BASTA!

.NET, VISUAL STUDIO & MORE

Share Connect 2011

SQLCON 2011

Rainer Stropek / software architects

Power Workshop
Der Weg zur C#-Softwarefabrik

Vorstellung

- software architects gmbh
- Rainer Stropek

  Developer, Speaker, Trainer

  MVP for Windows Azure

  rainer@timecockpit.com

  twitter FOLLOW ME @rstropek

AIT

time cockpit

CloudProTEAM

http://www.timecockpit.com

http://www.software-architects.com

Vom Handwerk zum Produktionsprozess

Was ist eine Softwarefabrik?

Probleme “typischer” Softwareerstellungsprozesse

- Verfügbarkeit von Softwareentwicklern/innen
- Kosten- und Zeitdruck
- Monolithische Systeme
- Einsatz zu allgemeiner Frameworks
  - Beispiel: .NET Framework vs. MS Access
- One-Off Development
  - Wiederverwendung bestenfalls auf Ebene einfacher Basisklassen oder Plattformkomponenten
- Geringer Reifegrad im Spezifikationsprozess
- Innovationszyklen bei Basistechnologien
- U.v.m.

Wiedererkennbare Muster innerhalb einer Familie von Anwendungen

Beispiel: Datenorientierte Geschäftsanwendungen

- Grundprozess
  - Lesen von Daten aus einer Datenbank
  - Anwenden definierter Geschäftslogik
  - Anzeige der Daten in tabellarischer oder grafischer Form
  - CRUD-Operationen innerhalb gewisser Geschäftsregeln
  - Schreiben der geänderten Daten in eine Datenbank
- Spezialanforderungen

Was ist eine Softwarefabrik? (aka MDD, CBD, etc.)

A **software factory** is an organizational structure that specializes in producing computer software applications or software components […] **through an assembly process**.

The software is created by **assembling predefined components**. Traditional coding, is left only for creating new components or services.

A **composite application** is the end result of manufacturing in a software factory.

Quelle: "Software Factory" in Wikipedia

Vorteile einer Softwarefabrik (1/2)

- Kernkompetenzen der Mitarbeiter werden hervorgehoben
  - Fachlich orientierte Berater konfigurieren (Vokabular näher bei der jeweiligen Domäne des Kunden)
  - Softwareentwickler erstellen Basiskomponenten
- Steigerung der Effizienz
  - Das Rad wird weniger oft neu erfunden
- Steigerung der Qualität
  - Anspruchsvolle QS-Maßnahmen für Basiskomponenten

Vorteile einer Softwarefabrik (2/2)

- Reduktion der Projektrisiken
  - Fachberater mit besserem Kundenverständnis
  - Höhere Qualität der Basiskomponenten
- Steigerung des Firmenwerts
  - Systematisches Festhalten von Wissen über die Herstellung einer Familie von Softwarelösungen
  - Design Patterns, Frameworks, Modelle, DSLs, Tools
- Vereinfachung des Vertriebs- und Spezifikationsprozesses
  - Konzentration auf projektspezifische Lösungsteile
  - Quick Wins durch Standardkomponenten

Was eine Softwarefabrik nicht will…

- Reduktion des Entwicklungsprozesses auf standardisierte, mechanische Prozesse
  - Im Gegenteil, mechanische Prozesse sollen Tool überlassen werden
- Verringerung der Bedeutung von Menschen im Entwicklungsprozess
- „Handwerk“ der Softwareentwicklung wird nicht gering geschätzt sondern gezielt eingesetzt
- Entwicklung von Frameworks statt Lösungen für Kunden

Software Factories
→ Economy of Scope

Economy of Scale

Multiple implementations (=Copies) of the same design

- Mehr oder weniger mechanische Vervielfältigung von Prototypen
  - Massengüter
  - Software (z.B. Auslieferung auf Datenträger)

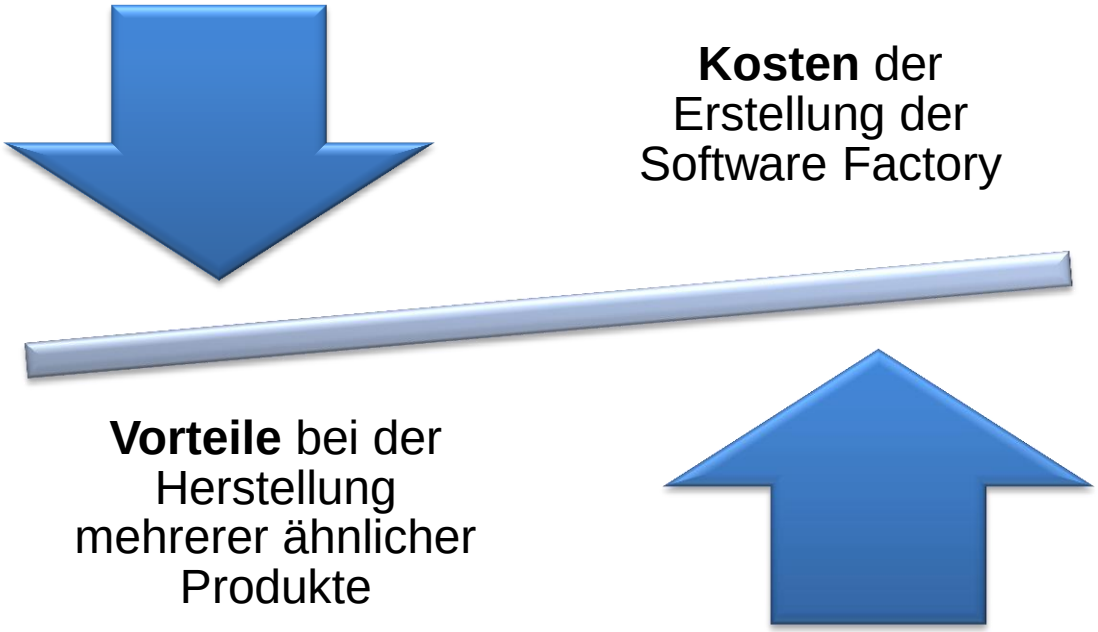
Economy of Scope

Production of multiple designs and their initial implementations

- Ähnliche Designs (=Familien von Anwendungen) auf Grundlage gemeinsamer Techniken und Technologien
  - Individuelle physische Güter (z.B. Brücken, Hochhäuser)
  - Individualsoftware, Softwareplattformen (vgl. PaaS)

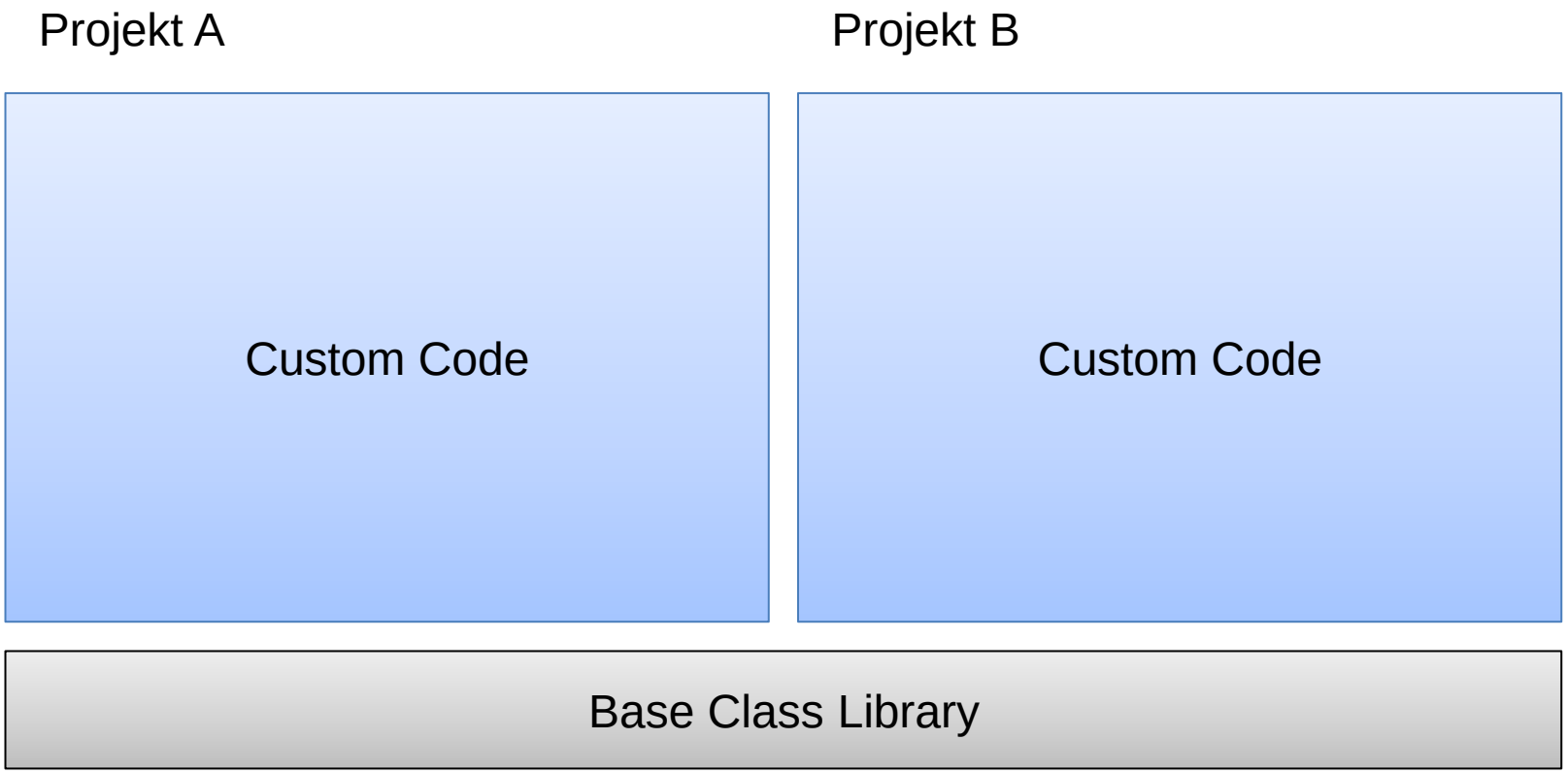
Quelle: Greenfield J., Short K.: Software Factories

Kosten/Nutzen Abwägung

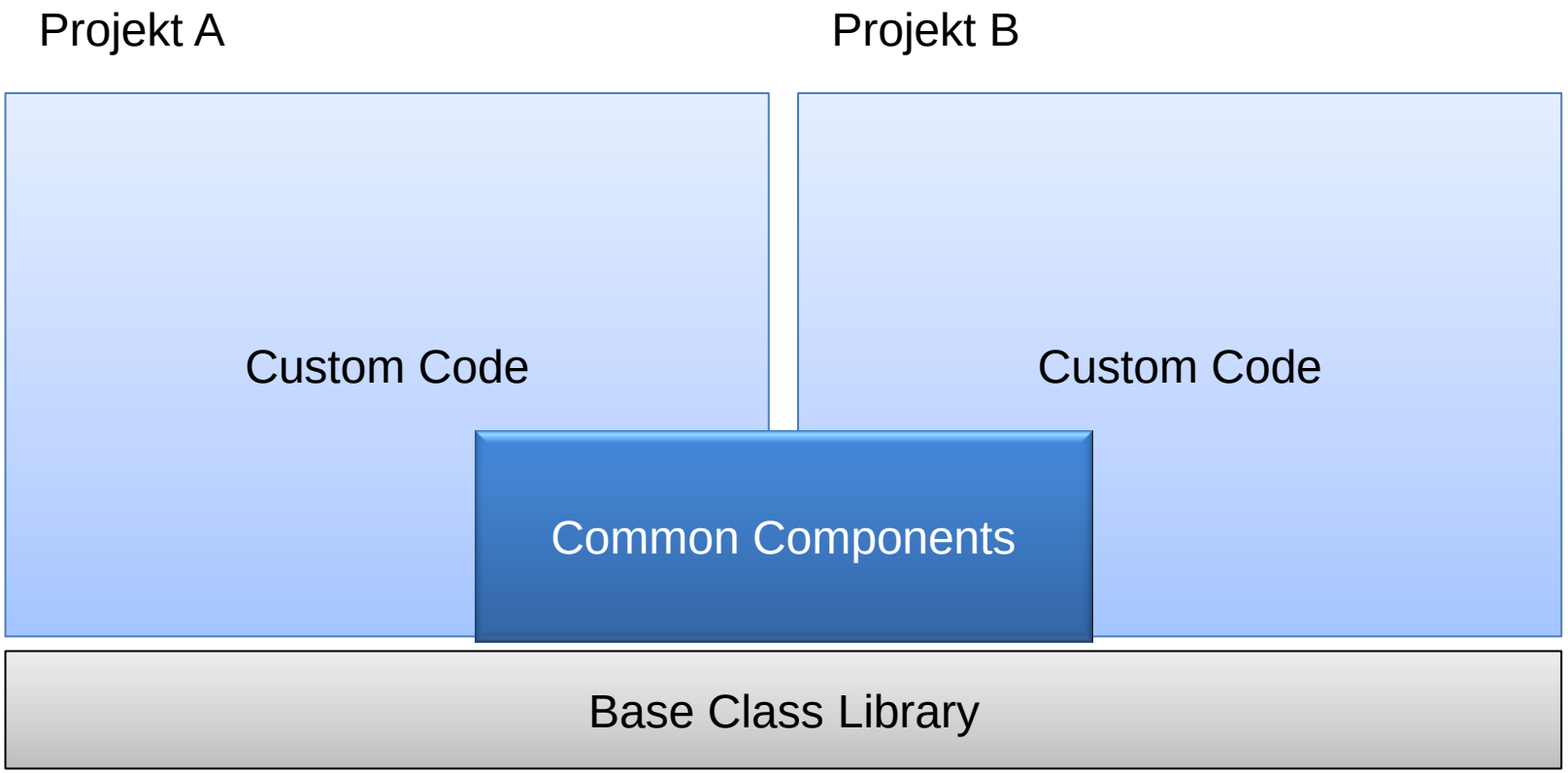


Spezialisierung als zentraler Erfolgsfaktor beim Einsatz der Software Factory Prinzipien

