.NET Enterprise Applications

Roland Hediger

January 16, 2014

Contents

Part I.

Prüfung 1

1. Basics,Intro,TFS

[1]

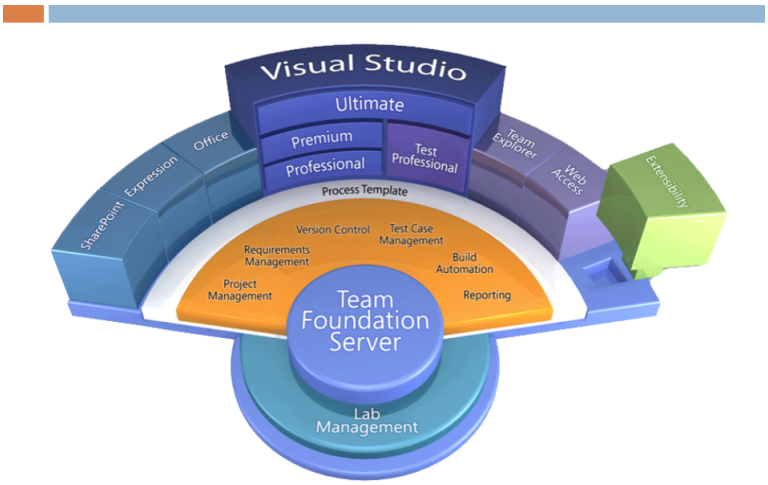


Figure 1.1.: TFS Features

1.1. TFS Setup

On Premises
- Server Setup
- Server Unterhalt + Backup
- Reporting (SQL Reporting Dienst)
- Sharepoint
- **Voll konfigurierbar im vergleich mit TFS**

TFS als SAAS Lösung
- Kein Server Setup
- Kein Unterhalt + Backup
- Saklierbar
- Data Storage off premises.

Additional TFS Features
- Sprint Planning
- Scrum Board
- Excel Reporting

General Dont waste too much time choosing the right process template. Out of the box experience is sufficient : Source Control, Work Item Types, Basic Reporting

[1] Author welcomes tokens of gratitude in the form of beer, if thy deem this documents helpful for thine test

2. Windows Presentation Foundation (WPF)

2.1. Goals of WPF

1. Unified approach to UI Docs and Media, replacing the individual technologies GDI, GDI+, Win32 (Winforms)
2. **Integrated vector-based composition engine** - One graphics engine for whole stack.
3. Declarative programming : Separate UI look and feel (XAML) from programming (Code Behind).
4. Ease of deployment : Allowing administrators to deploy and manage applications securely.

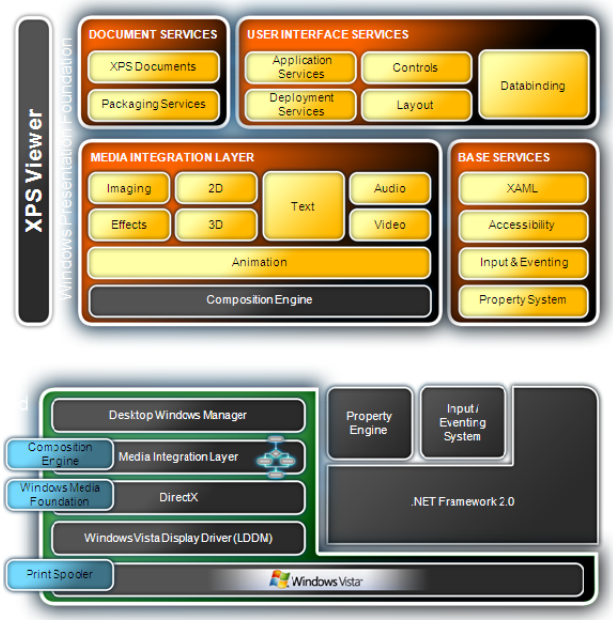


Figure 2.1.: WPF Architecture

2.2. WPF Features

2.2.1. Rendering

2.3. XAML

Extensible Application Markup Langunage, xml based to *instantiate and initialize objects with heirachichal relationships*

Listing 2.1: First XAML Example

```
1 <Window xmlns="http://schemas.microsoft.com/winfx/..."> <StackPanel
    HorizontalAlignment="Center" > <Image Source="Images/hello.jpg" Height="80" /> <TextBlock
    Text="Welcome to WPF!" FontSize="14"/> <Button Content="OK" Padding="10,4" />
    </StackPanel> </Window>
```

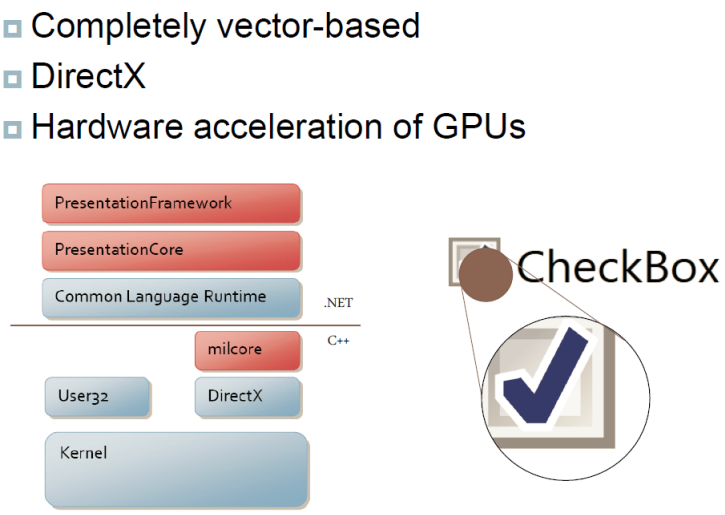


Figure 2.2.: figure

General Markup for windows : Build applications in simple declarative statements. Can be used for any CLR object heirarchy.Code and content are strictly separate, streamlines designer developer collaboration.

XAML vs Code Everything in XAML can also be done in code. The xml tags correspond to objects using their default constructors, and xml attributes correspond to the UI Object Properties.

```xml
<StackPanel>
    <TextBlock Margin="20">Hello</TextBlock>
</StackPanel>
```

XML

```
StackPanel stackPanel = new StackPanel();

TextBlock textBlock = new TextBlock();
textBlock.Margin = new Thickness(10);
textBlock.Text = "Welcome to WPF";
stackPanel.Children.Add(textBlock);
```

C#

Figure 2.3.: xaml vs c#

XAML Prefixes + Property Element Syntax

Listing 2.2: XAML Prefixes

```
\begin{lstlisting} [caption=XAML Property Element Syntax]
<Rectangle Width="20" Height="20"> <Rectangle.Fill> <LinearGradientBrush> <GradientStop
    Color="Red" Offset="0" /> <GradientStop Color="Blue" Offset="1" />
    </LinearGradientBrush> </Rectangle.Fill> </Rectangle>
```

XAML Compilation

UI Services

- Layout
- Controls Library
- Templates
- Styles and Resources

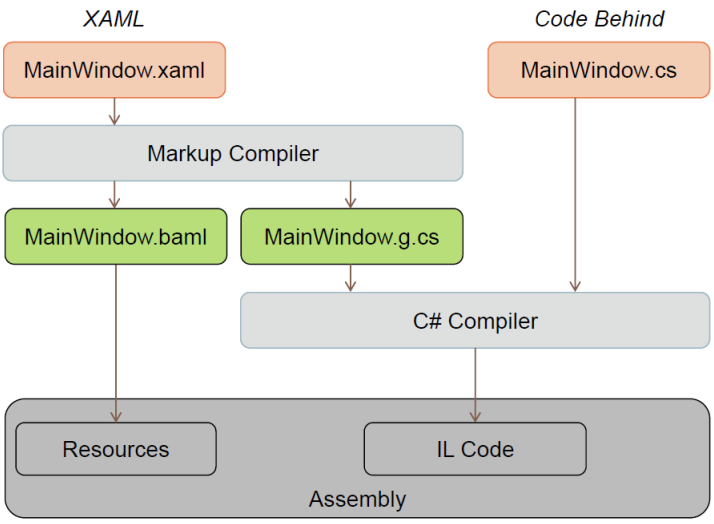


Figure 2.4.: figure

Listing 2.3: Combination of UI Services Example

```
<StackPanel>
<StackPanel.Triggers>
<EventTrigger RoutedEvent="Button.Click">
<EventTrigger.Actions>
<BeginStoryboard>
<BeginStoryboard.Storyboard>
<Storyboard>
<ColorAnimation To="Yellow" Duration="0:0:0.5"
Storyboard.TargetName="TheBrush"
Storyboard.TargetProperty="Color" />
<DoubleAnimation To="45" Duration="0:0:2"
Storyboard.TargetName="LowerEllipseTransform"
Storyboard.TargetProperty="Angle" />
...
</StackPanel.Triggers>
... remainder of contents of StackPanel, including x:Name'd
TheBrush and LowerEllipseTransform ...
</StackPanel>
```

Flexible Composition I can define an Item inside the content tag of an Item and it will be used as the parent item's content :

Listing 2.4: XAML Composition Example

```
 <Button Width="50">
<Button.Content>
<Image Source="images/windows.jpg"
Height="40"/>
</Button.Content>
</Button>
```

Attached Properties Allows a child element of an object to adjust properites of itsself in relation to the parent object (where it should dock, margins) which are only available due to the type of the parent element.

Listing 2.5: XAML Attached Properties Example

```
<DockPanel>
<Button DockPanel.Dock="Left" Content="Button" />
</DockPanel>
<Canvas>
<Button Canvas.Top="20" Canvas.Left="20"
Content="Button" />
</Canvas>
```

2.3.1. User Controls vs Custom Control

User controls are reusable compositions of other controls : Grid with items positioned on it the same way used many times over.

CustomControl Self explanatory - enhances existing control.

2.3.2. Layout Panels , Padding and Alignment

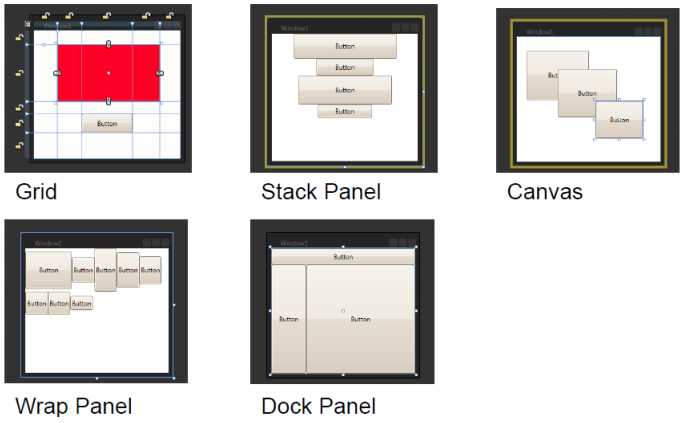
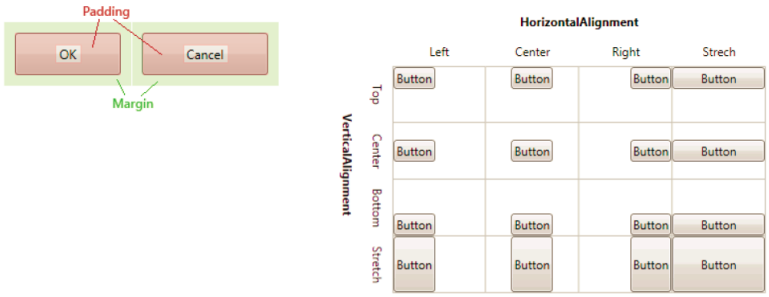


Figure 2.5.: WPF Panels



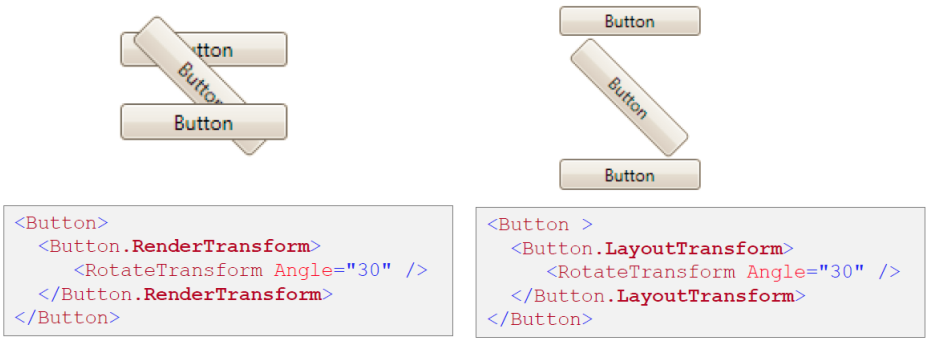

```
<Button HorizontalAlignment="Left"
        VerticalAlignment="Top"
        Margin="8,0,8,8" >
    Test
</Button>
```

Figure 2.6.: Padding and Alignment

- Use Alignment, Padding and Margin to position elements
- Avoid fix sizes for elements
- Do not misuse the Canvas Panel for fix positioning of elements (WinForms Style)

2.3.3. Transformations (Translations/Animations)

Element can be transformed in WPF LayoutTransform influences the layout, RenderTransform does not.




```
<Button>
  <Button.RenderTransform>
     <RotateTransform Angle="30" />
  </Button.RenderTransform>
</Button>
```

```
<Button >
  <Button.LayoutTransform>
     <RotateTransform Angle="30" />
  </Button.LayoutTransform>
</Button>
```

Figure 2.7.: figure

2.4. Databinding with XAML

```
<TextBlock Text="{Binding Path=Vorname}" />
```

- DataBinding synchronizes the values of two properties.
- Typically UI element is connected to an entity object (POCO / DAO) from a database.
- Binding cna be Uni or Bidirectional.
- A ValueConverter can adapt different data formats for syncronization.

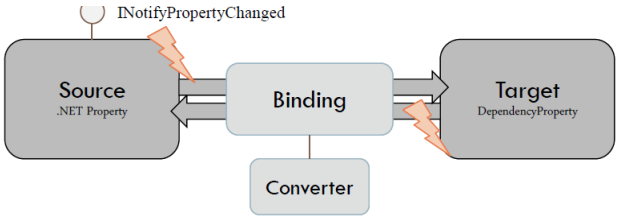


Figure 2.8.: Data Binding Diagram

2.4.1. Data Context

Every WPF element has a DataContext property.The DataContext is inherited to children.Allows the Binding to a Data-Object.The default source of a Bindings is always the DataContext.

Listing 2.6: Data Binding Example

```
<Button Content="{Binding Name}" />
DataContext = customer1;
```

Properties:

UpdateSourceTrigger ProperyChanged,LostFocus,Explicit(manual)

Mode Direction of Databinding : OneWay,TwoWay,OneWayToSource.

2.4.2. IPropertyChanged

Every Data-Object must implement INotifyPropertyChanged to allow the propagation of changes.

Listing 2.7: INotifyPropertyChanged implementation

```
public class Customer : INotifyPropertyChanged
{
```

```
private string _name;
public string Name
{
get { return _name; }
set
{
_name = value;
NotifyPropertyChanged("Name");
}
}
public event PropertyChangedEventHandler PropertyChanged;
private void NotifyPropertyChanged( string name)
{
if( PropertyChanged != null )
PropertyChanged(this, new PropertyChangedEventArgs(name));
}
}
```

2.4.3. ObservableCollection

Use ObservableCollections to allow the propagation of collection changes.

```
ObservableCollection<Auction> auctions
= new ObservableCollection<Auction>();
```

2.4.4. Value Converters

ValueConverters can change the format of the data in both directions:

Listing 2.8: ValueConverter Example

```
public class BoolToVisibilityConverter : IValueConverter
{
#region IValueConverter Members
public object Convert(object value, Type targetType, object parameter,
CultureInfo culture)
{
return (bool) value ? Visibility.Visible : Visibility.Collapsed;
}
public object ConvertBack(object value, Type targetType, object parameter,
CultureInfo culture)
{
throw new NotImplementedException();
}
#endregion
}
```

Listing 2.9: ValueConverter in XAML

```
<Window.Resources>
<conv:BooleanToStatusTextConverter
x:Key="booleanToStatusTextConverter" />
</Window.Resources>
<Button Content="{Binding IsOpen,
Converter={booleanToStatusTextConverter}}" />
```

2.4.5. Problems

DataBinding errors are written to the Debug Output.Use an empty ValueConverter to set breakpoints.

2.5. MVVM Pattern

Motivation

- Separation of GUI design ("style") and logic "behavior".Ability to use Expression Blend.
- No duplicated code to update views..No "myLabel.Text = newValue" sprinkled in code behind everywhere.

- Testability: Since your logic is completely agnostic of your view (no "myLabel.Text" references), unit testing is made easy.

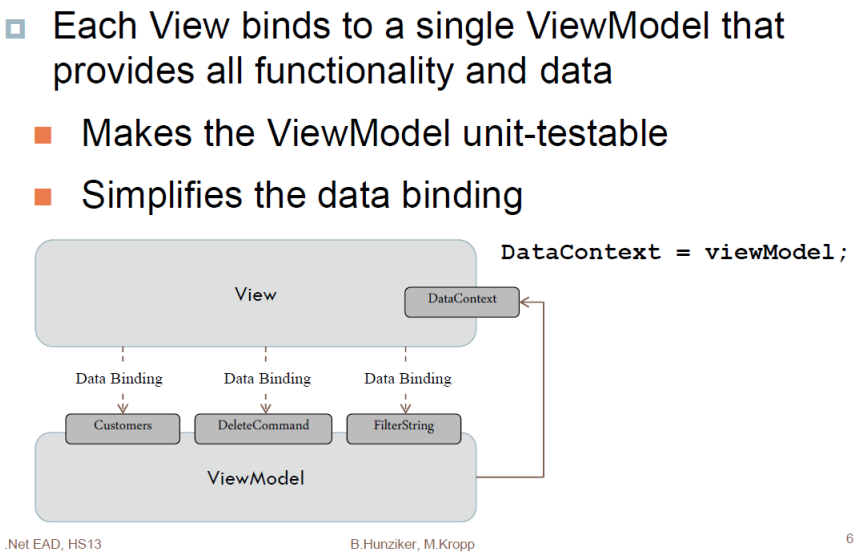


Figure 2.9.: MVVM Example

View(xaml) Usercontrol based, with xaml. Has minimal code behind. Datacontext is set to the associated View Model. No event Handlers. Data binding of view is set to view model.

