance of an effect class rather than from within a component (as we have seen previously). In the code snippets overleaf (the first of which declares an instance of the `<mx:Dissolve>` effect class), we can see the declaration of an *endEffect* event. As the name suggests, this event fires after the completion of the effect and in this case it invokes another ActionScript function (second snippet) which sets the *visible* attribute of *imgDetailImage* to Boolean *false* – causing it to disappear.

```

<mx:Dissolve id="effDissolveDissolve"
  target="{imgDetailImage}" targetArea="{rrImageArea}"
  alphaFrom="{Number(txtDissolveMax.text)}"
  alphaTo="{Number(txtDissolveMin.text)}"
  color="#ff0000" duration="{Number(txtDuration.text)*1000}"
  effectEnd="invisibleImage()" />

```

```

//function to make imgDetailImage disappear at end of the dissolve effect
private function invisibleImage():void
{
    imgDetailImage.visible=false;
}

```

Finally, as mentioned at the beginning of this section, Flex provides the facility to create more complex animations through the use of the `<mx:Sequence>` and `<mx:Parallel>` effect classes. An example of an implementation of this approach within the Images application can be seen in the code snippet below, which simultaneously instigates a Resize and Rotate effect on a single target:

```

<mx:Parallel id="effParallelParallel">
  <mx:Rotate
    target="{imgDetailImage}"
    angleFrom="0"
    angleTo="{Number(txtRotateDegrees.text)}"
    originX="{Number(txtRotateOrigin.text)}"
    originY="{Number(txtRotateOrigin.text)}"
    duration="{Number(txtDuration.text)*1000}" />
  <mx:Resize
    target="{imgDetailImage}"
    widthTo="{Number(txtResizeMin.text)}"
    heightTo="{Number(txtResizeMin.text)}"
    duration="{Number(txtDuration.text)*1000}" />
</mx:Parallel>

```

An example of the `<mx:Sequence>` effect can also be seen within the application. In this case (demonstrated in the code snippet below), the declarations of the relevant effects are made within the `<mx:itemRenderer>` component that they apply to and cause the instigation of an `<mx:Zoom>` effect followed by an `<mx:Glow>` effect:

```
<mx:itemRenderer>
  <mx:Component>
    <mx:VBox>
      horizontalScrollPolicy="off"
      verticalScrollPolicy="off"
      horizontalAlign="center"
      verticalAlign="middle"
      paddingTop="10"
      paddingRight="0">

      <mx:Sequence
        id="effSequenceMouseRollOver"
        duration="200">
        <mx:Zoom
          originX="{(imgListImage.width)/2}"
          originY="{(imgListImage.height)/2}"
          zoomHeightTo="2" zoomWidthTo="2" />
        <mx:Glow color="#ffffff"
          blurXTo="200" blurYTo="200"
          alphaTo="1" />
      </mx:Sequence>

      <mx:Sequence
        id="effSequenceMouseRollOut"
        duration="200">
        <mx:Zoom
          originX="{(imgListImage.width)/2}"
          originY="{(imgListImage.height)/2}"
          zoomHeightTo="1" zoomWidthTo="1" />
        <mx:Glow color="#ffffff"
          blurXTo="0" blurYTo="0"
          alphaTo="0" />
      </mx:Sequence>

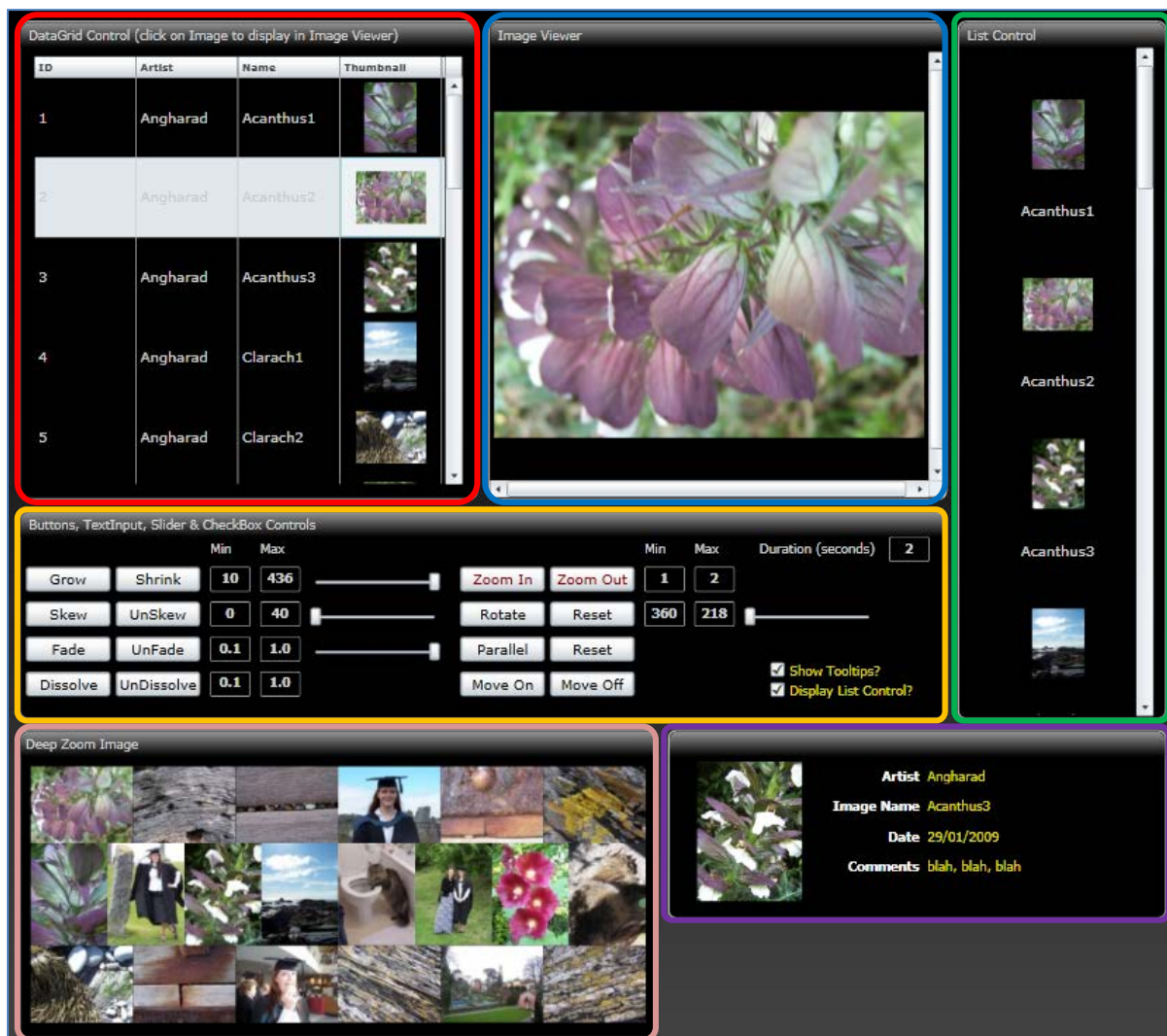
      <mx:Image id="imgListImage"
        source="imagefiles/thumbs/thm{data.imagePath}"
        width="70" height="70"
        rollOverEffect="{effSequenceMouseRollOver}"
        rollOutEffect="{effSequenceMouseRollOut}"
        horizontalAlign="center" verticalAlign="middle" />

      <mx:Label id="lblImageName"
        text="{data.imageName}" color="#cccccc" />
    </mx:VBox>
  </mx:Component>
</mx:itemRenderer>
```

## 10.4 Silverlight Application

### 10.4.1 Visual Structure

The screen clip below shows the current Silverlight release of the Images application. [29]



Full view of the *Images* application delivered via Microsoft Silverlight

The visual structure of the Images application in Silverlight is, by design, almost exactly the same as the one developed using Flex – the overall idea being of course to compare the deliverables of a range of the facilities within the two technologies. The following list provides a reminder of 5 of the core parts of the application which have been designed to replicate the application as it appears in the Flex version:

- A **DataGrid Control** area (bounded in **Red**) displaying a list of all images stored in the XML file together with a thumbnail of each image
- An **Image Viewer** area (bounded in **Blue**) which displays the detailed version (higher resolution) of whichever image has been selected by the user from those presented in the DataGrid

[29] <http://www.acanthus-advance.com/Silverlight/Images.html> (Last accessed 18/09/09)

- A **List Control** area (bounded in **Green**) which provides a vertical display of the image thumbnails and their name
- A **Control** area (bounded in **Orange**) that contains a set of buttons, textboxes, sliders and checkboxes which can be used to control the application
- An **Image Details** area (bounded in **Purple**) that is used to display the data about an image in an alternative form when that image is selected by the user from those presented in the List Control area.

(Note: The display of the List Control and Image Details areas is turned on or off by the use of the **Display List Control?** checkbox provided in the Control area.

In addition, the Silverlight version of the application contains a 6<sup>th</sup> component area (bounded in **Dark Pink**) which provides a display using a Multi-scale Image. This type of component (also known as a **Deep Zoom Image**) is available only in Silverlight - at least as a vendor-supported “out of the box” component. I’ll describe the use of multi-scale image technology in Chapter 11 of this report.

#### **10.4.2 Application Class**

As in a Flex application, the top level class within a Silverlight application is the application class and also, as in Flex, there can be only one instance of the application class defined - however there are some key differences between the implementations of Flex and Silverlight that are worth noting. Firstly, Silverlight applications exploit the .Net framework implementation of partial classes which allow for a combination of both XAML markup and managed code (C# in this project) to be used within one class. In the case of the application class for a Silverlight project (and assuming that Visual Studio or Expression Blend are used to create the project) two files are generated - *App.xaml* and *App.xaml.cs*.

Secondly, the *App.xaml* file is not used to contain / define any visual elements for the application. The whole application class is used to manage the **global** aspects and attributes of the Silverlight application, e.g. start-up, shut down and any resources that may be defined as being application wide rather than specific to any individual element. Visual elements (and to a certain degree any managed code used to support them) are defined within a class (or classes) which is separate from the main *Application* class instance. When using Visual Studio or Expression Blend for creating a Silverlight application, a separate *Page* class instance (comprising the two partial class files *Page.xaml* and *Page.xaml.cs*) is automatically generated to support this approach. The managed code partial class file (*App.xaml.cs*) of the application references this class to create a page object at runtime as can be seen in the code snippet overleaf:



```

.
.
namespace Images
{
    public partial class App : Application
    {
        public App()
        {
            this.Startup += this.Application_Startup;
            this.Exit += this.Application_Exit;
            this.UnhandledException +=
this.Application_UnhandledException;
            InitializeComponent();
        }

        private void Application_Startup(object sender, StartupEventArgs
e)
        {
            this.RootVisual = new Page();
        }

        private void Application_Exit(object sender, EventArgs e)
        {
        }
    }
}
.
.
.

```

An example of the XAML elements of the *Page* class (*Page.xaml*) is given in the code snippet below. Within this, we can see that the base or root element of the page is the `<UserControl>` element within which are defined the various namespaces needed to support the application. Any resources such as storyboards and styles (both of which I will cover later in this report) that can be used across the application are defined within the `<UserControl.Resources>` element. The core visual elements for the application are then defined as children to a top level container element with the name of *LayoutRoot*. Note: This is always set as a `<Grid>` element when using Visual Studio or Expression Blend to create a Silverlight project.

```

<UserControl x:Class="Images.Page"
    xmlns:data="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Controls.Data"
.
.
.
    <UserControl.Resources>
.    General resources to support visual elements such as storyboards,
        styles, etc. can go here
    </UserControl.Resources>
    <Grid x:Name="LayoutRoot">
.        Visual Element Definitions Go Here
    </Grid>
</UserControl>

```

### 10.4.3 Containers

As with Flex, Silverlight provides several visual components that act as containers for other components, e.g. Grid, Canvas, Stackpanel, Scrollviewer and Border. When building the application in Flex, I used that technology's `<mx:Panel/>` component as the top level component for each of the discrete application areas as it contains an attribute that allowed me to define a title. Unfortunately, none of the container components supplied in Silverlight provide comparable functionality and so I used a combination of Border & Canvas components to achieve the same effect as seen below:

```
<Border x:Name="brdDetailImage" Height="480" Width="460"
      Margin="480,10,0,0" Style="{StaticResource stlBorderStd}">
  <Canvas x:Name="cnvDetailImage" Height="476" Width="456"
        Margin="0,0,0,0" VerticalAlignment="Top">
    <Border x:Name="brdDetailImageHeader" Height="25" Width="456"
          CornerRadius="8,8,0,0">
      <Border.Background>
        <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0"
                          SpreadMethod="Pad">
          <GradientStop Color="#FF000000" Offset="1"/>
          <GradientStop Color="#FF999999" Offset="0"/>
        </LinearGradientBrush>
      </Border.Background>
      <TextBlock x:Name="tbDetailImageHeader" Text="Image Viewer"
                Width="446" Margin="5,5,0,0" Height="20"
                FontFamily="Verdana" FontSize="12"
                Foreground="LightGray"/>
    </Border>
    <Image x:Name="imgDetailImage" Margin="0,0,0,0"
          VerticalAlignment="Center"
          HorizontalAlignment="Center" Width="436" Height="436">
      <Image.RenderTransform>
        <TransformGroup>
          <ScaleTransform x:Name="trfScale"/>
          <SkewTransform x:Name="trfSkew"/>
          <RotateTransform x:Name="trfRotate"/>
          <TranslateTransform x:Name="trfTranslate"/>
        </TransformGroup>
      </Image.RenderTransform>
    </Image>
  </Canvas>
</Border>
```

It can be seen from the above snippet that several lines of XAML code are required to define just the title area – a `<Border>` and a `<TextBlock>` component are needed together with the declaration of several attributes to provide the required styling. When compared to the Flex approach (partially detailed in the code snippet overleaf) this would seem at first viewing to be excessive:

```
<mx:Panel
    id="pnlDetailImage" x="480" y="10" width="460" height="480"
    title="Image Viewer"
    verticalAlign="middle"
    horizontalAlign="center"
    .
    .
```

However, it should be noted that Silverlight (like Flex) contains the ability for the construction of custom components and supports CSS-like component styling. Therefore, although I have not demonstrated it here, it is entirely feasible that in a more complex application where many similar components are required, this amount of code (within the definition of the individual component) could be significantly reduced. Overall, the use of a combined set of `<Border>` + `<Canvas>` + `<Border>` components is slightly more clumsy (in this scenario) than a single `<mx:Panel>` component in Flex, but it could be argued that this in itself provides greater options to the application creator. Likewise, the number of parameters required to define the visual attributes for the `<Border.Background>` would appear to make this seemingly simple task (i.e. defining a title area) to be unnecessarily time consuming, but as most, if not all, of the code here is automatically generated during the design process by the Expression Blend design programme, productivity is not hindered greatly.