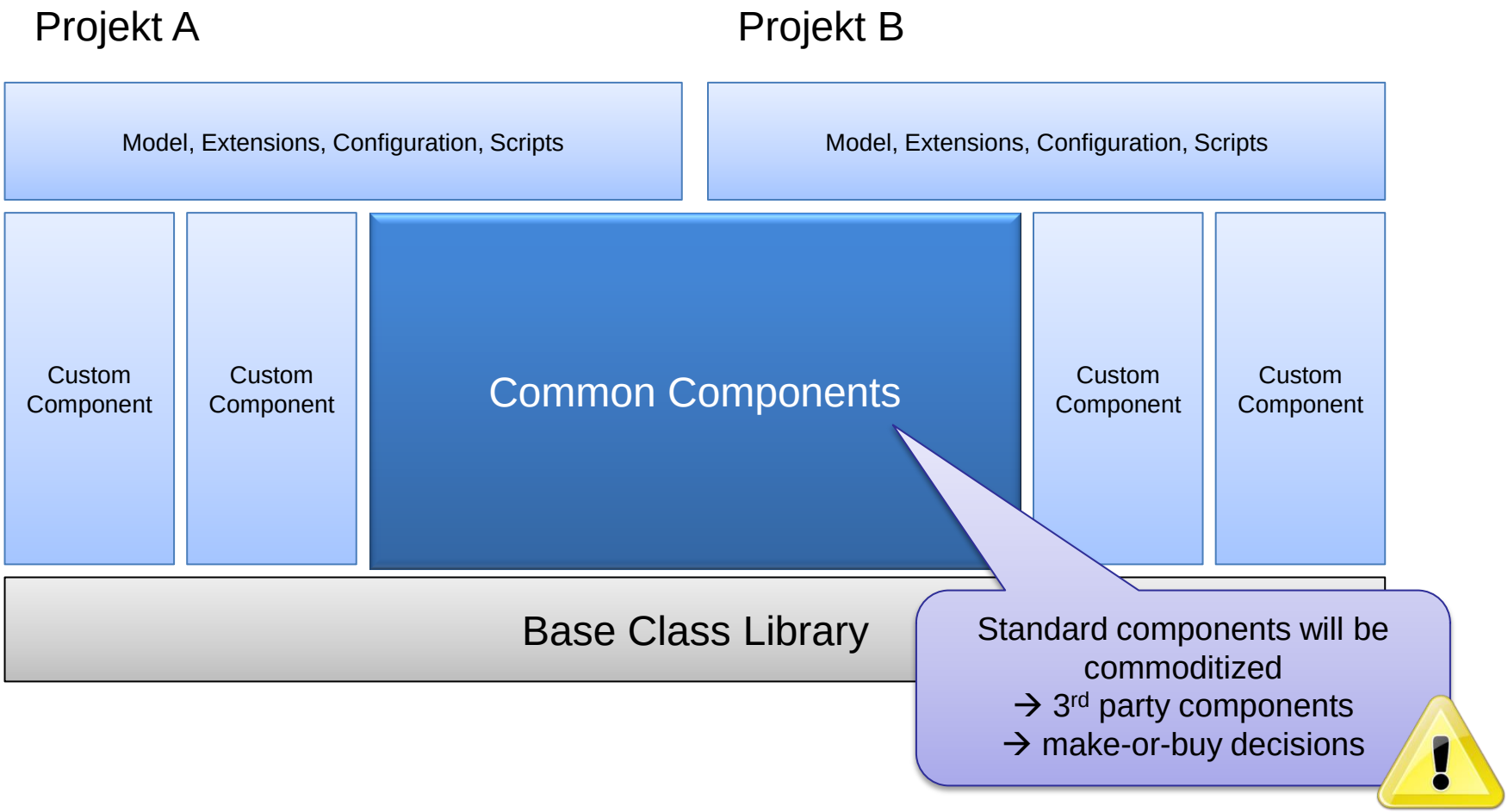
Entwicklung einer Software Factory



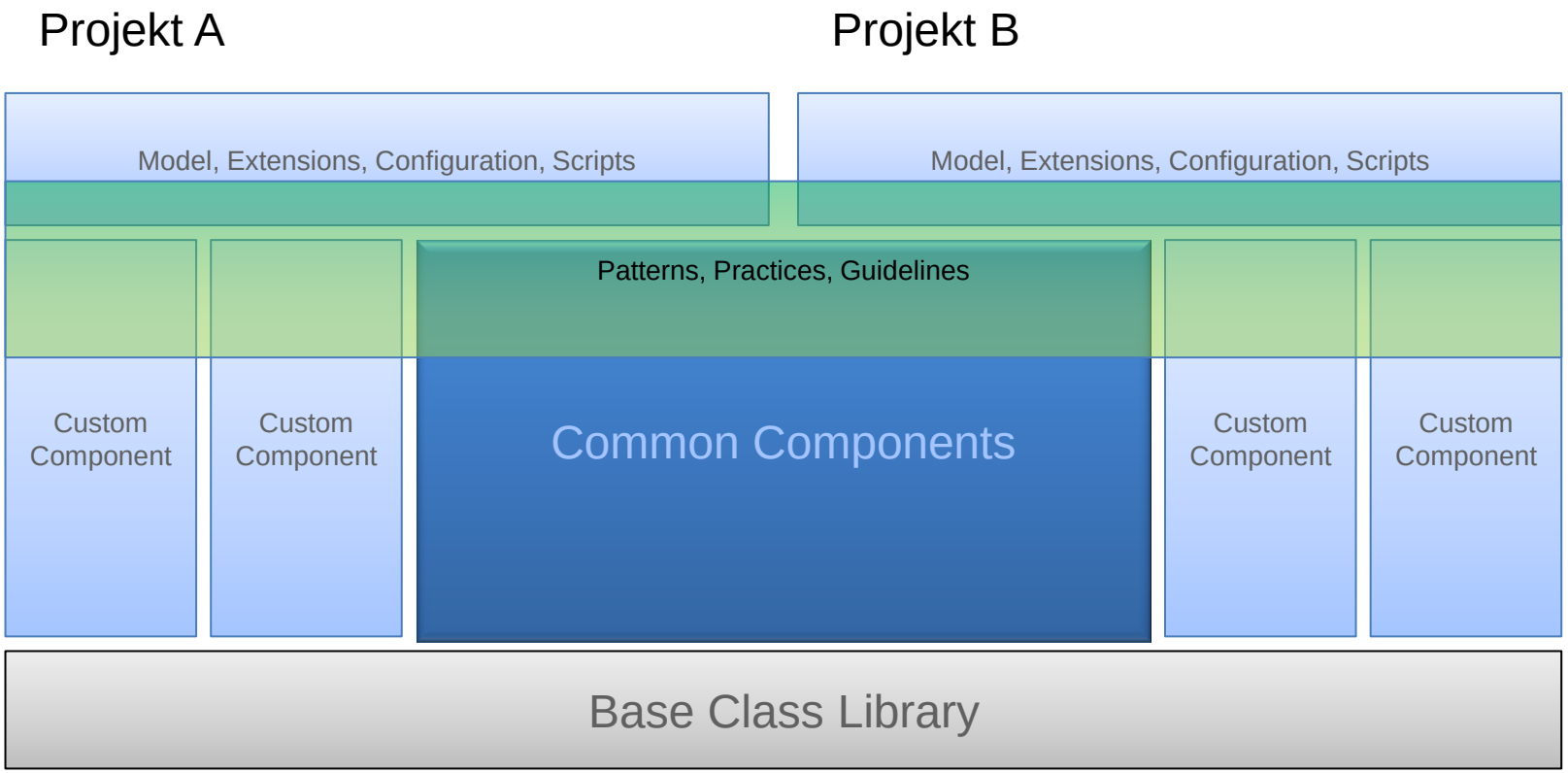
Entwicklung einer Software Factory



Entwicklung einer Software Factory



Entwicklung einer Software Factory

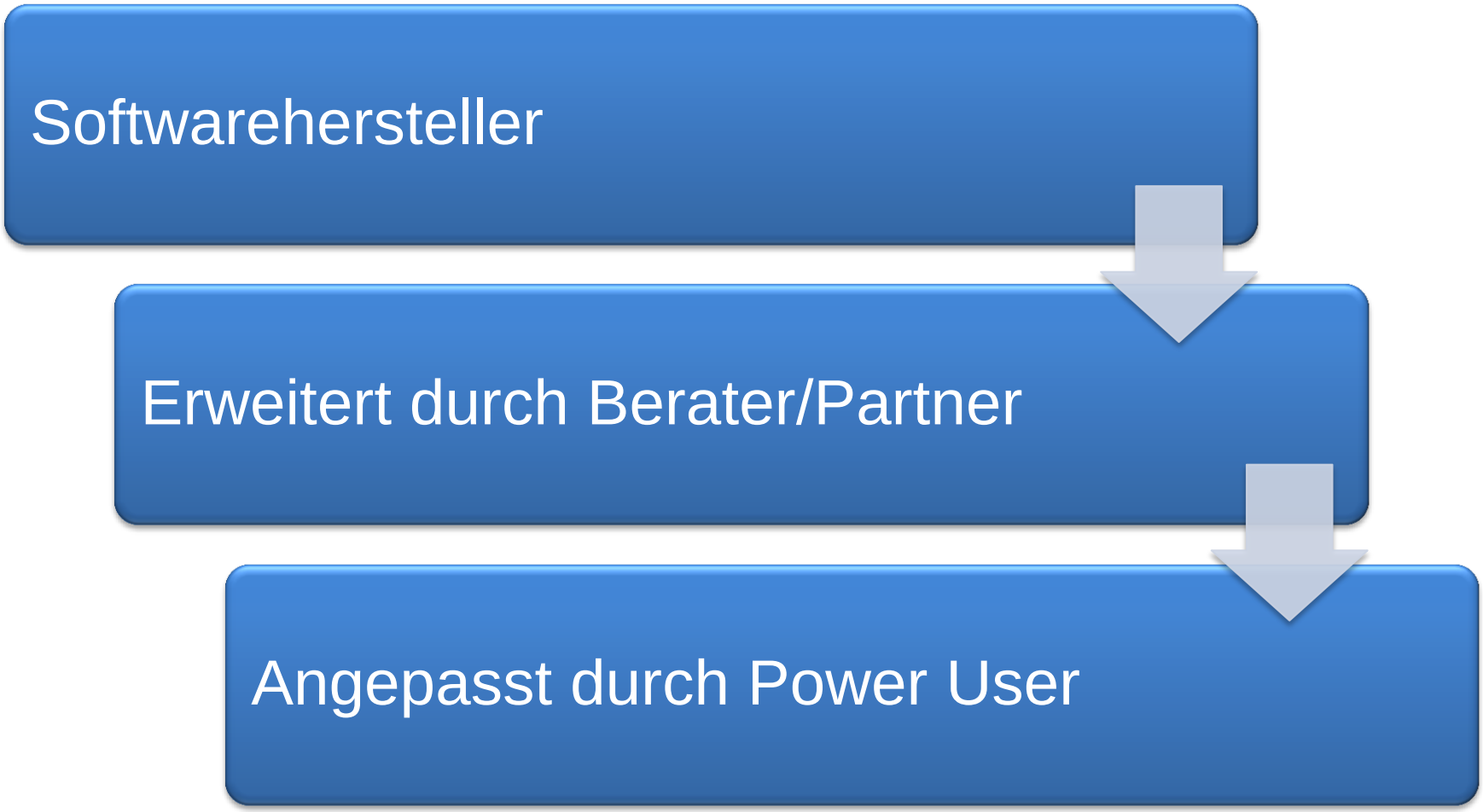


Nette Theorie, aber in der Praxis??

Herausforderungen

- Abstraktionsgrad
  - Je abstrakter desto mächtiger ☺
  - Je abstrakter desto spezifischer ☹
- Abhängigkeiten
  - Vertrauen in Werkzeuge
  - Vertrauen in Lieferanten
  - Vertrauen in Mitarbeiter
- Kleinster gemeinsamer Nenner
  - Ausnutzung aller Möglichkeiten der zugrunde liegenden Plattform
  - Performanceprobleme (Beispiel: OR-Mapper vs. T-SQL)

Benutzertypen und Software Factories



Software Factories im Kontext von Cloud Computing

- Platform as a Service (PaaS)
- Beispiele
  - Office 365 – SharePoint Online
  - SAP Business by Design
  - Microsoft Dynamics xRM
  - Force.com

Microsoft-Technologie

WERKZEUGE FÜR SOFTWARE FACTORIES

Werkzeuge

Beispiele aus der Microsoft-Welt

- Klassenbibliotheken
  - Dokumentation
  - Statische Codeanalyse
- Codegeneratoren
  - Vorlagen
  - Patterns in Form von Assistenten
- Domänenspezifische Sprachen
  - XML-basierend oder individuell (Compiler-Compiler)
  - Compiler (Codegenerator) vs. interpretiert
- Scriptsprachen
- Anwendungsmodularisierung
- Prozessautomatisierung
  - Qualitätssicherung
  - Build

- MS Framework Design Guidelines
  - Sandcastle
  - StyleCop, Code Analysis
- Codegeneratoren
  - T4, ANTLR StringTemplates
  - Visual Studio Templates
- Domänenspezifische Sprachen
  - XAML (UI und Workflows), EF ANTLR (Compiler-Compiler)
- DLR, Project „Roslin“
- MEF
- Prozessautomatisierung
  - Visual Studio Unit Tests
  - TFS Buildautomatisierung

Empfehlungen zur Erstellung wiederverwendbarer Klassenbibliotheken

KLASSENBIBLIOTHEKEN

Die wichtigsten fünf Gebote für Klassenbibliotheken

1. Je häufiger wiederverwendet desto höher muss die Qualität sein
   - An der zentralen Klassenbibliothek arbeiten Ihre besten Leute
   - Design, Code und Security Reviews
2. Folgen Sie den Microsoft Framework Design Guidelines (siehe Bibliographie nächste Slide)
   - Nutzen Sie StyleCop und Code Analysis (siehe folgende Slides)
3. Schreiben Sie Unit Tests
   - Gleiche Qualitätskriterien wie beim Framework selbst
   - Monitoring der Code Coverage
4. Steigern Sie die Produktivität mit Visual Studio Templates
5. Verwenden Sie Scenario Driven Design (siehe folgende Slides)

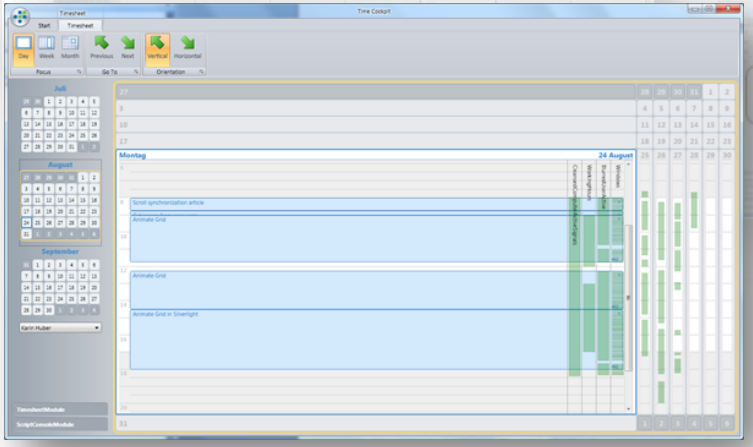
Bibliographie
Framework Design Guidelines

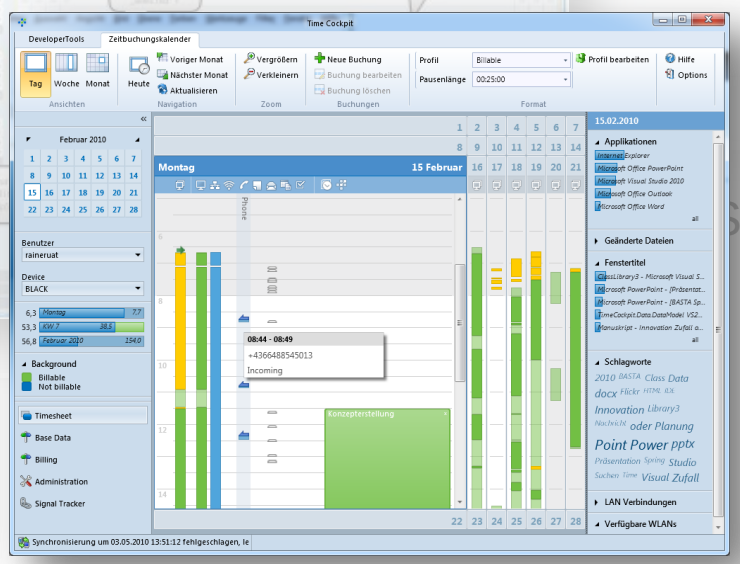
- Cwalina, Abrams: Framework Design Guidelines
  - Sporadische Aktualisierungen im Blog von Cwalina
  - Abrams ist nicht mehr bei MS (früherer Blog)
- Auszug aus dem Buch kostenlos in der MSDN verfügbar
  - Design Guidelines for Developing Class Libraries
- Scenario Driven Design
  - Blogartikel Cwalina
- Tipp: Portable Library Tools
  - Interessant für portable Libraries (.NET Framework, Silverlight, Windows Phone, XNA)
  - Noch nichts dergleichen für WinRT verfügbar

Früher Prototyp (Funktional)

UI Skizzen

Programmierte Prototypen





Skizzen

This solution would replace `Install.stg` and `CodeBatchCollection`. There would be a Xaml file compiled into time cockpit's resources. There has to be a function to apply all update batches in the Xaml file similar to today's `InstallBatchManager.Install`. Additionally there will be functions to

1. find out all update batches that are missing on a certain database.
2. find out if the application can work with a certain database.

The following code snippets show how the API to install update batches would work:

Get update batch from XAML file stored in the assembly's resources.

```
UpdateBatch updateBatch = this.ReadUpdateBatchFromResources();
```

Note that DbClient.Create *will* ***not*** *automatically install update batches in the future any more.*

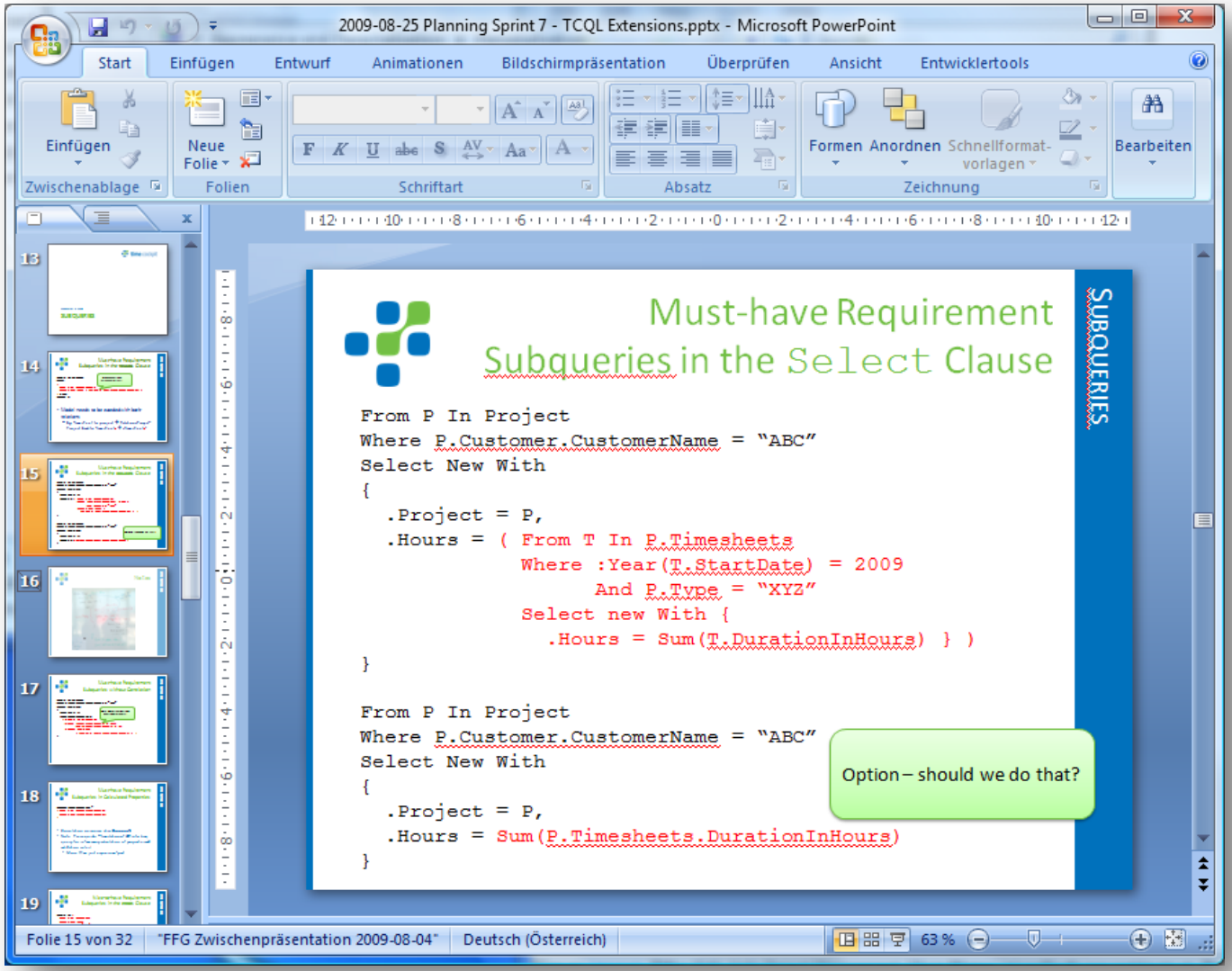
```
using (var dbClient = new DbClient.Create(...))
{
```

Find out which update batches are not installed in the database that dbClient *is pointing to.*

```
	IEnumerable<UpdateBatch> missingBatches =
		dbClient.GetMissingUpdateBatches(updateBatch);
	foreach (var missingBatch in missingBatches)
	{
		Console.WriteLine("{0} is missing", missingBatch.Guid);
	}

	Find out if app can run without executing any batches (i.e. if mandatory batches are missing).
	switch (dbClient.GetUpdateBatchStatus(updateBatch))
	{
		case BatchStatus.Complete:
			Console.WriteLine("Update batch is completely installed.");
			break;
		case BatchStatus.Acceptable:
			Console.WriteLine("Some non-mandatory batches are missing.");
			break;
		case BatchStatus.Incomplete:
			Console.WriteLine("Mandatory batches are missing.");
			break;
	}

	Install all missing update batches.
	dbClient.InstallMissingUpdateBatches(updateBatch);
}
```

Feature Files

On model level time cockpit will be extended by "features". A feature is a part of the logical data


2009-08-25 Planning Sprint 7 - TCQL Extensions.pptx - Microsoft PowerPoint
Start
Einfügen
Entwurf
Animationen
Bildschirmpräsentation
Überprüfen
Ansicht
Entwicklertools
Must-have Requirement
Subqueries in the Select Clause
SUBQUERIES
From P In Project
Where P.Customer.CustomerName = "ABC"
Select New With
{
.Project = P,
.Hours = (From T In P.Timesheets
Where :Year(T.StartDate) = 2009
And P.Type = "XYZ"
Select new With {
.Hours = Sum(T.DurationInHours) })
}
From P In Project
Where P.Customer.CustomerName = "ABC"
Select New With
{
.Project = P,
.Hours = Sum(P.Timesheets.DurationInHours)
}
Option – should we do that?
Folie 15 von 32
"FFG Zwischenpräsentation 2009-08-04"
Deutsch (Österreich)
63 %

Eine Software Factory in 5 Stunden

UNSER BEISPIEL

Szenarien - CTO

- Unsere Domäne
  - Kaufmännische, datenorientierte Anwendungen
- Unsere Ziele (Auszug)
  - Einheitliche Datenmodellierungskonventionen
  - Typische Anforderungen ohne Programmierung
    - Listen mit Suche und CRUD
  - Einfache Anforderungen durch Partner mit Scripts
    - Vorort, Workshopcharakter
  - Komplexe Anforderungen mit voller Power von VS
  - Versionierung Kundenlösung unabhängig vom Framework
  - Zukunftssicher
    - Win8, HTML/Javascript, SOA, Cloud, ...