View Model(c#) Implemenets INotifyPropertyChanged. Exposes ICommand, handles validation. Functions as an adapter class between view and model, as a result, it is testable.

Model POCO free of all WPF.

2.5.1. Commands

Commands are used to bind UI actions to the ViewModel functionality. A command implements 3 functions of the ICommand interface :

```
Execute(object param);
canExecute(object param);
even CanExecuteChanged;
```

Commands can be used on different UI Elements.

Listing 2.10: ICommand Implementation Example

```
public class FooCommand : ICommand
{
 public Action<object> ExecuteDelegate { get; set; }
 public Func<object, bool> CanExecuteDelegate { get; set; }

 #region ICommand Members

 public bool CanExecute(object parameter)
 {
 return CanExecuteDelegate(parameter);
 }

 public event EventHandler CanExecuteChanged;

 public void Execute(object parameter)
 {
 ExecuteDelegate(parameter);
```

```
    }

19

    #endregion
    }
```

3. Entity Framework

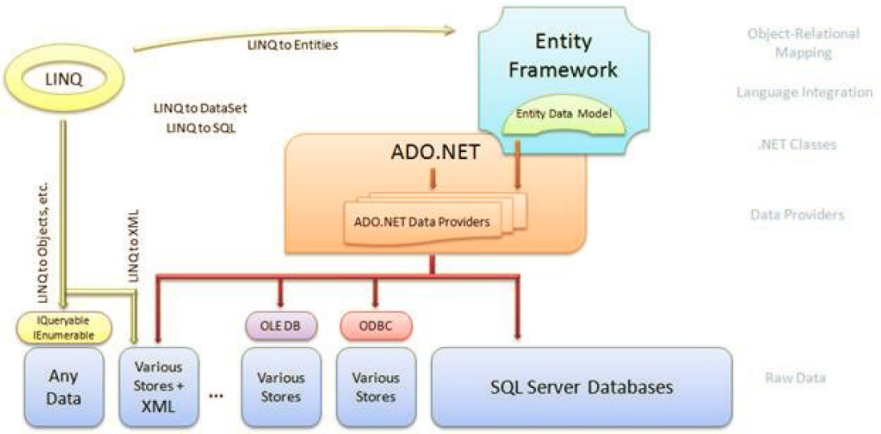
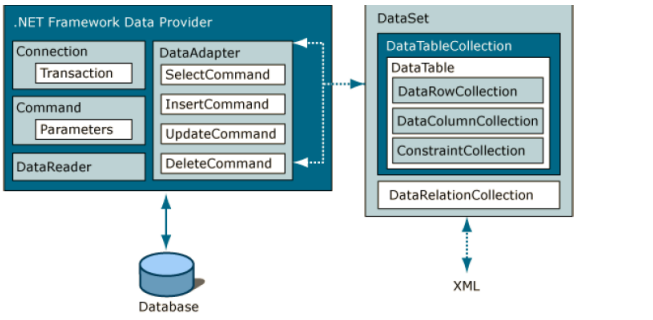


Figure 3.1.: Underlying Architecture 1



.NET Framework data provider	Description
.NET Framework Data Provider for SQL Server	Provides data access for Microsoft SQL Server version 7.0 or la
.NET Framework Data Provider for OLE DB	For data sources exposed by using OLE DB. Uses the System.D
.NET Framework Data Provider for ODBC	For data sources exposed by using ODBC. Uses the System.Da
.NET Framework Data Provider for Oracle	For Oracle data sources. The .NET Framework Data Provider fo
EntityClient Provider	Provides data access for Entity Data Model (EDM) applications.

Figure 3.2.: Underlying Architecture 2

3.1. ADO.NET

klassenauflistung:

- SqlConnection
- SqlCommand
- SqlCommandBuilder
- SqlDataAdapter
- SqlDataReader

- SqlException
- SqlParameter
- SqlBulkCopy
- SqlTransaction

3.1.1. Examples

Listing 3.1: DataSet Example

```
using (SqlConnection cn = new SqlConnection("<connection string>") {
  cn.Open();
  SqlDataAdapter ad = new OleDbDataAdapter("select * from products", cn);
4   DataSet ds = new DataSet();
    ad.Fill(ds,"Products");
    foreach(DataRow dr in ds.Tables["Products"].Rows) {
    Console.WriteLine(dr["ProductName"]); }
  }
```

3.2. Intro to Entity Framework

EF Maps conceptual model to physical tables, uses LINQ. Has features like change tracking identity resolution, lazy loading and more.

LINQ to SQL Lightweight no mapping.Replaced by EF but still supported.

NHibernate Open Source like Entity Framework, uses LINQ.

3.3. Architecture / Structure of EF

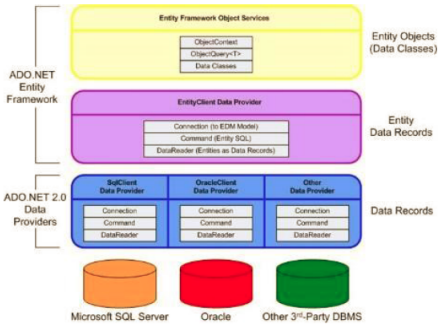


Figure 3.3.: EF Architecture

EDMX Files

Object Context (DB Context)

```
Methods:
SaveChanges()
CreateObjectSet() (->LINQ-Query)
Attach() / Detach()
efresh()
Properties:
Connection
ObjectStateManger
```

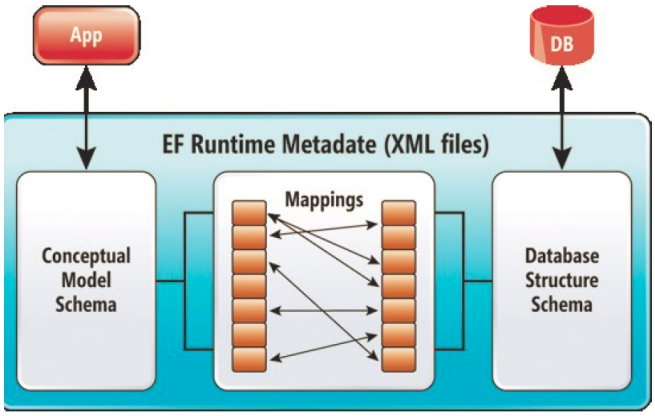


Figure 3.4.: EDMX Files

```
<?xml version="1.0" encoding="utf-8"?>
<edmx:Edmx Version="2.0" xmlns:edmx="http://schemas.microsoft.com/ado/2008/10/edmx">
  <!-- EF Runtime content -->
  <edmx:Runtime>
    <!-- SSDL content -->
    <edmx:StorageModels>...</edmx:StorageModels>
    <!-- CSDL content -->
    <edmx:ConceptualModels>...</edmx:ConceptualModels>
    <!-- C-S mapping content -->
    <edmx:Mappings>...</edmx:Mappings>
  </edmx:Runtime>
  <!-- EF Designer content (DO NOT EDIT MANUALLY BELOW HERE) -->
  <Designer xmlns="http://schemas.">...</Designer>
</edmx:Edmx>
```

Figure 3.5.: EDMX Files

Lazy Loading Default, Eager loading - load everything - must be specified explicitly.

StoredProcedure mapping Usually sucks, maps to a single entity with one method to fetch the result of stored procedure, as dataset.

3.4. "Variations" of Entity Framework

Database First : All code for Entities generated from database. (Model)

Code First No .edmx files, DB Schema and Entity Objects / Model from code.

Model first Uses UI Designer, and edmx file with T4 Templating to generate Entity Objects and Context.

3.5. Entity Objects

Notes Important to consider dev performance vs actual runtime performance because ORM means overhead.

Design Consderations
- Encapsulate DAL implementation? (technology, connections, queries, structure)
- Technology (ADO.NET vs. EF vs. .....)
- Logical – physical model mapping?
- Performance & scalability
- Batching: reduce round-trips (performance), Bulk insert/update: for large volumes
- Use Views?
- Use Stored Procedures?

Benefits of ORM
- Decouples logical model from database model
- Consistent, independent query language

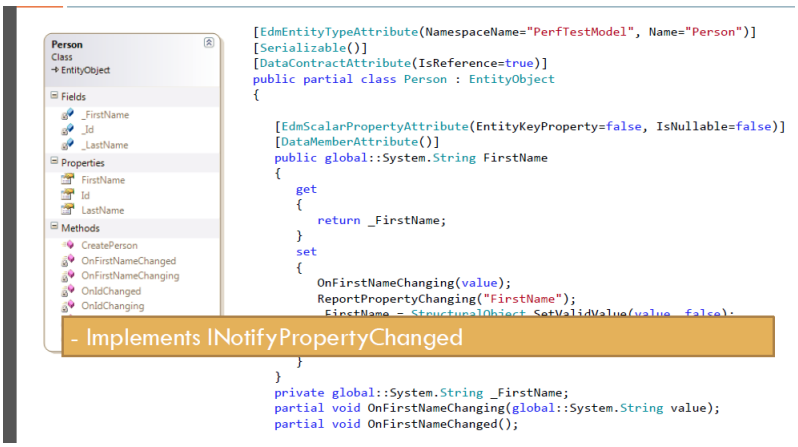


Figure 3.6.: Entity Framework Object using Model based approach

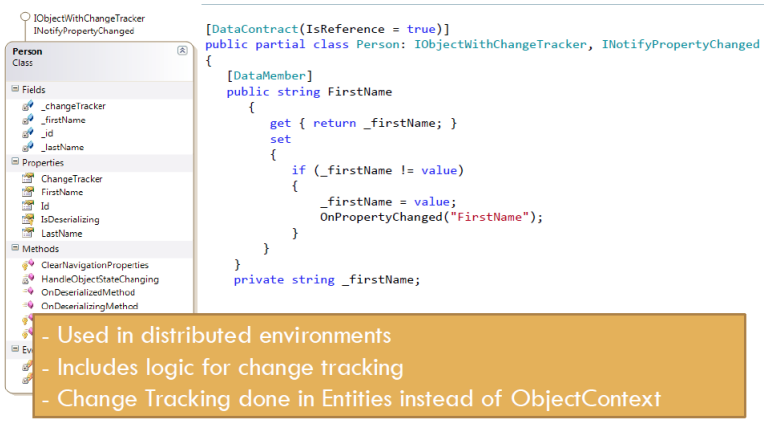


Figure 3.7.: Object with Change Tracking

- Rich designer support
- Most common patterns (1-to-many, many-to-1, many-to-many, self-referencing, inheritance,
- Slower than ADO.NET core
- Must regenerate EDM after DB changes

3.6. First Steps

Listing 3.2: Initial Entity Framework Example

```
// create dbcontext
AuctionContext context = new AuctionContext();
// get all auctions from database
 var myAuctions = context.Auctions;
// get all open auctions
var openAuctions = context.Auctions.Where(a => a.EndTime < DateTime.Now);
// add new auction to database
var newAuction = new Auction() {          } context.AddToAuctions(newAuction);
    context.SaveChanges();
```

4. Dependency Injection with Unity

4.1. What is a dependency?

Dependenencies in UML :
Common Dependencies in Enterprise Apps:

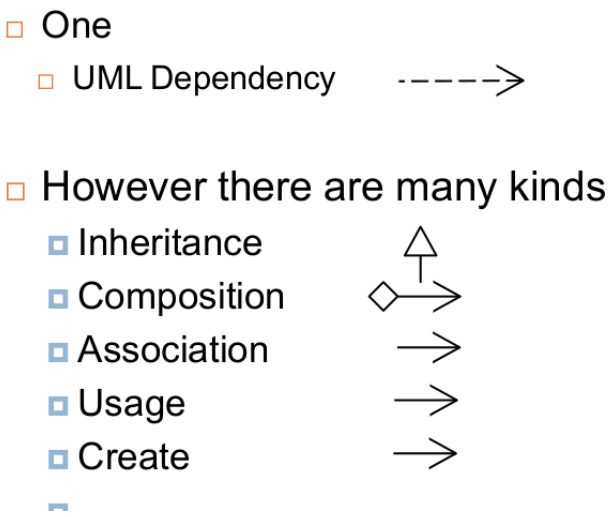


Figure 4.1.: Depedency examples in UML

Application Layers Data Access Layer & Database dependencies, Dependencies between business layer, and DB Layer.

External Services and Components Web Services, third party components.

.NET Components File Objects / Http Objects (MVC) - HttpContext, Session.

4.1.1. Tightly Coupled Dependencies (Bad)

Listing 4.1: Bad DI Example

```
public class CustomerService : ICustomerService {
 #region ICustomerService Members
 public CustomerDTO GetACustomerFrom(int id)
{
    CustomerDTOMapper dtoMapper = new CustomerDTOMapper();
    CustomerRepository custRepository = new CustomerRepository();
    return dtoMapper.mapFrom(custRepository.getFrom(id));
}
```

Problems :

- Code is tightly coupled,difficult to isolate when testing, difficult to maintain. Changing out components not possible.

Solutions:

- Inversion of Control - Hollywood principle.
- Dependency injection.

4.2. What is Inversion of Control

Higher level modules should not depend on lower level modules. Both should depend on abstractions (interfaces or abstract classes).*Abstractions should not depend on details*

4.2.1. IOC Implications

- Layers, modulariziation
- Interface based programming.
- Interface in seperate package to implementation.
- Implementations fetched through dependency injection.

4.2.2. IOC Advantages

- Increase loose coupling
  - Abstract interfaces don't change
  - Concrete classes implement interfaces
  - Concrete classes easy to throw away and replace
- Increase mobility
- Increase isolation
  - decrease rigidity
  - Increase testability
  - Increase maintainability

4.3. What is dependency injection

Purpose How do we wire up concrete interfaces? DI gives us the ability to inject external dependency into component

DI is a form of IoC, where implementations are passed into an object through constructors/setters/service look-ups, which the object will 'depend' on in order to behave correctly

4.3.1. Dependency Options

Option 1 – Factory User depends on factory,factory depends on destination.

Option 2 – Locator/Registry/Directory The component still controls the wiring. Instantiation Sequence.Dependency on the Locator.

Option 3 – Dependency Injection An assembler controls the wiring

4.3.2. Ways to inject dependencies

Constructor Injection

```
public class CustomerService : ICustomerService
{
    #region DI

    private ICustomerRepository repository;
    private ICustomerDTOMapper mapper;

    public CustomerService(
        ICustomerRepository repository,
        ICustomerDTOMapper mapper)
    {
        this.repository = repository;
        this.mapper = mapper;
    }
```

Figure 4.2.: figure

```
public class CustomerService : ICustomerService
{
    private ICustomerRepository customerRepository;
    public ICustomerRepository CustomerRepository
    {
        get
        {
            return customerRepository;
        }
        set
        {
            customerRepository = value;
        }
    }
```

Figure 4.3.: figure

Setter Injection

Method Injection

```
public class CustomerService : ICustomerService
{
    public ICustomer GetCustomerDetails
        (
            ICustomerRepository repository,
            int id
        )
    {
        ICustomer customer = repository.GetFrom(id);
        return customer;
    }
```

Figure 4.4.: figure

4.4. Pros and Cons of DI

+ Loosely Coupled
+ Better Testing
+ Allows IOC Container

4.5. IOC Container Services

- Service Locator - Finds implementation of Interface based on wiring.
- Managing lifetime of dependencies. (Singleton, per http request etc)
- Automatic injection.
- Wiring config.

4.6. IOC based Injection

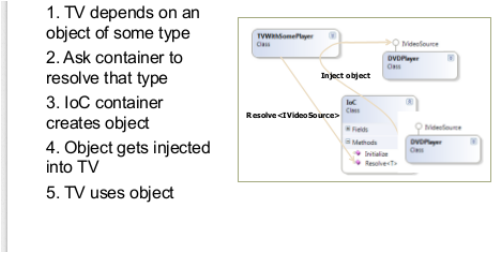


Figure 4.5.: IOC based Injection

4.7. How to use Unity

Unity container creates object instances during resolution of types.Container must be asked to resolve a type.

Listing 4.2: DI Unity First Example

```
IUnityContainer container = new UnityContainer();
IVideoSource source = container.resolve<IVideoSource>();
IEnumerable<VideoSource> sources = container.ResolveAll<IVideoSource>();
```

4.7.1. Resolving Dependencies (Wiring)

Dependencies are wired up for resolved types
Properties marked as [Dependency]
Constructor marked [InjectionConstructor]
Methods marked [InjectionMethod]
Unity can also wire up dependencies on objects not created by container

		Constructor	Properties	Method
Wiring Methods :	Resolve	Yes	Yes	Yes
	BuildUp	No	Yes	Yes

- Container has registration for object resolution Maps interfaces and abstract base classes to
- Concrete types
- Existing object instances
- Performed only once per container instance Can Use code or configuration Register multiple types based on names

```
IUnityContainer container = new UnityContainer();
container
 .RegisterType<TVWithSomePlayer, TVWithSomePlayer>()
 .RegisterType<IVideoSource, DVDPlayer>()
 .RegisterInstance<ILog>(someLogObject);
```

Figure 4.6.: Sample Registration using Unity

```
<configSections>
 <section name="unity"
      type="Microsoft.Practices.Unity.Configuration.
            UnityConfigurationSection,
            Microsoft.Practices.Unity.Configuration"/>
</configSections>
```

```
<unity>
 <namespace name="Ploeh.Samples.MenuModel" />
 <assembly name="Ploeh.Samples.MenuModel" />
 <containers>
  <register type="IIngredient" mapTo="Steak" />
 </containers>
</unity>
```

Figure 4.7.: Sample Registration in App.config

4.8. Managing Lifetimes of Objects with Unity

- Control the lifetime of objects created by Unity Use LifetimeManager derived classes during registration

- Container controlled: singleton
- Transient: new every time, resolve is called (default)
- Externally managed: keeps weak references
- RegisterInstance method: Singleton not created by container.