#### **10.4.4 Application Startup**

The Silverlight version of the Images application uses exactly the same XML data file and image files as those consumed by the Flex application, however the method used in the Silverlight version to transfer that information from the server to the application is slightly different. Silverlight provides many facilities for data transfer, including sockets (using the `System.Net.Sockets` library), `HTTPWebRequest` / `HTTPWebResponse` classes (`System.Net` library) and support for RSS and Atom feeds (`System.ServiceModel.Syndication` library). Silverlight also provides a `WebClient` class which shares functionality with the `HTTPWebRequest` class but provides a simpler implementation. In addition, various methods are provided for handling data from a variety of sources including XML and SQL databases.

The Silverlight version of the Images application uses a combination of the `WebClient` class and methods for parsing XML data to achieve the transfer process of the data and images from the server into the application. The first part of this data transfer process is to initiate a call to the server as part of the application start up routine – in conceptually the same way that the Flex version of the application does. Within the code behind file (`Page.xaml.cs`) which forms part of the `Page` class, I constructed a method (`GetServerData()`) which instantiates a new `WebClient` object. The `DownloadStringCompleted` function of this `WebClient` object creates a new instance of an event handler (which is used to process the contents of the data string once it has been downloaded) and

attaches it to the *WebClient* object (*wclXMLData* in this instance). The *DownloadStringAsync* function is then called (with the URI of the *images.xml* file passed to it as its first argument) which causes the actual download of the file as a string. The definition of the *GetServerData()* method can be seen in the following code snippet (Note the *XMLFileLoaded* name assigned in the *DownloadStringCompletedEventHandler* object):

```
public void GetServerData()
{
    WebClient wclXMLData = new WebClient();
    wclXMLData.DownloadStringCompleted += new
        DownloadStringCompletedEventHandler(XMLFileLoaded);
    wclXMLData.DownloadStringAsync(new Uri("xmldata/images.xml",
        UriKind.RelativeOrAbsolute));
}
```

In order to facilitate the download of the data both automatically and at application start up, I have included a call to this method in the constructor of the *Page* object that is created as part of the Silverlight application:

```
public Page()
{
    //Required to initialize variables
    InitializeComponent();
    //Fetch data from server
    GetServerData();
    .
    .
}
```

Before developing the next part of the process, i.e. using the string of data that has been transferred through the use of the *GetServerData()* method to populate the various application components, it was necessary to create a separate class (*ImageItem.cs*) that would be used to create and assist in the management of the discrete image objects. The definition of this simple class can be seen in the code snippet on the next page. You will note that with the exception of *imageThumbNail*, the various instance variables of the class have a direct 1:1 mapping with the elements of the *images.xml* file. (The *imageThumbNail* instance variable is used to contain the actual thumbnail (low resolution) images rather than plain text-based data):

```
using System;
using System.Net;
using System.Windows;
using System.Windows.Media.Imaging;

namespace ImagesDevV9
{
    public class ImageItem
    {
        public int imageID { get; set; }
        public string imageArtist { get; set; }
        public string imageName { get; set; }
        public string imagePath { get; set; }
        public DateTime imageDate { get; set; }
        public string imageComments { get; set; }
        public BitmapImage imageThumbNail { get; set; }
    }
}
```

Returning now to the *GetServerData()* method, you will recall that I declared that the event handler that would be used after the data transfer is called *XMLFileLoaded*. The definition of this event handler is in the code snippet overleaf:

```

void XMLFileLoaded(object sender, DownloadStringCompletedEventArgs e)
{
    if (e.Error == null)
    {
        string strXML = e.Result;
        XDocument xmlDoc = XDocument.Load(XmlReader.Create(new
            StringReader(strXML)));
        var imageData = from imageElement in
            xmlDoc.Descendants("image")
        select new ImageItem
        {
            imageID =
Convert.ToInt16(imageElement.Element("imageID").Value),
            imageArtist =
Convert.ToString(imageElement.Element("imageArtist").Value),
            imageName =
Convert.ToString(imageElement.Element("imageName").Value),
            imagePath =
Convert.ToString(imageElement.Element("imagePath").Value),
            imageDate =
Convert.ToDateTime(imageElement.Element("imageDate").Value),
            imageComments =
Convert.ToString(imageElement.Element("imageComments").Value),
            imageThumbnail = new BitmapImage(new
Uri("imagefiles/thumbs/thm" +
Convert.ToString(imageElement.Element("imagePath").Value),
UriKind.Relative))
        };

        //Create a List<> collection and pass each item from imageData to it
        var lstImageItems = new List<ImageItem>();
        foreach (ImageItem item in imageData)
        {
            lstImageItems.Add(item);
        }

        //Use this List<> collection as datasource for DataGrid
so that we can implement sorting
        dgImageGrid = this.FindName("dgImages") as DataGrid;
        dgImageGrid.ItemsSource = lstImageItems;

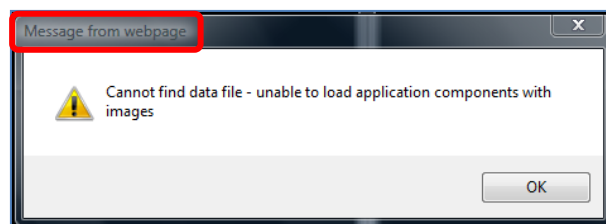
        //As we're not worried about sorting the ListBox, leave
it's datasource as imageData
        lbImageList = this.FindName("lbImages") as ListBox;
        lbImageList.ItemsSource = imageData;
    }
    else
    {
        HtmlPage.Window.Alert("Cannot find data file - unable to
load application components with images");
    }
}

```

Ignoring for the moment the *if* statement, we can see that the first part of the method takes the text string that is encapsulated in the *Result* property of the *DownloadStringCompletedEventArgs* which is then used as the source to create an XML document, i.e. to re-construct the XML file that is available on the server within the Images application client.

Next, using the *ImageItem* class constructed earlier, the child elements (or descendants) of the top level *<image>* element of this XML document are then parsed into a generic collection of *ImageItem* objects called *imageData* together with the relevant thumbnail images. Finally, for these objects to be used as the source for the DataGrid control (a Silverlight component that supports sorting), it is necessary to place them into a *List<>* collection. (Note: The same generic collection of *ImageItem* objects is also used by the application as the source for the ListBox component, but as no sorting functionality is needed for this component, the generic collection *imageData* is adequate, i.e. it is unnecessary for them to be placed in a *List<>* collection).

Returning to the *if* statement which bounds the principal code in the above method, it has been included because the *WebClient* class only provides a *DownloadStringCompleted* function – unlike in Flex, no corresponding “download failed” function is available. Any error that might occur in the transfer of the data from the server is manifested as an error in the Result property of the event and the *if* statement has been included to trap this and display an appropriate message. Note: Unlike Flex, which has an in-built *Alert* class, Silverlight manages alerts by providing access to the browser’s Document Object Model through the *HtmlPage* class. This exposes the browser’s Window object which can then be used to invoke an *Alert()* method – the resultant visual alert can be seen below:



*Graphical display of an Alert in Silverlight (Supplied through browser’s DOM)*

As a result, the Alert facility provided by Silverlight is visually less integrated than the one provided in Flex and customisation of the alert message is limited to the message text only – the dialog box title (bounded in Red in the above screenshot) is not customisable and additional buttons and icons cannot be included like they can in Flex.

#### **10.4.5 List Controls**

As in Flex, Silverlight provides controls that can be used for the presentation of data in list format. The two that are provided as standard are the DataGrid and ListBox controls and both are directly comparable to the DataGrid and List controls in Flex.

Focussing on the DataGrid control, we can see from the code snippet overleaf that there are strong similarities between the MXML (Flex) and XAML declarations for a DataGrid. Columns that make up the DataGrid have their own declaration, and for simple text columns this can be limited to

just a *Header* attribute to provide a title for the column and a *Binding* attribute to link the contents of the column to a data source (Binding is covered in the following section). Likewise, the definition for the column that contains the thumbnail images is slightly more complex and involves the encapsulation of the *<Image>* element within a set of elements which create a data template for the image and is analogous to the *<mx:itemRenderer>*, *<mx:Component>*, *<mx:Vbox>* element set used in Flex. What are also included in the following snippet are the declarations to instruct Silverlight to allow the user to be able to **reorder**, **resize** and **resort** the columns within the DataGrid. These are directly comparable to the Flex *<mx:DataGrid>* element's *draggable*, *resizable* and *sortable* attributes and give the developer great flexibility in customising the functionality that is to be provided to the user.

```
<data:DataGrid x:Name="dgImages"
  Height="430" Width="430" AutoGenerateColumns="False"
  Margin="13,33,0,0" ColumnWidth="102" RowHeight="80"
  SelectionChanged="dgImageSelectedChangeHandler"
  Background="Transparent" Foreground="#FFCCCCCC"
  FontSize="14" FontFamily="Verdana"
  GridLinesVisibility="Vertical"
  AlternatingRowBackground="{x:Null}"
  BorderBrush="{x:Null}" RowBackground="{x:Null}"
  SelectedIndex="-1"
  CanUserReorderColumns="True"
  CanUserResizeColumns="True"
  CanUserSortColumns="True"
  HorizontalAlignment="Center" VerticalAlignment="Center">
  <data:DataGrid.Columns>
    <data:DataGridTextColumn Header="ID" Binding="{Binding
      Path=imageID}" IsReadOnly="True"/>
    <data:DataGridTextColumn Header="Artist" Binding="{Binding
      Path=imageArtist}" IsReadOnly="True"/>
    <data:DataGridTextColumn Header="Name" Binding="{Binding
      Path=imageName}" IsReadOnly="True"/>
    <data:DataGridTemplateColumn Header="Thumbnail" IsReadOnly="True">
      <data:DataGridTemplateColumn.CellTemplate>
        <DataTemplate>
          <Image Source="{Binding Path=imageThumbNail}"
            MaxHeight="70" MaxWidth="70"/>
        </DataTemplate>
      </data:DataGridTemplateColumn.CellTemplate>
    </data:DataGridTemplateColumn>
  </data:DataGrid.Columns>
</data:DataGrid>
```

#### 10.4.6 Data Binding

Binding data in Silverlight can be either a simple or complex process depending on what it is that needs to be achieved, but one glaring comparison with Flex is the lack of support for XAML element-to-element binding. Recalling an example from Flex, in the code snippet overleaf we can see that the duration attribute of the *<mx:WipeRight>* element is bound to the *text* attribute (i.e. the value) in the component called *txtDuration*.



```
<mx:WipeRight id="effWipeRightWipeIn"
duration="{Number(txtDuration.text)*1000}"/>
```

If the user changes that text value, the `<mx:WipeRight>` element will use the new value automatically. However, if the user makes the same change in the Silverlight version of the application, it will only be picked up if a reference is made in the managed code that has been developed to manage the other components or animations elsewhere in the application. The example of the Silverlight equivalent (with both XAML and managed C# code) is given in the two code snippets provided below:

```
<!--Storyboard for wipe on & off of List Box and Image Details area -->
<Storyboard x:Name="stbWipe">
  <DoubleAnimationUsingKeyFrames BeginTime="00:00:00"
    Storyboard.TargetName="rcgListBox"
    Storyboard.TargetProperty="ScaleX">
    <SplineDoubleKeyFrame KeyTime="00:00:00" x:Name="sdkfWipeListBox"/>
  </DoubleAnimationUsingKeyFrames>
  <DoubleAnimationUsingKeyFrames BeginTime="00:00:00"
    Storyboard.TargetName="rcgImageDetails"
    Storyboard.TargetProperty="ScaleX">
    <SplineDoubleKeyFrame KeyTime="00:00:00"
x:Name="sdkfWipeImageDetails"/>
  </DoubleAnimationUsingKeyFrames>
</Storyboard>
```

```
private void chkList_Click(object sender, RoutedEventArgs e)
{
  if (chkList.IsChecked == true)
  {
    sdkf1 = this.FindName("sdkfWipeListBox") as SplineDoubleKeyFrame;
    sdkf2 = this.FindName("sdkfWipeImageDetails") as SplineDoubleKeyFrame;
    sdkf1.Value = 1;
    sdkf2.Value = 1;
    sdkf1.KeyTime =
      KeyTime.FromTimeSpan(TimeSpan.FromSeconds(Convert.ToInt32(txtDuration.
        Text)));
    sdkf2.KeyTime =
      KeyTime.FromTimeSpan(TimeSpan.FromSeconds(Convert.ToInt32(txtDuration.
        Text)));
    stbWipe.Begin();
  }
}
```

I'll cover the animation specific issues that are included in above code later in this report, but what can be seen here is that the XAML code does not reference the value of the `txtDuration` component directly. Instead, the C# code is needed to handle this behind the scenes.