Szenarien - Modellierung

- Konzentration auf fachliche Anforderungen
  - Keine technischen Details wie Felder für Sync, Archivierung, Audit, etc.
  - Abstrakte, domänenspezifische Datentypen
  - Sicherstellung von Konventionen im Hintergrund
    - Primary Keys, Namensschema, Datentypen je RDBMS, ...
  - Geschäftslogik im Modell
    - Excel-ähnlich
    - Prüfungen, Berechnungsformeln, Security,...
  - Mehrsprachigkeit
- Text am Anfang ok, später grafisches Tool

Szenarien - Modellierung

```
<DataModel>
    <DataModel.Entities>
        <Entity Code="Customer" InvariantFriendlyName="Kunden">
            <Entity.Properties>
                <TextProperty Code="CustomerCode"
                    InvariantFriendlyName="Kundennummer" MaximumLength="20" />
                <TextProperty Code="CustomerDescription"
                    InvariantFriendlyName="Kundenname" MaximumLength="100" />
                <NumericProperty Code="RevenueYTD„
                    InvariantFriendlyName="Jahresumsatz (akt.)" Scale="10" />
                <NumericProperty Code="RevenuePrevYear"
                    InvariantFriendlyName="Jahresumsatz Vorjahr" Scale="10" />
                <CalculatedProperty Code="TargetRatio„
                    InvariantFriendlyName="Zielerreichungsgrad (in %)„
                    Formula="RevenueYTD / RevenuePrevYear * 100" />
            </Entity.Properties>
        </Entity>
</DataModel>
```

Möglichkeit von Mehr-sprachigem Modell

Domänen-spezifische Datentypen

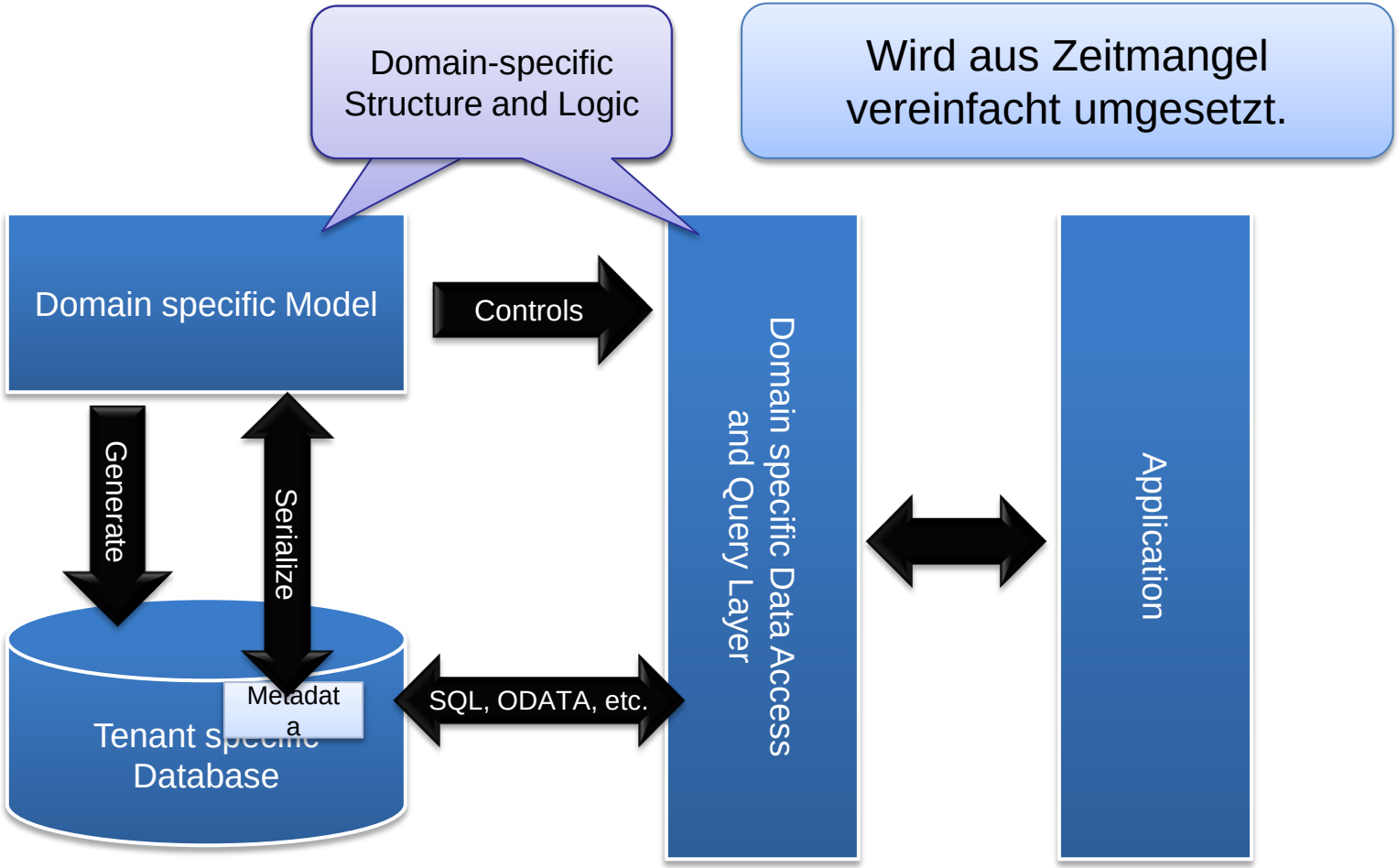
Excel-ähnliche Formelsprache

Szenarien - Modellierung

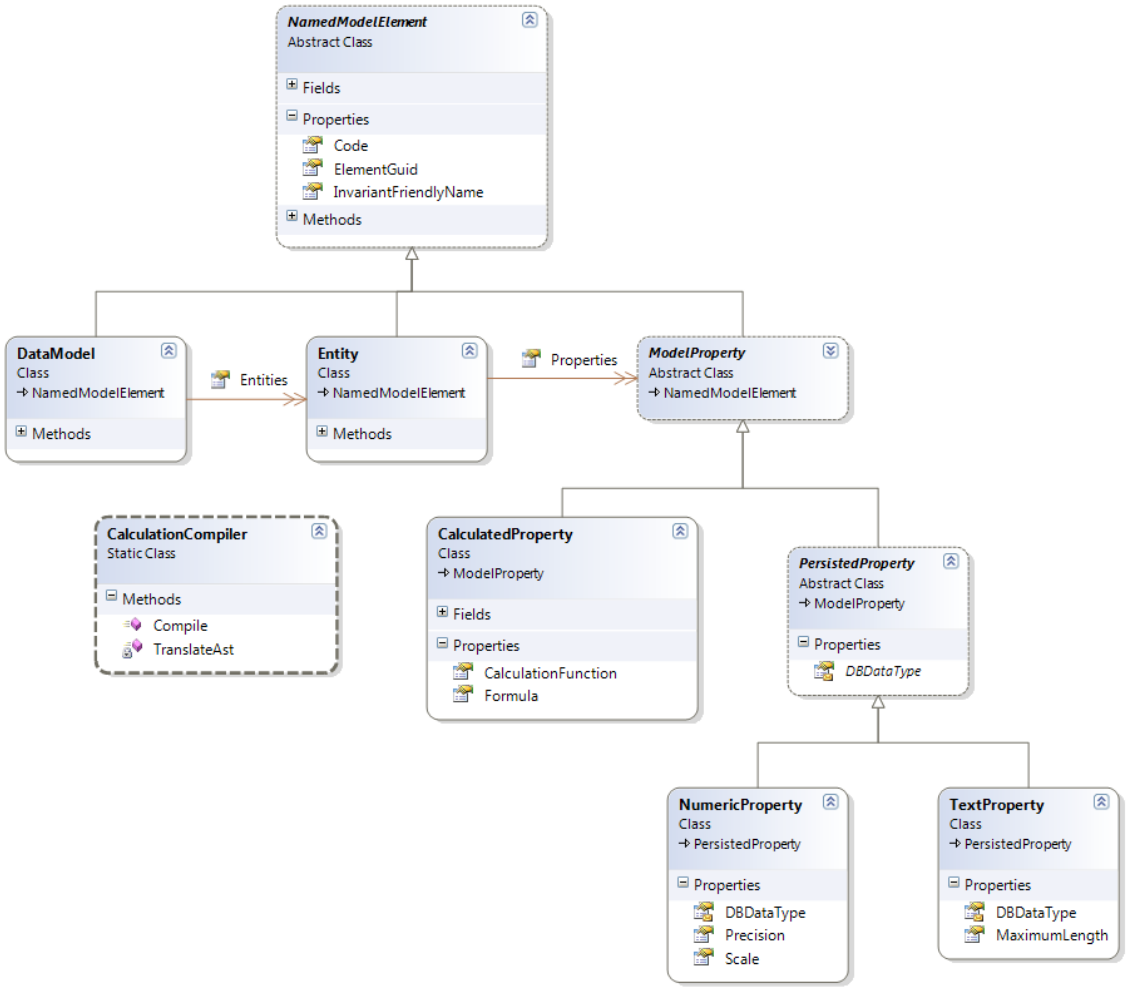
```
// Grundrechenarten, Klammernsetzung
Feld1 – Feld2 + (Feld3 – Feld4)

// Kostanten
Feld1 + 100

// Funktionen
Feld1 * :CovertToEur(Feld2 * 2)

...
```

Globale Bibliothek,
kundenspezifisch erweiterbar

Metadata Management



Metamodellierung + XAML



Anregung für domänenspezifische Modellierung

- Domänenspezifische Datentypen
  - Wertebereiche, Gültigkeitsregeln, Speicherformat, etc.
  - Zusammengesetzte Datentypen
  - Konstant hohe Qualität durch Vereinheitlichung
  - Mehrsprachigkeit
- Domänenspezifische, abstrakte Strukturen
  - Abbildung von Hierarchien
  - Abbildung von historischen Beziehungen
- Deklarative Geschäftslogik
  - Berechnete Spalten
  - Gültigkeitsregeln

Einleitung in StyleCop

STYLECOP

Was leistet StyleCop?

- Analysiert C# Source Code und überprüft Regeln bzgl. Stil und Konsistenz
  - Dokumentationsregeln
  - Layoutregeln
  - Regeln für bessere Wartbarkeit
  - Regeln für Namensschema
  - Reihenfolge der Memberdefinition
  - Lesbarkeitsregeln
  - Spacing Rules
  - Beschreibung aller Regeln in StyleCop Rules Guide (CHM)
- Open Source Projekt
  - Codeplex Site, NuGet Package
  - Blog
  - Version 4.6 Unterstützt C# 4 Syntax, ReSharper plugin
- Kann sowohl interaktiv als auch im Buildprozess verwendet werden

Bedarf nach mehr?
→ Versteckte Slides

- Zusammenfassung StyleCop Regeln
  - Dokumentationsregeln
  - Layoutregeln
  - Regeln für Spacing
  - Lesbarkeitsregeln
  - Reihenfolgeregeln
  - Namensregeln
  - Setting Files
  - Regeln deaktivieren
  - MSBuild Integration

Dokumentationsregeln

- SA16xx Regeln
- Jedes Element muss konsistent dokumentiert werden.
- Dokumentation muss valides XML sein.
- Leere/zu kurze/wiederholende Dokumentation nicht erlaubt ;-)
- Textprüfungen; Beispiele:
  - get/set ←→ Summary
  - Konstruktoren
- File Headers

```
//-----------------------------------------------------------
// <copyright file="Bucket.cs" company="Contoso Ltd.“>
//     Copyright (c) Contoso Ltd. All rights reserved.
// </copyright>
//-----------------------------------------------------------
```

  - Mit optionaler Inhaltsprüfung

Layoutregeln

- SA15xx Regeln
- Sind die geschwungenen Klammern dort wo sie sein sollen?
- Jedes Statement in einer eigenen Zeile
- Unnötige/fehlende Leerzeilen

Regeln für Spacing

- SA10xx Regeln
- Richtige Leerzeichen bei
  - C# Keywords
  - Komma
  - Semicolon
  - Symbolen
  - Dokumentation
  - Operatoren
  - Etc.
- Tabs must not be used
  - Darüber kann man streiten…

Lesbarkeitsregeln

- SA11xx Regeln
- Aufruf eines Members mit `base.` Nicht erlaubt wenn keine lokale Implementierung existiert
  - SA1100
- Zugriff auf Instanzmembers immer mit `this.`
  - SA1101
- Regeln für die Formatierung von LINQ Abfragen
- Keine leeren Statements, Kommentare, etc.
- Diverse Formatierungsregeln (Position von Klammern, etc.)

Reihenfolgeregeln

- SA12xx Regeln
- `usings`
  - Innerhalb des Namespace
  - Sortiert
- Reihenfolge innerhalb der Klasse:
  - Fields, Constructors, Finalizers (Destructors), Delegates, Events, Enums, Interfaces, Properties, Indexers, Methods, Structs, Classes
- Innerhalb bestimmt Access Modifier die Reihenfolge
  - `public, internal, protected internal, protected, private`

Generelle Namensregeln

- Diskussionen darüber im Team? Warum nicht einfach die Regeln von Microsoft übernehmen?
- `PascalCasing` für alle Identifier mit Ausnahme von Parameternamen
  - Ausnahme: Zweistellige, gängige Abkürzungen (z.B. `IOStream`, aber `HtmlTag` statt `HTMLTag`)
- `camelCasing` für Parameternamen
  - Ausnahme: Zweistellige, gängige Abkürzungen (z.B. `ioStream`)
- Fields? Nicht relevant, da nie public oder protected ;-)
- Dont‘s
  - Underscores
  - Hungarian notation (d.h. kein Präfix)
  - Keywords als Identifier
  - Abkürzungen (z.B. `GetWin`; sollte `GetWindow` heißen)

Namen für Assemblies und DLLs

- Keine Multifileassemblies
- Oft empfehlenswert, den Namespacenamen zu folgen
- Namensschema
  - `<Company>.<Component>.dll`
  - `<Produkt>.<Technology>.dll`
  - `<Project>.<Layer>.dll`
- Beispiele
  - `System.Data.dll`
  - `Microsoft.ServiceBus.dll`
  - `TimeCockpit.Data.dll`

Namen für Namespaces

- Eindeutige Namen verwenden (z.B. Präfix mit Firmen- oder Projektname)
- Namensschema
  - `<Company>.<Product|Technology>[.<Feature>][.<Subnamespace>]`
- Versionsabhängigkeiten soweit möglich vermeiden (speziell bei Produktnamen)
- Organisatorische Strukturen beeinflussen Namespaces nicht
- Typen nicht gleich wie Namespaces benennen
- Nicht zu viele Namespaces
- Typische Szenarien von seltenen Szenarien durch Namespaces trennen (z.B. `System.Mail` und `System.Mail.Advanced`)
- `PascalCasingWithDotsRecommended`

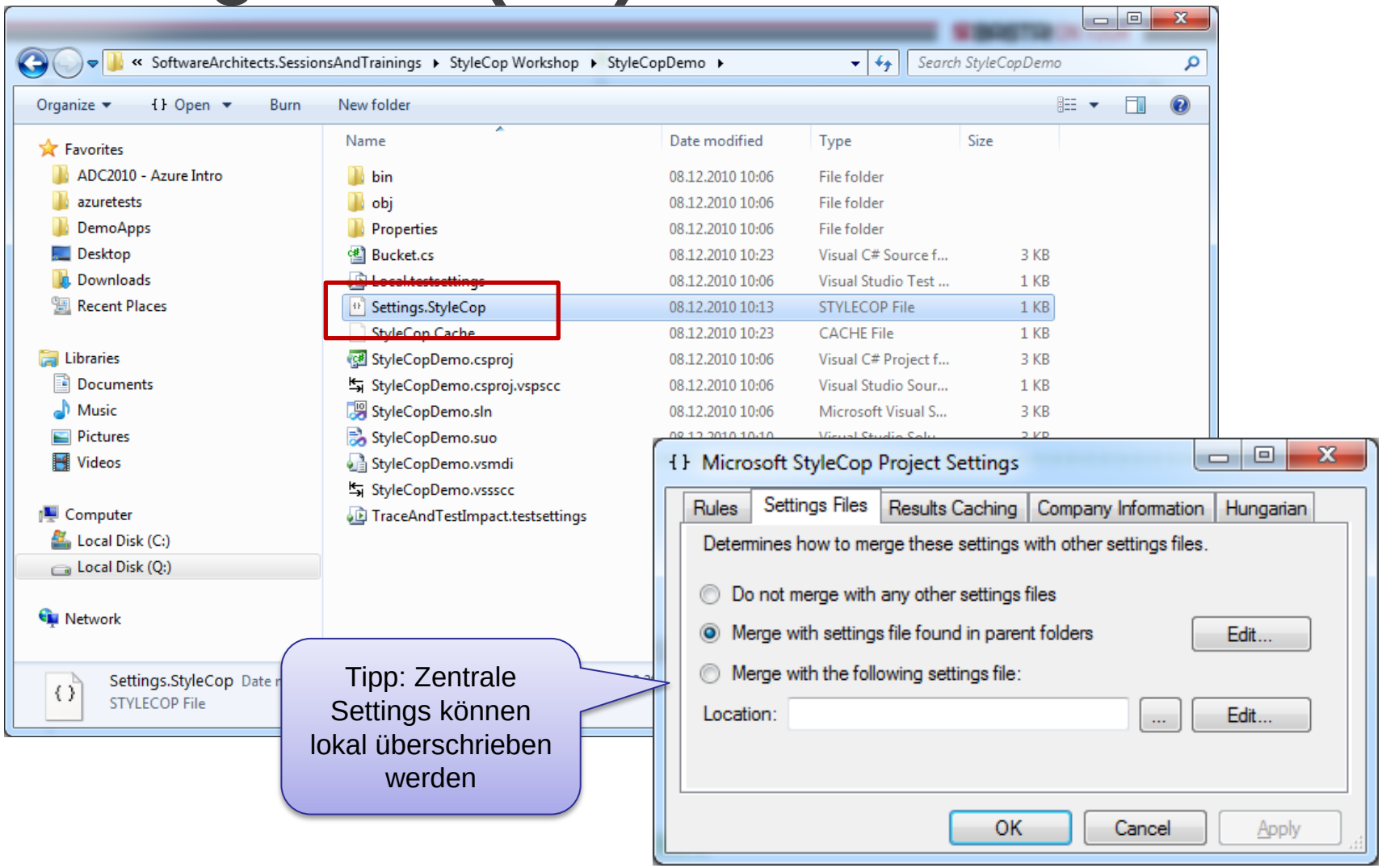
Klassen-, Struktur- und Interfacenamen

- `PascalCasingRecommended`
- Keine üblichen oder bekannten Typnamen verwenden (z.B. keine eigene Klasse `File` anlegen)
- Klassen- und Strukturnamen
  - Meist Hauptwörter (z.B. `Window`, `File`, `Connection`, `XmlWriter` etc.)
- Interfacenamen
  - Wenn sie eine Kategorie repräsentieren, Hauptwörter (z.B. `IList`, etc.)
  - Wenn sie eine Fähigkeit ausdrücken, Adjektiv (z.B. `IEnumerable`, etc.)
- Kein Präfix (z.B. „C…“)
  - „I“ für Interfaces ist historisch gewachsen und deshalb sehr bekannt