Part II.

Prüfung 2

5. ASP MVC4

5.1. Advantages of MVC

- A new option for ASP.NET, not a replacement for ASP.NET WebForms
- Simple way to program ASP.NET
- Easy testable
- Much control over your ¡html¿
- Much control over your URLs
- Can be used as a base infrastructure for SPA
- Plays well with others : Entity Framework, Unity, Unit Testing (Single Page Apps)

5.2. Naming Conventions

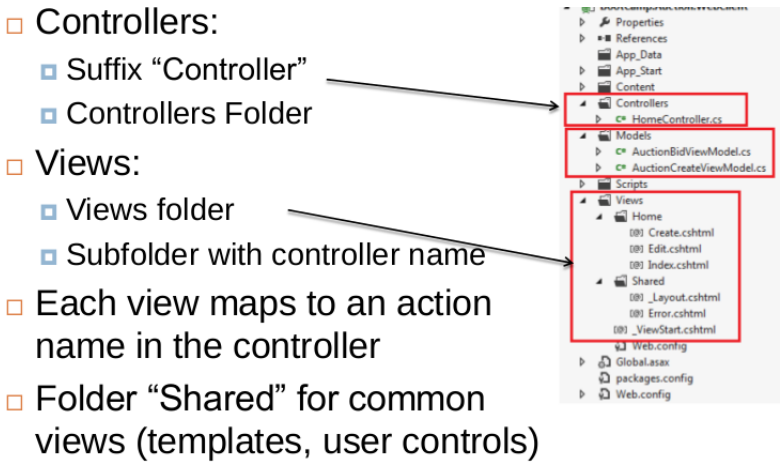


Figure 5.1.: MVC Naming Conventions

5.3. MVC Sequence Diagram - How it works

5.4. Controllers

- Controllers handle all incoming requests
- Retrieve data from storage
- Store posted data in storage (http post)
- Pass the data to a View to generate the
- HTML/CSS/JavaScript for display
- Controllers can implement REST APIs (derive controller from ApiController)

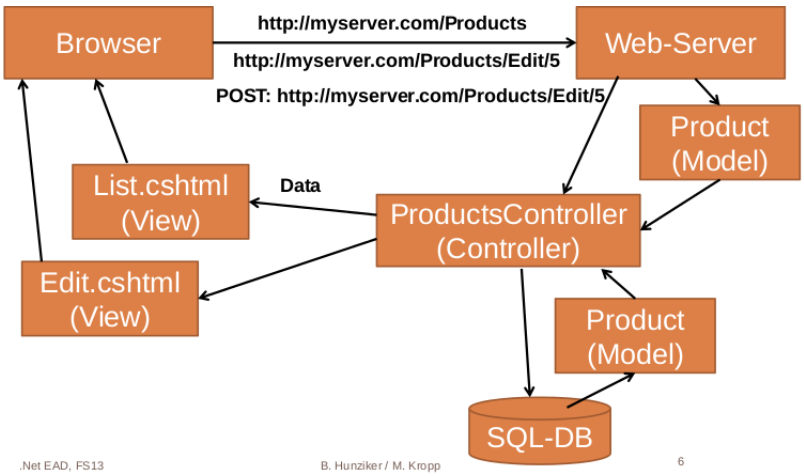


Figure 5.2.: MVC Architecture

- The simplest controller just returns HTML to the Web browser
- IMPORTANT: By default, all public methods in a controller class can be called from the Web browser
- Controllers give a ResultObject as return value and can be tested without the need for a webserver - Normal object.

5.4.1. Controller Action Types

Return Type	Content	Example
ViewResult	Html-Page	return View();
PartialViewResult	Part of HTML Page	return PartialView()
RedirectResult	Redirect to other page	return Redirect("url");
FileResult	Binary data	return File("path");
JsonResult	Serialized JS Objects	return JSON(MyPOCO);
JavaScriptResult	JS (AJAX)	return JavaScript("alert('hi')");

5.4.2. Controllers and AJAX

Ajax calls can also be handled by the controller These are also public methods:

```
 View:
@Ajax.ActionLink("Click me", "Click", null)
Controller:
public JavaScriptResult Click()
{
return JavaScript("alert('Hallo World');");
}
```

5.5. Passing Data between Controller and View

This part was not taken entirely from the slides because it was badly explained.

1. I can return a view with the View(Modelobject) method as follows :

```
  public ActionResult Index()
{
return View(db.Item.ToList());
}
```

This looks for view index.cshtml under views (matches method name). The parameter is the model object. Unless otherwise specified the view expects either no parameter or a parameter of type object, which can be accessed

5.5.1. Viewbag

This is referred to in the slides as the untyped variant. It IS NOT strictly speaking a variant. You are able to send data to the View using a model as above, and fill up the viewbag with objects (properties) at the same time.

Listing 5.1: ViewBag Example

```
ViewBag.Persons = new SelectList(context.People, "Id", "Name");
//In View
@Html.DropDownListFor(model => model.SellerId, (SelectList)ViewBag.Persons)

//Combined Variant
//Controller
  public ActionResult Index()
{
Object blab = new Object(); //can be anything
Viewbag.bla = blab
return View(db.Item.ToList());
}

//In View
<h2> Viewbag.blab.toString</h2>
@Html.DropDownListFor(model => model.SellerId, (SelectList)ViewBag.Persons
```

5.6. Razor View Engine

Asp.Net classic view engine can be used but replaced by Razor. Html tags implicitly identified and cause the code block to end. If it doesnt work one can manually force a code block :

Listing 5.2: Normal Razor ASP.NET MVC

```
<h2>Products</h2>
<ul>
   @foreach (var p in products) {
    <li> @p.ProductName </li>
   }
<li>
```

Listing 5.3: Specified Code Block ASP.NET MVC

```
\begin{lstlisting}
//@{code here} html stuff here.
<div>
<b>
@if (HttpContext.Current.Request.IsAuthenticated)
{
@:Change your password:
}
else
{
@:Please change your initial password:
}
</b>
</div>
```

5.7. View Helpers

Methods that you can use in the view to generate HTML tags that correspond with properties of @model.

```
@Html.ActionLink( Edit Record , Edit , new {Id=3}):
//generates:
<a href= /Home/Edit/3 >Edit Record</a>
@Html.LabelFor(model => model.FirstName):
//generates:
<label for="FirstName">FirstName</label>

@Html.EditorFor(model => model.FirstName):

//generates
<input class="text-box single-line" data-val="true"
data-val-required="The FirstName field is required."
id="FirstName" name="FirstName" type="text" value="" />
```

5.8. Layout

This was also badly explained in the slides. As we learn in web frameworks we need a template for every view. The template itsself is a view in ASP.NET mvc - layout.cshtml. The name for the template is specified in _ViewStart.cshtml, doesnt have to be named layout.

The layout also has access to the Viewbag - **Whatever I put in the viewbag in a Controller method, can be displayed in the layout in non dynamic parts, if the property doesnt exist - an empty String is displayed (null)** Example :

_Layout.cshtml

```
<html>
<title>@ViewBag.Title</title>
<body>
  @RenderBody()
</body>
</html>
```

Index.cshtml

```
@{
    ViewBag.Title = "Your Page Title";
 }
<div>Hello World!</div>
```

Figure 5.3.: Layout Example

Requirement : Layout must contain @RenderBody that is all

5.9. Routing

Listing 5.4: Routing Example

```
App_Start/RouteConfig.cs
 routes.MapRoute(
"Default", // Route name
"{controller}/{action}/{id}", // URL with parameters
new { controller = "Home", action = "Index", id = UrlParameter.Optional } // Default
);
```

First route that matches url pattern "wins".

5.10. Model binding

Just like in Spring MVC or JSF the properties of the Model Object can be bound to form fields provided the fields are given suitable names so that the model binder knows what to bind. This is taken care of with the html helpers above. In other words manually making form fields is generally a bad idea.

If you really want to you can extract the values manually from the Request object :

Listing 5.5: Manual reading of form values (not recommended but just in case)

```
public ActionResult Create()
```

- The MVC model binder provides a simple way to map posted values to a .Net Framework type passed as an action parameter (Brute-Force!)

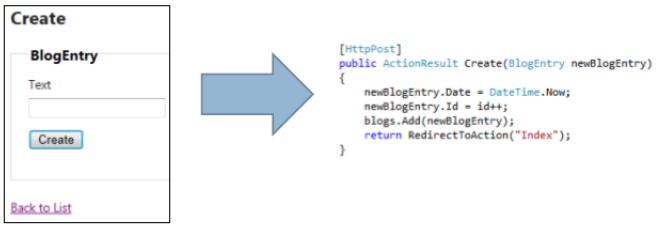


Figure 5.4.: Model binder

```
{
var product = new Product() {
AvailabilityDate =
DateTime.Parse(Request["availabilityDate"]),
CategoryId = Int32.Parse(Request["categoryId"]),
Description = Request["description"],
Kind =
(ProductKind)Enum.Parse(typeof(ProductKind),
Request["kind"]),
Name = Request["name"],
UnitPrice = Decimal.Parse(Request["unitPrice"]),
UnitsInStock =
Int32.Parse(Request["unitsInStock"]),
};
// do work
}
```

Listing 5.6: Model binding with primitive values

```
public ActionResult Create(
DateTime availabilityDate, int categoryId,
string description, ProductKind kind, string name,
decimal unitPrice, int unitsInStock
)
{
var product = new Product() {
AvailabilityDate = availabilityDate,
CategoryId = categoryId,
Description = description,
Kind = kind,
Name = name,
UnitPrice = unitPrice,
UnitsInStock = unitsInStock,
};
// do work
}
```

Listing 5.7: Model binding sensibly

```
public ActionResult Create(Product product)
{
// do work
}
```

5.10.1. Hidden values

@HiddenFor(model=¿model.property) in the view. Generates a hidden form field that is also mapped to the model later.

5.10.2. Step by Step behind the curtain

Value Sources: querystring parameters, form
fields and route data
Evaluation order:

- Previously bound action parameters, when the action is a child action
- Form fields (Request.Form)
- The property values in the JSON Request body (Request.InputStream), but only when the request is an AJAX request
- Route data (RouteData.Values)
- Querystring parameters (Request.QueryString)
- Posted files (Request.Files)

5.11. Sessions

Listing 5.8: using session in controller

```
1       public ActionResult Index(SessionAspDotNetMvcModel data)
          {
              // Assign value into session
              Session["SessionUserID"] = data.sUserID;
              // Redirect view
6             return RedirectToAction("EmployeeSection");
          }
```

Session States :
InProc: In memory on the Web server (default)

StateServer: Separate process called the ASP.NET state service

SQLServer: SQL Server database Custom: Custom storage provider

Off

6. Azure

Cloud service : On demand , self service, and runs on pay per use principal. Why cloud : Outsourcing of infrastructure, scalability.

Cloud Service Types:

IaaS: Infrastructure as a Service - Virtual Machines (Virtual Private Server)

PaaS: Platform as a Service -

- Mobile Services
- Cloud Services
- SQL Databases & Reporting
- Storage
- Service Bus
- Virtual Networks
- Web Sites

Important Characteristics : Usually automates deployment with possibilities for customization through scripts that you can add to the deployment process. Services like openshift for example deploy when you commit your project to the git repo.

[SaaS:] Software as a Service : Site builders and so on fully managed.

6.1. Data Center Architecture

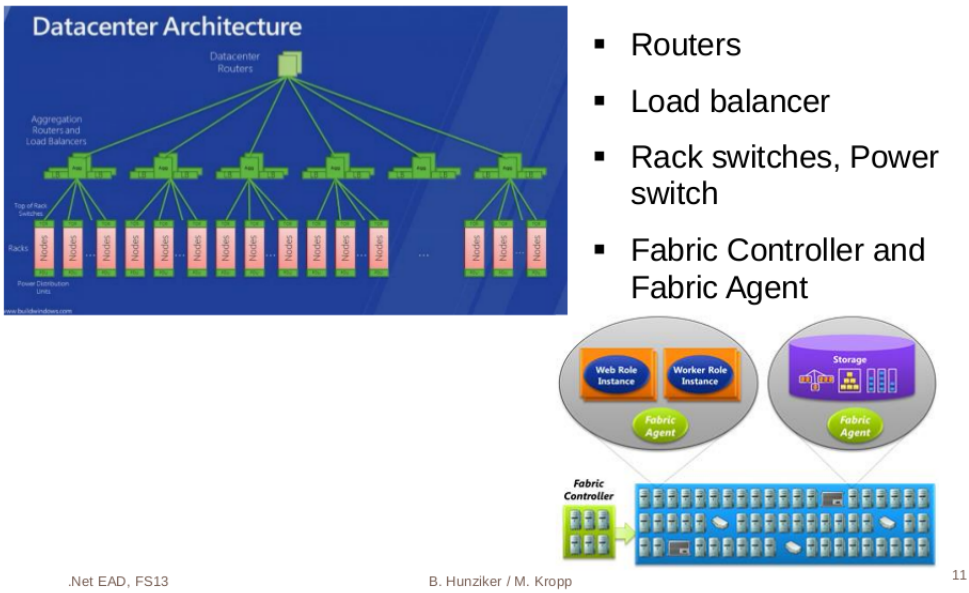


Figure 6.1.: figure

6.2. Windows Azure Portal

Service Management Portal UI Subscriptions (roles, URL, billing account) Hosted services, storage, SQL Azure and all the other services. Staging, production, updates.Create, configure, scale, monitor.
Built on top Azure Service Management. REST APIs.
Multiple service administrators.
Get detailed billing information

Modes Free - Limited VM
Shared VM - $9.68
Standard : Separate VM - $75 to $298

PAAS Websites Visual Studio helps, use publish button to send to Azure.

IAAS Features To migrate legacy applications. Full control over the OS Image. Create your VHD locally and upload
Deploy a service package that references and uses the custom OS image
You maintain the OS (patch) With your tools.

6.2.1. PAAS in depth for Azure

- Auto scaling possible Use Azure Diagnostics, avoid logging to local discs
- Azure may "reimage" the VM any time (after patch) -¿ this removes all local data!
- Production & staging environment
- Virtual IP address swap (easy testing in staging, quick undo possible)

Implementation

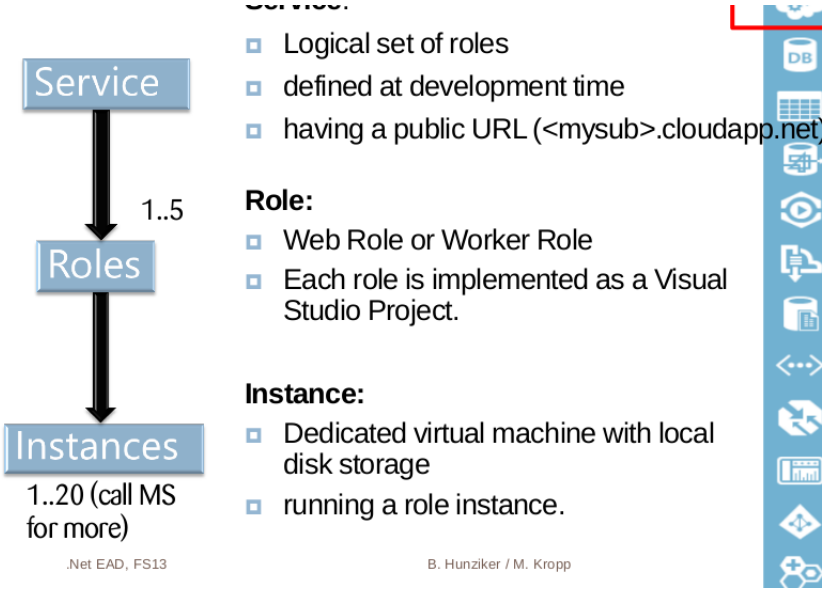


Figure 6.2.: figure

6.3. Databases/Storage Account

- Familiar SQL Server Support for existing APIs & tools

- Failover cluster, data stored on 3 different backend data nodes 150 GB limit - "sharding"
- No point in time backup!
- Accessible with Management Studio

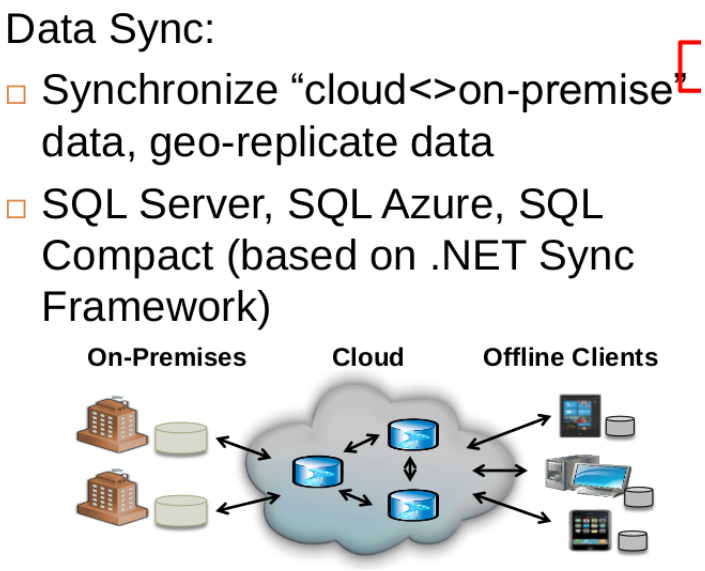


Figure 6.3.: Database Syncing

Storage in DB :

- blobs[1], Tables & Queues
- Georeplication (immediately consistent)
- 100TB per storage account
- Accessible via RESTful Web Service API Security: shared secret keys, HTTPS endpoint

6.3.1. Storage Types (Table,Queue)

Table
- Rows of entities
- Up to 255 properties per row
- Max. 1 MB per entity
- Partitioned by (partition) key
- Indexed by row key
- Scales for large number (billions) of entities
- Structured data, no fixed schema, not an RDBMS.