Silverlight however does support numerous methodologies for binding data sources to targets within an application and also supports two-way data binding - updates can occur in either direction between source and target), the default one-way (from source to target) and one-time -

from source to target but the update only happens once with no update to the target if the source changes and this potentially offers a degree of flexibility that Flex doesn't easily offer.

Due to the nature of the Images application, the only imperative requirement for data binding is within the DataGrid and ListBox controls. In the section above on Application Startup I demonstrated how the data (once downloaded) was assigned as the *ItemSource* for both of these controls – as a *List<>* collection for the DataGrid and as a generic collection for the ListBox. Although this was achieved using managed code as part of the application initialisation procedure, the binding of the individual attributes or fields of each image item to their respective child elements was achieved through the binding facility available in XAML. The code snippet below shows how this was achieved for the DataGrid control (Note: As in MXML, XAML uses curly brace notation {} to signify that a binding is required, but the *Binding* keyword is also required):

```
<data:DataGrid x:Name="dgImages"
  Height="430" Width="430"
  AutoGenerateColumns="False" Margin="13,33,0,0"
  ColumnWidth="102" RowHeight="80"
  SelectionChanged="dgImage_SelectionChanged"
  Background="Transparent" Foreground="#FFCCCC"
  FontSize="14" FontFamily="Verdana"
  GridLinesVisibility="Vertical" AlternatingRowBackground="{x:Null}"
  BorderBrush="{x:Null}" RowBackground="{x:Null}"
  SelectedIndex="-1" CanUserReorderColumns="True"
  CanUserResizeColumns="True" CanUserSortColumns="True"
  HorizontalAlignment="Center" VerticalAlignment="Center">
  <data:DataGrid.Columns>
    <data:DataGridTextColumn Header="ID" Binding="{Binding
      Path=imageID}" IsReadOnly="True"/>
    <data:DataGridTextColumn Header="Artist" Binding="{Binding
      Path=imageArtist}" IsReadOnly="True"/>
    <data:DataGridTextColumn Header="Name" Binding="{Binding
      Path=imageName}" IsReadOnly="True"/>
    <data:DataGridTemplateColumn Header="Thumbnail" IsReadOnly="True">
      <data:DataGridTemplateColumn.CellTemplate>
        <DataTemplate>
          <Image Source="{Binding Path=imageThumbNail}" MaxHeight="70"
            MaxWidth="70"/>
        </DataTemplate>
      </data:DataGridTemplateColumn.CellTemplate>
    </data:DataGridTemplateColumn>
  </data:DataGrid.Columns>
</data:DataGrid>
```

Within this code can be seen four Binding declarations – 3 for the text information required for the *<data:DataGridTextColumn>* elements for ID, Artist and Name and 1 for the Image thumbnail. No other declarations are needed for the *<data:DataGrid>* component itself as the association between the DataGrid and its data source has already been made within managed code via the *dgImageGrid.ItemsSource = lstImageItems;* declaration.

The same general approach was used to achieve the data binding for the `ListBox` control, but the implementation is different due to the fact that the items in the `ListBox` needed to be animated when the user moved the mouse over them. I will cover this in further detail in the next section on Animations.

As mentioned above, it is possible to create far more complex data binding scenarios through Silverlight, but the implementation of the Images application did not require anything deeper than that already demonstrated. However, I would recommend Chapter 18 of “Silverlight 2 Unleashed” by Laurent Bugnion as a good place to start if further information is required on Data Binding.

#### **10.4.7 Animations**

As with the Flex version of the application, the Silverlight version of Images implements many animation techniques. The primary way to achieve animations in Silverlight is through the *Storyboard* class which allows changes to be made to a component over a period of time. Storyboards can be defined in XAML and in managed code, but within the Images application I have defined all of the storyboards in XAML – principally because I found it easier to do so. The components of a storyboard object can be used to initiate three different animation types:

- *DoubleAnimation* is used for animating attributes that are related to spatial dimensions (e.g. *Canvas.Top*) or visual properties (e.g. *Opacity*)
- *ColorAnimation* is used to animate component properties that contain colour values (e.g. *Background* or *Stroke*)
- *PointAnimation* is used to animate component properties that contain points such as line segments.

The Images application only employs the *DoubleAnimation* type within its storyboards, but the principles used are the same for the other two animation types. Within the animation type it is necessary to define both the name of the component that requires animating and the property of that component that is to be animated. Other animation properties that can be set include the *BeginTime* (to set the delay before the animation begins), the *Duration*, the target property's *From* and *To* values, the *SpeedRatio*, the *RepeatBehaviour* and the *AutoReverse* property. Again, like with storyboards, these can be defined both in XAML and in managed code and this time in my implementation, I have used a combination of the two. Recalling from a previous section on Binding, Silverlight's inability to support XAML element-to-element binding, I have used managed code to associate the property values of one element as feeds for the property values in the animations.

Note: Before investigating my implementation of storyboards, it is worth noting that it is possible to define both the storyboard and the event triggering of that storyboard as child elements within a visual component in XAML, but Silverlight 2 (the release on which this project is based) only

supports one event trigger – *RoutedEvent* and this only supports one event type – *Loaded*. Therefore to respond to other events, such as *Click* or *MouseOver*, a different approach is required.

Within the Images application I have defined several storyboards and then used managed code to actually initiate or trigger those storyboards when certain events are raised and to set some of the animation properties. The two code snippets below demonstrate this approach:

```
<Storyboard x:Name="stbDetailImageResize">
  <DoubleAnimation x:Name="danResize1"
    BeginTime="00:00:00"
    Storyboard.TargetName="imgDetailImage"
    Storyboard.TargetProperty="(FrameworkElement.Width)"/>

  <DoubleAnimation x:Name="danResize2"
    BeginTime="00:00:00"
    Storyboard.TargetName="imgDetailImage"
    Storyboard.TargetProperty="(FrameworkElement.Height)"/>

  <DoubleAnimation x:Name="danResize3"
    BeginTime="00:00:00"
    Storyboard.TargetName="sldResize"
    Storyboard.TargetProperty="(FrameworkElement.Value)"/>
</Storyboard>
```

```
private void btnGrow_Click(object sender, RoutedEventArgs e)
{
  dan1 = this.FindName("danResize1") as DoubleAnimation;
  dan2 = this.FindName("danResize2") as DoubleAnimation;
  dan3 = this.FindName("danResize3") as DoubleAnimation;
  dan1.To = Convert.ToDouble(txtResizeMax.Text);
  dan2.To = Convert.ToDouble(txtResizeMax.Text);
  dan3.To = Convert.ToDouble(txtResizeMax.Text);
  dan1.Duration = TimeSpan.FromSeconds(Convert.ToInt32(txtDuration.Text));
  dan2.Duration =
  TimeSpan.FromSeconds(Convert.ToInt32(txtDuration.Text));
  dan3.Duration =
  TimeSpan.FromSeconds(Convert.ToInt32(txtDuration.Text));
  stbDetailImageResize.Begin();
}
```

The XAML code defines a storyboard object (*stbDetailImageResize*) and within this there are three *DoubleAnimations* defined. Using the first as an example (named *danResize1* so that it can be referenced in managed code) it can be seen that I have defined the *BeginTime* to be 0 (i.e. there will be no delay in the launch of the animation when it is initiated), that the target for the animation is the image component called *imgDetailImage* and that the property of that component that is to be animated is its *Width*.

Moving to the managed code snippet, you can see that within an event handler that responds to the *Click* event of the button control called *btnGrow*, I have instantiated a blank *DoubleAnimation* object and associated it with the *danResize1 DoubleAnimation* that was defined in

the XAML. I have then used this new object to a) modify the *To* property – setting it whatever value is currently in the *txtResizeMax* textbox and b) modify the *Duration* property to be equal to the value currently held in the *txtDuration* textbox. The same methodology is used to set the relevant values for the *Height* property of *imgDetailImage* and the maximum *Value* for the slider component called *sldResize*. (Note: It is possible to target more than one component and component property within a single storyboard providing each target component name / target component property pair is defined within its own animation type declaration). The final statement of the event handler then initiates the storyboard by calling its *begin()* method. (Note: In the code snippet above, by setting the target component's property values in managed code, and only using the XAML storyboard declarations to identify the target components and their properties, I am able to re-use the same storyboard as part of the event handler for the Shrink button's (*btnShrink*) click event).

Exactly the same principles have been used to provide the animation that is invoked to cause the fading and sharpening of the detail image in the application – initiated via the click events for the *btnFade* and *btnUnFade* buttons. However, the animations that are initiated via the majority of the other buttons in the application (Skew / Unskew, Zoom In / Zoom Out, Rotate / Reset, Parallel / Reset and Move On / Move Off) use a slightly different methodology that is worth investigating – firstly because they use another key facility supplied in Silverlight called **Transforms** and b) because their use initially caused a very unwanted side-effect – at least in some development versions of the application, and I think the lessons learned are worth articulating.

Transformations differ from animations in that they define changes in the mapped position or dimensions of an object but are not concerned with anything to do with timing. There are 5 transformations provided in Silverlight, 4 of which have been used in some form in the Images application:

- A *ScaleTransform* is used to modify the size of a component.
- A *RotateTransform* rotates a component about a specified point by a specified angle.
- A *SkewTransform* can be used to independently move the X- and / or Y-axis of a component (e.g. to turn a rectangle into a lozenge) and is therefore very useful if you need to use a combination of 2D components to simulate a 3D representation.
- A *TranslateTransform* is used to move a component from one location to another.

Note: The 5<sup>th</sup> transform type is known as a *MatrixTransform* and which can be used to simultaneously apply multiple transforms to a visual object. The first 4 transform types are effectively special instances of a *MatrixTransform* and I have not employed the *MatrixTransform* within this project.

Transforms are very useful facilities to have available, especially when the effect you want to achieve isn't accessible via the normal properties associated with a component. For example, an Image component has an Opacity property, and it is this that I used within the Images application to

achieve the Fade and UnFade animations, however it does not have RotateAngle or RotateOrigin properties that could be used to effect a rotation.