Membernamen

- `PascalCasingRecommended`
- Methoden
  - Verb als Name verwenden (z.B. `Print`, `Write`, `Trim`, etc.)
- Properties
  - Hauptwörter oder Adjektive (z.B. `Length`, `Name`, etc.)
  - Mehrzahl für Collection Properties verwenden
  - Aktiv statt passiv (z.B. `CanSeek` statt `IsSeekable`, `Contains` statt `IsContained`, etc.)
- Events
  - Verb als Name verwenden (z.B. `Dropped`, `Painting`, `Clicked`, etc.)
  - Gegenwart und Vergangenheit bewusst einsetzen (z.B. `Closing` und `Closed`, etc.)
  - Bei Eventhandler typisches Pattern verwenden (`EventHandler`-Postfix, `sender` und `e` als Parameter, `EventArgs` als Postfix für Klassen
- Fields
  - Keine `public` oder `protected` Fields
  - Kein Präfix

Setting-Files (1/2)



Setting-Files (2/2)

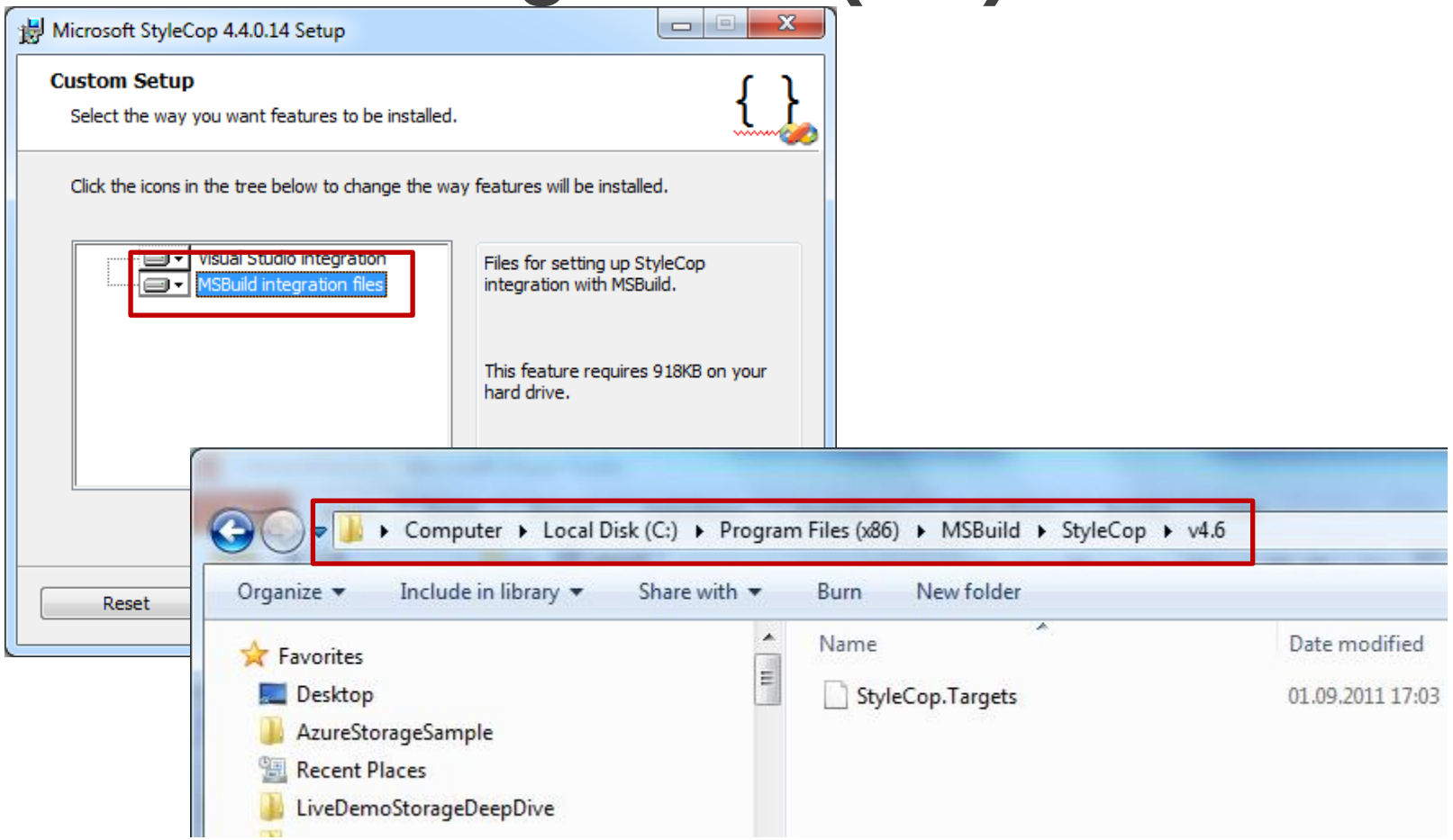
```
<StyleCopSettings Version="4.3">
  <Analyzers>
    <Analyzer AnalyzerId="Microsoft.StyleCop.CSharp.ReadabilityRules">
      <Rules>
        <Rule Name="DoNotUseRegions">
          <RuleSettings>
            <BooleanProperty Name="Enabled">True</BooleanProperty>
          </RuleSettings>
        </Rule>
      </Rules>
      <AnalyzerSettings />
    </Analyzer>
    <Analyzer AnalyzerId="Microsoft.StyleCop.CSharp.SpacingRules">
      <Rules>
        <Rule Name="TabsMustNotBeUsed">
          <RuleSettings>
            <BooleanProperty Name="Enabled">False</BooleanProperty>
          </RuleSettings>
        </Rule>
      </Rules>
      <AnalyzerSettings />
    </Analyzer>
  </Analyzers>
</StyleCopSettings>
```

StyleCop Regeln deaktivieren

- StyleCop Regeln können mit dem Attribut `SuppressMessage` situativ deaktiviert werden
  - Details zum Attribut siehe MSDN
- Beispiel:

```
[SuppressMessage(
  "Microsoft.StyleCop.CSharp.DocumentationRules",
  "SA1600:ElementsMustBeDocumented",
  Justification = "No time to write documentation...")]
  public class Bucket<T>
  {
     …
  }
```

MSBuild Integration (1/2)



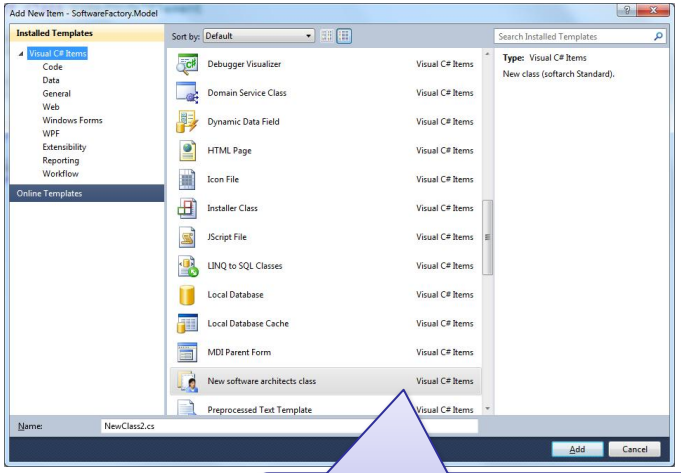
MSBuild Integration (2/2)

- Import für StyleCop zum Project-File hinzufügen
  - `<Import Project="$(ProgramFiles)\MSBuild\StyleCop\v4.6\StyleCop.targets" />`
- Falls StyleCop Warning als Errors behandelt werden sollen
  - Flag zur ersten PropertyGroup im Projekt-File hinzufügen
  - `<StyleCopTreatErrorsAsWarnings>`
    `false`
    `</StyleCopTreatErrorsAsWarnings>`

Routinetätigkeiten vermeiden, Patterns verbreiten

VISUAL STUDIO TEMPLATES

Einleitung in VS Templates



"Add New Item" Wizard

- Vorlagen für
  - Projekte
  - Projektelemente
- Bestehen aus
  - Vorlagendateien
  - Template Metadata Files (`.vstemplate`)
  - (Optional) Individuelle Wizards (Custom Code; siehe `Microsoft.VisualStudio.TemplateWizard.IWizard`)
- Manuelle Verteilung oder Installation
  - VSIX Dateien

Bibliographie VS Templates

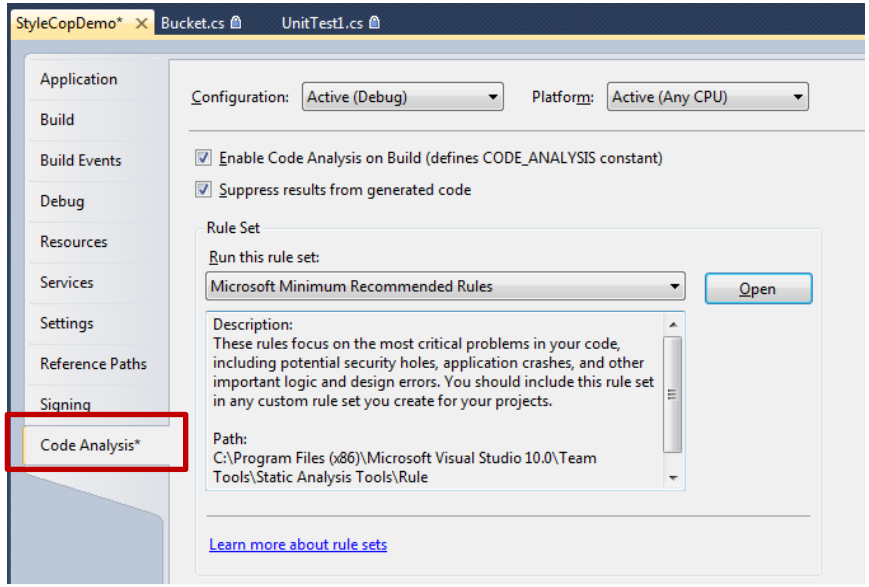
- Weitere Unterlagen
  - Visual Studio Templates
  - Creating Project Templates
  - Creating Item Templates
  - How to: Publish Project Templates
    - VSIX Project Template Files

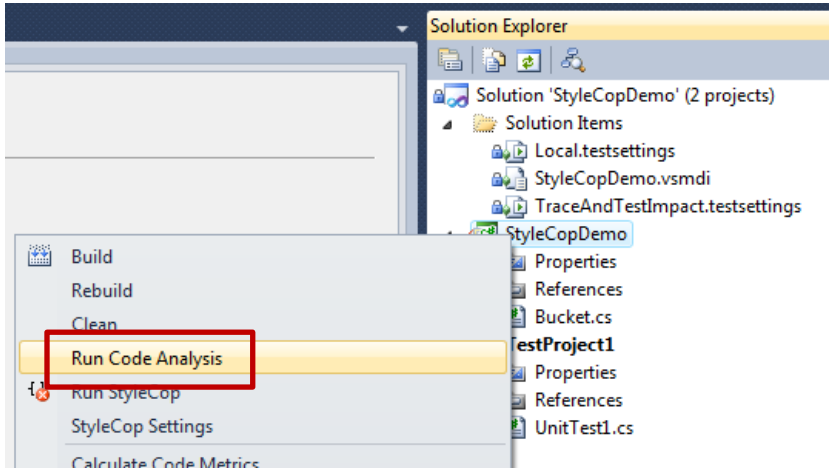
Früher FxCop, jetzt…
Leider nur Ultimate und Premium ☹

VISUAL STUDIO CODE ANALYSIS

Was leistet Code Analysis?

- Analysiert managed assemblies (nicht Sourcecode)
- Kontrolliert die Einhaltung der Framework Design Guidelines für .NET





Rule Sets (1/2)

- Eingebaute Rule Sets
  (Beschreibung in den Slides würde den Rahmen sprengen)
  - Siehe MSDN
- Können angepasst werden
  - Siehe MSDN
- Situative Unterdrückung von Code Analysis Warnings/Error wie bei StyleCop mit `SuppressMessage`
  - Beispiel:

```
[SuppressMessage("Microsoft.Design",
  "CA1039:ListsAreStrongTyped")]
Public class MyClass
{
    // code
}
```

Rule Sets (2/2)

- Unterdrückung globaler Code Analysis Warnings/Error
  - Beispiel:

```
[assembly: SuppressMessage(
 "Microsoft.Design",
 "CA1020",
 Justification = "More will come later",
 Scope = "namespace",
 Target = "SoftwareFactory.DataAccess")]
```

Aktuelle Entwicklungen

- In Zukunft alles async (vgl. WinRT)
  - Task-based Async Pattern
  - Must Read: TAP White Paper
- C#: `async` und `await`
  - Async CTP (mit go-live Support)
  - VS 2011 Developer Preview
- Thread-Safe programmieren
  - Locking
  - Lock-free wo möglich
  - Immutables
  - Weitere Informationen: Visual Studio Asynchronous Programming
- **Eigene BASTA! Session am Dienstag**

http://www.timecockpit.com/en/blogs/10-12-08/Hands-On_Labs_StyleCop_and_Code_Analysis.aspx

HANDS-ON LAB FÜR ZU HAUSE

T4

TEMPLATES

dataModel	{SoftwareFactory.Model.DataModel}
base	{SoftwareFactory.Model.DataModel}
Entities	Count = 2
[0]	{SoftwareFactory.Model.Entity}
base	{SoftwareFactory.Model.Entity}
Properties	Count = 5
[0]	{SoftwareFactory.Model.TextProperty}
[1]	{SoftwareFactory.Model.TextProperty}
[2]	{SoftwareFactory.Model.NumericProperty}
[3]	{SoftwareFactory.Model.NumericProperty}
[4]	{SoftwareFactory.Model.CalculatedProperty}
Raw View	
[1]	{SoftwareFactory.Model.Entity}
base	{SoftwareFactory.Model.Entity}
Properties	Count = 3
[0]	{SoftwareFactory.Model.TextProperty}
[1]	{SoftwareFactory.Model.TextProperty}
[2]	{SoftwareFactory.Model.TextProperty}
Raw View	
Raw View	

Template Engines

Sequentieller Inhalt mit Struktur

Baumstruktur im Speicher

Sequentielle Ausgabe

Operationen

```
<DataModel xmlns="clr-namespace:SoftwareFactory.Model;assembly=Software
    <DataModel.Entities>
        <Entity Code="Customer" InvariantFriendlyName="Kunden">
            <Entity.Properties>
                <TextProperty Code="CustomerCode" InvariantFriendlyName
                <TextProperty Code="CustomerDescription" InvariantFrien
                <NumericProperty Code="RevenueYTD" InvariantFriendlyNam
                <NumericProperty Code="RevenuePrevYear" InvariantFriend
                <CalculatedProperty Code="TargetRatio" InvariantFriendl
                                    Formula="RevenueYTD / RevenuePrevYe
            </Entity.Properties>
        </Entity>

        <Entity Code="Employees" InvariantFriendlyName="Mitarbeiter">
            <Entity.Properties>
                <TextProperty Code="EmployeeCode" InvariantFriendlyName
                <TextProperty Code="FirstName" InvariantFriendlyName="V
                <TextProperty Code="LastName" InvariantFriendlyName="Na
            </Entity.Properties>
        </Entity>
    </DataModel.Entities>
</DataModel>
```

```
nerated Text Transformation
<#@ template language="C#" #>
<#@ import namespace="System.Linq" #>

CREATE TABLE <#= this.Schema #> (
    <#= this.Entity.Code #>_ID UNIQUEIDENTIFIER PRIMARY K
    <#
        foreach ( var property in this.Entity.Properties.
        {
            #>, <#= property.Code #> <#= property.DBDataT
            this.WriteLine(string.Empty);
        }
    #>
);
GO
```

Template Engines

- Erzeugen beliebige Textausgaben
- Ausgangspunkt ist ein Template mit Platzhaltern (und Code)
- Input ist eine beliebige Objektstruktur im Speicher
- Anwendungsbereiche
  - Codegenerierung
  - Generierung von XML oder HTML
  - Compile time vs. Runtime
- Zwei Beispiele
  - Microsoft T4
  - ANTLR StringTemplate

ANTLR StringTemplate

- Open Source
- Implementierungen
  - Java
  - C#
    - Full Client und Silverlight
  - Python
  - Objective-C
- Rudimentäre Debugging-UI verfügbar (String Template Inspector GUI)

ANTLR StringTemplate Beispiel

Deklarativ, keine Schleifen, IFs, etc.

```
group DataModelTemplates;

CreateTable(context, entity) ::=
<<
CREATE TABLE [<context.Tenant>].[<entity.Name>]
(
        <entity.Name>Uuid uniqueidentifier NOT NULL,
        <entity.Properties:{ p | <p:(p.Type.Name)()>}; separator=",\n">
)
ALTER TABLE [<context.Tenant>].[<entity.Name>]
        ADD CONSTRAINT DF_<entity.Name>_<entity.Name>Uuid
        DEFAULT newid() FOR <entity.Name>Uuid
ALTER TABLE [<context.Tenant>].[<entity.Name>]
        ADD CONSTRAINT PK_<entity.Name> PRIMARY KEY CLUSTERED
        ( <entity.Name>Uuid )
>>

TextProperty(property) ::= "<property.Name> varchar(50) NULL„

NumericProperty(property) ::= "<property.Name> numeric(18,4) NULL”
```

Verschachteltes Template

Microsoft T4 Template Engine

- Tief in Visual Studio integriert
- Development Time, Compile Time, Runtime
- Implementierungen (Runtime)
  - C#
  - VB
- VS-integrated Editor von Tangible
  - Visual Studio Gallery

T4
Beispiel

```
<#@ template language="C#" #>
<#@ import namespace="System.Linq" #>

CREATE TABLE <#= this.Schema #> (
    <#= this.Entity.Code #>_ID UNIQUEIDENTIFIER PRIMARY KEY
    <#
        foreach ( var property in
this.Entity.Properties.OfType<PersistedProperty>() )
        {
            #>, <#= property.Code #> <#= property.DBDataType #><#
            this.WriteLine(string.Empty);
        }
    #>
);
GO
```

Sehr Code-lastig! Gefahr, Logik in Templates zu verpacken.