Queue
- Any binary object can be queued (64kB limit)
- Use for Inter-role communication
- non transactional
- read at least once
- message invisible for timeout
- delete to remove message - otherwise is returned to queue

[1]Large binary - 200gb

7. Windows Communication Foundation

7.1. Fundamentals

- Is a runtime for message based communication Is a development API
  For creating systems that send messages between services and clients
  Unified API across different transport mechanisms
- For applications that communicate with other applications
  On the same computer system
  On external computer system
  In another company accessible over the internet

7.1.1. Service Orientation

Boundaries are Explicit Developers opt-in to consuming, exposing, and defining public-facing service façade.

Autonomous Evolution Services and consumers are independently versioned, deployed, operated, and secured.

Share Schema + Contract not class Data never includes behavior; Objects with data and behavior are a local phenomenon.

Compatability based on policy Capabilities and requirements represented by a unique public name; Used to establish service suitability.

7.1.2. Design Goals WCF

Unifaction
- Unified former distributed technology stacks
- Composable functionality
- Appropriate for use on-machine, in the intranet, and cross the Internet

Productive Service-Oriented Programming
- Service-oriented programming model
- Supports 4 tenets of service-orientation
- Improve developer productivity

Interoperability & Integration
- WS-* interoperability with applications running on other platforms
- Interoperability with today's distributed stacks.

7.2. WCF Concepts

Adressses `http://www.mystore.com/StoreFront net.tcp://mycomputer:9000/StoreFront`
Scheme:
HTTP, TCP, Named pipes, MSMQ, ... Machine

$$Port$$

Path

Contracts Defines what a service communicates.

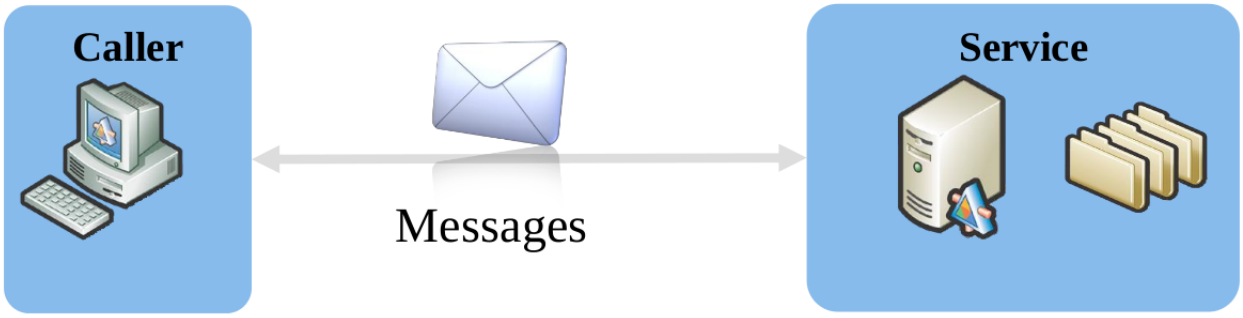


Figure 7.1.: figure

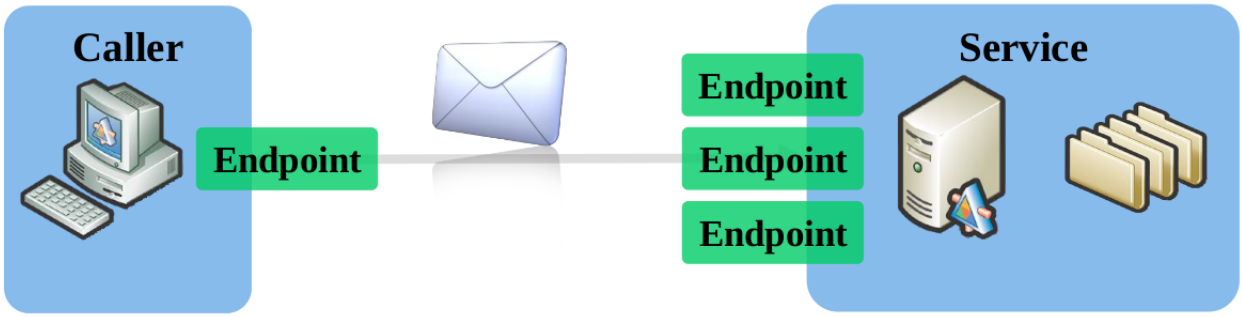


Figure 7.2.: figure

7.3. Contracts,Behavior,Ways to Talk

Service Contract Defines Operations, Behaviors and Communication Shape Map CLR types to WSD[1] (what does your service do?)

Data Contract Defines Schema and Versioning Strategies Maps CLR types to XSD[2]

Message Contract Allows defining application-specific headers and unwrapped body content Maps CLR types to SOAP messages.

Binding Describes how a service communicates : Set of binding elements :

- Set of predefined standard bindings
- Transport; http, tcp, np, msmq
- Encoding format; text, binary, MTOM, ...
- Security requirements
- Reliable session requirements
- Transaction requirements
- Can be customized
- Custom binding

Behavior Modifies or extends service or client runtime behavior. Instancing; Singleton, PrivateSession, SharedSession, PerCall Concurrency; Multiple, Reentrant, Single, Throttling; connections, threading Metadata customization Transactions; AutoEnlist, Isolation, AutoComplete.

Ways to talk One way, Request - Reply, Duplex/Dual.

[1]Web Services Description Language
[2]XML Schema Definition

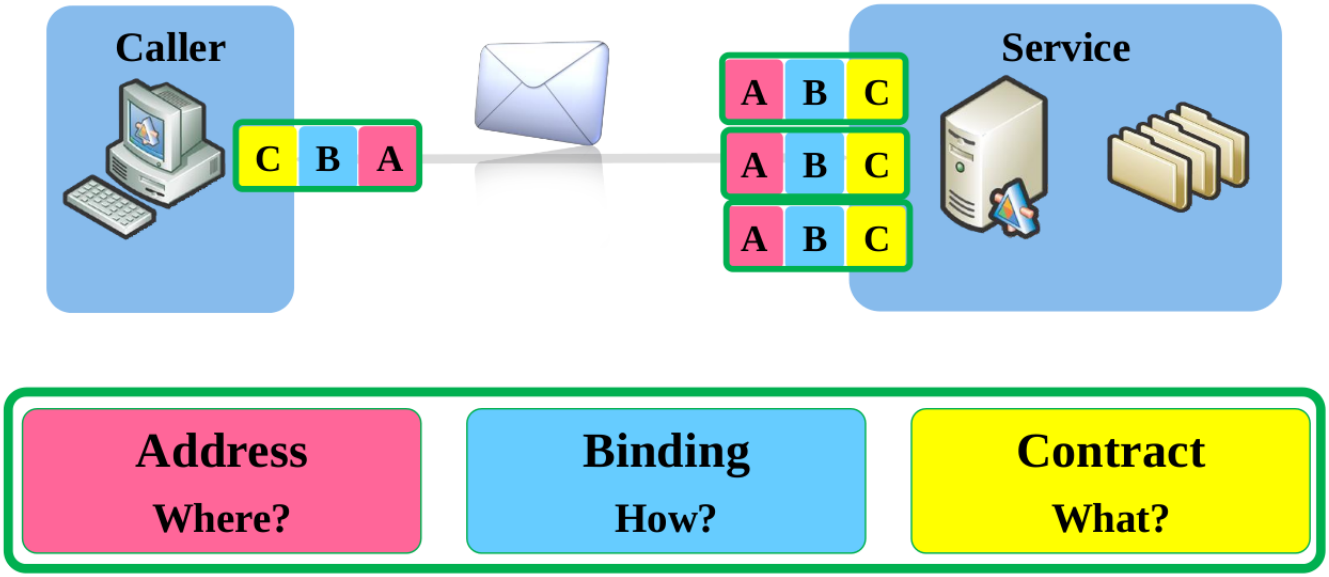


Figure 7.3.: figure

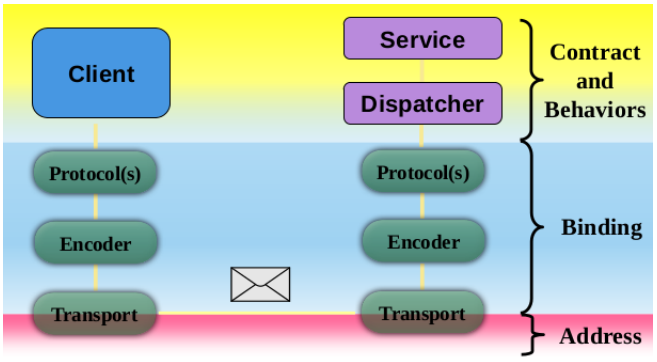


Figure 7.4.: WCF Messaging Runtime

7.4. Steps for Building a WCF App

1. Service
   - Define Contracts
   - Implement Contracts
   - Define Endpoints.
   - Host and Run Service.
2. Client.
   - Generate Proxy from Metadata.
   - Implement Run Client.

7.5. WCF Examples

7.5.1. Service Contract

Listing 7.1: Define Service Contract

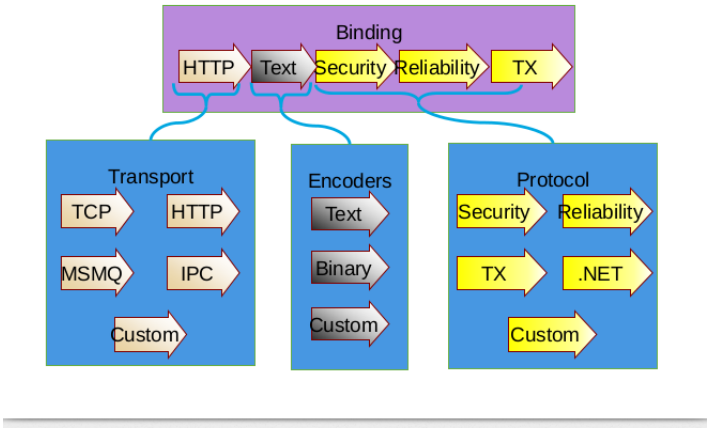


Figure 7.5.: figure

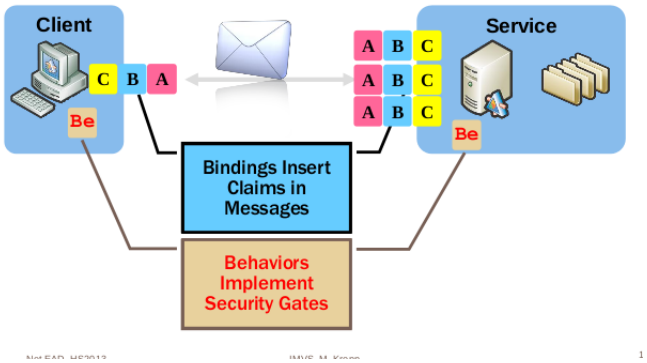


Figure 7.6.: figure

```
 [ServiceContract]
public interface ICalculator
{
[OperationContract]
int Add(int a, int b);
// Not exposed as part of the external contract :-)
void MethodRequiredForImplementation(bool b);
}
```

Listing 7.2: Service Contract: One Way

```
 [ServiceContract]
public interface ICalculator
{
[OperationContract]
int Add(int a, int b);
[OperationContract(IsOneWay=true)]
void StoreProblem (ComplexProblem p);
}
```

Listing 7.3: Service Contract:Duplex Asymmetric

```
 [ServiceContract(Session=true,
CallbackContract=typeof(ICalculatorResults))]
public interface ICalculatorProblems
{
[OperationContract(IsOneWay=true)]
void SolveProblem (ComplexProblem p);
}
public interface ICalculatorResults
```

```
{
[OperationContract(IsOneWay=true)]
void Results(ComplexProblem p);
}
```

7.5.2. Data Contract

Listing 7.4: Defininbg a Data Contract

```
 [DataContract]
public class ComplexNumber
{
[DataMember]
public double Real = 0.0D;
[DataMember]
public double Imaginary = 0.0D;
public ComplexNumber(double r, double i)
{
this.Real = r;
this.Imaginary = i;
}
}
```

7.5.3. Message Contract

Listing 7.5: Define Message Contract

```
 [MessageContract]
public class ComplexProblem
{
[MessageHeader(Name="Op", MustUnderstand=true)]
public string operation;
[MessageBodyMember]
public ComplexNumber n1;
[MessageBodyMember]
public ComplexNumber n2;
[MessageBodyMember]
public ComplexNumber solution;
// Constructors...
}
```

7.5.4. Bindings

Listing 7.6: WCF Bindings

```
 <endpoint address="Calculator"
bindingSectionName="basicHttpBinding"
bindingConfiguration="UsernameBinding"
contractType="ICalculator" />
<bindings>
<basicHttpBinding>
<binding name="UsernameBinding"
messageEncoding="Mtom">
<security mode="Message">
<message clientCredentialType="UserName"/>
</security>
</binding>
</basicHttpBinding>
</bindings>
```

7.6. Features Summary

7.7. Config (xml) based WCF

Listing 7.7: Endpoints WCF Config

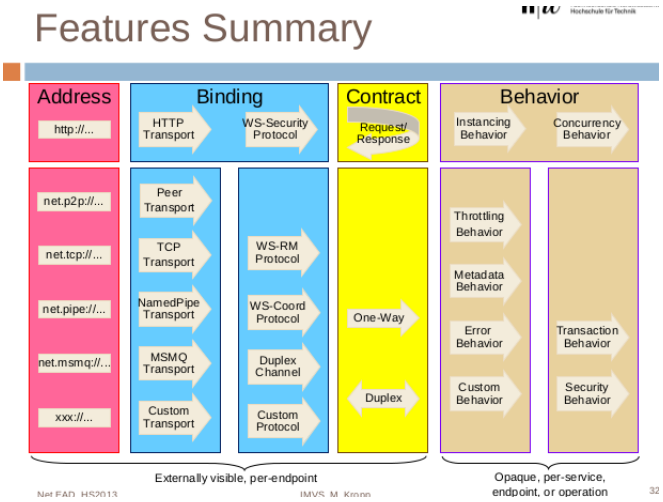


Figure 7.7.: figure

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
<system.serviceModel>
<services>
<service serviceType="CalculatorService">
<endpoint address="Calculator"
bindingSectionName="basicProfileBinding"
contract="ICalculator" />
</service>
</services>
</system.serviceModel>
</configuration>
```

Listing 7.8: Bindings WCF Config

```
<endpoint address="Calculator"
bindingSectionName="basicProfileBinding"
bindingConfiguration="Binding1"
contractType="ICalculator" />
<bindings>
<basicProfileBinding>
<binding configurationName="Binding1"
hostnameComparisonMode="StrongWildcard"
transferTimeout="00:10:00"
maxMessageSize="65536"
messageEncoding="Text"
textEncoding="utf-8"
</binding>
</basicProfileBinding>
</bindings>
```

Listing 7.9: Custom Bindings

```
<bindings>
<customBinding>
<binding configurationName="Binding1">
<reliableSession bufferedMessagesQuota="32"
inactivityTimeout="00:10:00"
maxRetryCount="8"
ordered="true" />
<httpsTransport manualAddressing="false"
maxMessageSize="65536"
hostnameComparisonMode="StrongWildcard"/>
<textMessageEncoding maxReadPoolSize="64"
maxWritePoolSize="16"
messageVersion="Default"
```

```
encoding="utf-8" />
</binding>
</customBinding>
</bindings>
```

7.8. WCF Summary

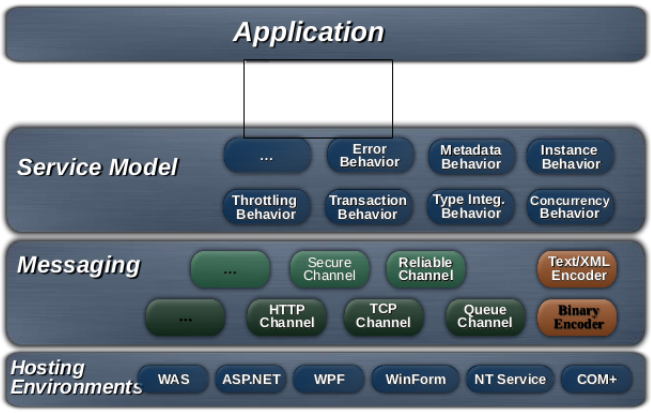


Figure 7.8.: figure

8. REST API MVC

8.1. Rest Basics

Definition Style of software architecture for distributed systems Term originated in a 2000 PhD written by Roy Fielding one of the principal authors of the HTTP protocol specification Term is also often used to describe any simple interface that transmits domain-specific data over HTTP without an additional messaging (e.g. SOAP) or session tracking (e.g. cookies)

Principles

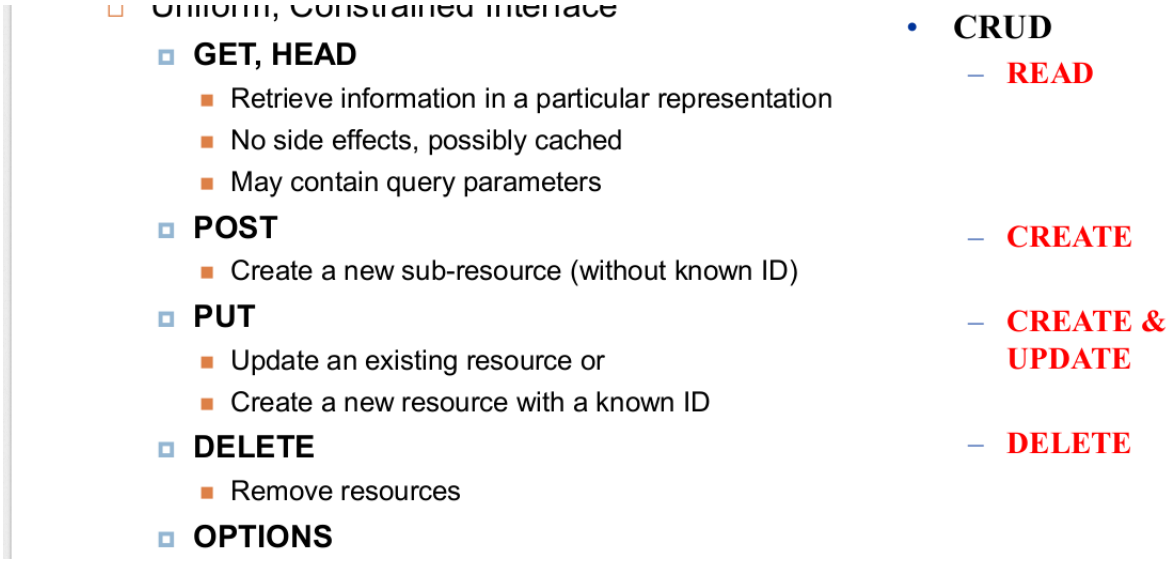


Figure 8.1.: figure

Listing 8.1: Verlingung mit Rest