The two code snippets (below and overleaf) demonstrate an implementation of a *RotateTransform* in the Images application. Firstly, as I wish to animate the rotation of the detail Image component, I need to let that component know that it is capable of being rotated. This is achieved by including within the declaration of the detail image component, a second declaration that addresses the *RenderTransform* property of the *<Image>* component and that then defines a *<TransformGroup>*. Within this it is necessary to define each of the transforms that you may wish to apply to the component, e.g. *<ScaleTransform>*, *<RotateTransform>*, etc. (Note: In this context, the assignments of values to the *x:Name* properties are superfluous – they have been included in the code for other purposes. It is only necessary to identify the type of transforms required).

```
<Image x:Name="imgDetailImage" Margin="0,0,0,0"
  VerticalAlignment="Center"
  HorizontalAlignment="Center"
  Width="436" Height="436">
  <Image.RenderTransform>
    <TransformGroup>
      <ScaleTransform x:Name="trfScale"/>
      <SkewTransform x:Name="trfSkew"/>
      <RotateTransform x:Name="trfRotate"/>
      <TranslateTransform x:Name="trfTranslate"/>
    </TransformGroup>
  </Image.RenderTransform>
</Image>
```

Now that the relevant component has been modified, such that it is capable of being transformed, it is necessary to create a storyboard to manage the animation aspects of the transformation (see code snippet overleaf):

```

<Storyboard x:Name="stbDetailImageRotate">
  <DoubleAnimation x:Name="danRotate1" BeginTime="00:00:00"
    Storyboard.TargetName="imgDetailImage"
    Storyboard.TargetProperty="(FrameworkElement.RenderTransform).
    (TransformGroup.Children)[2].(RotateTransform.Angle)"/>

  <DoubleAnimation x:Name="danRotate2" BeginTime="00:00:00"
    Storyboard.TargetName="sldRotate"
    Storyboard.TargetProperty="(FrameworkElement.Value)"/>

  <DoubleAnimation x:Name="danRotate3" BeginTime="00:00:00"
    Storyboard.TargetName="imgDetailImage"
    Storyboard.TargetProperty="(FrameworkElement.RenderTransform).
    (TransformGroup.Children)[2].(RotateTransform.CenterX)"/>

  <DoubleAnimation x:Name="danRotate4" BeginTime="00:00:00"
    Storyboard.TargetName="imgDetailImage"
    Storyboard.TargetProperty="(FrameworkElement.RenderTransform).
    (TransformGroup.Children)[2].(RotateTransform.CenterY)"/>
</Storyboard>

```

Here we can see that the target properties of the relevant *DoubleAnimation* types in the storyboard are not simply referencing a standard property of component (such as width or opacity), but referencing the properties of that component's *<RenderTransform>* element's properties. Note: The above *Storyboard.TargetProperty* statements not only contain references to the particular type of transform being used, e.g. *RotateTransform*, but also which *Children* type of the *TransformGroup* that this represents – [2] in the above example. The following list details this relationship for the 4 standard *RenderTransform* types:

- *ScaleTransform* = Children type 0
- *SkewTransform* = Children type 1
- *RotateTransform* = Children type 2
- *TranslateTransform* = Children Type 3

Finally as part of this particular process, there is a need to define event handlers to respond to the click events that are raised when the Rotate or Reset (*btnRotate* & *btnUnRotate*) buttons are pressed – the click event associated with the *btnRotate* control is detailed in the code snippet overleaf:

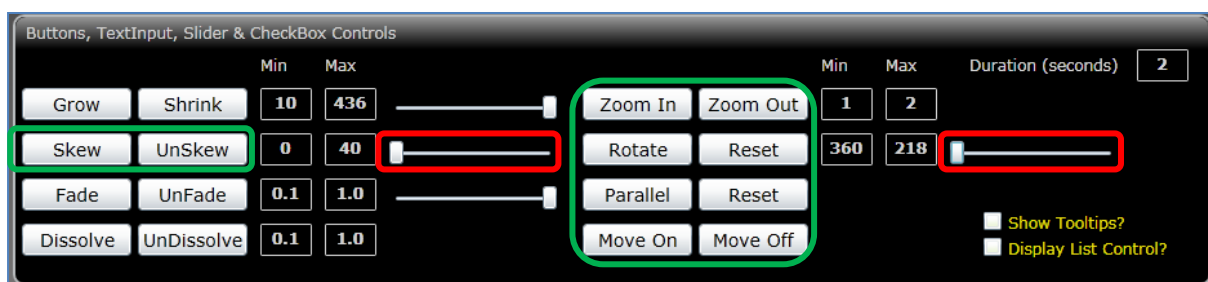
```

private void btnRotate_Click(object sender, RoutedEventArgs e)
{
    dan1 = this.FindName("danRotate1") as DoubleAnimation;
    dan2 = this.FindName("danRotate2") as DoubleAnimation;
    dan3 = this.FindName("danRotate3") as DoubleAnimation;
    dan4 = this.FindName("danRotate4") as DoubleAnimation;
    dan1.To = Convert.ToDouble(txtRotateDegrees.Text);
    dan2.To = Convert.ToDouble(txtRotateDegrees.Text);
    dan3.To = Convert.ToDouble(txtRotateOrigin.Text);
    dan4.To = Convert.ToDouble(txtRotateOrigin.Text);
    dan1.Duration =
    TimeSpan.FromSeconds(Convert.ToInt32(txtDuration.Text));
    dan2.Duration =
    TimeSpan.FromSeconds(Convert.ToInt32(txtDuration.Text));
    dan3.Duration =
    TimeSpan.FromSeconds(Convert.ToInt32(txtDuration.Text));
    dan4.Duration =
    TimeSpan.FromSeconds(Convert.ToInt32(txtDuration.Text));
    stbDetailImageRotate.Begin();
}

```

As with my previous example (which detailed the managed code event handler for the *btnGrow* button), this event handler references the relevant *DoubleAnimation* types, sets certain values and durations for these based on values held in other components and then initiates the animation by calling the *begin()* method of the storyboard.

As I mentioned previously, there is a downside in this current implementation. When experimenting with the various functions provided through the buttons and sliders in the application, I noticed that the sliders for the Skew and Rotate only worked if they were used **before** any buttons that invoked a storyboard that relied on a *RenderTransform* were pressed. (See screen clip below for clarity).



Screenshot of facilities that were affected by the storyboard animation / transform problem

Once **any one** of the buttons that invoked one a storyboard associated with a *RenderTransform* was pressed, i.e. Skew, Zoom, Rotate, Parallel and Move (bounded in Green in above screen clip), the sliders for both the Skew and Rotate functions (bounded in Red) become inoperable. If **only** the buttons for the Grow / Shrink or Fade / UnFade functions were pressed, then these sliders continued to work. (Note: A working example of these annoying behaviours can be found at <http://www.acanthus-advance.com/ImagesDevV8/Images.html>).



My initial assumption was that this was probably the result of actually including the sliders for these functions (i.e. ones that included a reference to the *RenderTransform* properties) within the respective storyboards and that even when the storyboards appeared to have completed they still retained control of the sliders. I tested this by removing the references to the Rotate and Skew sliders from within their respective storyboards and trying the functions again. This had no effect and led me to my next assumption that storyboards (once invoked) retain some sort of control over the actual image component they target. Invoking another storyboard (e.g. clicking on the Rotate button after clicking on the Skew button) gave expected (normal) results leading me to the conclusion that a storyboard can wrest control from another storyboard, but a managed code function can't. However, further experimentation then showed that the sliders that manage the animation effects for the Grow and Fade functions continued to work after invoking any of the storyboards, and these were controlled via managed code.

The animation effects that are controlled by the slider components for the Skew and Rotate functions are handled in managed code, as shown in the following code snippet:

```
private void sldSkew_ValueChanged(object sender,
    RoutedPropertyChangedEventArgs<double> e)
{
    trfSkew.AngleX = sldSkew.Value;
    trfSkew.AngleY = sldSkew.Value;
    trfSkew.CenterX = imgDetailImage.ActualWidth / 2;
    trfSkew.CenterY = imgDetailImage.ActualHeight / 2;
}
```

However, the managed code that controls the slider animations for the Grow and Fade functions are (like their storyboards) slightly different, as can be seen in the example in the following code snippet:

```
private void sldResize_ValueChanged(object sender, RoutedEventArgs e)
{
    imgDetailImage.Height = sldResize.Value;
    imgDetailImage.Width = sldResize.Value;
}
```

The first code snippet above shows that the slider associated with the Skew functions (*sldSkew*) effects its changes by referencing the *<SkewTransform>* that was defined in the *TransformGroup* for the detail image, whereas the slider associated with the Grow functions references the width and height properties of the detail image directly. Therefore, the use of the Transforms in combination with animations through storyboards seemed to be the cause of the problem. Note: In addition to a *begin()* method, storyboards also have a *stop()* method and I also experimented with this. Unfortunately, when the *stop()* method is used it not only fails to release

control but it also restores the component being animated to the state (or position) it was in before the *begin()* method was called.

Despite trawling various web fora, I struggled to find an answer to this particular annoyance. I also lodged the problem with the University's Academic Evangelist at Microsoft but had no response, so at this stage I felt it best to leave the problem to one side – there were more pressing issues within the project to address. However, the problem continued to rankle so toward the end of the project I attempted another search for a solution. On the basis that the problem only occurred with the two sliders that were associated with components whose animations were effected through the use of *RenderTransforms*, I consulted the web fora again but this time searched for problems being posted that were related with *RenderTransform* properties, rather than problems associated with storyboards. Fortunately, I came across a series of posts from January 2009 [30] that described a problem that resonated with mine, together with a solution that pointed to the development of a storyboard dedicated to the slider control. To that end, I developed new storyboards that would only be invoked in response to a change in the value of the sliders and modified the slider event handlers' managed code. Example code snippets for the slider for the Skew functions are detailed below – firstly, the XAML declaration of the required storyboard and secondly, the amended managed code for the slider movement event handler (the commented out lines in the second snippet show the original code):

```
<Storyboard x:Name="stbDetailImageSkewSlider">
  <DoubleAnimation x:Name="danSkewSlider1"
    BeginTime="00:00:00" Duration="00:00:00"
    Storyboard.TargetName="imgDetailImage"
    Storyboard.TargetProperty="(FrameworkElement.RenderTransform).
    (TransformGroup.Children)[1].(SkewTransform.AngleX)"/>
  <DoubleAnimation x:Name="danSkewSlider2"
    BeginTime="00:00:00" Duration="00:00:00"
    Storyboard.TargetName="imgDetailImage"
    Storyboard.TargetProperty="(FrameworkElement.RenderTransform).
    (TransformGroup.Children)[1].(SkewTransform.AngleY)"/>
</Storyboard>
```

```
private void sldSkew_ValueChanged(object sender,
  RoutedPropertyChangedEventArgs<double> e)
{
  dan1 = this.FindName("danSkewSlider1") as DoubleAnimation;
  dan2 = this.FindName("danSkewSlider2") as DoubleAnimation;
  dan1.To = e.NewValue;
  dan2.To = e.NewValue;
  stbDetailImageSkewSlider.Begin();
  //trfSkew.AngleX = sldSkew.Value;
  //trfSkew.AngleY = sldSkew.Value;
  //trfSkew.CenterX = imgDetailImage.ActualWidth / 2;
  //trfSkew.CenterY = imgDetailImage.ActualHeight / 2;
}
```

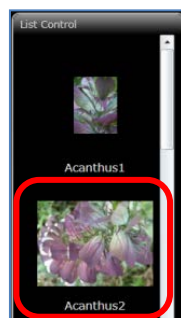
---

[30] <http://betaforums.silverlight.net/forums/t/68888.aspx> (Last accessed 18/09/09)

Here we can see that instead of targeting the properties of the `<SkewTransform>` (`trfSkew`) directly with whatever the current value for the slider is, the event handler is referencing the `DoubleAnimation` types in the storyboard (which themselves reference the properties of the `<SkewTransform>`) and passing the values via them. The storyboard's `begin()` method then causes the invocation of the animation.

If I had followed my own earlier assumption, i.e. that it **appears** that when `RenderTransforms` are involved, only the invocation of a second storyboard can be used to wrest control from an initial storyboard then I may have got there on my own. But the moral of the tale reinforces what many developers have learnt over the years, that it is unlikely that the problem facing you has not been hit before, and that with enough searching (and by asking the right question) you can often find someone who has already found a solution. (Note: A later post in the same forum thread that led me to develop the now working solution implies that in reality, the `RenderTransform` object that I was trying to manipulate with the slider with my initial code may have actually been a different `RenderTransform` object than the one being manipulated by the original storyboard and it was this that led to the effect (or lack of one) that I experienced. However, I was unable to determine any difference in the object references in the Visual Studio debugger pane and so I must assume this needs further verification).

The final effect I want to cover in this section on Animations is the one provided in the `List Box` component (see screen clip below):



*Demonstration of the resize of an individual image in the List Box area*

Replicating the functionality of the Flex implementation of Images application, I wanted any of the images in this component to grow in size (over a period of 200ms) whenever the user moved their mouse over them.

Achieving the growth of the image instantaneously proved to be a relatively simple task. Firstly, (as can be seen in the code snippet overleaf), I defined the `<Image>` component (with its data binding declaration) and placed this within a `<Grid>` component (needed to support my requirement to also display a textbox below the image) and this was placed within a `<DataTemplate>` and

<ListBox.ItemTemplate> - both of these being needed to provide a visual rather than string representation of my images.

```
<ListBox x:Name="lbImages"
    Height="670" Width="185" Margin="10,25,0,0"
    Background="Transparent" BorderBrush="{x:Null}"
    FontFamily="Verdana">
    <ListBox.ItemTemplate>
        <DataTemplate>
            <Grid Height="170" Width="140" x:Name="grdListImages"
                Margin="10,0,0,0">
                <Image Source="{Binding Path=imageThumbNail}"
                    x:Name="listImage"
                    MaxHeight="70" MaxWidth="70"
                    HorizontalAlignment="Center"
                    VerticalAlignment="Center"
                    Margin="0,0,0,0"
                    MouseEnter="listImageMouseEnterHandler"
                    MouseLeave="listImageMouseLeaveHandler" />
            .
            .
            .
        </DataTemplate>
    </ListBox.ItemTemplate>
</ListBox>
```

To complete the functionality, I added two event handler declarations to respond to the mouse either entering or leaving the area of the image. The corresponding managed code for these event handlers (using the *MouseEnter* event as an example) is detailed in the code snippet below:

```
private void listImageMouseEnterHandler(object sender, MouseEventArgs e)
{
    Image imgEnter = sender as Image;
    imgEnter.MaxHeight = 140;
    imgEnter.MaxWidth = 140;
}
```

This approach certainly provided the initial part of the effect I wanted to achieve – that of growing or shrinking the image (and only that image) in response to the presence or otherwise of the mouse. However, the effect was instantaneous and I wanted it to animate over a period of 200ms. In attempting to provide this animation I tried working with storyboards, and making the call to the storyboard from managed code – in the same manner that I had employed for the animations I had achieved elsewhere within the application. However these all proved fruitless as, despite the fact that the images had been placed in templates to ensure that they were referenced discretely, passing this reference to the storyboard caused failures indicating that the storyboard could not identify the image to be manipulated.