Yet another ORM???

DATENZUGRIFF

Datenzugriffsmöglichkeiten

- Direkter DB-Zugriff
  - Wenig Aufwand, wenig Kontrolle
  - SQL ist bereits dynamisch
- Code generieren
  - Early Binding → viele Vorteile
  - Schlecht für Multi-Tenancy
  - Komiler notwendig
- Deklarativ
  - Nur für absehbare Szenarien denkbar
- Key-Value pair Modell
  - Nicht sehr bequem
  - Dynamische Sprache für Arme
- Dynamische Sprachen
  - Das probieren wir

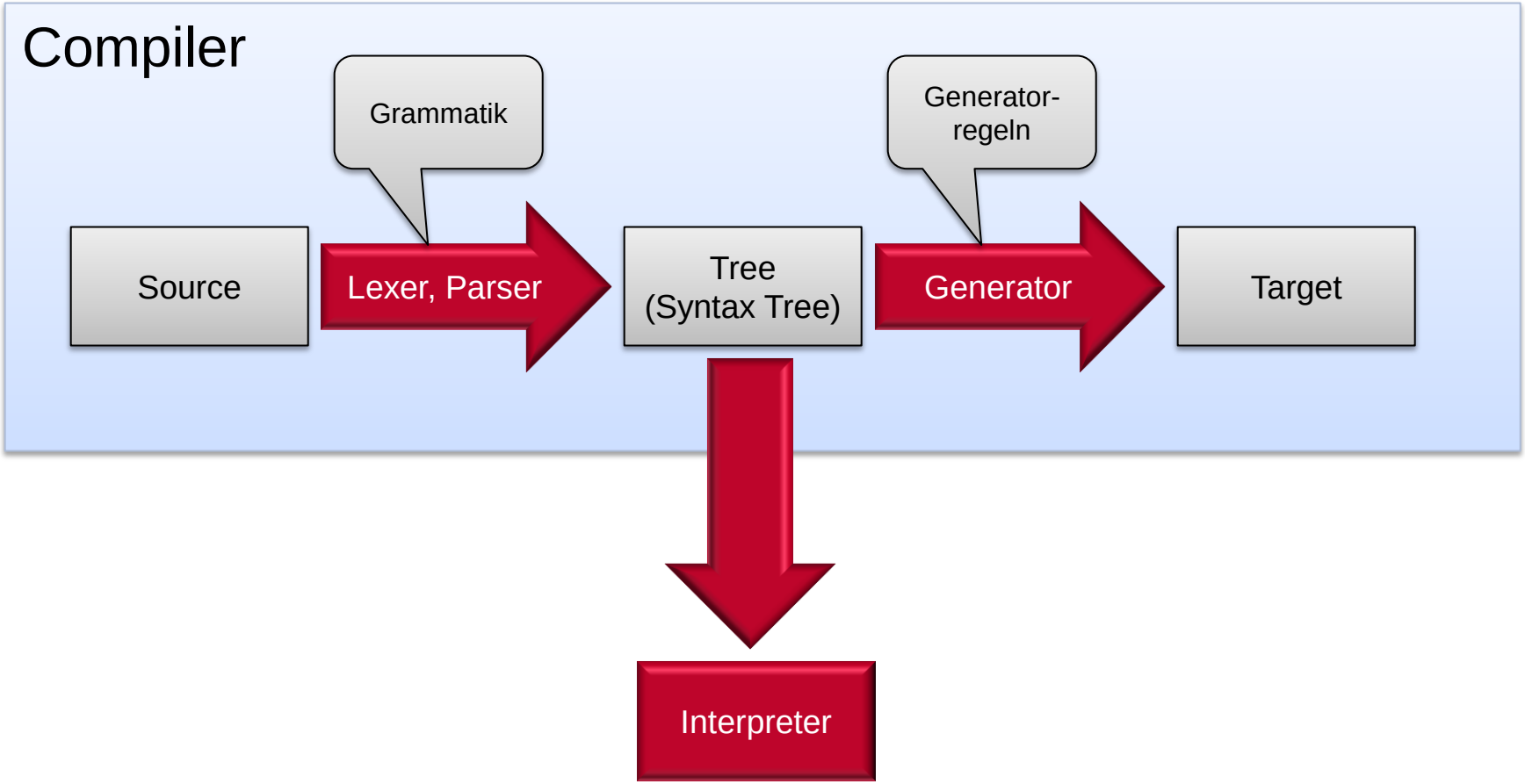
Szenarien - Datenzugriff

```
customer = CreateObject("Customer")
customer.CustomerCode = "Cust" + str(i)
customer.CustomerDescription = "Customer " + str(i)
customer.RevenueYTD = i * 1000
customer.RevenePrevYear = i * 1000 - 100
```

DSLs

DOMÄNENSPEZIFISCHE SPRACHEN

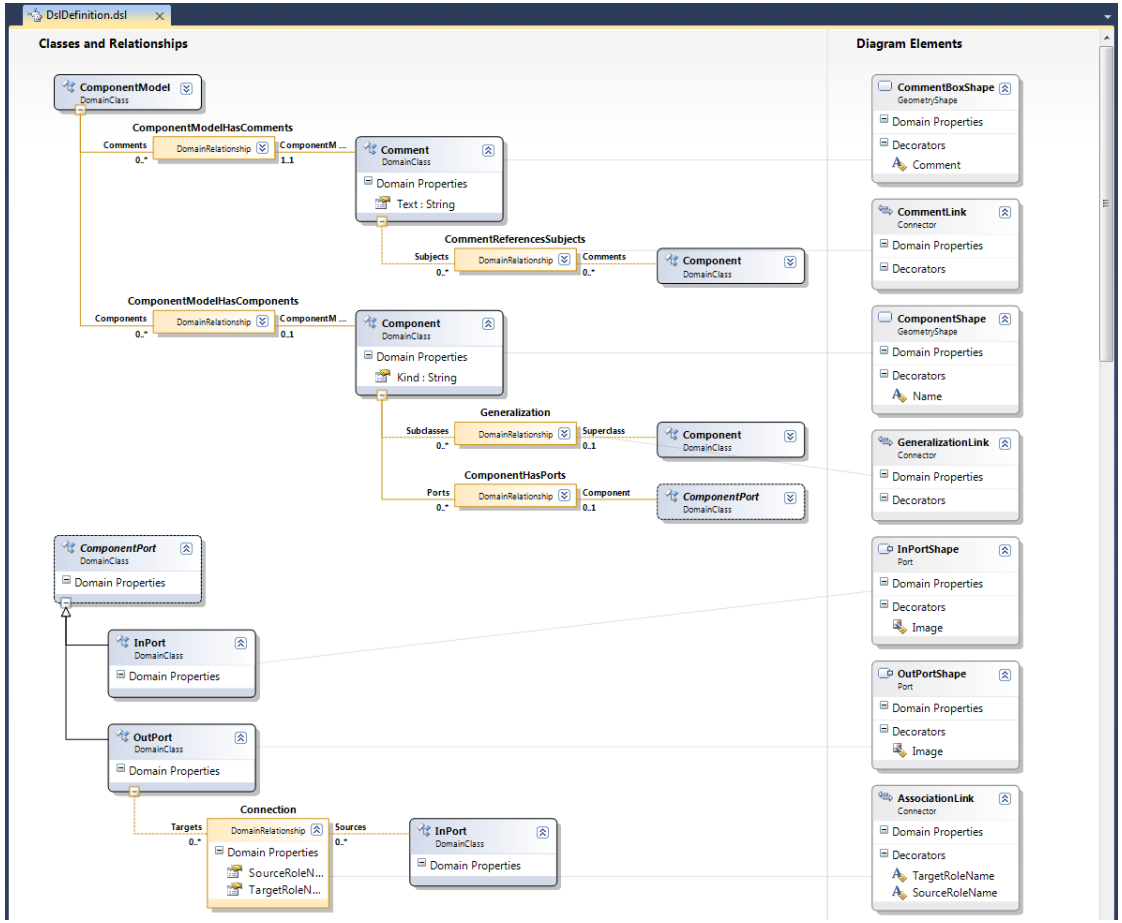
Wichtige Begriffe



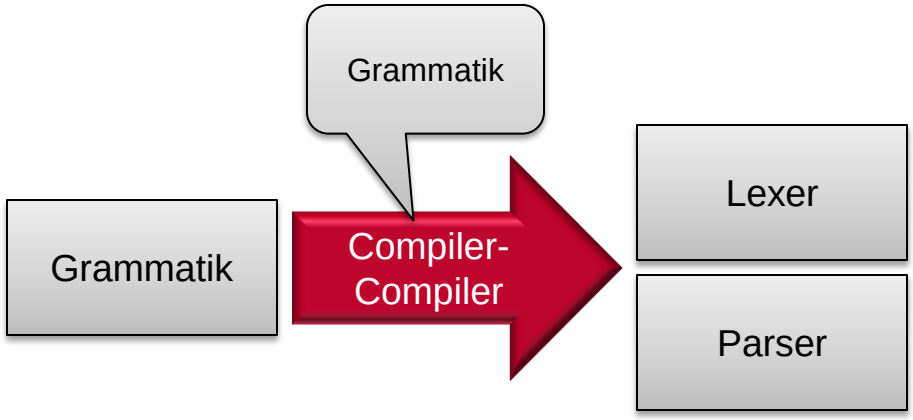
Einige Beispiele

- Lexer/Parser
  - XML in DOM
  - SQL in Execution Plan
- Compiler bzw. Lexer/Parser/Generator
  - C# in IL
  - FetchXML in SQL (MS CRM)
- Interpreter
  - SQL Server Execution Plan
- Compiler-Compiler
  - ANTLR
  - Visual Studio 2010 Visualization and Modeling SDK
  - Coco/R

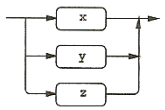
Grafische DSL



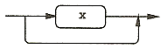
Wichtige Begriffe



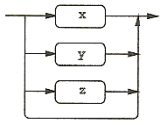
EBNF



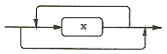
(«x»|«y»|«z»)
Match any alternative within the subrule exactly once.



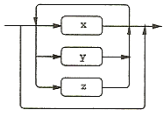
x?
Element *x* is optional.



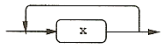
(«x»|«y»|«z»)?
Match nothing or any alternative within subrule.



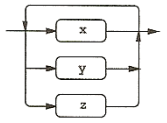
*x**
Match element *x* zero or more times.



(«x»|«y»|«z»)*
Match an alternative within subrule zero or more times.



x+
Match element *x* one or more times.



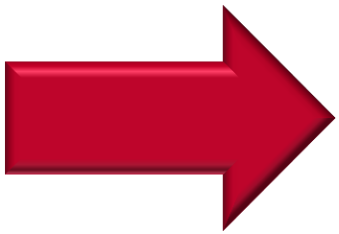
(«x»|«y»|«z»)+
Match an alternative within subrule one or more times.

Quelle: The Definitive ANTLR Reference, Terence Parr

Figure 4.3: EBNF GRAMMAR SUBRULES WHERE «...» REPRESENTS A GRAMMAR FRAGMENT

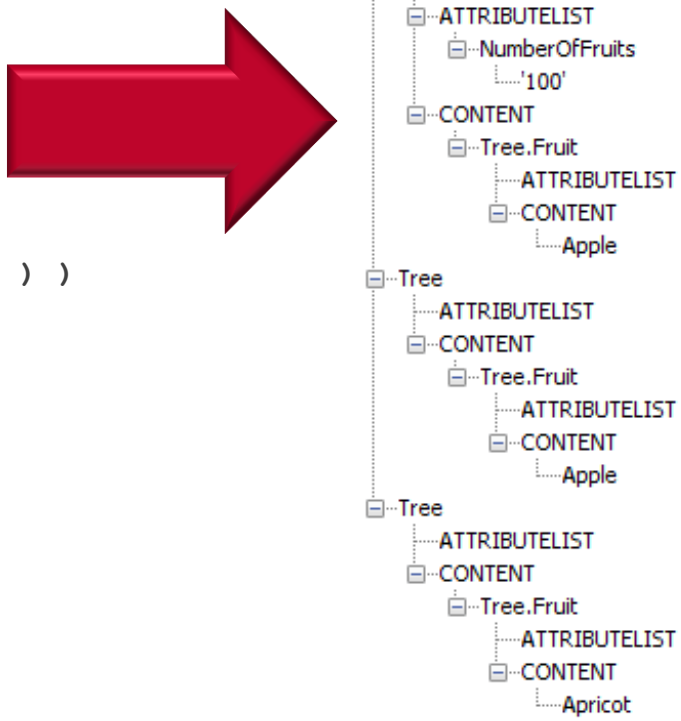
Praktisches Beispiel

```
grammar XmlLanguage2;
options { output = AST; }

// PARSER ---------------------------------
xmlDocument : node;
node
    : '<'! ELEMENTNAME attributeList '>'!
        ( node )*
      '</'! ELEMENTNAME '>'!
    | '<'! ELEMENTNAME '/>'!;

attributeList : attribute*;
attribute : ELEMENTNAME '='! LITERAL;

// LEXER ----------------------------------
ELEMENTNAME
    : IDENTIFIER ( '.' IDENTIFIER )?;
LITERAL
    : '\'' ( ~'\'' )* '\'';
fragment IDENTIFIER
    : ( 'a'..'z' | 'A'..'Z' | '_' ) ( 'a'..'z' | 'A'..'Z' | '0'..'9' )*;
NEWLINE
    : ('\r'? '\n')+ { $channel = HIDDEN; };
WHITESPACE
    : ( '\t' | ' ' )+ { $channel = HIDDEN; } ;
```



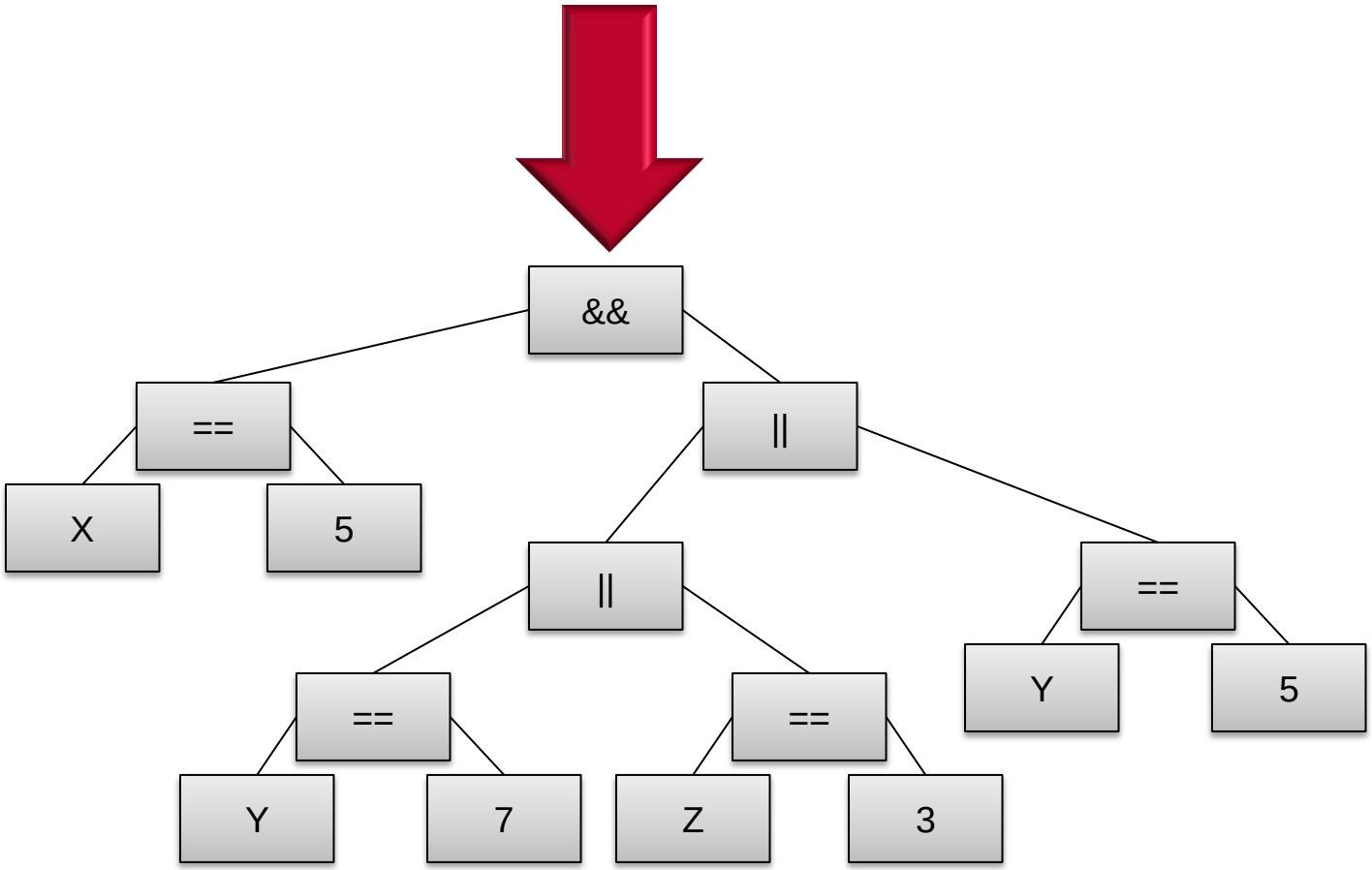
Praktisches Beispiel

```
grammar XmlLanguage;
options { output = AST; }
tokens {
    NODE = 'Node';
    ATTRIBUTELIST = 'AttributeList';
    ATTRIBUTE = 'Attribute';
    CONTENT = 'CONTENT';
}

// PARSER ---------------------------------
xmlDocument
    : node;
node
    : '<' start=ELEMENTNAME attributeList '>' ( node )*
      '</' end=ELEMENTNAME '>'
        -> ^( NODE [$start] attributeList ^( CONTENT node* ) )
    | '<' tag=ELEMENTNAME '/>'
        -> ^( NODE [$tag] );
attributeList
    : attribute*
        -> ^( ATTRIBUTELIST attribute* );
attribute
    : attribName=ELEMENTNAME '=' LITERAL
        -> ^( ATTRIBUTE [$attribName] LITERAL );

// LEXER ----------------------------------
[…]
```