```
 <order>
<date>16.03.2013</date>
<amount>23</amount>
<product ref="http://example.com/products/4711" />
<customer ref="http://example.com/customers/1234" />
</order>
```

Error Codes

- 201 Created
- 200 Success/204 Success but No Content
- 301 Uri Moved
- 403 Not authorized
- 404 Not Found / Does not exist
- 405 Method not Allowed
- 415 Unsupported Media Type
- 500 Server Error

Rest Steps 1. Give everything id, id is an uri.

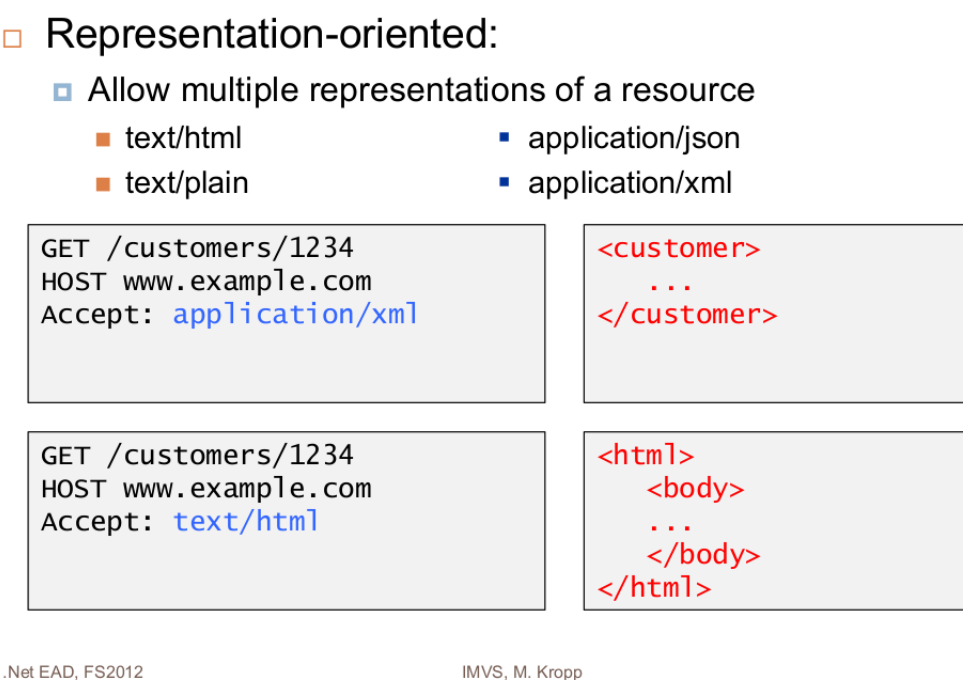


Figure 8.2.: figure

2. Link things together using resource references.
3. Standard HTTP Methods (GET;POST;PUT;DELETE;)
4. Content negotiation URI based representations (Accept: application/xml) (foo.xml / foo.html / foo.json)

Common Patterns

- GET /container List container content
- POST /container Add item to container (with item in request)
- URI of item returned in HTTP response header, e.g. Location: http://host/container/item
- DELETE /container Remove container (including items)
- GET /container/item Read item
- PUT /container/item Update item (with updated item in request) DELETE /container/item Remove item

Rest vs SOAP

8.2. WCF vs Web Api

- Service -¿ Web API controller
- Operation -¿ Action
- Service contract -¿ Not applicable
- Endpoint -¿ Not applicable
- URI templates -¿ ASP.NET Routing
- Message handlers -¿ Same
- Formatters -¿ Same
- Operation handlers -¿ Filters, model binders

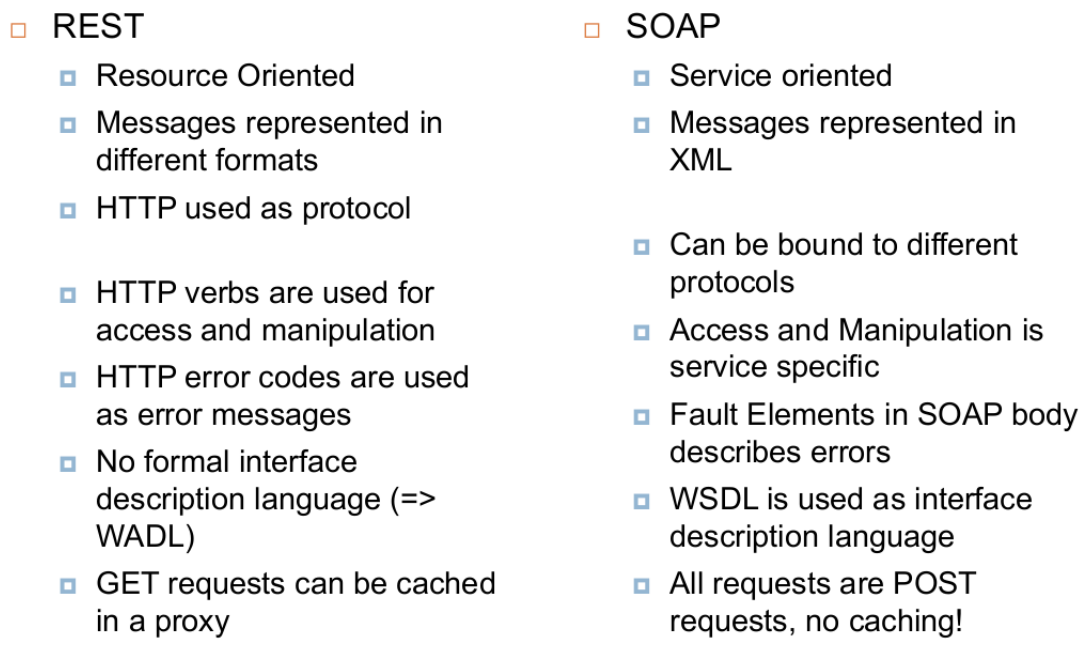


Figure 8.3.: figure

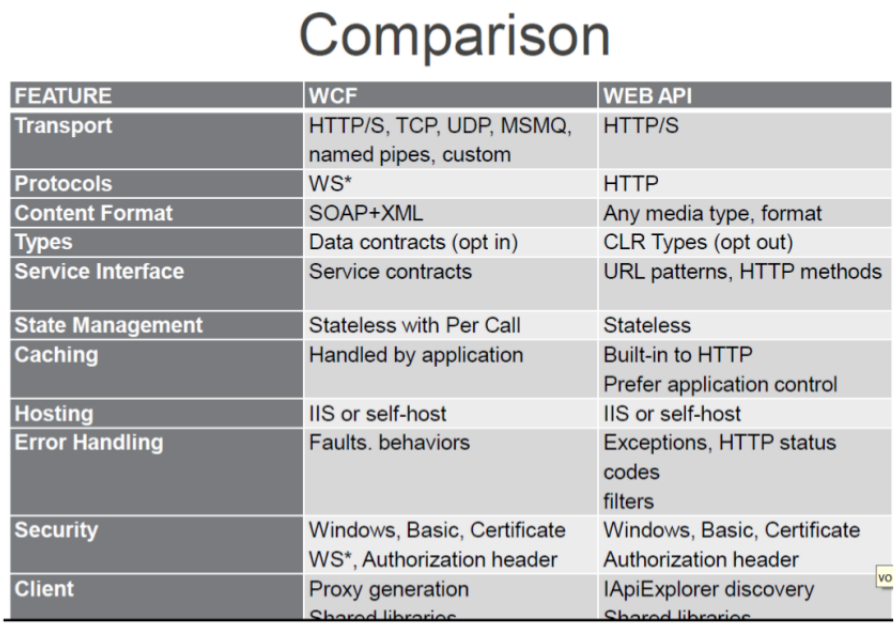
8.3. Web Api Vorteile

- Has nothing to do with ASP.Net
- New http object Model : HttpRequestMessage, HttpResponseMessage, Headers, HttpContent
- Code based config only.
- Symmetry client, server, async apis.

8.3.1. Integrated Stack

- Modern HTTP programming model
- Full support for ASP.NET Routing
- Content negotiation and custom formatters
- Model binding and validation
- Filters
- Query composition
- Easy to unit test
- Improved Inversion of Control (IoC) via
- DependencyResolver
- Code-based configuration
- Self-host

8.4. Default Route (Important)



Comparison

FEATURE	WCF	WEB API
Transport	HTTP/S, TCP, UDP, MSMQ, named pipes, custom	HTTP/S
Protocols	WS*	HTTP
Content Format	SOAP+XML	Any media type, format
Types	Data contracts (opt in)	CLR Types (opt out)
Service Interface	Service contracts	URL patterns, HTTP methods
State Management	Stateless with Per Call	Stateless
Caching	Handled by application	Built-in to HTTP Prefer application control
Hosting	IIS or self-host	IIS or self-host
Error Handling	Faults. behaviors	Exceptions, HTTP status codes filters
Security	Windows, Basic, Certificate WS*, Authorization header	Windows, Basic, Certificate Authorization header
Client	Proxy generation Shared libraries	IApiExplorer discovery Shared libraries

Figure 8.4.: figure

- Default route will use http method for action
- Controller/action/id
- API/Controller/id GET/POST/PUT/DELETE

8.5. Formate (XML JSON)

- XML tag based document formatting
- Javascript Notation by Douglas Crockford
- JSON less verbose than XML, more lightweight
- Mobile devices have limited bandwidth
- Set Xml or JSON based on Content-Type or Accept header

  Accept: application/xml
- Can also use Odata

8.6. Security in Web API

```
[Authorize()]
https over port 443
Security Tokens
OAuth
```

Scenario	WCF 4.5	ASP.NET Web API
Need to support specific scenarios like Message queues, duplex communication, end to end message security, distributed transactions, one way messaging etc....	It's me!	
Already you have existing working WCF services and would like to add HTTP support additionally.	It's me!	
One code base to support both SOAP and RESTful endpoints	It's me!	
Need to create a resource-oriented services over HTTP		It's me!
Your project is a MVC application and want to expose some functionality over HTTP		It's me!
Want to build only a HTTP / RESTful services		It's me!
Duplex communication over HTTP	SignalR	
SQL backend and need to expose OData endpoints	WCF Data Services	

Figure 8.5.: figure

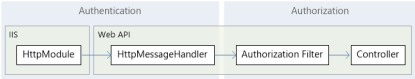


Figure 8.6.: figure

8.6.1. Exkurs : Authorize

- Globally: To restrict access for every Web API controller, add the AuthorizeAttribute filter to the global filter list

```
public static void Register(HttpConfiguration config)
{
    config.Filters.Add(new AuthorizeAttribute());
}
```

- Controller: To restrict access for a specific controller, add the filter as an attribute to the controller:

```
// Require authorization for all actions on the controller.
[Authorize]
public class ValuesController : ApiController
{
    public HttpResponseMessage Get(int id) { ... }
    public HttpResponseMessage Post() { ... }
}
public class ValuesController : ApiController
{
    public HttpResponseMessage Get() { ... }

    // Require authorization for a specific action.
    [Authorize]
    public HttpResponseMessage Post() { ... }
}
// Restrict by user:
[Authorize(Users="Alice,Bob")]
public class ValuesController : ApiController
{
}

// Restrict by role:
[Authorize(Roles="Administrators")]
public class ValuesController : ApiController
{
```

```
}
```

8.7. API Hosting

Web Asp.net IIS

Self Hosting Any .NET application can become a Web API server

In Memory Whole pipeline in memory

OWIN (Open Web Interface for .NET) Decouple web application from the host.

8.8. Building the API

Controllers Represents a HTTP resource Doesn't have to map 1-to-1 to an entity!, Use http helpers to conform to http standards : [HttpGet] usw. Can Use IAssemblyResolver to load.

Actions Maps to an incoming HTTP request.

```
Actions matched:


  - by prefix (GetAll, GetById, Post(id),Put(id))
- by action attribute (i.e. [HttpPost])
```

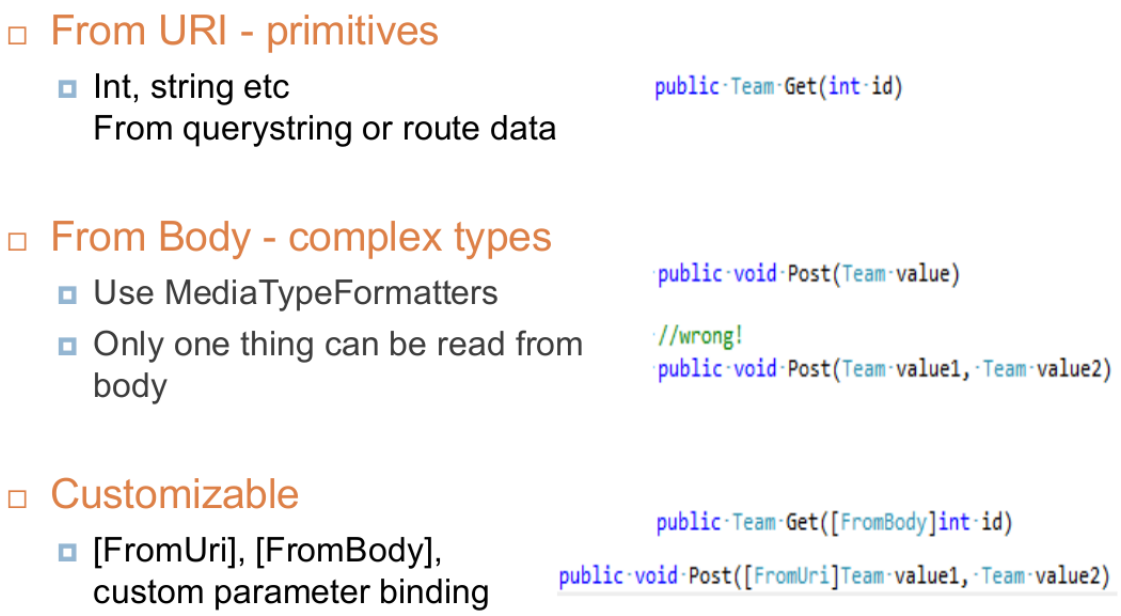


Figure 8.7.: figure

Action Return Model / void, HttpResponseMessage , any CLR type, Work with HTTP concepts directly: IHttpActionResult Predefined factories creating HttpResponseMessage