Coincidentally, work on the implementation of multi-scale image technology for the application (which I will cover later in this report) introduced me to the concept of Visual States, and it was working with these that provided the answer to my problem. According to the Visual Studio help pages... “A **VisualState** specifies how the control looks when it is in a certain state. For example,

when a Button is pressed, its border might be a different color than normal. The **VisualState** class has a *Storyboard* property that changes the appearance of the control.” [31]

Taking this forward, the first part of the required code was the definition of the various components of the *VisualState* and its associated elements as shown in the code snippet below:

```
<vsm:VisualStateManager.VisualStateGroups>
  <vsm:VisualStateGroup x:Name="CommonStates">
    <vsm:VisualState x:Name="Normal">
      <Storyboard>
        <DoubleAnimation Storyboard.TargetName="imgListImage"
          Storyboard.TargetProperty="(FrameworkElement.Width)"
          To="70" Duration="00:00:00.2000000"/>
        <DoubleAnimation Storyboard.TargetName="imgListImage"
          Storyboard.TargetProperty="(FrameworkElement.Height)"
          To="70" Duration="00:00:00.2000000"/>
      </Storyboard>
    </vsm:VisualState>
    <vsm:VisualState x:Name="MouseOver">
      <Storyboard>
        <DoubleAnimation Storyboard.TargetName="imgListImage"
          Storyboard.TargetProperty="(FrameworkElement.Width)"
          To="140" Duration="00:00:00.2000000"/>
        <DoubleAnimation Storyboard.TargetName="imgListImage"
          Storyboard.TargetProperty="(FrameworkElement.Height)"
          To="140" Duration="00:00:00.2000000"/>
      </Storyboard>
    </vsm:VisualState>
  </vsm:VisualStateGroup>
</vsm:VisualStateManager.VisualStateGroups>
```

First, the code declares which of the *VisualStateGroups* that will be used. There are 3 *VisualStateGroups*, each of which contain *VisualStates* which equate to the various states a visual component might have. The 3 *VisualStateGroups* and their members are:

- Common States: Normal, MouseOver, Pressed, Disabled
- Focus States: Focused, ContentFocused, Unfocused
- Check States, Checked, Unchecked, Indeterminate

The two relevant states that are applicable to the functionality I wanted to create are the *Normal* & *MouseOver* states – both of which are Common States, hence the reason why only these have been declared within the above code.

The code then defines two storyboards – one for each of the two visual states I wanted my *ListBox* items to react to. The first storyboard is for the *Normal* state, i.e. what the *ListBox* item should look like when the user’s mouse is not positioned over it. The second storyboard defines what happens when the mouse is moved over a particular image (or item) in the *ListBox* – in this

---

[31] Visual Studio 2008 Help Page [ms-help://MS.VSCC.v90/MS.VSIPCC.v90/MS.SilverlightSDK.v20/dv\\_SilverLTMLRef/html/bccbb091-87a6-6c88-47dd-b6f8b2f11532.htm](http://ms-help://MS.VSCC.v90/MS.VSIPCC.v90/MS.SilverlightSDK.v20/dv_SilverLTMLRef/html/bccbb091-87a6-6c88-47dd-b6f8b2f11532.htm)

case I alter the *Height* and *Width* properties of the *imgListItem* component to 140 pixels (as defined in the *To* property) and that I want this change to occur over a period of 200ms (as defined in the *Duration* property).

Note: The same effect could be achieved by employing *VisualTransitions* which instruct Silverlight about the actions to be performed when moving between a pair of states, but as there is no other complexity involved in this particular example, the use of *VisualTransitions* would be superfluous. (Note: Ian Griffiths provides an excellent blog entry [32] on *VisualStates* which provides a good grounding on their use together with further information and examples on *VisualTransitions*).

In order to ensure that the animations that have been defined only apply to the discrete images as the mouse moves over or away from them, it is also necessary to encapsulate the above definitions in a *<Grid>* element together with the associated definitions for the image and textbox. This is all then encapsulated within a *<ControlTemplate>* within a *<Style>* element:

```
<Style x:Key="stlListItem" TargetType="ListItem"
  xmlns:vsm="clr-namespace:System.Windows;assembly=System.Windows">
  <Setter Property="Template">
    <Setter.Value>
      <ControlTemplate TargetType="ListItem">
        <Grid Height="170" Width="140"
          x:Name="grdListImages" Margin="10,0,0,0">
          <vsm:VisualStateManager.VisualStateGroups>
            </vsm:VisualStateManager.VisualStateGroups>
            Code for Visual States from above goes here...
            <Image Source="{Binding imageThumbnail}"
              x:Name="imgListItem" Height="70" Width="70"
              HorizontalAlignment="Center" VerticalAlignment="Center"
              Margin="0,0,0,0" />
            <TextBlock Text="{Binding imageName}"
              x:Name="tbListItem" FontFamily="Verdana"
              FontSize="14" HorizontalAlignment="Center"
              VerticalAlignment="Bottom" Margin="0,0,0,0"
              Foreground="LightGray"/>
          </Grid>
        </ControlTemplate>
      </Setter.Value>
    </Setter>
  </Style>
```

This *<Style>* element is then defined within the *<UserControl.Resources>* element for the application.

Note: The *<UserControl.Resources>* element is a part of the *Page* object which contains the resources and elements that need to be made available to the rest of the components within that

---

[32] <http://www.interact-sw.co.uk/iangblog/2008/06/10/visual-state> (Last accessed 15/08/09)

*Page* object – remember that the `<UserControl>` element is the root element of the XAML part of the definition of the *Page* object for the application.

Finally, it is necessary within the definition of the `<ListBox>` component to reference the use of this `<Style>` component (called `stlListBoxItem`). The complete definition of the `<ListBox>` component can be seen in the following code snippet with the reference to the style component highlighted:

```
<ListBox x:Name="lbImages"
    Height="670" Width="185" Margin="10,25,0,0"
    Background="Transparent" BorderBrush="{x:Null}"
    FontFamily="Verdana" SelectionChanged="lbImages_SelectionChanged"
    ItemContainerStyle="{StaticResource stlListBoxItem}">
</ListBox>
```

The net result of this approach is that the images in the `<ListBox>` component animate over a specified duration from normal to large (and back again) in response to the mouse. It is also worth noting that this is all completed without the use of any managed code.

## 10.5 Comparison

One of the most striking comparisons that can be made between the development processes for the two versions of the Images application is the amount of code that is needed for each. This shows that equivalent functionality takes approximately twice as many lines in Silverlight as it does in Flex. Making a straight line extrapolation through to productivity is of course dangerous, as other factors need to be taken into account, e.g. the effectiveness of the automatic code generation features in each IDE and online support tools, and the ongoing maintainability of the code. (Note: Although I won't be covering multi-scale image technology until the next chapter it is worth noting at this point that the Silverlight version of Images application also contains a lot of code to support the mouse movements for a multi-scale image / Deep Zoom component. Silverlight's multi-scale image component is supported as part of the Silverlight product set whereas the corresponding Flex-based OpenZoom component, which as we'll see uses significantly less code, is not currently supported by Adobe and so this disjoint needs to be taken into account).

Both technologies have strengths and weaknesses when it comes to the code that needs to be used to generate an application – some of which are apparent in the respective versions of the Images application, and some that can be inferred. For example, the data binding capabilities within Flex are both comprehensive and intuitive, and the automatic component-to-component binding feature within MXML is a big boon when compared to Silverlight XAML. However, the Images applications only touch the surface of the capabilities of the two technologies and the potential power of the binding facilities in Silverlight (with its two-way, one-way and one-time modes) promises much for more complex applications.

In addition, both technologies provide comparable support for controls and custom-made components and both technologies are supported by communities (both commercially and non-commercially oriented) who can provide additional functionality. However, it is my belief that the support for animations provided as standard within Flex is currently superior to that provided within Silverlight – certainly, the processes by which animations / effects can be generated and managed within Flex are both more efficient in code terms and more intuitive in human terms. Again, there is the argument that the storyboard features of Silverlight offer potentially far greater power in more complex applications, but for simple applications Flex is just so much easier to work with – especially when combining the simpler and more intuitive binding techniques between effects and components. The machinations I had to go through in Silverlight in order to overcome the problem with the sliders becoming inoperative felt like an awful lot of effort (in terms of coding) to address a what should be a simple issue.



## 11. Multi-scale Images

### 11.1 Introduction

A facility that has been popularised through the launch of Silverlight is the multi-scale image – a facility that enables the application creator to provide a visually stunning alternative to standard zoom functions. As will be seen in this chapter, from at least a technological standpoint, the multi-scale image facility is a significant development – only the future will tell whether it is used extensively in web applications. Initially, my investigations indicated that the lack of a multi-scale image type facility in Flex made its presence in Silverlight a really key differentiator between the two technologies, however I then came across an independent initiative that aims to redress the balance and which I will cover in the latter half of this chapter.

### 11.2 Silverlight (Deep Zoom)

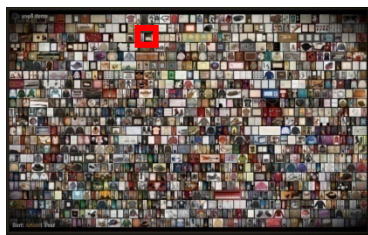
The `<MultiScaleImage>` is perhaps one of the most significant components provided in Silverlight (at least from the perspective of an “out-of-the-box” component). Initially it doesn’t sound too revolutionary as it might be seen to just provide a way of grouping several images together within a single visual space – something that could probably be achieved with the careful positioning of several image components, but it is the **MultiScale** part of the name (or its pseudonym **Deep Zoom**) that intimates its power and appeal. Firstly, `<MultiScaleImage>` components very easily allow images to be nested within other images and secondly, they allow very high definition images to be embedded within an application without impacting the performance of that application. That is to say that when multi-scale images are encountered within an application, the application only displays them at a resolution that is appropriate for the portion of the screen being used to provide their display, and at the zoom level selected by the user.

Taking this concept further, if the user zooms in to a `<MultiScaleImage>` component, Silverlight will only request the network download of the detail for the part of the image (or set of images) that will be displayed to the user – any other part of the image (or set of images) that falls outside of the display area is not retrieved, thus significantly reducing the load on the network and thereby providing a correspondingly responsive result for the user. The most well known example of a commercial implementation of the technology (which Microsoft took ownership of when it acquired Seadragon Software in February 2006) can be seen on the **Memorabilia** page of the Hard Rock Cafe website [33]. The most rewarding experience of the functionality provided through this site (or rather that of the multi-scale image / Deep Zoom technology) is best gained live in order to appreciate the smoothness of the transitions between the various resolutions, but the screenshots overleaf attempt to demonstrate the facility – albeit in a very static and non-interactive way. (Note:

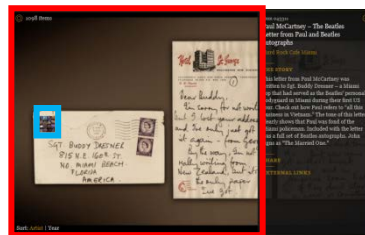
---

[33] <http://memorabilia.hardrock.com/> (Last accessed 16/09/2009)

The large coloured frame in one image shows an expanded view of the same small coloured frame from the previous image).



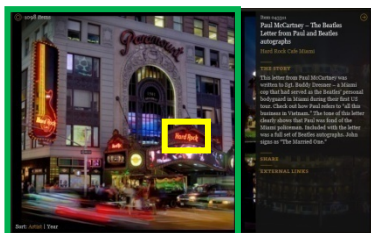
Initial Image...



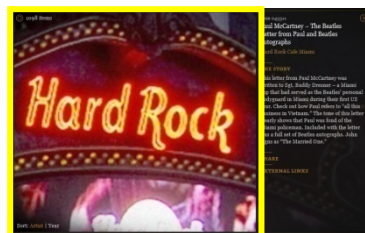
Zoomed in on letter...



Zoomed in on stamp...



Zoomed in on part of stamp...



Zoomed in on one last time...

Before using a *<MultiScaleImage>* component within a Silverlight application, it is necessary to create the “deep zoom” image set that it will contain. This is achieved through the use of the Deep Zoom Composer tool that can be downloaded free of charge from the **Get Started** page of Microsoft’s Silverlight portal site [34]. I do not intend to explain the mechanics of how to use the Deep Zoom Composer as these can be found elsewhere in many Silverlight publications (see the Bibliography for some of these), but I would like to demonstrate how I have implemented the resultant output in the Silverlight version of the Images application and also demonstrate how I addressed a particular issue with the mouse wheel that I encountered as part of that implementation.