Wo ist der Baum?

```
X=5 And ( Y=7 Or Z=3 Or Y=5 )
```



Praktisches Beispiel

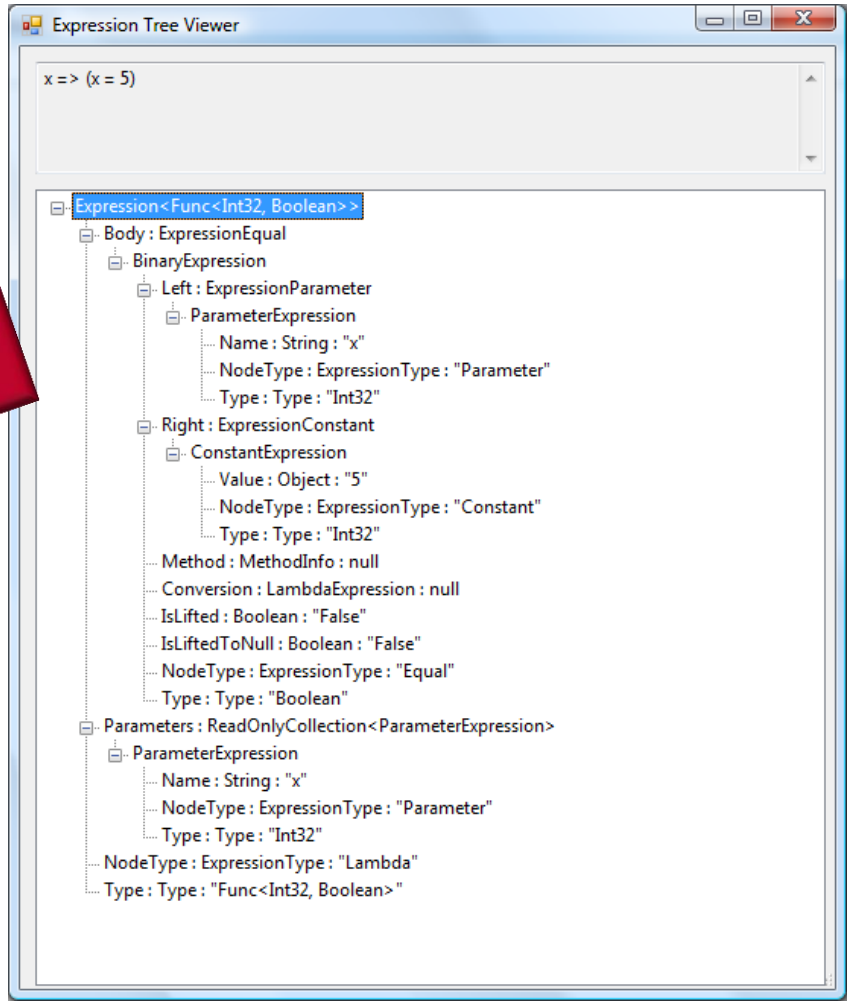
```
public void ParseBackgroundFormula()
{
    var stream = new ANTLRStringStream(this.BackgroundFormula);
    var lexer = new ExpressionLanguageLexer(stream);
    var tokens = new CommonTokenStream(lexer);
    var parser = new ExpressionLanguageParser(tokens);
    parser.TreeAdaptor = new ExpressionLanguageTreeAdaptor();
    this.Ast = new [] { parser.expression().Tree as ExpressionLanguageCommonTree };
}
```

Microsoft Expression Trees

AST IN C#

ExpressionTrees in C#

```
Func<int, bool> f =
    (x) => x==5;

Expression<Func<int, bool>> ex =
    (x) => x == 5;
```

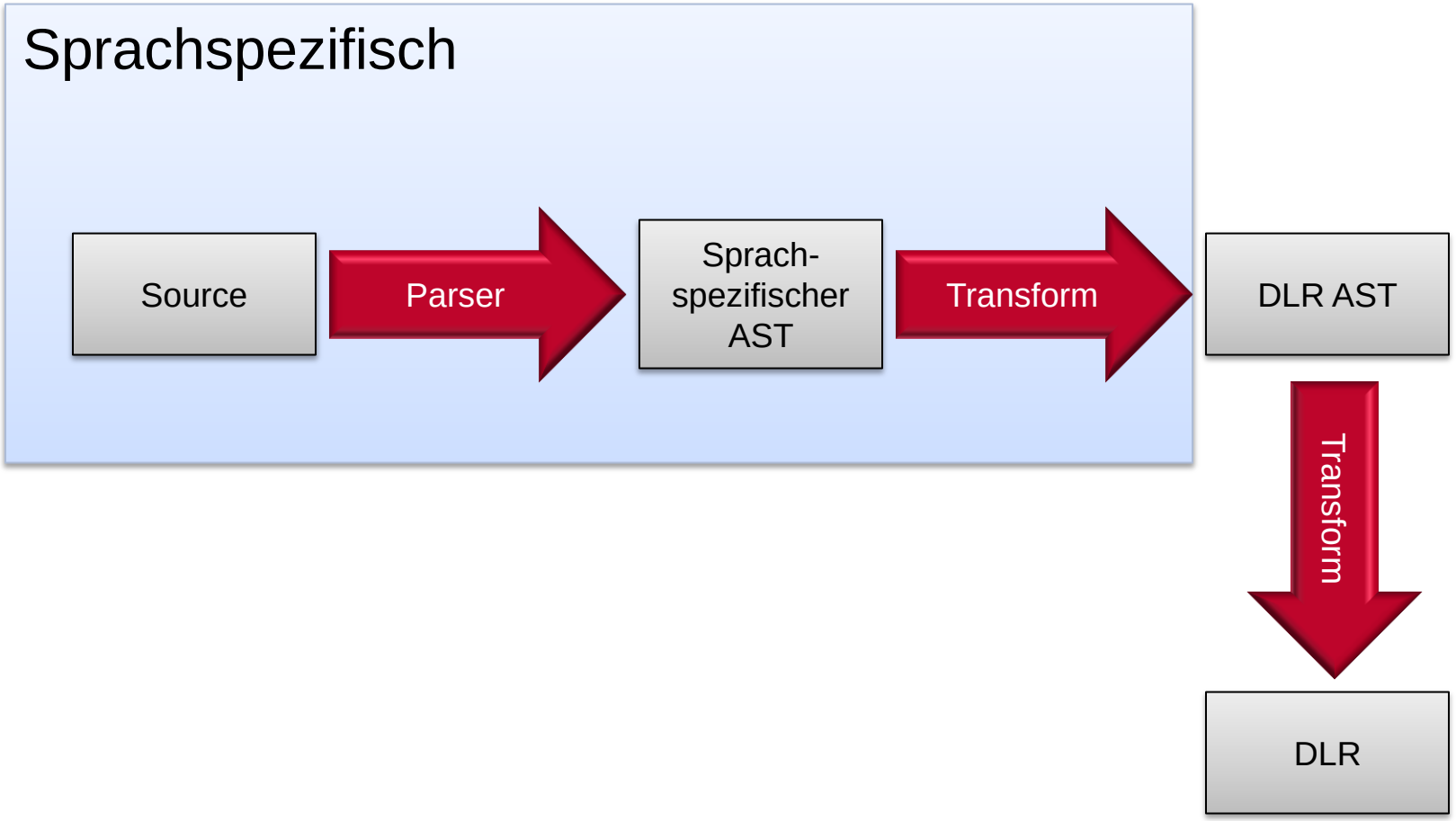


Expression Trees in C#

```
private static void Main(string[] args)
{
    Func<int, bool> f;
    Expression<Func<int, bool>> ex;
    […]

    ex = Expression.Lambda<Func<int, bool>>(
        Expression.Equal(
            CS$0$0000 = Expression.Parameter(typeof(int), "x"),
            Expression.Constant((int) 5, typeof(int))
        ),
        new ParameterExpression[] { CS$0$0000 });

     return;
}
```

Compiler bietet Zugriff auf den Syntax Tree zur Laufzeit

AST in DLR


Sprachspezifisch
Source
Parser
Sprach-
spezifischer
AST
Transform
DLR AST
Transform
DLR

ExpressionTrees in C#

2010

Inheritance Hierarchy

System.Object
System.Linq.Expressions.Expression
- System.Linq.Expressions.BinaryExpression
- System.Linq.Expressions.BlockExpression
- System.Linq.Expressions.ConditionalExpression
- System.Linq.Expressions.ConstantExpression
- System.Linq.Expressions.DebugInfoExpression
- System.Linq.Expressions.DefaultExpression
- System.Linq.Expressions.DynamicExpression
- System.Linq.Expressions.GotoExpression
- System.Linq.Expressions.IndexExpression
- System.Linq.Expressions.InvocationExpression
- System.Linq.Expressions.LabelExpression
- System.Linq.Expressions.LambdaExpression
- System.Linq.Expressions.ListInitExpression
- System.Linq.Expressions.LoopExpression
- System.Linq.Expressions.MemberExpression
- System.Linq.Expressions.MemberInitExpression
- System.Linq.Expressions.MethodCallExpression
- System.Linq.Expressions.NewArrayExpression
- System.Linq.Expressions.NewExpression
- System.Linq.Expressions.ParameterExpression
- System.Linq.Expressions.RuntimeVariablesExpression
- System.Linq.Expressions.SwitchExpression
- System.Linq.Expressions.TryExpression
- System.Linq.Expressions.TypeBinaryExpression
- System.Linq.Expressions.UnaryExpression

2008

Inheritance Hierarchy

System.Object
System.Linq.Expressions.Expression
- System.Linq.Expressions.BinaryExpression
- System.Linq.Expressions.ConditionalExpression
- System.Linq.Expressions.ConstantExpression
- System.Linq.Expressions.InvocationExpression
- System.Linq.Expressions.LambdaExpression
- System.Linq.Expressions.ListInitExpression
- System.Linq.Expressions.MemberExpression
- System.Linq.Expressions.MemberInitExpression
- System.Linq.Expressions.MethodCallExpression
- System.Linq.Expressions.NewArrayExpression
- System.Linq.Expressions.NewExpression
- System.Linq.Expressions.ParameterExpression
- System.Linq.Expressions.TypeBinaryExpression
- System.Linq.Expressions.UnaryExpression

Formelsprache in Action

```
var param = Expression.Parameter(typeof(int), "x");
Expression<Func<int, int>> ex2 =
   Expression.Lambda<Func<int, int>>(
      Expression.MakeBinary(
         ExpressionType.Subtract,
         param,
         Expression.Constant(1)),
      param);
```

Formelsprache in Action

```
private static Func<Project, object> CompileFormula(
    ExpressionLanguageCommonTree ast)
{
    var parameter = Expression.Parameter(typeof(Project), "p");
    var expression = ProjectList.ConvertToExpressionTree(ast, parameter);
    var lambda = Expression.Lambda<Func<Project, object>>(
        Expression.Convert(expression, typeof(object)),
        parameter);

    return lambda.Compile();
}
```

Bibliographie ANTLR und DSLs

- ANTLR
  - http://www.antlr.org/
  - Buch „The Definitive ANTLR Reference“ auf Amazon

Why does the world need MEF?

THE PROBLEM

Original Goals

- Before MEF
  - Multiple extensibility mechanism for different Microsoft tools (e.g. Visual Studio, Trace Listeners, etc.)
  - Developers outside of MS had the same problem
- MEF: Provide standard mechanisms for hooks for 3rd party extensions
- Goal: *Open and Dynamic Applications*
  - make it easier and cheaper to build extensible applications and extensions

MEF „Hello World“

```
[Export(typeof(Shape))]
public class Square : Shape
{
  // Implementation
}

[Export(typeof(Shape))]
public class Circle : Shape
{
  // Implementation
}

[Export]
public class Toolbox
{
  [ImportMany]
  public Shape[] Shapes { get; set; }
  // Additional implementation...
}

[…]

var catalog = new AssemblyCatalog(typeof(Square).Assembly);
var container = new CompositionContainer(catalog);
Toolbox toolbox = container.GetExportedValue<Toolbox>();
```

Export with name or type

Defaults to `typeof(Toobox)`

„Attributed Programming Model“

MEF „Hello World“ (continued)

- *Parts*
  - `Square`, `Circle` and `Toolbox`
- *Dependencies*
  - Imports (`Import`-Attribute)
  - E.g. `Toolbox.Shapes`
- *Capabilities*
  - Exports (`Export`-Attribute)
  - E.g. `Square`, `Circle`

Exports And Imports

- `Export` attribute
  - Class
  - Field
  - Property
  - Method
- `Import` attribute
  - Field
  - Property
  - Constructor parameter
- Export and import must have the same contract
  - Contract name and contract type
  - Contract name and type can be inferred from the decorated element

MEF Catalogs

- Catalogs provide components
- Derived from `System.ComponentModel.Composition.Primitives.ComposablePartCatalog`
  - `AssemblyCatalog`
    - Parse all the parts present in a specified assembly
  - `DirectoryCatalog`
    - Parses the contents of a directory
  - `TypeCatalog`
    - Accepts type array or a list of managed types
  - `AggregateCatalog`
    - Collection of `ComposablePartCatalog` objects

MEF In Silverlight

- Additional catalog `DeploymentCatalog`
  - Load exported parts contained in XAP files
  - Provides methods for asynchronously downloading XAP files containing exported parts (`DeploymentCatalog.DownloadAsync`)
- Goal
  - Minimize initial load times
  - Application can be extended at run-time

Bedarf nach mehr?
→ Versteckte Slides

- Details zu MEF
  - Inherited Exports
  - Lazy Imports
  - Optional Imports
  - Creation Policies
  - Metdata

Inherited Exports

```
[Export]
public class NumOne
{
    [Import]
    public IMyData MyData
       { get; set; }
}

public class NumTwo : NumOne
{
}

[InheritedExport]
public class NumThree
{
    [Export]
    Public IMyData MyData { get; set; }
}

public class NumFour : NumThree
{
}
```

Import automatically inherited

Export NOT inherited
→ NumTwo has no exports

Member-level exports are never inherited

Inherits export with contract `NumThree` (including all metadata)

Lazy Imports

- Imported object is not instantiated immediately
  - Imported (only) when accessed
- Sample:

```
public class MyClass
{
    [Import]
    public Lazy<IMyAddin> MyAddin `
      { get; set; }
}
```

Prerequisite Imports

- Composition engine uses parameter-less constructor by default
- Use a different constructor with `ImportingConstructor` attribute
- Sample:

```
[ImportingConstructor]
public MyClass(
   [Import(typeof(IMySubAddin))]IMyAddin
     MyAddin)
{
   _theAddin = MyAddin;
}
```

Could be removed here; automatically imported

Optional Imports

- By default composition fails if an import could not be fulfilled
- Use `AllowDefault` property to specify optional imports
- Sample:

```
public class MyClass
{
    [Import(AllowDefault = true)]
    public Plugin thePlugin { get; set; }
}
```

Creation Policy

- `RequiredCreationPolicy` property
- `CreationPolicy.Any`
  - `Shared` if importer does not explicitly request `NonShared`
- `CreationPolicy.Shared`
  - Single shared instance of the part will be created for all requestors
- `CreationPolicy.NonShared`
  - New non-shared instance of the part will be created for every requestor

Advanced exports

METADATA AND METADATA VIEWS

Goal

- Export provides additional metadata so that importing part can decide which one to use
- Import can inspect metadata without creating exporting part
- Prerequisite: Lazy import

Metadata

```
namespace MetadataSample
{
  public interface ITranslatorMetadata
  {
    string SourceLanguage { get; }

    [DefaultValue("en-US")]
    string TargetLanguage { get; }
  }
}
```

```
namespace MetadataSample
{
  [Export(typeof(ITranslator))]
  [ExportMetadata("SourceLanguage", "de-DE")]
  [ExportMetadata("TargetLanguage", "en-US")]
  public class GermanEnglishTranslator : ITranslator
  {
    public string Translate(string source)
    {
      throw new NotImplementedException();
    }
  }
}
```

Export Metadata can be mapped to metadata view interface

Metadata (continued)

```
namespace MetadataSample
{
    class Program
    {
        static void Main(string[] args)
        {
            var catalog = new AssemblyCatalog(
                typeof(ITranslator).Assembly);
            var container = new CompositionContainer(catalog);

            // We need a translator from hungarian to english
            Lazy<ITranslator, ITranslatorMetadata> translator =
                container
                .GetExports<ITranslator, ITranslatorMetadata>()
                .Where(t => t.Metadata.SourceLanguage == "hu-HU"
                    && t.Metadata.TargetLanguage == "en-US")
                .FirstOrDefault();
        }
    }
```

Custom Export Attributes

- `[TranslatorExport("de-DE", "en-US")]`

```
public class GermanEnglishTranslator
  : Itranslator
{
  public string Translate(
    string source)
  {
    throw new NotImplementedException();
  }
}
```

Custom export attributes makes code much cleaner.