```
 public interface IHttpActionResult
{
Task<System.Net.Http.HttpResponseMessage> ExecuteAsync(CancellationToken cancellationToken);
}
```

Routing

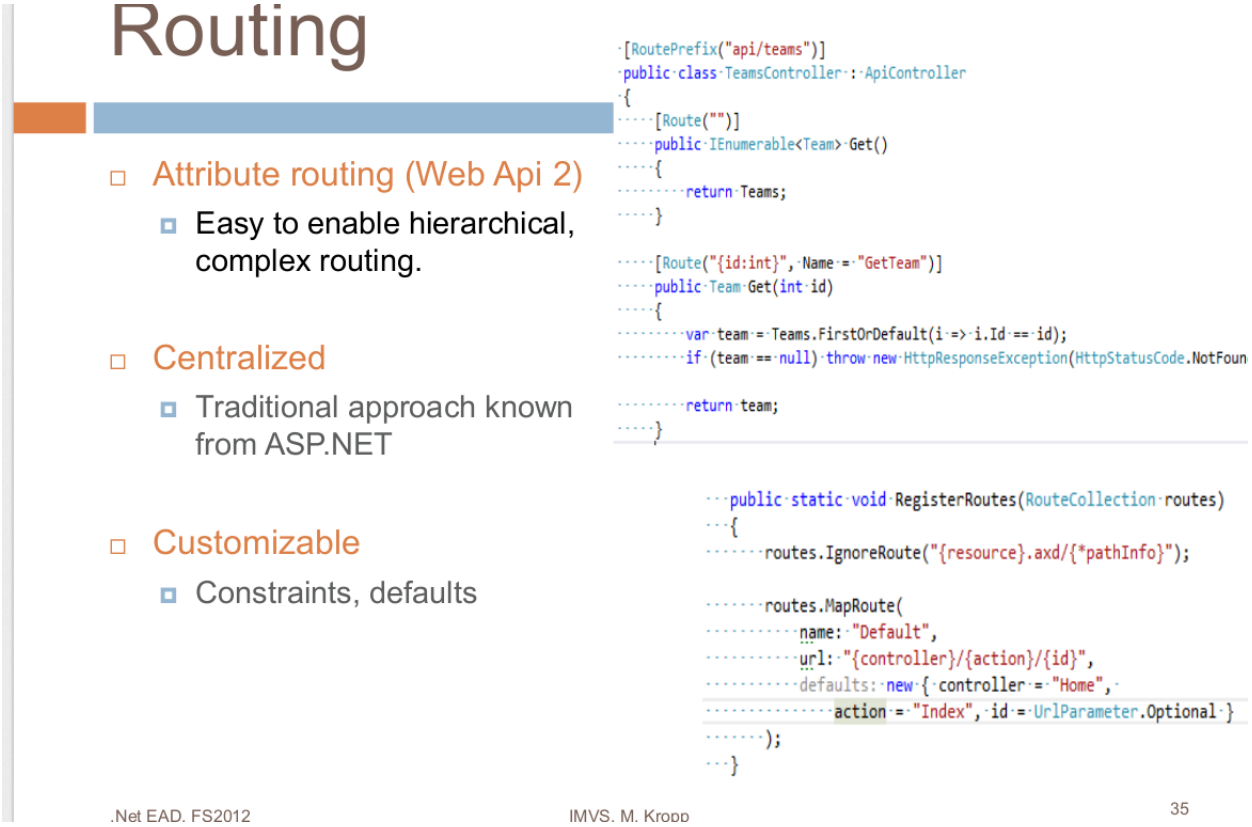


Figure 8.8.: figure

Content Negotiation

Testing Browser tools Fiddler4

- MediaTypeMapping
  - /api/resource.json, /api/resource?format=json
- Accept header
  - Accept: application/json
- Content-type header
  - Content-type: text/xml
- MediaTypeFormatter order
  - Can be configured to return 406 (Not acceptable)

Figure 8.9.: figure

- JSON
  - Uses JSON.NET or DataContractSerializer
- XML
  - Uses DataContractSerializer or XmlSerializer
- Pluggable
  - Many other media types can be supported
- Conneg can be run by hand
  - Using IContentNegotiatior

Formatters

- WebApiContrib.Formatting.Bson
- WebApiContrib.Formatting.Html
- WebApiContrib.Formatting.JavaScriptSerializer
- WebApiContrib.Formatting.Jsonp
- WebApiContrib.Formatting.MsgPack
- WebApiContrib.Formatting.ProtoBuf
- WebApiContrib.Formatting.Razor
- WebApiContrib.Formatting.ServiceStack

Figure 8.10.: figure

9. Signal R

Schlussendlich websockets über verschiedene Tech. Higher Abstraction Level
Abstracts different technologies to push data
Websockets not required
Falls back to available technologies
Open source
Easier to use
SignalR extends ASP.NET
etect and handle connection states of clients
Auto-reconnect after network failures -¿ persistent connections
Real-time" HTTP-based communication
framework Runs in all browsers (JavaScript needed)
Server and client APIs:
Client calls server
Server calls client
APIs for different client technologies (beyond JavaScript)

9.1. Hubs

Hubs offer an application-level RPC style
Derived from Microsoft.AspNet.SignalR.Hub
Abstraction on top of persistent connections
Hub conventions:

- Public methods are callable from the outside
- Send messages to clients by invoking hub methods which are delegated to the clients (dynamic, no type safety)

9.2. Hub Protocol

Base endpoint is ¡MyURL¿/signalr
Get JS metadata from ¡MyURL¿/ signalr/hubs
Transport negotiation sequence:

1. Web Sockets
2. 2: Server Sent Event (SSE) (-¿ not supported by IE)
3. Forever frame
4. Long polling
5. Parameters are serialized to/from JSON

9.3. Pusing Data to Clients

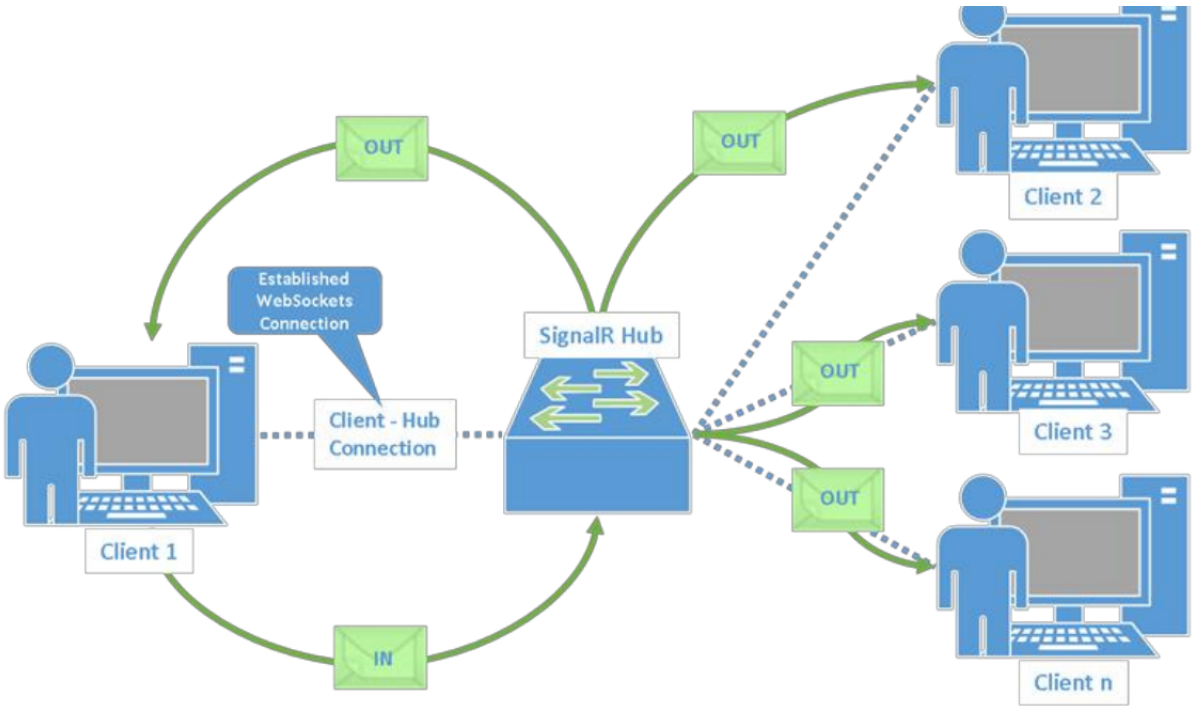


Figure 9.1.: figure

Listing 9.1: Pushing Data to Clients

```
public class MyHub : Hub
{
public void SendHelloToAll()
{
Clients.All.broadcastMessage("Hello");
}
}
```

Listing 9.2: Hub Lifecycle

```
public class MyHub1 : Hub
{
public override Task OnConnected()
{
...
}
public override Task OnDisconnected()
{
...
}
public override Task OnReconnected()
{
...
}
}
```

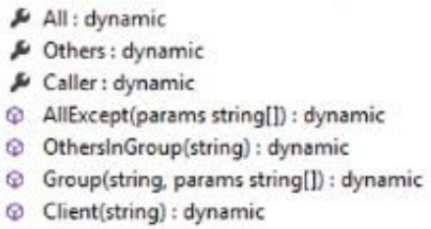
9.4. Hub Customers

Many all platforms. Popular jquery api - uses Late Binding / Generated Proxy.

9.5. Client Connection Steps

- Create Hubconnection

- Clients property of Hub on server hold dynamic properties and methods:



- Target method with parameters is dynamically serialized and embedded into the protocol response

Figure 9.2.: figure

```
var connection = $.hubConnection();
var proxy = connection.createHubProxy('chat');
proxy.on('newMessage', onNewMessage);
connection.start();

proxy.invoke('sendMessage', $('#message').val());
```

Figure 9.3.: figure

- Create hub proxy
- Wire up handlers via On(...)
- Call methods via invoke /lambda expr

Listing 9.3: client example signalr

```
 static void Main(string[] args)
{
var hubConnection =
new HubConnection("http://xxx.azurewebsites.net");
var chat = hubConnection.CreateHubProxy("myHub");
chat.On<string>("newMessage", x => Console.WriteLine(x));
hubConnection.Start();
Console.ReadLine();
}
```

10. Windows 8 Apps

10.1. Basics

- For Desktop+Tablet
- Businesses use Windows store or Side loading[1]
- 2/3 c# 1/3 html js (not run in browser)
- Touch enabled, full screen, Immersive, Managed lifecycle.
- Metro apps can be "snapped" - View more then 1 app at the same time on screen. Make one smaller.
- Apps can have live tiles with updated info (Push notification)
- Charms - menu right hand side, context sensitive.
- "App Bar" at bottom or top of screen replaces menus and toolbars.

10.2. App Lifecycle

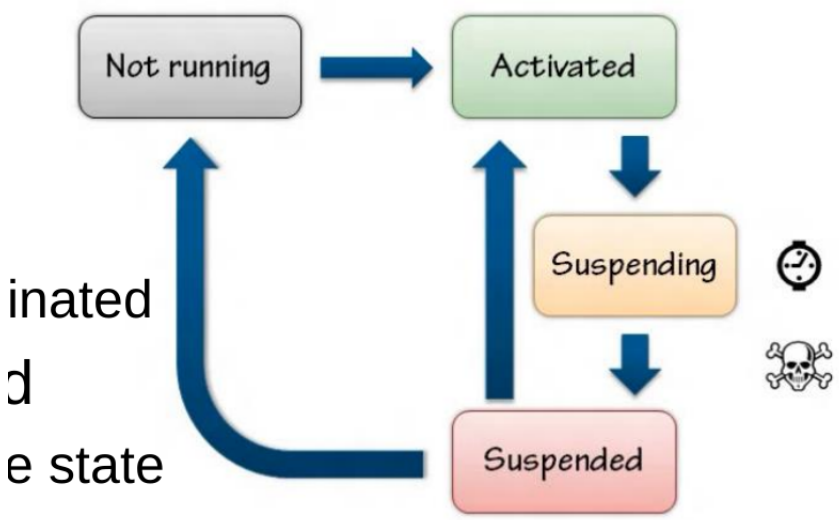


Figure 10.1.: figure

- 3 App Sites :

  Running suspended, Not running / terminated.
- If app suspended, 5 seconds to restore state.

  Subscribe to events, Incrementially save data, Save or restore data - user context.

[1] Involves purchasing "Enterprise Key" from Microsoft

- Templates in VS : Hub Application Class Library ,Portable Class Library (-¿usable on Desktop, WinRT, Windows Phone 8), Unit Test Library (all Windows Runtime)
- Async api many win rt functions async await.

10.3. Capabilities

Abilities that need to be enabled - permissions / features of device.

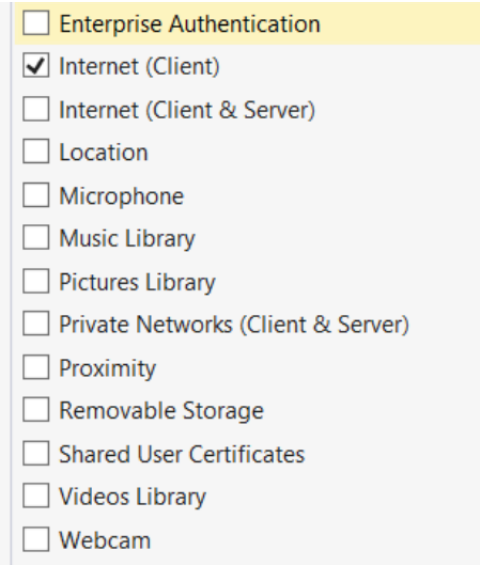


Figure 10.2.: figure

10.4. Contracts

Ability to make data available globally in winrt (to other applications) : Search
Share Target
File Open Picker (Skydrive, ...)
File Save Picker
Contact Picker
Print Task Settings
Protocol
File Association

10.5. Layout Conventions

- Align everything to 20x20 pixel grid
  - Shape outlines
  - Text baselines
- Standard font sizes/line heights
  - 12/20, 14.67/20, 26.67/32, 56/64
- Standard content area margin
  - Top: 7 units (140px)
  - Heading Baseline: 5 units (100px)
  - Left: 6 units (120px)

10.6. Semantic Zoom

Two Elements with Syncronzied Scrolling :

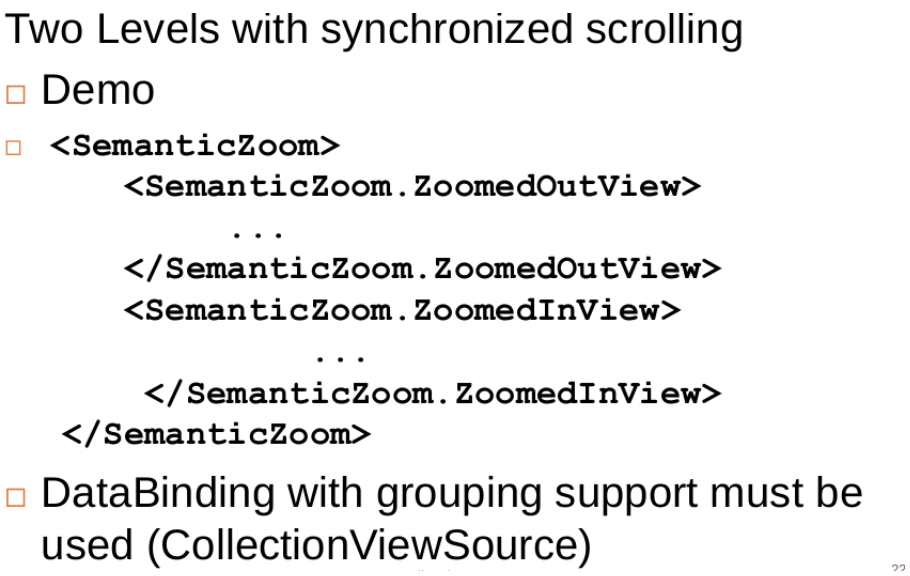


Figure 10.3.: figure

10.7. Application State and Session State, Disc Access

App State No explicit save for app state, do it manually often.

Session State Navigation, UI, scroll positions-

Disc Access In container, no direct access to disc:

```
 Windows.ApplicationModel.Package.Current.Inst
alledLocation
ApplicationData.Current.LocalFolder
 ApplicationData.Current.TemporaryFolder
```

10.8. How to launch an App using Contracts

10.9. Application Manifest + Packaging + Deployment

10.9.1. Packaging apps

Right click project click store, click create app packages, follow wizard.

10.9.2. Deployment

Deployed in store after verification / with "Side Loading Key" - Then an .appx package is created with a powershell script that installs it using the given key on the computer.

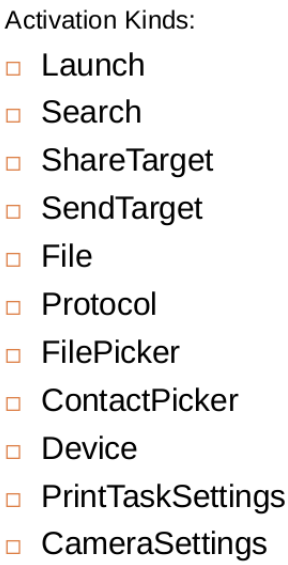


Figure 10.4.: figure

10.10. Azure Mobile Services

10.10.1. Definition

- Generic framework to create backend services. Built with Node js based on Azure websites. Creates a common backend that works for multiple mobile platforms.
- Data Source : Crud Rest API orientated Models.
- Identity / OAUTH : Out of the box (Facebook, Google , Twitter)
- Compatability for Android PUSH, APN (Apple) + Windows Mobile / Store.
- Dynamic DB Schema Creation : Created on first use. Can be switched off for production.

10.10.2. Data Access

Listing 10.1: Azure web services Data Access