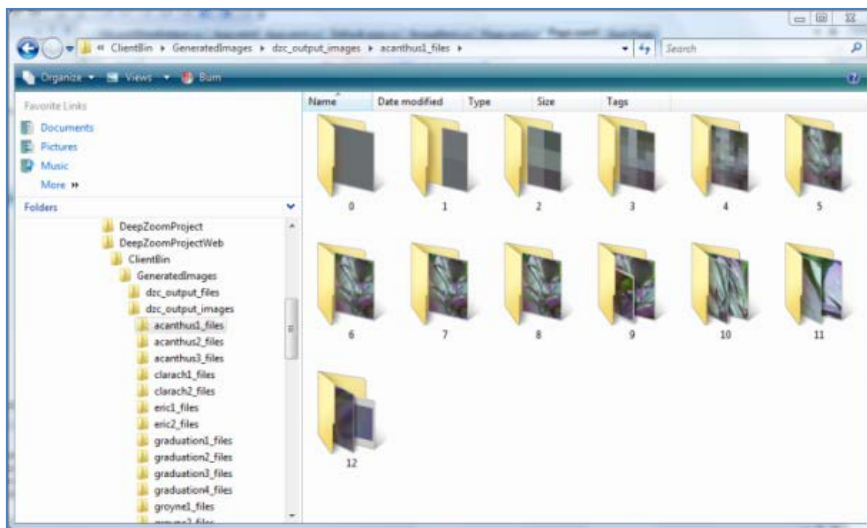
However, before looking at these I believe it would be interesting to look at some of the output that is created through the Deep Zoom Composer, as I believe this will help to demonstrate (at least in a small way) what is going on behind the scenes.

Once a project has been generated (exported) with the Deep Zoom Composer, a series of files are created by the program and placed within a set of sub-folders of a folder called *GeneratedImages* which itself is buried within the Deep Zoom project’s main folder.

Drilling into the *dzc\_output\_images* folder we find the several folders that have been created – one for each of the images incorporated into the Deep Zoom project. Within each of these there is a further series of folders which start to give a clue as to what the Deep Zoom composer has

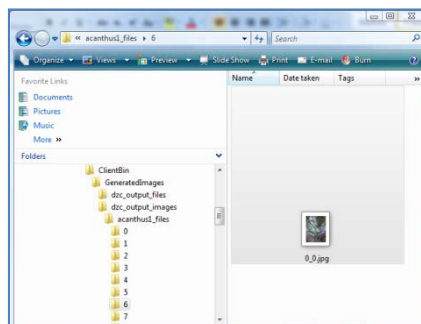
[34] <http://silverlight.net/GetStarted/> (Last accessed 15/08/2009)

done with the images and go some way to explaining how a Silverlight application manages to maintain performance:



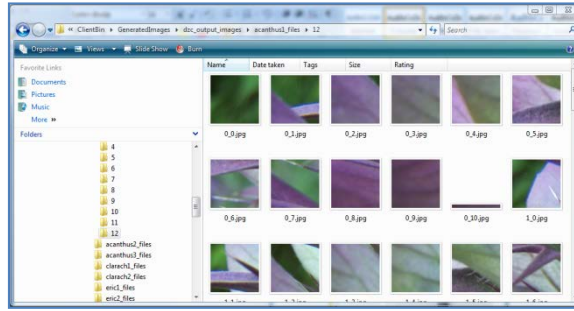
*Graphical display of the folders containing the “images” output from Deep Zoom Composer*

Looking inside the folder **0**, we find a single file (0\_0.jpg) which is so small at 633 bytes it can hardly be rendered – even when using the large icon view option of Windows Explorer. Looking inside folder **6** we still find a single image, but which is now 2.4Kb in size and is at least discernable as an image:



*Display of image from folder 6 in Deep Zoom output*

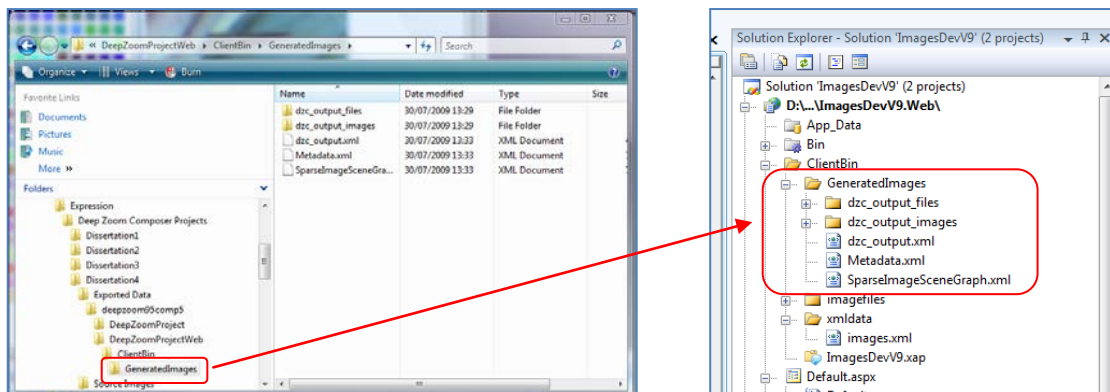
When looking inside folder **12** (the last in the series for this particular image) we find a veritable plethora of image files (each averaging 30Kb in size) – each one being a small part (or **tile**) of the original:



*Display of multiple images from folder 12 in Deep Zoom output*

From this we can deduce that what has been made available is a series of image tiles that represent a **zoom pyramid** - each level or floor of which can be used to provide different extents and resolutions as the user zooms in or out of the `<MultiScaleImage>` component.

Returning to the Images application, actually implementing the `<MultiScaleImage>` component is remarkably simple. Firstly, the `GeneratedImages` folder (and its contents) should be copied from the `ClientBin` folder in the Deep Zoom project into the `ClientBin` folder of the Silverlight application project in Visual Studio:



*Moving Deep Zoom output into Visual Studio project*

Following this, a `<MultiScaleImage>` element needs to be declared as in the code snippet overleaf (Note: The properties relating to *Margin*, *Height* and *Width* obviously need to be modified to suit each application):

```

<Grid x:Name="grdDeepZoom"
    Width="636" Height="306" VerticalAlignment="Stretch">
    <MultiScaleImage x:Name="msiDeepZoom"
        Source="..\GeneratedImages/dzc_output.xml"
        Margin="0,35,0,0" Width="616" Height="260"
        HorizontalAlignment="Center" VerticalAlignment="Top"/>
    <Canvas x:Name="cnvDeepZoomButton" Height="20"
        HorizontalAlignment="Left" Margin="500,0,0,0"
        Background="{x:Null}" Width="100" VerticalAlignment="Top"
        Opacity="0">
        <Button x:Name="btnDeepZoomIn" Height="20" Width="42"
            Canvas.Left="0" Template="{StaticResource ctmpZoomIn}"
            Click="btnDeepZoomIn_Click"/>
        <Button x:Name="btnDeepZoomOut" Height="20" Width="42"
            Canvas.Left="30" Template="{StaticResource ctmpZoomOut}"
            Click="btnDeepZoomOut_Click"/>
        <Button x:Name="btnDeepZoomRestore" Height="20" Width="42"
            Canvas.Left="60" Template="{StaticResource ctmpRestore}"
            Click="btnDeepZoomRestore_Click"/>
    </Canvas>
</Grid>

```

Ignoring for the moment the declaration of the `<Canvas>` and `<Button>` elements, we can see that only a single element declaration is needed for the `<MultiScaleImage>` component and that within this the `Source` property only needs to contain a reference to a single xml file – `dzc_output.xml`. Note: As part of this project I have used several books for reference. Unfortunately, some suggested that the source file generated from the Deep Zoom Composer tool that I would need to include in my Silverlight project would be a file called `info.bin`, in a folder called `dzcsample`. I suspect that some of these books were written when the Deep Zoom Composer was in its beta version because the release now available generates its output to the file detailed in the above code snippet.

An extract of the `dzc_output.xml` file is detailed in the code snippet below:

```

<?xml version="1.0" encoding="utf-8"?>
<Collection MaxLevel="8" TileSize="256" Format="jpg" NextItemId="20"
    xmlns="http://schemas.microsoft.com/deepzoom/2008">
    <Items>
        <I Id="0" N="0" Source="dzc_output_images/acanthus2.xml">
            <Size Width="2576" Height="1932" />
            <Viewport Width="6.0006654835847382" X="-0" Y="-0" />
        </I>
        <I Id="1" N="1" Source="dzc_output_images/acanthus1.xml">
            <Size Width="1932" Height="2576" />
            <Viewport Width="8.0008873114463182" X="-0" Y="-0.99999999999999989"
        />
    </Items>
    .
    .   Other <I> elements defined here
    .
    </Items>
</Collection>

```

Within this file we can see a collection of `</>` elements – one representing each of the images that collectively comprise the multi-scale image. The `</>` elements also contain child elements in which are declared the size and position (*Viewport*) of these images.

Running the application at this point (see <http://acanthus-advance.com/ImagesDevV4/Images.html>) we can see that the multi-scale image is displayed (after scrolling down to the bottom of the page), but that nothing can be done with it – that is, the image does not respond to any mouse click or mouse wheel events.

In order to overcome this I experimented with code provided by Laurence Moroney [35] which uses a combination of managed code and JavaScript in the page hosting the Silverlight application. This code effectively provides support for mouse wheel movement functionality to the whole application but as the `<MultiScaleImage>` component in the Images application is only one of many components (and is also placed at the bottom of the page) it gave rather unpleasant results in my test system - the mouse wheel moved the whole page up and down whilst simultaneously zooming the `<MultiScaleImage>` component in and out which, of course, was not an acceptable state of affairs. Also, when I transferred this test to a live web server, it wouldn't even do that which in itself was very frustrating. (Note: Although this version of the application is not the released version, I have included the Visual Studio project's source code for it on the CD accompanying this report).

Fortunately as part of its generated project output, the Deep Zoom Composer program also generates code (in C# only) that can be used to handle mouse events and by copying this into the project the project for my application, I was able to achieve the right results. (Note: As the code for this is not my own, and also rather long to include and describe in this report, I have instead included it in the Visual Studio project's source code on the CD accompanying this report. It should also be noted that the mouse wheel helper code generated (or supplied) by the Deep Zoom Composer is in C#, however an untested (by me) VB.Net version can be found in the Silverlight portal's forum pages. [36]).

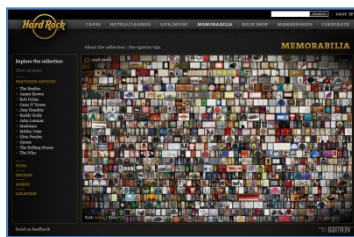
Finally, it is worth noting that within Deep Zoom Composer you have the option to generate output as either a **composition** or a **collection**. Although potentially hundreds of actual images are generated in either case (to support the tiling and zoom pyramid floors), a composition generates the output in such a way that these can only be handled by the Silverlight application as a single entity. Alternatively, a collection output retains the identity of each of the source images such that they can be addressed and manipulated individually within the application. Again, using the Hard

---

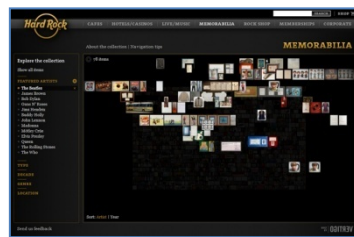
[35] Introducing Microsoft Silverlight 2 (2<sup>nd</sup> Edition) – Laurence Moroney. ISBN(13) 978-0-7356-2528-0. Microsoft Press – 2008. pp283-288.

[36] <http://silverlight.net/forums/t/19893.aspx> (Last accessed 20/08/09)

Rock Cafe Memorabilia page to demonstrate an example of this in action, the following three screen clips show the collection of images as they display a) at initial application load, b) during the transition to the display of just the Beatles images (initiated by clicking on the **Beatles** filter on the left-hand side of the page) and c) at the completion of the transition:



*Display of all Images*

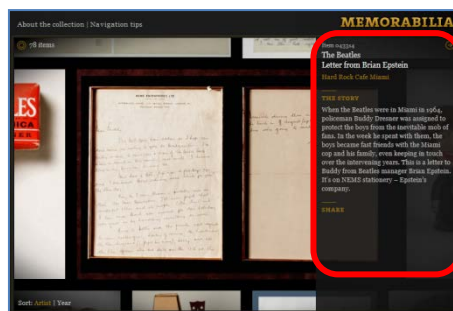


*Part of transition to "Beatles" Images*



*Transition completed*

By exploiting this image addressability, the functionality within the site also drives the display of information related to each image when that image is clicked or selected by the user – an example of this can be seen in the screen clip below:



*Demonstration of display of text associated with single image*

### 11.3 Flex (OpenZoom)

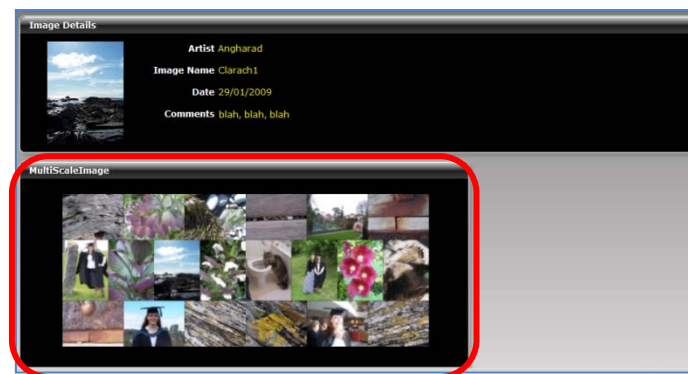
Throughout the development of the two applications that support this project, I have steadfastly resisted the temptation to include the use of components that are not provided as standard within the two manufacturer's published or released software – a) because of the potential cost implications and b) because it could lead to unfavourable comparisons being made. In addition, once one starts down that particular route it would be difficult to know where and when to stop.