- `[Export(typeof(ITranslator))]`
  `[ExportMetadata("SourceLanguage", "de-DE")]`
  `[ExportMetadata("TargetLanguage", "en-US")]`

```
public class GermanEnglishTranslator
  : Itranslator
{
  public string Translate(
    string source)
  {
    throw new NotImplementedException();
  }
}
```

Custom Export Attributes (continued)

```
•[MetadataAttribute]
[AttributeUsage(AttributeTargets.Class, AllowMultiple = false)]
public class TranslatorExportAttribute
        : ExportAttribute, ITranslatorMetadata
{
        public TranslatorExportAttribute(
                string sourceLanguage, string targetLanguage)
                : base(typeof(ITranslator))
                {
                        this.SourceLanguage = sourceLanguage;
                        this.TargetLanguage = targetLanguage;
                }
                public string SourceLanguage { get; private set; }
                public string TargetLanguage { get; private set; }
        }
}
```

Bibliographie
Managed Extensibility Framework

- Hands-On Lab MEF
  - Blogartikel Rainer Stropek
- Managed Extensibility Framework on MSDN
- Managed Extensibility Framework for .NET 3.5 on Codeplex
- Visual Studio 2010 and .NET Framework 4 Training Kit

Dynamic Language Runtime

DLR AND SCRIPTING

Typische Einsatzgebiete von DLR und Scripting

aus Sicht von C# Entwicklern ;-)

- Typische Anwendungen
  - Automatisieren von Routinetätigkeiten (Makros)
  - Schnittstellen
  - Plug-In ohne Visual Studio
  - Installation, Wartung, Updates
  - Prototyping
- Nutzen
  - Anpassungsmöglichkeiten vorort beim Kunden eventuell durch den Kunden
  - Kein VS, kein Kompilieren notwendig
  - Dynamisches Programmieren manchmal effektiver (z.B. bei Prototyping)
  - Python und JavaScript sind coole Sprachen

Voraussetzungen für IronPython

- Herunterladen von IronPython für .NET
- Referenzen auf
  - `IronPython.dll`
  - `Microsoft.Scripting.dll`

Hosting API Grundlagen

- `Microsoft.Scripting.Hosting`
- `ScriptRuntime`
  - Möglichkeit, verschiedene Laufzeitumgebungen voneinander zu trennen (z.B. für Security)
  - Python runtime mit `IronPython.Hosting.Python.CreateRuntime()`
- `ScriptEngine`
  - Enthält alle wichtigen Funktionen zum Ausführen von Python Code, zum Zugriff auf Variablen, etc.
  - Python Engine mit `IronPython.Hosting.Python.CreateEngine()`

Pythondatei ausführen

```
var engine = Python.CreateEngine();
using (var stream = new ScriptOutputStream( s => {
        this.AppendToScriptOutput(s);
        App.Current.Dispatcher.BeginInvoke(
            new Action(() => this.OnPropertyChanged("ScriptOutput")));
    }, Encoding.UTF8))
{
    engine.Runtime.IO.SetOutput(stream, Encoding.UTF8);
    var scriptSource = engine.CreateScriptSourceFromFile("SampleScript01.py");
    try
    {
        scriptSource.Execute();
    }
    catch (SyntaxErrorException e)
    {
        this.AppendToScriptOutput("Syntax error (line {0}, column {1}): {2}",
            e.Line, e.Column, e.Message);
        App.Current.Dispatcher.BeginInvoke(
            new Action(() => this.OnPropertyChanged("ScriptOutput")));
    }
}
```

Wegen asynchroner Ausführung

Exkurs: ScriptOutputStream

```
public sealed class ScriptOutputStream : Stream
{
    public ScriptOutputStream(Action<string> write, Encoding encoding)
    {
        […]
        chunks = new BlockingCollection<byte[]>();
        this.processingTask = Task.Factory.StartNew(() => {
                foreach (var chunk in chunks.GetConsumingEnumerable()) {
                    write(this.encoding.GetString(chunk));
                }
            }, TaskCreationOptions.LongRunning);
    }
    public override void Write(byte[] buffer, int offset, int count)
    {
        var chunk = new byte[count];
        Buffer.BlockCopy(buffer, offset, chunk, 0, count);
        this.chunks.Add(chunk);
    }
    public override void Close()
    {
        this.chunks.CompleteAdding();
        try { this.processingTask.Wait(); }
        finally { base.Close(); }
    }
    […]
}
```

Consumer

Producer

Beispielscript in Python

```
import clr
clr.AddReference("mscorlib")

from System.Threading import Thread

for i in range(0, 10):
  print str(i+1)
  Thread.Sleep(500)

print "Done!"
```

Referenzen auf Assemblies

~using

Methode aus dem .NET Framework

Beispielscript in Python

```
[…]

class ViewModel:
  numberOfSpeakers = 0
  def __init__(self, speakers):
    self.numberOfSpeakers = speakers

def getNumberOfSpeakers():
  vm = ViewModel(Application.Current.MainWindow.DataContext.Speakers.Length)
  stream = Application.Current.GetType().Assembly.GetManifestResourceStream(
    "IronPython.UI.Scripts.ResultWindow.xaml")
  reader = StreamReader(stream)
  window = XamlReader.Parse(reader.ReadToEnd())
  reader.Close()
  stream.Close()
  window.DataContext = vm
  window.FindName("CloseButton").Click += lambda s, e: window.Close()
  window.Show()

Application.Current.Dispatcher.BeginInvoke(Action(lambda: getNumberOfSpeakers()))
print "Done!"
```

ViewModel geschrieben in Python (Python implementiert ICustomTypeDescriptor)

Zugriff auf Elemente der C# Anwendung

Dynamisches Laden des XAML-Codes

Auf WPF-Event in Python reagieren

BeginInvoke wegen Hintergrundthread

Advanced: LINQ in Python

```
this.Speakers.AsParallel().ForAll(
    eo => eo.AddCalculatedProperty("NumberOfApprovedSessions", @"
import clr
clr.AddReference(""System.Core"")
from System.Linq import Enumerable
lambda s: Enumerable.Count(s.Sessions, lambda p: p.Approved)"));
```

Linq in Python Lambda

```
this.Speakers.AsParallel().ForAll(
    eo => eo.AddCalculatedProperty("NumberOfApprovedSessions",
    "lambda s: len([session for session in s.Sessions if
        session.Approved])"));
```

Das gleiche mit Python list comprehension

Python Beispielcode

```
this.ApproveSessionCommand = new GenericCommand(
    x => this.SelectedSession != null,
    x =>
    {
        var engine = Python.CreateEngine();
        var scope = engine.CreateScope();
        scope.SetVariable("viewModel", this);
        engine.CreateScriptSourceFromString(@"
viewModel.SelectedSession.Approved = True
viewModel.SaveChanges()
").Execute(scope);
    }, this);
```

Variablen an Script übergeben

Excel-Export

```
import clr
clr.AddReferenceByName(
  'Microsoft.Office.Interop.Excel, Version=11.0.0.0, Culture=neutral,
PublicKeyToken=71e9bce111e9429c')
clr.AddReference(""System.Core"")
from Microsoft.Office.Interop import *
from Microsoft.Office.Interop.Excel import *
from System import *

def export(speakers):
  ex = Excel.ApplicationClass()
  ex.Visible = True
  ex.DisplayAlerts = False
  workbook = ex.Workbooks.Add()
  ws = workbook.Worksheets[1]
  rowIndex = 1
  for speaker in speakers:
    ws.Cells[rowIndex, 1].Value2 = speaker.FirstName
    ws.Cells[rowIndex, 2].Value2 = speaker.LastName
    ws.Cells[rowIndex, 3].Value2 = speaker.FullName
    ws.Cells[rowIndex, 4].Value2 = speaker.NumberOfApprovedSessions
    rowIndex = rowIndex + 1
```

Import des Excel Interop Assemblies

Export einer Funktion

Excel-Automatisierung

Excel-Export

```
var engine = Python.CreateEngine();
var scope = engine.CreateScope();
var scriptSource = @"[…]";
engine.CreateScriptSourceFromString(scriptSource).Execute(scope);
dynamic exportFunc = scope.GetVariable("export");
engine.Operations.Call(exportFunc, this.Speakers);
```

Funktionsdefinition abfragen

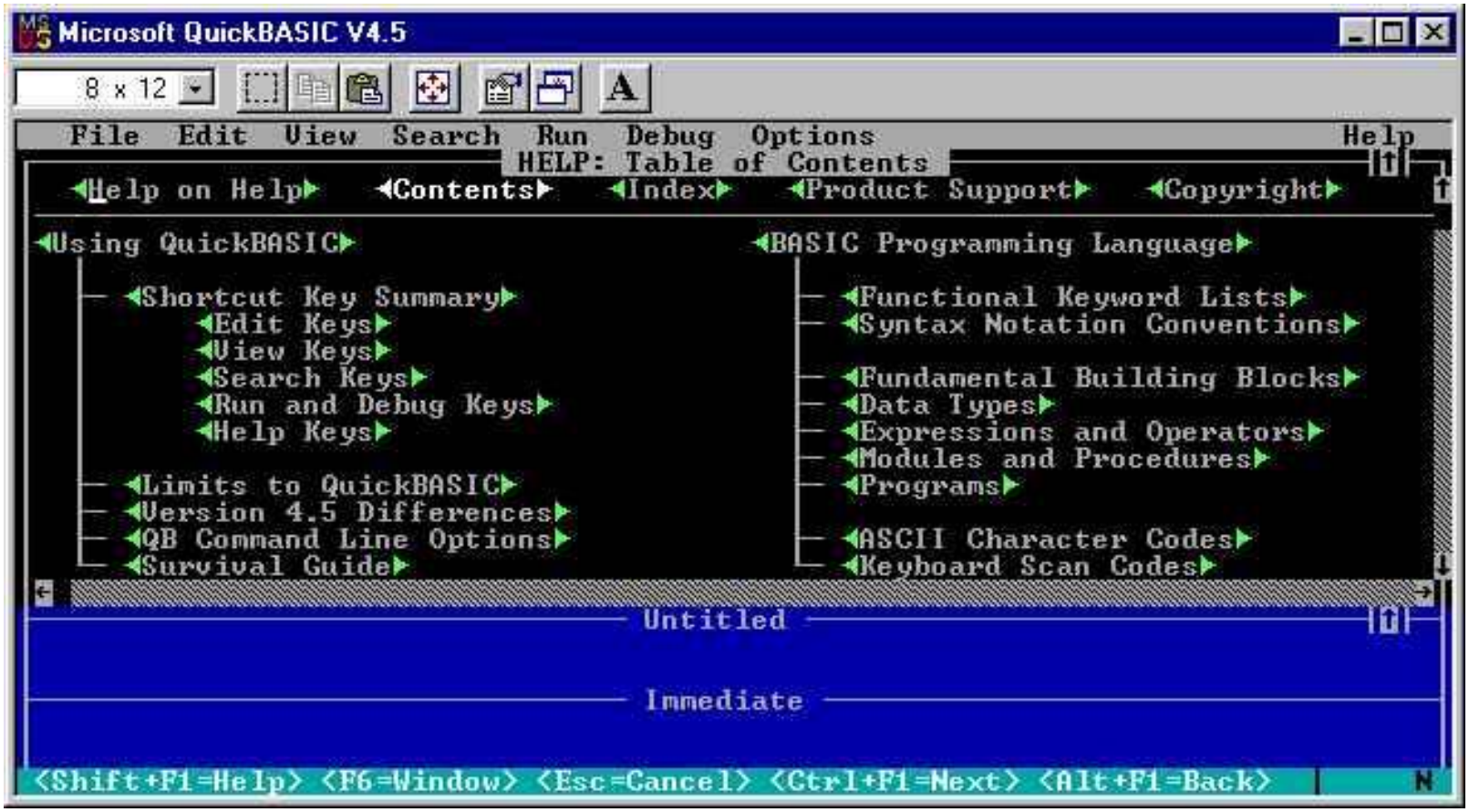
Aufruf der Funktion mit `ObjectOperations`

Bibliographie IronPython und Scripting

- IronPython Dokumentation
  - http://www.ironpython.net
  - http://docs.python.org
  - Sourcecode (DLR und IronPython sind auf codeplex)
- Lust, IronPython in einer echten Anwendung auszuprobieren?
  - http://www.timecockpit.com

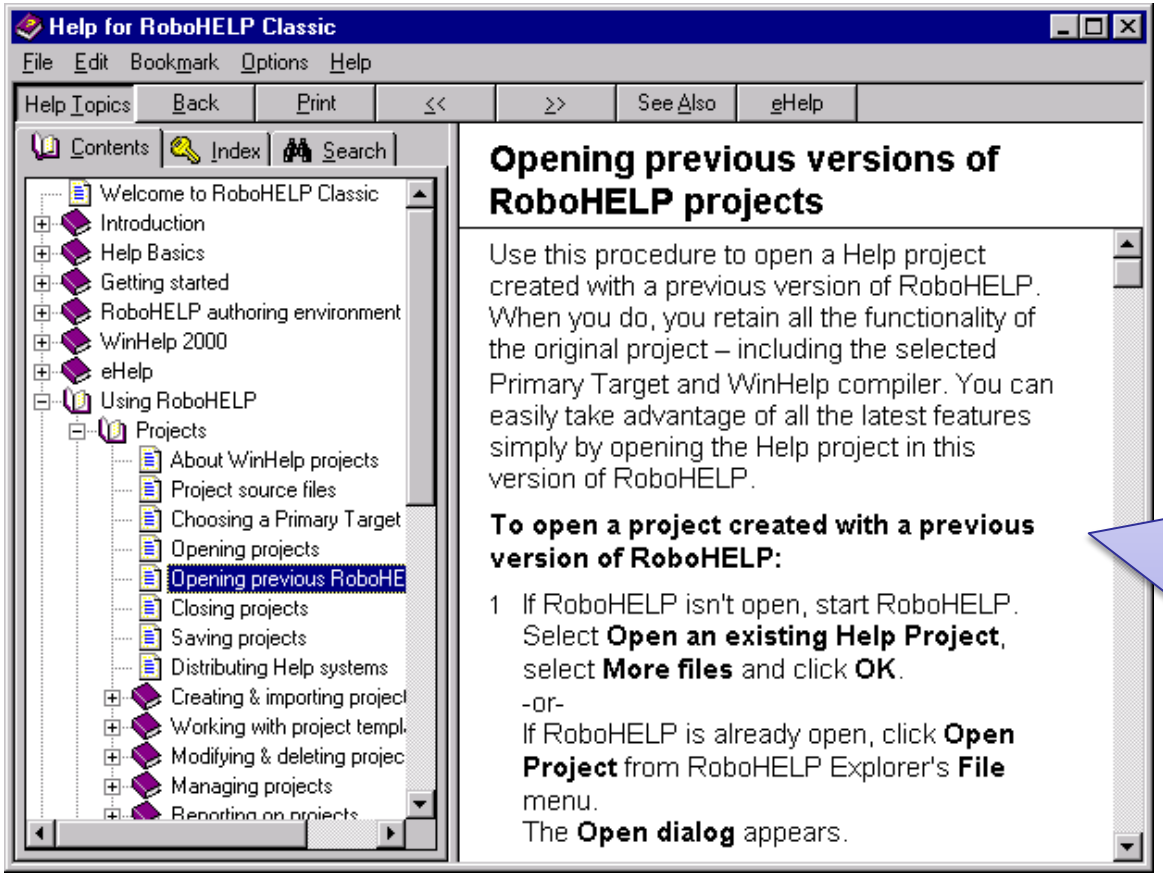
The Windows Help System

THE HISTORY



Where everything started: *QuickHelp*

End of 1980's



*"Microsoft strongly recommends that software developers **discontinue** using the Windows Help application."*

Source: MSDN

Next Evolution: *WinHelp*

Combines RTF + Images + VBA Macros

No out-of-the-box support starting with Windows Vista

HTML Reaches The World Of Help: *HTMLHelp*

HTML + Images + JavaScript compiled into a single file

Detailed history of HTMLHelp see Wikipedia article on [Microsoft Compiled HTML Help]

HTMLHelp

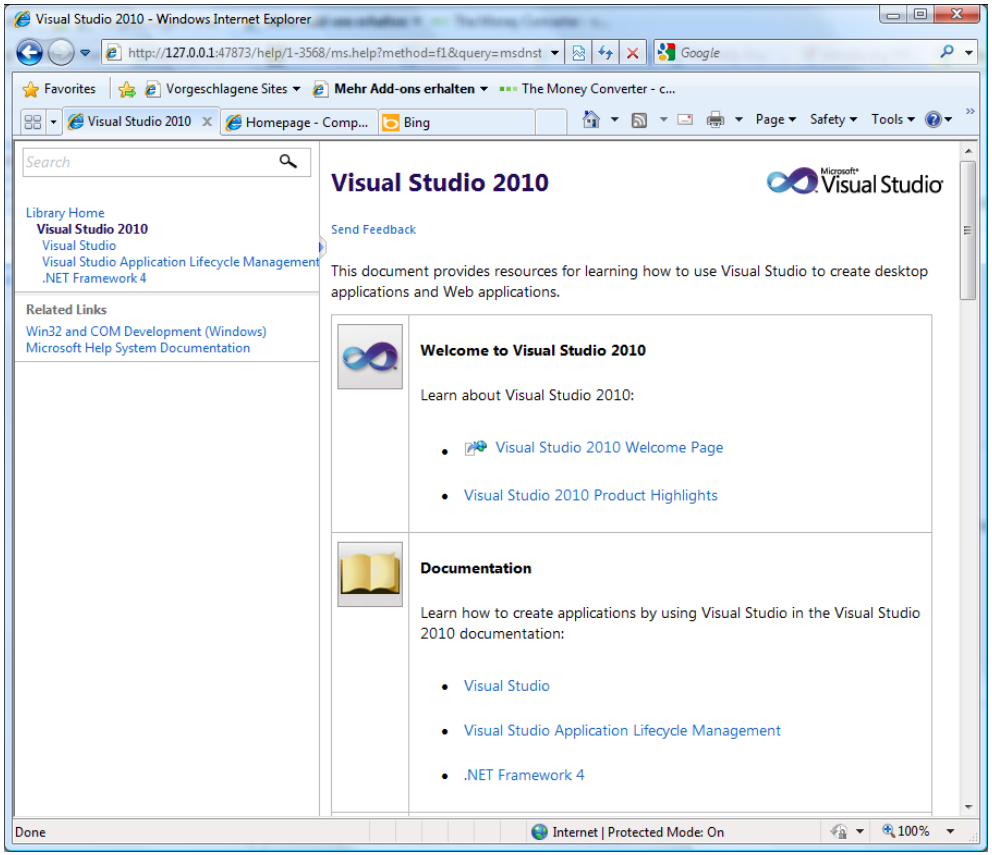
- Embedded Internet Explorer is used to display help content
  - Problem: Untrusted network locations
- Current Version is 1.4
  - Included in Windows 7

Assistance Platform 1.0 Client SDK

- Formerly *Windows Vista Help SDK*
- “Can only be used to display the Windows Help content set.”
- “[…] cannot be used by third-party programs”
- Source: MSDN

HTMLHelp 2

- Designed for the VS .NET environment only
- Not released as a general Help platform (Source: MS press announcements)
- Need more details? MS Help 2 FAQ



MS Help Viewer 1.0

Making everything better…

MS Help Viewer 1.0

- Called *Help3* during development
- Shipped with Visual Studio 2010
  - Replaced MS Help 2.0
  - Details see series of blog posts by Jeff Braaten (Microsoft)
  - See also video
- Help for VS 2010 SP1
  - Offline Viewer
  - TOC Tree
  - Support for tabs
  - Traditional index tab
  - History and favorites

So What???