```
1  IMobileServiceTable<TodoItem> todoTable =
App.MobileService.GetTable<TodoItem>();
await todoTable.InsertAsync(newItem);

 exports.post = function(request, response)
2 {
// Use "request.service" to access
// features of your mobile service:
// var tables = request.service.tables;
// var push = request.service.push;
7 response.send(200, "Hello World");
};
```

Scheduling

Scripts can be executed as a type of cron job.

10.10.3. Authentication with OAuth

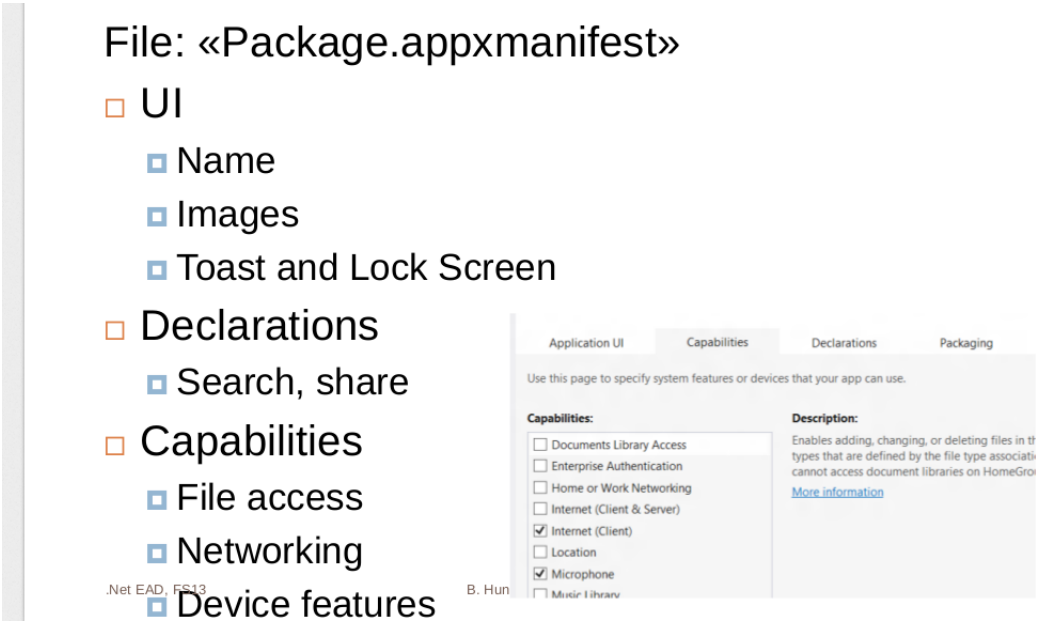


Figure 10.5.: figure

> Restrict permissions in settings afterward

10.11. Push Notifications

Common high level notification pattern:
1. Client retrieves its handle
2. Client stores handle in app backend
3. App backend contacts PNS
4. PNS forwards notification

Note : Platform specific infrastructure, no common interface - problematic.
Solution windows azure notification hub

10.11.1. Notification Hub

Multiplatform push notification infrastructure
- Platform-independent messages
- Scale out
- Sends push notifications from any backend (in the cloud or on-premises) to any mobile platform

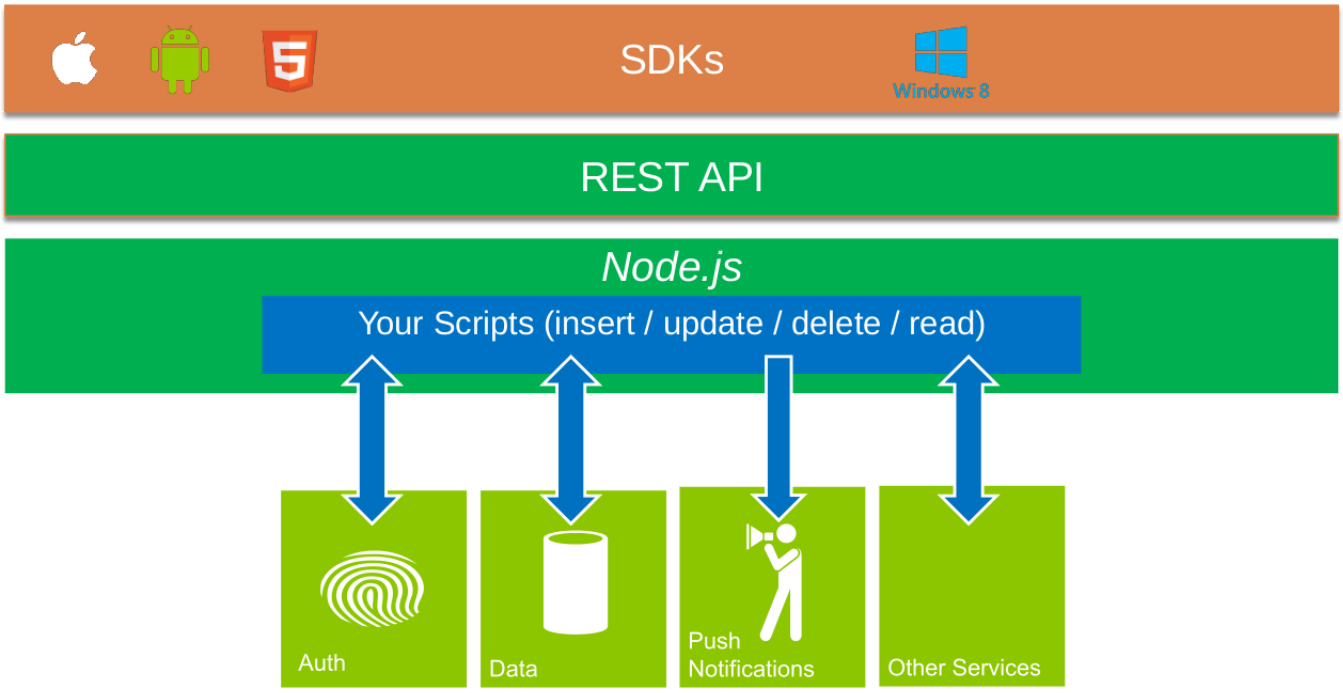


Figure 10.6.: figure

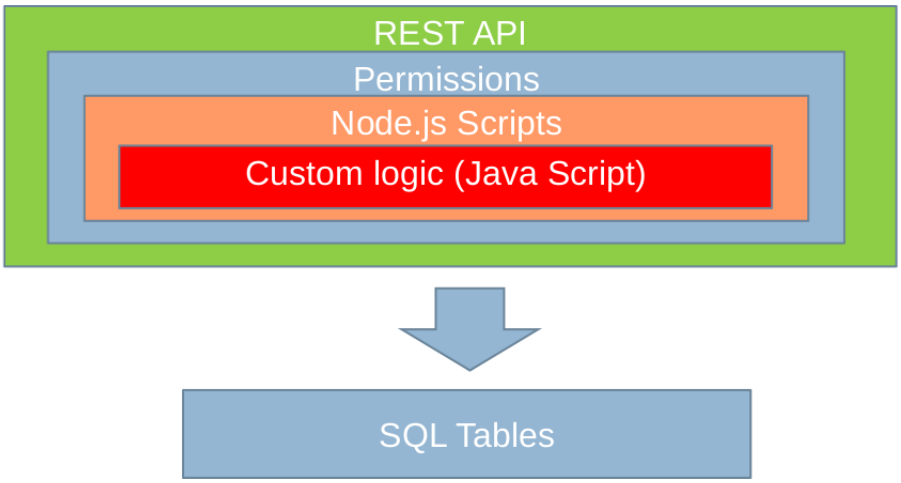


Figure 10.7.: figure

Data Access Interception

Add business logic to CRUD data access functions (Java Script):

- function insert(item, user, request) {…}
- function update(item, user, request) {…}
- function del(id, user, request) {…}
- function read(query, user, request) {…}

Figure 10.8.: figure

VORGANG Einfügen

```
1 function insert(item, user, request) {
2
3     item.createdAt = new Date();
4     console.log(item.text);
5     request.execute();
6
7 }
```

Figure 10.9.: figure

- Query interception:

OPERATION Read

```
1 function read(query, user, request) {
2     query.where({Complete:false}).orderByDescending('Title');
3   request.execute();
4
5 }
```

Figure 10.10.: figure

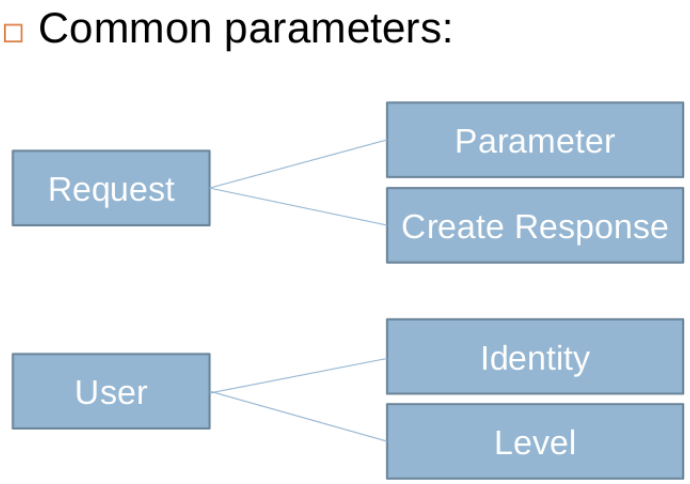


Figure 10.11.: figure

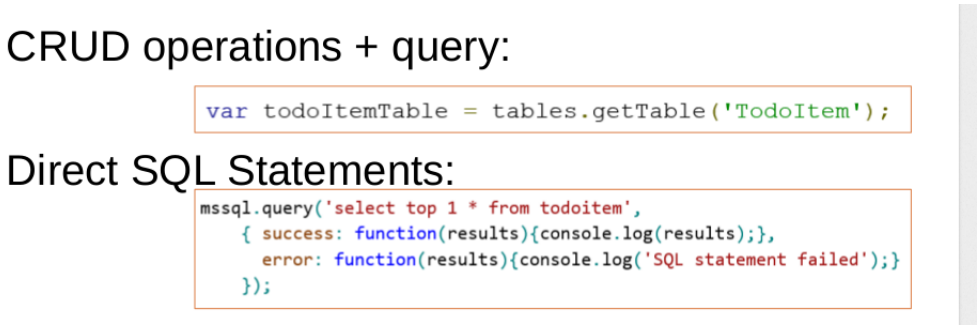


Figure 10.12.: figure

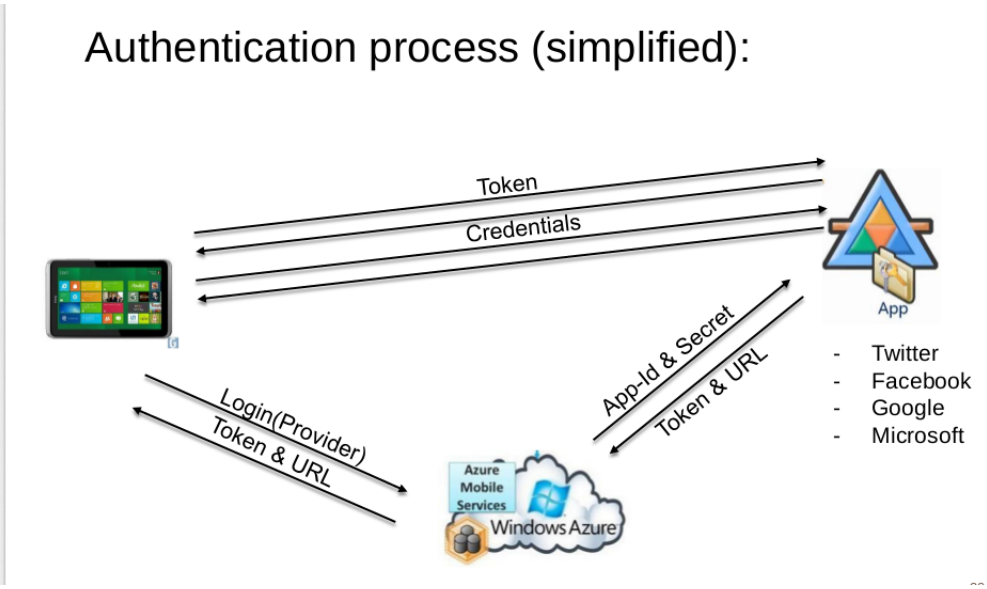


Figure 10.13.: figure

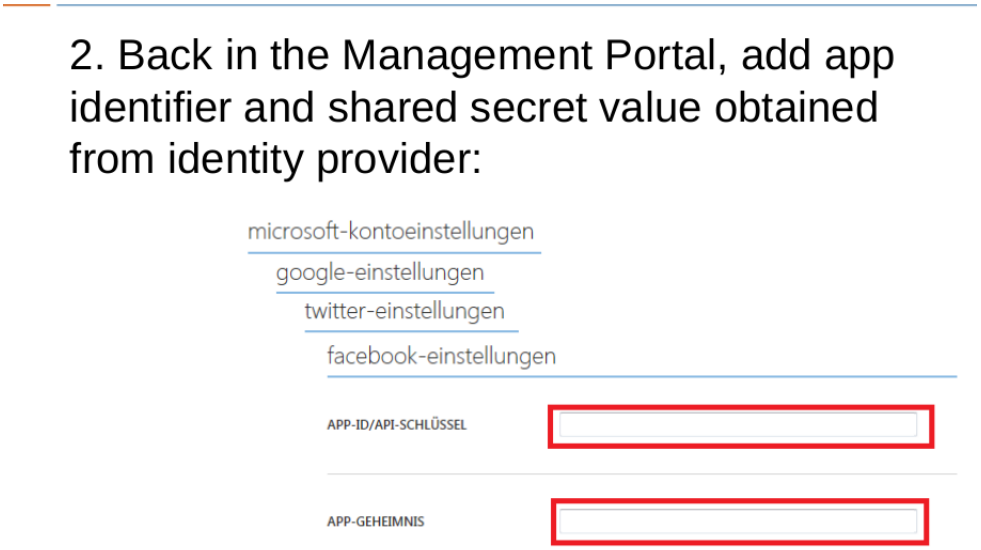


Figure 10.14.: figure

Figure 10.15.: figure

Part III.

Code

11. Code Prüfung 1

Listing 11.1: Converter WPF

```
using System;
using System.Windows.Data;

namespace Fhnw.Dnead.Auction.WpfClient.Converter
{
    /// <summary>
    /// Converts booleans to Offen/Abgeschlossen strings
    /// </summary>
    [ValueConversion(typeof(bool), typeof(String))]
    public class BooleanToStatusTextConverter : IValueConverter
    {
        public object Convert(object value, Type targetType, object parameter,
            System.Globalization.CultureInfo
culture)
        {
            if (value == null)
            {
                return null;
            }

            return (bool)value?"Offen":"Abgeschlossen";
        }
        public object ConvertBack(object value, Type targetType, object parameter,
            System.Globalization.CultureInfo
culture)
        {
            // we don't intend this to ever be called
            return null;
        }
    }
}
```

Listing 11.2: Using a viewModel in View

```
 using System.Windows;
using Fhnw.Dnead.Auction.WpfClient.ViewModels;

namespace Fhnw.Dnead.Auction.WpfClient.Views
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
       MainViewModel viewModel = new MainViewModel();

       public MainWindow()
        {

            InitializeComponent();

            DataContext = viewModel;

        }

        private void btnSell_Click(object sender, RoutedEventArgs e)
        {
            var sellView = new SellView();
```

```
            if (sellView.ShowDialog().Value == true)
            {
                viewModel.Auctions.Add(sellView.ViewModel.NewAuctionItem);
            }
        }

        private void btnBuy_Click(object sender, RoutedEventArgs e)
        {
            Auction selectedAuction = (Auction)dgAuctionItems.SelectedItem
            var bidView = new BidView(selectedAuction);
            bidView.ShowDialog();
        }
    }
}
```

Listing 11.3: ViewModel/Business logic

```
using System;
using System.Globalization;
using System.IO;
using System.Windows;
using System.Windows.Input;
using Fhnw.Dnead.Auction.WpfClient.Commands;

namespace Fhnw.Dnead.Auction.WpfClient.ViewModels
{
    public class SellViewModel : ViewModelBase
    {
        private DelegateCommand<Window> _okCommand;

       public Auction NewAuctionItem { get; private set; }

        public string Name { get; set; }
        public string StartPrice { get; set; }
        public string EndTime { get; set; }
        private string picturePath ;
        public string PicturePath
        {
            get { return picturePath;  }
            set{

                picturePath = value;
                    if (picturePath != null && (picturePath.Length > 0))
                {
                                #region 2. Extension: Image Preview - Enables preview of
                                    the selected image
                Picture = ReadPicture(picturePath);
                #endregion
                }
            }
        }

        private byte[] picture;

        public byte[] Picture { get { return picture; }
            set { picture = value;
                NotifyPropertyChanged("Picture");
            }
        }

        public SellViewModel()
        {
            EndTime = DateTime.Now.AddDays(7).ToString(CultureInfo.CurrentCulture);

            Name = "?";
            StartPrice = "1";

        }
        /// <summary>
        /// Validates if all data are correctly provided for the new auction
```

```
        /// Creates a new NewAuctionItem property instance.
        /// </summary>
56      /// <returns>true if OK, else false</returns>
        public bool ValidateEntries()
        {
            if (Name.Length == 0) return false;
            int startPrice;
61
            if (!int.TryParse(StartPrice, out startPrice)) return false;
            if (startPrice == 0) return false;
            DateTime endTime;
            if (!DateTime.TryParse(EndTime, out endTime)) return false;
66          if (endTime < DateTime.Now) return false;
            if (Picture != null && PicturePath.Length > 0 && !File.Exists(PicturePath))
                return false;

            // create new auction item and initialize it

71          NewAuctionItem = new Auction

            {
                Name = this.Name,
                StartPrice = startPrice,
76              Seller = Environment.UserName,
                EndTime = endTime,
                Open = true,
                StartTime = DateTime.Now,

81              #region 1. Extension: Read picture
                // optional: read picture
                Picture = ReadPicture(PicturePath)
                #endregion
            };
86          return true;
        }

        #region 1. Extension: Read picture (2. part)
        /// <summary>
91      /// Helper method to read the selected picture
        /// </summary>
        /// <returns></returns>
        private byte[] ReadPicture(string filename)
        {
96          // basic check if there is a filename
            if (string.IsNullOrEmpty(filename))
            {
                return null;
            }
101
            using (FileStream fileStream = new FileStream(filename, FileMode.Open))
            {
                using (BinaryReader reader = new BinaryReader(fileStream))
                {
106                 byte[] buffer = new byte[fileStream.Length];
                    reader.Read(buffer, 0, Convert.ToInt32(fileStream.Length));
                    return buffer;
                }
            }
111     }
        #endregion

        public ICommand OkCommand
        {
116         get
            {
                if (_okCommand == null)
                {
                    _okCommand = new DelegateCommand<Window>(DoOk, CanDoOk);
121
                }
                return _okCommand;
            }
```

```
        }

        void DoOk(Window win)
        {
            if (ValidateEntries() == true)
            {
                win.DialogResult = true;
            }
        }

        bool CanDoOk(Window win) { return true; }
    }
}
```

Listing 11.4: sample edmx file

```
<?xml version="1.0" encoding="utf-8"?>
<edmx:Edmx Version="3.0" xmlns:edmx="http://schemas.microsoft.com/ado/2009/11/edmx">
  <!-- EF Runtime content -->
  <edmx:Runtime>
    <!-- SSDL content -->
    <edmx:StorageModels>
      <Schema Namespace="AuctionModel.Store" Alias="Self" Provider="System.Data.SqlClient"
          ProviderManifestToken="2008"
xmlns:store="http://schemas.microsoft.com/ado/2007/12/edm/EntityStoreSchemaGenerator"
xmlns="http://schemas.microsoft.com/ado/2009/11/edm/ssdl">
        <EntityContainer Name="AuctionModelStoreContainer">
          <EntitySet Name="Auction" EntityType="AuctionModel.Store.Auction"
              store:Type="Tables" Schema="dbo" />
        </EntityContainer>
        <EntityType Name="Auction">
          <Key>
            <PropertyRef Name="Id" />
          </Key>
          <Property Name="Id" Type="int" Nullable="false" StoreGeneratedPattern="Identity" />
          <Property Name="Name" Type="nchar" Nullable="false" MaxLength="50" />
          <Property Name="StartPrice" Type="int" Nullable="false" />
          <Property Name="Bid" Type="int" />
          <Property Name="StartTime" Type="datetime" Nullable="false" />
          <Property Name="EndTime" Type="datetime" Nullable="false" />
          <Property Name="Seller" Type="nchar" Nullable="false" MaxLength="50" />
          <Property Name="Buyer" Type="nchar" MaxLength="50" />
          <Property Name="Open" Type="bit" Nullable="false" />
          <Property Name="Picture" Type="image" />
        </EntityType>
      </Schema>
```