However, given the significance of the functionality provided by Silverlight's *<MultiScaleImage>* component I felt that it behoved me to investigate if any comparable technology could be found that could be used to support multi-scale images within a Flex application. Searching through various sites I came across one [37] constructed by Daniel Gasienica, a Zurich-based student / developer who has developed (and is continuing to develop) a Software Development Kit that provides functionality similar to that provided by Microsoft's implementation.

[37] <http://gasi.ch/blog/> (Last accessed 20/08/09)

Daniel's implementation (known as **OpenZoom**) currently relies on the presence of the Flex SDK Release 3.3 although he is currently working on a version that will be backward compatible with the 3.2 Release. He has also worked on supporting both the Microsoft Deep Zoom image description format and that provided via Zoomify [38] (a commercial program used to develop zoomable images for Flash), as well as developing his own image description format.

Rather than amend my final incarnation of the Flex version of the Images application (primarily on the basis that his multi-scale image technology is not an Adobe supported facility), I created a development version to test Daniel's component [39], the output of which can be seen bounded in **Red** in the following screen clip (it is at the bottom of the page on the web site):



*Example of Daniel Gasienica's **OpenZoom** multi-scale image technology in a Flex Application*

Implementing an `<openzoom:MultiScaleImage>` component in a Flex application initially requires that the OpenZoom SDK be made available to the project by using the following steps in the Flex Builder (Eclipse) IDE:

- 1) Add a new linked resource:
  - a. Window > Preferences > General > Workspace > Linked Resources > New...
  - b. Enter OPENZOOM\_HOME in the Name: box
  - c. Enter the relevant location of the SDK folder in the Location: box, e.g. "D:\Developer's Workspace\Downloads\openzoom-sdk-0.4.2\openzoom"
  
- 2) Add this resource to the source path for the project:
  - a. Project > Properties > Flex Build Path > Add Folder...
  - b. Enter the relevant location of the SDK folder in the Location: box, e.g. "D:\Developer's Workspace\Downloads\openzoom-sdk-0.4.2\openzoom" (This will enter `${OPENZOOM_HOME}` as an additional source folder).
  
- 3) Add the following **namespace declaration** to the `<mx:Application>` element of your project:
 

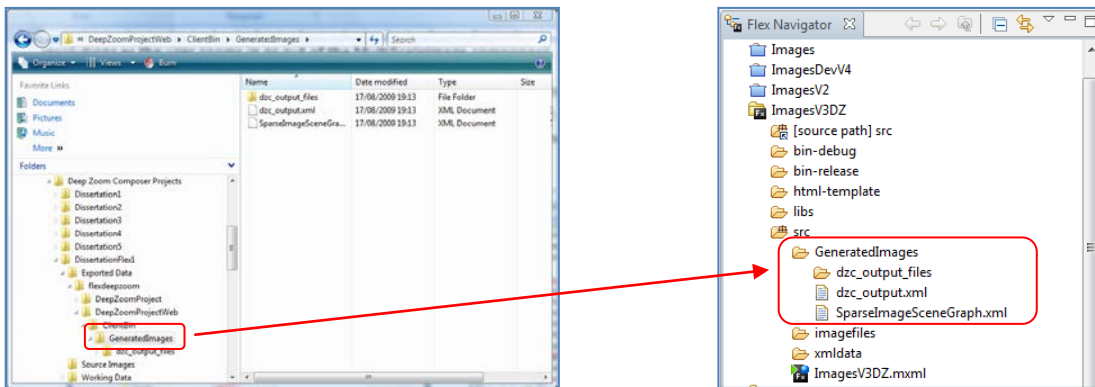
```
xmlns:openzoom="org.openzoom.flex.components.*"
xmlns:controllers="org.openzoom.flash.viewport.controllers.*"
xmlns:transformers="org.openzoom.flash.viewport.transformers.*"
```

[38] <http://zoomify.com/> (Last accessed 20/08/09)

[39] <http://www.acanthus-advance.com/FlexDZC/ImagesV3DZ.html> (Last accessed 20/08/09)



- 4) Copy the *GeneratedImages* folder (and its contents) from the *ClientBin* folder in the Deep Zoom project into the *src* folder of the Flex application project:



#### Moving Deep Zoom output into Flex project

The `<openzoom:MultiScaleImage>` component can then be declared as in the following code snippet:

```
<mx:Panel id="pnlMultiScaleImage" title="MultiScaleImage"
    x="10" y="918" width="620" height="286" layout="absolute">
    <openzoom:MultiScaleImage
        source="GeneratedImages/dzc_output.xml"
        width="580" height="210" x="10" y="20">
        <openzoom:transformer>
            <transformers:TweenerTransformer/>
        </openzoom:transformer>
        <openzoom:controllers>
            <controllers:MouseController/>
            <controllers:KeyboardController/>
        </openzoom:controllers>
    </openzoom:MultiScaleImage>
</mx:Panel>
```

From this we can see that we have simply declared a value for the source attribute of the `<openzoom:MultiScaleImage>` component together with some values for the component's dimensions and location. I have also included some empty element declarations which a) enhance the smooth transition between the zoom pyramid levels (the `<transformers:TweenerTransformer>`) and b) provide reactions to mouse and keyboard input - `<controllers:MouseController>` and `<controllers:KeyboardController>` respectively.

Although Daniel will admit that his efforts are still classed as work in progress, this current implementation is by no means insignificant. Although the `<openzoom:MultiScaleImage>` component only currently supports composite image output from the Deep Zoom Composer, (and this must be viewed as a significant difference between his and Microsoft's released and supported technology), he is currently working on a *DeepZoomContainer* class in which he aims to add support for Deep Zoom collection images.

## 11.4 Comparison

The performance of Daniel's component within my implementation of the component certainly compares favourably with that provided by the Silverlight component. However it is possible that, at least for the time being, larger implementations may highlight differences in response times and rendering quality that are not apparent in my implementation.

However, one significant difference between the two technologies that definitely goes in Daniel's favour is the support he has provided "out of the box" for both keyboard and mouse control. The volume of managed code required in the *page.xaml.cs* and *MouseWheelHelper.cs* classes in Microsoft's implementation is several orders of magnitude greater when compared to the declaration of the two empty MXML elements that are required in Daniel's implementation.

As with all home grown software, developers may be wary of using it for industrial strength commercial or mission critical applications, and from one perspective it might be more comfortable for many people if Adobe ultimately take over responsibility for supporting this initiative and formally encompass it within the Flex SDK. However, together with a full API reference set, Daniel has made all source code for the technology available under the MPL 1.1 / GPL 3 / LGPL 3 licensing schemes and so it is to be hoped that even if Adobe fail to take this facility on board, an OpenZoom development community will emerge.

However, until such time as Daniel's work is fully complete, (either through his efforts or those of others), Microsoft's implementation must remain as the market leader and will certainly draw many of those who wish to use this type of technology into the Silverlight fold.

## 12. User Experience

### 12.1 Comparison

Although my objective was to get each application to provide the same user experience, the reality is that without getting involved in some very complex and lengthy coding exercises (which I did not have time to undertake), I would never be able to achieve this. At the basic level, the two technologies do not provide exactly the same set of animations / effects, so reproducing in Silverlight an effect equivalent to Flex's *Blur* or a Flex equivalent of Silverlight's *SkewTransform* had to be passed over.

However, there are some other differences in the look and feel of the two applications that I think are worth noting.

- Within the Flex version I was able, through the use of the CSS code generated in the Flex Style Explorer, to significantly alter the style of the buttons (as can be seen in the left-hand screen clip below). This is because the Button class in Flex exposes many style-related properties which can be manipulated. The buttons in Silverlight on the other hand are fairly static in style (central screen clip below), as there are far fewer style properties exposed to the application creator. Although it is entirely feasible to create one's own images and get them to work as buttons, the effort involved in doing this is extensive. The right-hand screen clip below shows an example of customised buttons which are provided as part of the Deep Zoom composer output and which I copied into my application. Both products provide support for the customisation of the appearance of controls (known as skinning), through integration with "designer oriented" programmes such as Adobe CS3 and Microsoft Expression Design, but the effort involved is not trivial – especially for someone not trained as a designer.



Flex Buttons



Silverlight Standard Buttons



Silverlight Customised Buttons

- When using the Zoom In facility in the Flex version of the application, Flex can be easily instructed to automatically provide scroll bars to the side and beneath the zoomed image (see left hand screen clip overleaf). Without extensive programming effort (which I have not implemented) Silverlight cannot replicate this functionality. The best I could achieve with standard components was to either permanently display or hide the scroll bar component, and even when displayed, the context sensitivity of the scroll bars (to the degree of the zoom) is not available to the user resulting in an inability to use the scroll bars to pan around the whole image.



Flex Scroll Bars (note context sensitivity)



Silverlight Scroll Bars (no context sensitivity)

- Although it is impossible to show this through the use of static displays, through the use of storyboards the Silverlight application synchronises the animation of the detail image with the slider associated with it, e.g. when the Skew button is pressed, and the detail image animates towards a skewed image, the associated slider also animates within the same timeframe from left to right. I was unable to replicate this functionality in the Flex application, and although an example of this functionality is shown in [40], the effect is not very smooth.
- After spending much time with Silverlight to get the *<MultiScaleImage>* component to respond to mouse wheel movements, it became apparent that there was another significant difference between the two applications. The Flex version of the application is far more sensitive to the location of the mouse with the result that when the user places the mouse over the *<mx:DataGrid>* component, operating the mouse wheel only moves the images within that component. Likewise, the same sensitivity is shown when the user places the mouse over the list component. However, within the Silverlight version no such sensitivity exists, and no matter which component the user places their mouse over, a move of the mouse wheel causes the whole page to move up and down. I personally find this very annoying and would expect that other users would feel the same.

Although I firmly believe that by providing the same detailed application specification to an experienced Flex developer and experienced Silverlight developer, it would be possible to produce two applications that worked exactly the same, it is also my view that overall it is easier (quicker) to achieve a more pleasant user experience in Flex than Silverlight. I am sure that many Silverlight developers may complain about this assertion, but from the perspective of the learner / novice the amount of standard properties of the various component classes that are made available through the Flex framework allow for more professional results to be achieved in a shorter time.

---

[40] <http://javey.net/bike/map/player2/map.html> (Last accessed 18/09/09)

### 13. Conclusion

There is no doubt that both technologies investigated provide very professional and capable methods of building effective and visually appealing applications. With the possible exception of Silverlight's multi-scale image technology, most of the differences in the visual output were either minor (e.g. lack of Blur in Silverlight, lack of Skew in Flex) or I am sure could have been addressed with more time and effort.

Likewise, although there are differences in approach by both technology vendors in relation to learning support materials and development platforms there is nothing that really stands out as being a "show stopper" when one technology is compared with the other, although when considering the deployment and consumption of applications there is a fairly fundamental issue that could cause Silverlight developers grave cause for concern and which I will discuss shortly.

Although I stated at the beginning of this report that it was to be aimed (from a commercial perspective) specifically at the needs of "one man band" organisations, I think it is fair to say that any differences I identified would not influence larger development / web design organisations – other factors such as existing developer experience and investment in existing platforms would carry far more weight in any decision making process. With regards to small organisations such as my own, these factors are likely to weigh less, and certainly where no existing experience is relevant or no applicable platform investment has been made, then the decision on which technology should be used could well be as simple as "which will deliver what I want with the least effort and time?". From the experience I have gained thus far, my preference has to be with Adobe Flex.

Taking into account the fact that many independent software vendors like me (and specifically Access database developers) exploit the Microsoft Action Pack offering (which provides relatively cheap use of the Visual Studio IDE), many will be tempted from a purely financial perspective to work with Silverlight. Also it is possible that many developers already experienced in one or other of the .Net framework languages of C# or VB.Net would want to continue to capitalise on this. However, the analogy that keeps springing in to my head is that comparing these two IDE offerings is like taking a basic modern family car installed with all the gadgets you want (Flex Builder) and comparing it with a BMW with no speedometer or rev counter (Visual Studio). There is no doubt that both will get you to where you want to go, however Flex Builder will likely give you all you need for the journey and makes maintenance easier, but it's a bit rough around the edges. In comparison Visual Studio is smooth and very comfortable to drive, looks good but has some fundamentals missing – at least in its support for the development of Silverlight applications. I also don't believe the capital investment differences are significant, and nor do I believe that any developer already familiar with C# or VB.Net would take long to come to terms, and be effective, with constructing ActionScript code.

Of course, as with most analogies, this can sometimes fall short and if fully supported multi-scale image technology is vital to an organisation's web application then Silverlight almost certainly has to be the only serious contender at this time. I also have to recognise that although Microsoft has not been a player in this market for as long, it is working exceedingly hard at catching up and no doubt has ambitions to overtake Adobe as soon as possible – the rapidity with which it is releasing new versions of its Silverlight platform is testimony to this.

Whilst I firmly believe that all of the above conclusions and assertions are valid, providing testable validations of them was always going to be a difficult task, because by definition the outcomes of this project cannot be compared in a clinical or mathematically precise way – there are very few absolutes to test. The actual applications I have developed are, in effect, a means to an end, i.e. they have been developed in order for me to investigate and gain experience about the languages, development platforms, learning support materials, user controls, etc. I freely admit that in themselves they are not complex and are not breaking any new ground or introducing any revolutionary coding techniques. Indeed, I could argue that a base state of relative ignorance was one of the prerequisites for this project in order that the experiences gained would fuel the comparison in a real rather than artificial way. (Having said that, I do hope that others may find certain parts of the applications useful for their own work – be it commercial or academic).

The principal outcomes of this project were always going to be the comparisons made and the conclusions drawn in this report, but the argument could well be made that the factors, functions or facilities investigated were not objective, possibly irrelevant or were influenced by personal bias. Certainly on numerous occasions during the project, when consulting various web fora in my attempts to find a solution to a particular problem, I found many examples of “flame wars” between parties defending their own choice of platform (e.g. [41] and [42]) and the arguments used by both sides ranged from the technical, through the flowery to the plain biased. Obviously I will claim that maintaining my own sense of objectivity has been paramount during this project, but can I prove this categorically? Would presenting my findings to another developer and asking them to prove (or challenging them to disprove) my arguments be a reasonable attempt at validation – after all, how would one identify and quantify their particular experiences and prejudices and factor these into their conclusions? Other, more absolute, measures could theoretically have been used to support my arguments such as LoC (lines of code), application load times or numbers of classes, but these could only serve to help compare the results for one component of the project, i.e. the applications themselves. As already stated, the applications are neither complex (making such statistics irrelevant at best and misleading at worst) nor were they designed to be comparable from a coding efficiency perspective – the objective of this particular application development exercise

---

[41] <http://silverlight.net/forums/p/634/1076.aspx> (Last accessed 18/09/09)

[42] <http://stackoverflow.com/questions/20910/silverlight-vs-flex> (Last accessed 18/09/09)

was to investigate the efficacy of each technology in providing an understandable and usable development platform and their relative capabilities in supporting the delivery of a reasonable level of end-user functionality, rather than comparing their respective abilities to support the development of tight and coherent code.

Also, I fully recognise that during the authoring of any academic paper, one should always seek out other academic sources to support or challenge one's own conclusions. I can only surmise that the nature of the technologies I have investigated, i.e. they are either not relevant to the current research interests of others or are too commercially oriented, has so far meant that I have not been able to identify any published results from recognised academic or research sources that are in any way related or applicable to this project – therefore I am unable to offer any for comparison. (Of course I must concede that I might not have looked in the right places, but I only do so to demonstrate that I have already considered it as a possibility and accept it as feasible, rather than as an excuse).

The people contributing to the flame wars I came across had no reason to be non-partisan but reading through their contributions I could see that many valid points were made by both sides. However, there was one argument frequently lodged by the supporters of Flex which I believe was given far too much weight – that Flash player is currently installed on 95%+ of the world's PC systems whereas the Silverlight runtime has barely scratched the surface. Given that any Silverlight application automatically checks for the presence of the runtime and directs the user to the download site if it isn't there, I think that it is only a question of time before the respective populations of runtimes are comparable.

At least, I believe this will be true for Windows and MAC-based systems but it is on this point where I think there could be a real "show stopper" for using Silverlight for the development of applications that need to be truly browser agnostic. Linux and Unix-based systems don't currently support the Silverlight runtime in its native form and Novell, through the Mono project, has developed a formal relationship with Microsoft to provide an open source equivalent to Silverlight - known as Moonlight.

Moonlight aims to provide the same runtime functionality for Linux and Unix-based systems as the Silverlight runtime does for Windows and Mac-based systems but as can be seen in the project's roadmap [43], the currently released version is unfortunately only compatible with Silverlight 1.1. The beta version which promises compatibility with Silverlight 2 (the Silverlight version used to support this project) is not due for release until November 2009, and the Moonlight equivalent runtime that will be Silverlight 3 compatible does not even appear in the published plans. I can certainly confirm that although both versions of the Images application successfully loaded and

---

[43] <http://www.mono-project.com/MoonlightRoadmap> (Last accessed 18/09/09)

ran on various (popular) browsers on a Windows-based system (Internet Explorer, Opera, Firefox, Safari, and Google Chrome), the Silverlight version failed to load, let alone run, on a colleague's Ubuntu-based Linux system with the Moonlight 1.1 runtime installed.

One final interesting comparison between Flex and Silverlight is that a Google search for "controls for Silverlight applications" returns results that point to both community sites and commercial ones, i.e. sites for companies or organisations who provide custom controls on a commercial basis, as well as Microsoft managed sites. A similar search (replacing the word Silverlight with Flex) primarily returns results for Adobe managed sites, community sites and training organisations – in other words, there is something fundamentally different about the mores and attitudes of the marketplaces in which the two products sit.

In the final analysis, I believe that a commercial developer's choice of platform will depend on many issues other than straight technical functionality and that any existing commercial factors, intended target audiences and personal bias will in many cases outweigh any like-for-like capability comparison. However, I also believe that **both** technologies provide significant opportunities for use within secondary / tertiary education as a means of demonstrating new and alternative methods for the construction of web-based applications and that the factors that will influence commercial organisations are unlikely to worry academic institutions.



## 14. Critical Evaluation

As the saying goes – “hindsight is a wonderful thing”. Ignoring the fact that if I had my time to undertake this project again, I would wish to have managed that time more effectively, I have also considered where this project could have benefited from a different approach whilst trying to take into account that my starting point (or knowledge) would be the same.

One of my primary objectives in undertaking this project, indeed in undertaking the whole of the MSc programme, was to explore and understand more about the technologies that I could consider exploiting in order to provide alternative delivery mechanisms for the products and services that I, as an independent developer of database systems, provide to my customers. By doing a simple analysis of the time spent (and therefore the costs incurred) in developing a database using Microsoft Access (my current RDBMS of choice), I came to the conclusion that the vast majority of the effort I expend is on the construction of the user interface. For example, on a project that takes 20 weeks, the design of the tables and their key relationships is normally completed (to about 95%) in the first few days. The development of the queries to support the user forms and user reports (although ongoing throughout a project) normally takes no more than a week or two in total, and the rest of the time is spent fiddling about with the UI elements (both those in forms and for reports) and the back-end code to support them.

Looking back, I have indeed concentrated my time within this project investigating how the two respective technologies can be used to design and develop user interfaces using the various controls and components they both provide. However, I now recognise that this should not have been the primary focus as I failed to take into consideration some of the subtleties that are encompassed in delivering a true end user solution. Microsoft Access is, for me at least, a great product in that 99% of the functionality I am invariably asked to provide can be developed on and delivered through a single platform. The base data tables, queries, forms, reports and underlying code that are needed to develop an application are tightly coupled in the package, and so they really cannot be considered in isolation. When developing a form, for example, one must be mindful of its relationship to the underlying queries that supply its data, of the tables that support the queries, of its relationship to other forms, or sub-forms and the capabilities and required complexities of the VBA code that ties them all together. In addition, some functionality that needs to be carefully considered when combining two or more platforms (e.g. a UI in Flex using a web service to communicate with a back-end SQL Server installation) can be ignored when using a single package – the inter- and intra-component communication functions are invariably “just there”. It is the implications of working with this inter- and intra-component coupling that I feel I have not addressed sufficiently within this project, and that the analysis that I originally constructed of the time spent on developing a database application was far too simplistic. In other words, by focussing

primarily on the UI capabilities of Flex and Silverlight I failed to take into consideration the fact that a user interface is not an isolated component, but one that is part of a mix that has varying degrees of complexity.

The other naive assumption that I made at the start of the project was that the needs of an academic audience and commercial audience would be the same – at least in the context of these two technologies. I would like to believe that the information provided in this report could be of some reasonable value to an audience of new students who wished to gain an introductory understanding of the implementation of Rich Internet Applications through either or both of the technologies investigated. The aims of the **Product Overview**, **Development Platforms** and **Learning Support Materials** chapters were indeed set to support this and I also hope that the chapters on **Application Development** and **Multi-scale Images** could at least provide a reasonable context from which further learning could be undertaken. However, if I was to make a presentation of this project and its outcomes to the UK Access User Group (a bunch of very experienced and some might say hardened individuals) I believe that I would be inundated with many questions and comments targeted at the very shortcomings that I have highlighted above – they would be grateful for my time, but only to a point.

Obviously, I can and will address these shortcomings in work that I will do in my own time and for my own benefit (and hopefully also for the benefit of my current and future customers), but I must accept that if this project was to deliver against the objectives that were initially set, then further work remains to be done – particularly in the areas of inter-technology communication, (e.g. web services, local & remote database connections, etc.), multi-page applications, security and accessibility.

As I discussed in the **Project Overview** chapter, my initial idea was to compare Flex, Silverlight and JavaFX. However, it became very apparent very quickly that as my investigation would require me to undertake learning and develop my set of experiences from the very first base of each technology, there would not be time within the project scope to tackle all three. I could argue that the same reason (that the initial scope was too wide) also accounts for my not having time to address the other areas outlined above, but I cannot honestly say whether this failure is down to a mismanagement of time, lack of basic knowledge or a poorly reasoned proposal.

On the issue of comparing the two technologies, my objective has been to provide a non-partisan view of the advantages and disadvantages of each platform as I have found and understood them. Although I have been used to working primarily with Microsoft products over the years (and to a large degree am very comfortable with its offerings) and have had little experience of other suppliers' products prior to undertaking this MSc, I have attempted to maintain a consistency of

approach to both products and in some respects I believe my prior ignorance has helped – I had no extensive previous knowledge that would make my task easier for one technology over the other.

As I mentioned earlier in my report, I started the development of my application in Flex and the initial design of Images application was driven by a desire to see what Flex could do. The Silverlight version was then based on what I had already achieved in Flex. Given this approach, it is not unfair to ask whether the outcomes of this report would have been any different if I had started the project with an investigation of Silverlight. At the risk of being dubbed indecisive (again!), I think the proper answer is both yes and no. I think it is highly probable that the Images application would have looked different, e.g. the comparative difficulty in putting title text in a Canvas in Silverlight would probably have influenced me to not bother (or to not even think about it), and I then might have not even looked for a way to do it in Flex. Also I think there were many lessons learned through the Flex stage of the investigation which made the development in Silverlight easier, and if it had been the other way round I might have found the Silverlight development experience harder which then would have been reflected in both the final application and my opinions.

Counter to this, I think there are enough similarities between the two technologies that I would have most likely ended up comparing a good proportion of the ones I have compared. Bearing all of this in mind, it is tempting to think that the lack of an up-front formal definition of what the application should look like took something away from the comparison, i.e. if I had used a rigid “statement of requirements” then this might have led to the discovery of more glaring deficiencies in one or other of the products. I would contend however, that within any given set of application definitions, both technologies would fail and succeed in roughly equal proportions, i.e. not every technology can cope with every user desire and I hope this is properly reflected in this report.

## 15. Bibliography

- [1] **Introducing Microsoft Silverlight 2 - 2<sup>nd</sup> Edition.** *Laurence Moroney.*  
Microsoft Press. ISBN(13) 978-0-7356-2528-0.
- [2] **Silverlight 2 Unleashed.** *Laurent Bugnion.*  
SAMS. ISBN(13) 978-0-672-33014-8.
- [3] **Microsoft Expression Blend.** *Brennon Williams.*  
SAMS. ISBN(13) 978-0-672-32931-9.
- [4] **Beginning Web Development, Silverlight and ASP.Net Ajax.** *Laurence Moroney.*  
Apress. ISBN(13) 978-1-59059-959-4.
- [5] **Silverlight 2 Bible.** *Brad Dayley & Lisa DaNae Dayley.*  
Wiley Publishing. ISBN(13) 978-0-470-37500-6.
- [6] **Foundation Flex for Developers.** *Jas Jacobs with Koen De Weggheleire.*  
Friends of Ed. ISBN(13) 978-1-59059-894-8.
- [7] **Adobe Flex 3 – Training from the Source.** *Jeff Tapper, Michael Labriola and Matthew Boles with James Talbot.*  
Adobe Press. ISBN(13) 978-0-321-52918-3
- [8] **Adobe Flex 3 Bible.** *David Gassner.*  
Wiley Publishing. ISBN(13) 978-0-470-28764-4