```
    </edmx:StorageModels>

    <!-- CSDL content -->

    <edmx:ConceptualModels>

      <Schema Namespace="AuctionModel" Alias="Self" p1:UseStrongSpatialTypes="false"
xmlns:annotation="http://schemas.microsoft.com/ado/2009/02/edm/annotation"
xmlns:p1="http://schemas.microsoft.com/ado/2009/02/edm/annotation"
xmlns="http://schemas.microsoft.com/ado/2009/11/edm">

        <EntityContainer Name="AuctionContext" p1:LazyLoadingEnabled="true">

          <EntitySet Name="Auctions" EntityType="AuctionModel.Auction" />

        </EntityContainer>

        <EntityType Name="Auction">

          <Key>

            <PropertyRef Name="Id" />

          </Key>

          <Property Name="Id" Type="Int32" Nullable="false"
              p1:StoreGeneratedPattern="Identity" />

          <Property Name="Name" Type="String" Nullable="false" MaxLength="50" Unicode="true"
              FixedLength="true" />

          <Property Name="StartPrice" Type="Int32" Nullable="false" />

          <Property Name="Bid" Type="Int32" />

          <Property Name="StartTime" Type="DateTime" Nullable="false" Precision="3" />

          <Property Name="EndTime" Type="DateTime" Nullable="false" Precision="3" />

          <Property Name="Seller" Type="String" Nullable="false" MaxLength="50"
              Unicode="true" FixedLength="true" />

          <Property Name="Buyer" Type="String" MaxLength="50" Unicode="true"
              FixedLength="true" />

          <Property Name="Open" Type="Boolean" Nullable="false" />

          <Property Name="Picture" Type="Binary" MaxLength="Max" FixedLength="false" />

        </EntityType>

      </Schema>

    </edmx:ConceptualModels>

    <!-- C-S mapping content -->

    <edmx:Mappings>

      <Mapping Space="C-S" xmlns="http://schemas.microsoft.com/ado/2009/11/mapping/cs">

        <EntityContainerMapping StorageEntityContainer="AuctionModelStoreContainer"
CdmEntityContainer="AuctionContext">

          <EntitySetMapping Name="Auctions">

            <EntityTypeMapping TypeName="AuctionModel.Auction">

              <MappingFragment StoreEntitySet="Auction">

                <ScalarProperty Name="Id" ColumnName="Id" />
```

```
123                 <ScalarProperty Name="Name" ColumnName="Name" />
                    <ScalarProperty Name="StartPrice" ColumnName="StartPrice" />
                    <ScalarProperty Name="Bid" ColumnName="Bid" />
128
                    <ScalarProperty Name="StartTime" ColumnName="StartTime" />
                    <ScalarProperty Name="EndTime" ColumnName="EndTime" />
133                 <ScalarProperty Name="Seller" ColumnName="Seller" />
                    <ScalarProperty Name="Buyer" ColumnName="Buyer" />
                    <ScalarProperty Name="Open" ColumnName="Open" />
138
                    <ScalarProperty Name="Picture" ColumnName="Picture" />
                  </MappingFragment>
143             </EntityTypeMapping>
              </EntitySetMapping>
            </EntityContainerMapping>
148       </Mapping>
      </edmx:Mappings>
153   </edmx:Runtime>
    <!-- EF Designer content (DO NOT EDIT MANUALLY BELOW HERE) -->
    <Designer xmlns="http://schemas.microsoft.com/ado/2009/11/edmx">
158   <Connection>
        <DesignerInfoPropertySet>
163       <DesignerProperty Name="MetadataArtifactProcessing" Value="EmbedInOutputAssembly" />
        </DesignerInfoPropertySet>
      </Connection>
168   <Options>
        <DesignerInfoPropertySet>
173       <DesignerProperty Name="ValidateOnBuild" Value="true" />
          <DesignerProperty Name="EnablePluralization" Value="False" />
          <DesignerProperty Name="IncludeForeignKeysInModel" Value="True" />
178       <DesignerProperty Name="CodeGenerationStrategy" Value="None" />
        </DesignerInfoPropertySet>
183   </Options>
      <!-- Diagram content (shape and connector positions) -->
      <Diagrams></Diagrams>
188 </Designer>
  </edmx:Edmx>
```

Listing 11.5: Unity Example

```
 IUnityContainer myContainer = new UnityContainer();
myContainer.RegisterType<MyBaseService, CustomerService>();
MyBaseService myServiceInstance = myContainer.Resolve<MyBaseService>();


IUnityContainer myContainer = new UnityContainer();
myContainer.RegisterType(typeof(MyBaseService), typeof(CustomerService));
MyBaseService myServiceInstance = (MyBaseService)myContainer.Resolve(typeof(MyBaseService));

// Create container and register types
IUnityContainer myContainer = new UnityContainer();
myContainer.RegisterType<IMyService, DataService>("Data");
myContainer.RegisterType<IMyService, LoggingService>("Logging");

// Retrieve an instance of each type
IMyService myDataService = myContainer.Resolve<IMyService>("Data");
IMyService myLoggingService = myContainer.Resolve<IMyService>("Logging");

// Create container and register types using a name for each one
IUnityContainer myContainer = new UnityContainer();
myContainer.RegisterType<IMyService, DefaultService>();
myContainer.RegisterType<IMyService, DataService>("Data");
myContainer.RegisterType<IMyService, LoggingService>("Logging");

// Retrieve a list of non-default types registered for IMyService
// List will only contain the types DataService and LoggingService
IEnumerable<IMyService> serviceList = myContainer.ResolveAll<IMyService>();


// Create container and register types using a name for each one
IUnityContainer myContainer = new UnityContainer();
myContainer.RegisterType(typeof(MyServiceBase), typeof(DefaultService));
myContainer.RegisterType(typeof(MyServiceBase), typeof(DataService), "Data");
myContainer.RegisterType(typeof(MyServiceBase), typeof(LoggingService), "Logging");

// Retrieve a list of non-default types registered for MyServiceBase
// List will only contain the types DataService and LoggingService
foreach(Object mapping in myContainer.ResolveAll(typeof(MyServiceBase)))
{
  MyServiceBase myService = (MyServiceBase)mapping;
}
```

Listing 11.6: Constructor Injection

```
public class MyObject
{
  public MyObject(MyDependentClass myInstance)
  {
    // work with the dependent instance
    myInstance.SomeProperty = "SomeValue";
    // or assign it to a class-level variable
  }
}


IUnityContainer uContainer = new UnityContainer();
MyObject myInstance = uContainer.Resolve<MyObject>();

IUnityContainer uContainer = new UnityContainer()
   .RegisterType<IMyInterface, FirstObject>()
   .RegisterType<MyBaseClass, SecondObject>();
MyObject myInstance = uContainer.Resolve<MyObject>();



//Multiple-Constructor Injection Using an Attribute

public class MyObject
{
```

```
27    public MyObject(SomeOtherClass myObjA)
      {
        ...
      }

32    [InjectionConstructor]
      public MyObject(MyDependentClass myObjB)
      {
        ...
      }
37
  }
  IUnityContainer uContainer = new UnityContainer();
  MyObject myInstance = uContainer.Resolve<MyObject>();
```

12. Code Prüfung 2

Listing 12.1: WebApi Example

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Web.Http;
using ProductStoreWebApiService.Models;

namespace ProductStoreWebApiService.Controllers
{
    public class ProductsController : ApiController
    {

        static readonly IProductRepository repository = new ProductRepository();

        public IEnumerable<Product> GetAllProducts()
        {
            return repository.GetAll();
        }

        public Product GetProduct(int id)
        {
            Product item = repository.Get(id);
            if (item == null)
            {
                throw new HttpResponseException(HttpStatusCode.NotFound);
            }
            return item;
        }

```

```
        public IEnumerable<Product> GetProductsByCategory(string category)
        {
            return repository.GetAll().Where(
                p => string.Equals(p.Category, category, StringComparison.OrdinalIgnoreCase));
        }

        public HttpResponseMessage PostProduct(Product item)
        {
            item = repository.Add(item);
            var response = Request.CreateResponse<Product>(HttpStatusCode.Created, item);

            string uri = Url.Link("DefaultApi", new { id = item.Id });
            response.Headers.Location = new Uri(uri);
            return response;
        }

        public void PutProduct(int id, Product product)
        {
            product.Id = id;
            if (!repository.Update(product))
            {
                throw new HttpResponseException(HttpStatusCode.NotFound);
            }
        }

        public void DeleteProduct(int id)
        {
            Product item = repository.Get(id);
            if (item == null)
            {
                throw new HttpResponseException(HttpStatusCode.NotFound);
            }

            repository.Remove(id);
        }
    }
```

```
}
```

Listing 12.2: signalr example 1

```
 namespace MvcApplication1
2
{
    public class MyHub : Hub
7   {
        public void AddMessage(string nick, string message)
        {
12          Clients.All.newMessage(nick, message);
        }
17  }
}


22
namespace MvcApplication1
{
27      public class RouteConfig
        {
            public static void RegisterRoutes(RouteCollection routes)
32          {
                routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
37

                routes.MapRoute(
                    name: "Default",
42                  url: "{controller}/{action}/{id}",
                    defaults: new { controller = "Home", action = "Chat", id =
                        UrlParameter.Optional }
47              );
            }
        }
52  }


57  <div class="container">
        <input type="text" id="message" />
        <input type="text" id="nick" />
62  <input type="button" id="sendmessage" value="Send" />
    <ul id="discussion"></ul>
```

```
</div>
@section scripts {
<script src="~/Scripts/jquery.signalR-2.0.0.min.js"></script>
<script src="~/signalr/hubs"></script>
<script>
    $(function () {
        $.connection.myHub.client.newMessage = function (nick, message) {
            // Add the message to the page.
            $('#discussion').append('<li>' + nick + ', ' + message + '</li>');
        };
        $.connection.hub.start();
        $('#sendmessage').click(function () {
            // Call the Send method on the hub.
            $.connection.myHub.server.addMessage($('#nick').val(), $('#message').val());
        });
    });
</script>
}


namespace MvcApplication1
{
    public class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            app.MapSignalR();
        }
    }
}
```

Listing 12.3: wcf example

```
 namespace CurrencyConverterServices
{
    [ServiceContract]
    public interface ICurrencyConverterService
    {
        [OperationContract]
```

```
12
        string GetData(int value);


17      [OperationContract]
        CompositeType GetDataUsingDataContract(CompositeType composite);


22
        [OperationContract]
        CurrencyExchange Convert(Money sourceMoney, CurrencyCode targetCurrency);
27  }
    // Use a data contract as illustrated in the sample below to add composite types to
        service operations.
    // You can add XSD files into the project. After building the project, you can directly
        use the data types defined
32 there, with the namespace "CurrencyConverterService.ContractType".
    [DataContract]
    public class CompositeType
37
    {
        bool boolValue = true;
42      string stringValue = "Hello ";
        [DataMember]
        public bool BoolValue
47
        {
            get { return boolValue; }
52          set { boolValue = value; }
        }
        [DataMember]
57      public string StringValue
        {
62          get { return stringValue; }
            set { stringValue = value; }
        }
67  }
}

72
namespace CurrencyConverterServices
{
77  public class CurrencyConverterService : ICurrencyConverterService
    {
```

```
82          public string GetData(int value)
            {
                return string.Format("You entered: {0}", value);
87          }
            public CompositeType GetDataUsingDataContract(CompositeType composite)
92          {
                if (composite == null)
                {
97                  throw new ArgumentNullException("composite");
                }
102             if (composite.BoolValue)
                {
                    composite.StringValue += "Suffix";
107             }
                return composite;
112         }
            public CurrencyExchange Convert(Money sourceMoney, CurrencyCode targetCurrency)
            {
117             CurrencyConverter cc = new CurrencyConverter();
                return cc.Convert(sourceMoney, targetCurrency);
122         }
        }
    }
127

    namespace CurrencyConverterLib
132 {
        [DataContract]
        public enum CurrencyCode { [EnumMember]USD, [EnumMember]EUR, [EnumMember]CHF };
137 }


142 namespace CurrencyConverterLib
    {
        [DataContract]
147     public class Money
        {
152         public Money(CurrencyCode currencyCode, double amount)
```

```
        {
            Currency = currencyCode;
            Amount = amount;
        }
        [DataMember]
        public CurrencyCode Currency { get; set; }

        [DataMember]
        public double Amount { get; set; }
    }
}

<?xml version="1.0" encoding="utf-8" ?>
<configuration>

  <appSettings>
    <add key="aspnet:UseTaskFriendlySynchronizationContext" value="true" />
  </appSettings>
  <system.web>
    <compilation debug="true" />
  </system.web>
  <!-- When deploying the service library project, the content of the config file must be
      added to the host's
  app.config file. System.Configuration does not support config files for libraries. -->
  <system.serviceModel>
    <services>
      <service name="CurrencyConverterServices.CurrencyConverterService">
        <endpoint address="" binding="basicHttpBinding"
            contract="CurrencyConverterServices.ICurrencyConverterService">
          <identity>
            <dns value="localhost" />
          </identity>
        </endpoint>
        <endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange" />
        <host>
          <baseAddresses>
            <add
                baseAddress="http://localhost:8733/Design_Time_Addresses/CurrencyConverterService/Service1
```

```
                />
            </baseAddresses>
          </host>
227
        </service>
      </services>
232   <behaviors>
        <serviceBehaviors>
          <behavior>
237
            <!-- To avoid disclosing metadata information,
            set the values below to false before deployment -->
242         <serviceMetadata httpGetEnabled="True" httpsGetEnabled="True"/>
            <!-- To receive exception details in faults for debugging purposes,
            set the value below to true.  Set to false before deployment
247
            to avoid disclosing exception information -->
            <serviceDebug includeExceptionDetailInFaults="False" />
252       </behavior>
        </serviceBehaviors>
      </behaviors>
257
    </system.serviceModel>


262 </configuration>
```

Listing 12.4: wcf self host

```
 // program.cs
using System;
3 using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.ServiceModel;
using GettingStartedLib;
8 using System.ServiceModel.Description;

namespace GettingStartedHost
{
    class Program
13  {
        static void Main(string[] args)
        {
            // Step 1 Create a URI to serve as the base address.
            Uri baseAddress = new Uri("http://localhost:8000/GettingStarted/");
18
            // Step 2 Create a ServiceHost instance
            ServiceHost selfHost = new ServiceHost(typeof(CalculatorService), baseAddress);

            try
23          {
                // Step 3 Add a service endpoint.
                selfHost.AddServiceEndpoint(typeof(ICalculator), new WSHttpBinding(),
                    "CalculatorService");
```

```
                // Step 4 Enable metadata exchange.
28              ServiceMetadataBehavior smb = new ServiceMetadataBehavior();
                smb.HttpGetEnabled = true;
                selfHost.Description.Behaviors.Add(smb);

                // Step 5 Start the service.
33              selfHost.Open();
                Console.WriteLine("The service is ready.");
                Console.WriteLine("Press <ENTER> to terminate service.");
                Console.WriteLine();
                Console.ReadLine();
38
                // Close the ServiceHostBase to shutdown the service.
                selfHost.Close();
            }
            catch (CommunicationException ce)
43          {
                Console.WriteLine("An exception occurred: {0}", ce.Message);
                selfHost.Abort();
            }
        }
48  }
}
```

Listing 12.5: wcf client side

```
1 svcutil.exe /language:cs /out:generatedProxy.cs /config:app.config
      http://localhost:8000/ServiceModelSamples/service
  using System;
  using System.Collections.Generic;
  using System.Linq;
  using System.Text;
6 using GettingStartedClient.ServiceReference1;

  namespace GettingStartedClient
  {
      class Program
11    {
          static void Main(string[] args)
          {
              //Step 1: Create an instance of the WCF proxy.
              CalculatorClient client = new CalculatorClient();
16
              // Step 2: Call the service operations.
              // Call the Add service operation.
              double value1 = 100.00D;
              double value2 = 15.99D;
21            double result = client.Add(value1, value2);
              Console.WriteLine("Add({0},{1}) = {2}", value1, value2, result);

              // Call the Subtract service operation.
              value1 = 145.00D;
26            value2 = 76.54D;
              result = client.Subtract(value1, value2);
              Console.WriteLine("Subtract({0},{1}) = {2}", value1, value2, result);

              // Call the Multiply service operation.
31            value1 = 9.00D;
              value2 = 81.25D;
              result = client.Multiply(value1, value2);
              Console.WriteLine("Multiply({0},{1}) = {2}", value1, value2, result);

36            // Call the Divide service operation.
              value1 = 22.00D;
              value2 = 7.00D;
              result = client.Divide(value1, value2);
              Console.WriteLine("Divide({0},{1}) = {2}", value1, value2, result);
41
              //Step 3: Closing the client gracefully closes the connection and cleans up
                  resources.
              client.Close();
```

```
        }
    }
}
```

Listing 12.6: signalr js

```
var myConnection = $.connection("/chat");
Next, the received event is set up to display a received chat message as a new list item
    element in the messages
unordered list DOM element:

myConnection.received(function (data) {
        $("#messages").append("<li>" + data.Name + ': ' + data.Message + "</li>");
    });
```