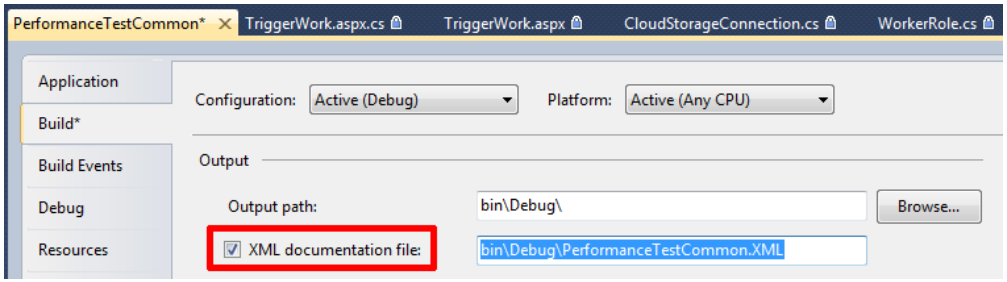
- General Windows application
  - Use HTML Help 1.x (.CHM files)
  - HTML Help ships as part of the Windows OS.
- OEM with the need to produce documentation to integrate with Windows Help
  - Assistance Platform 1.0 (.H1S files).
- Extending Visual Studio
  - Visual Studio 2010
    - Help Viewer 1.0 (.MSHC files)
    - Help Viewer 1.0 ships with Visual Studio 2010.
  - Earlier versions (2002 – 2008)
    - Microsoft Help 2.x (.HxS files)
    - Microsoft Help 2.x ships with Visual Studio 2002 – 2008.
- Assistance Platform 1.0, Microsoft Help 2.x and Help Viewer 1.0 are not available for general 3rd party use or redistribution
- Source: Paul O'Rear (Program manager MS help team)

Introduction

SANDCASTLE

The Problem

- .NET introduced XML-based code comments
  - See MSDN for details
  - We will go into more details later
- C# compiler can extract XML-based code documentation
  - `csc /doc:<targetFile>.xml`
  - See MSDN for details

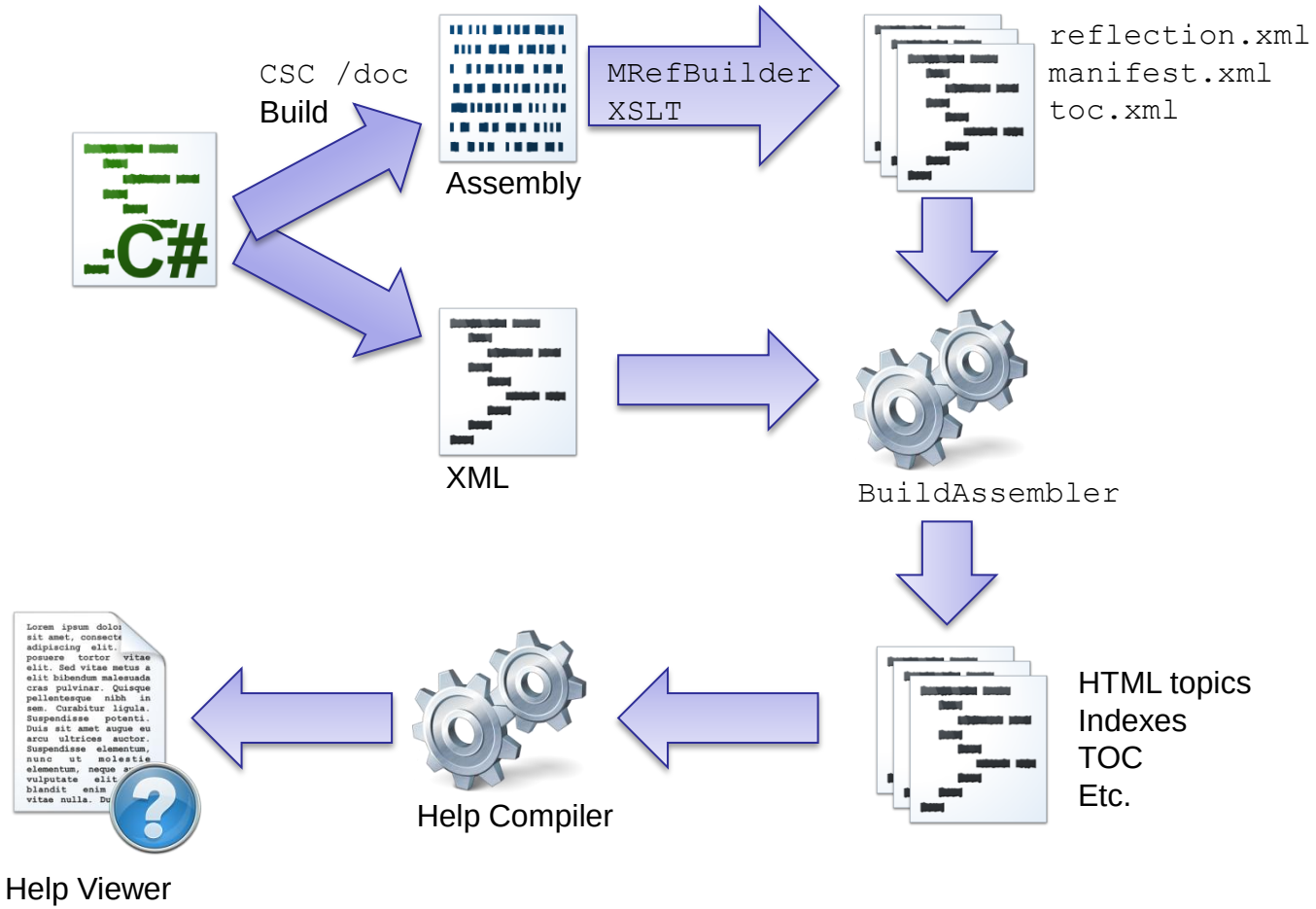


- What to do with this XML documentation??

The Idea

- Take XML documentation,
- extract type and member information from assemblies into XML-files,
- apply a set of XSL templates
- and finally build help files from that.

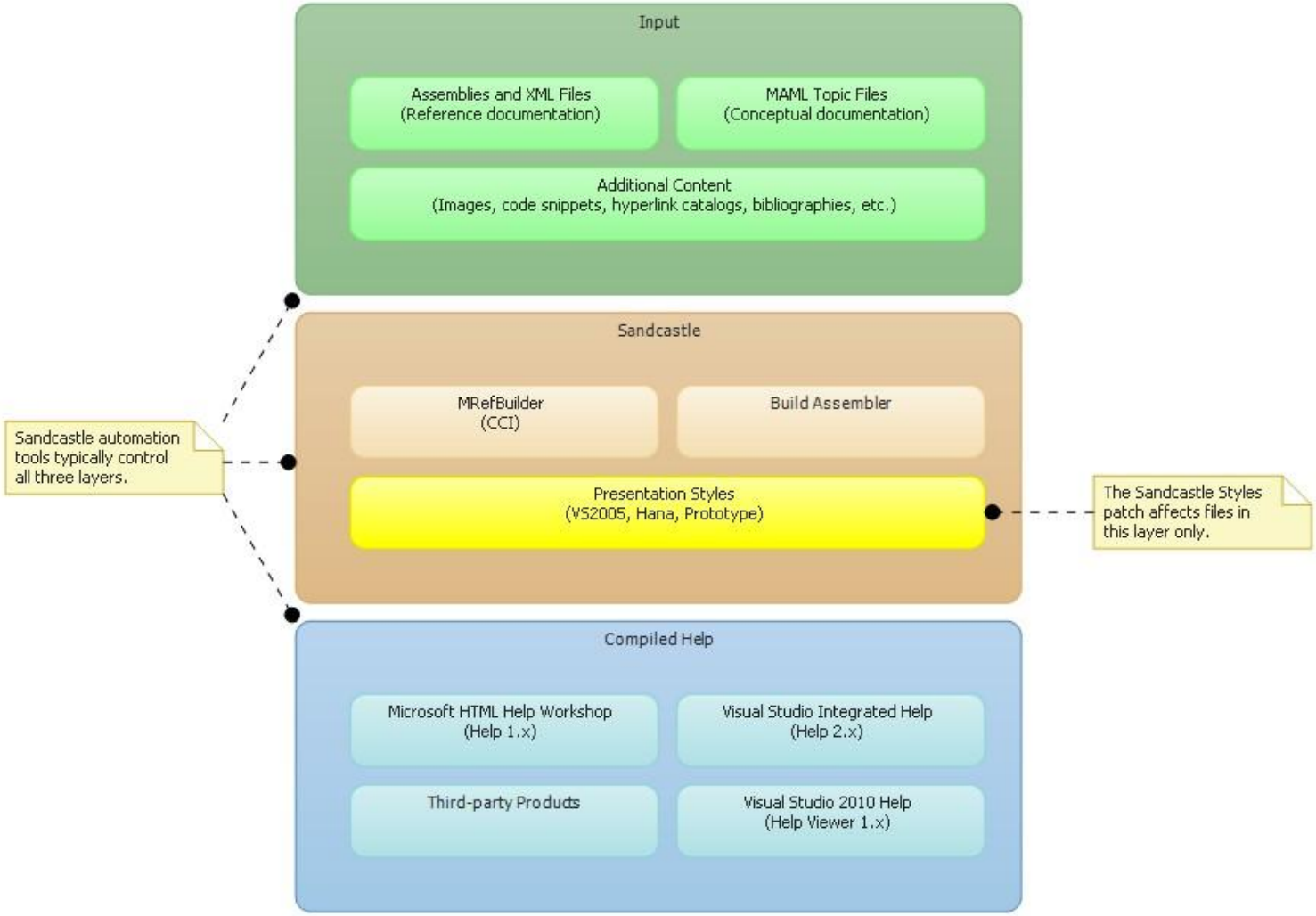
Sandcastle Process (CHM File)



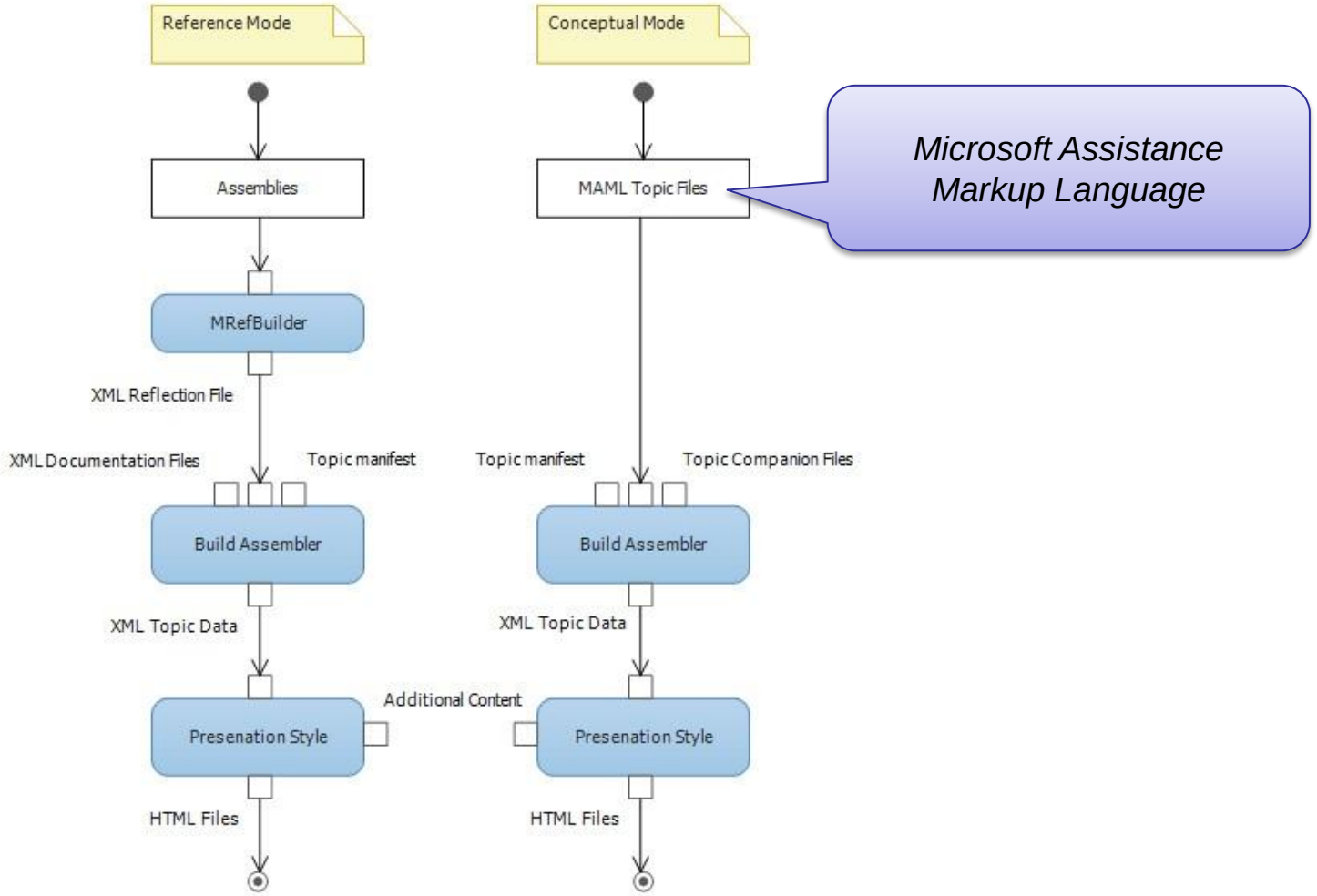
MRefBuilder

- Analyze the physical properties of the assemblies
- Generates XML reflection file
  - Contains all of the information required to document the physical properties of the input assemblies
- XML reflection file is mapped to XML documentation file from C# compiler

Role of Sandcastle Styles



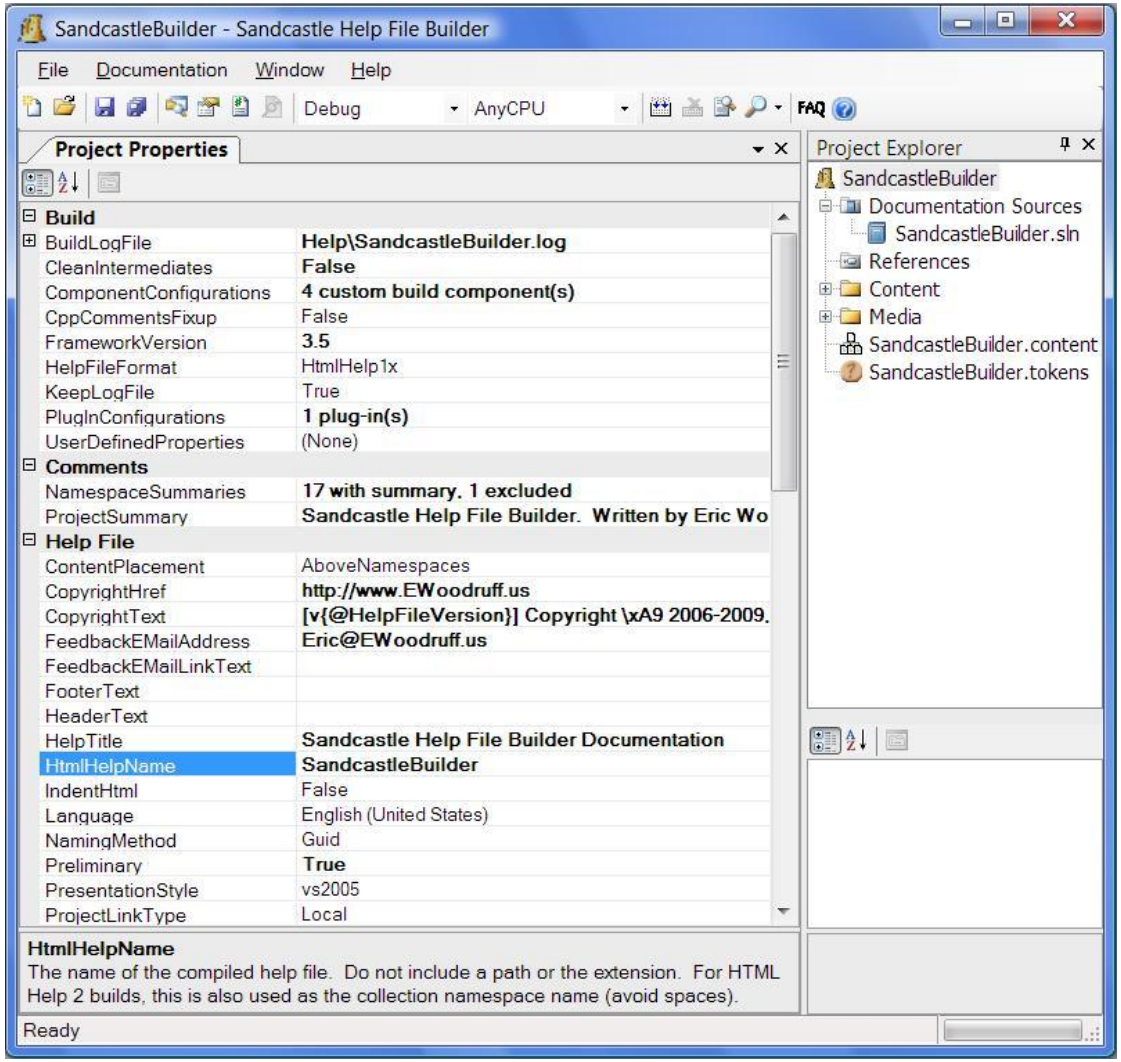
Conceptual Mode



Getting rid of config files and the command line

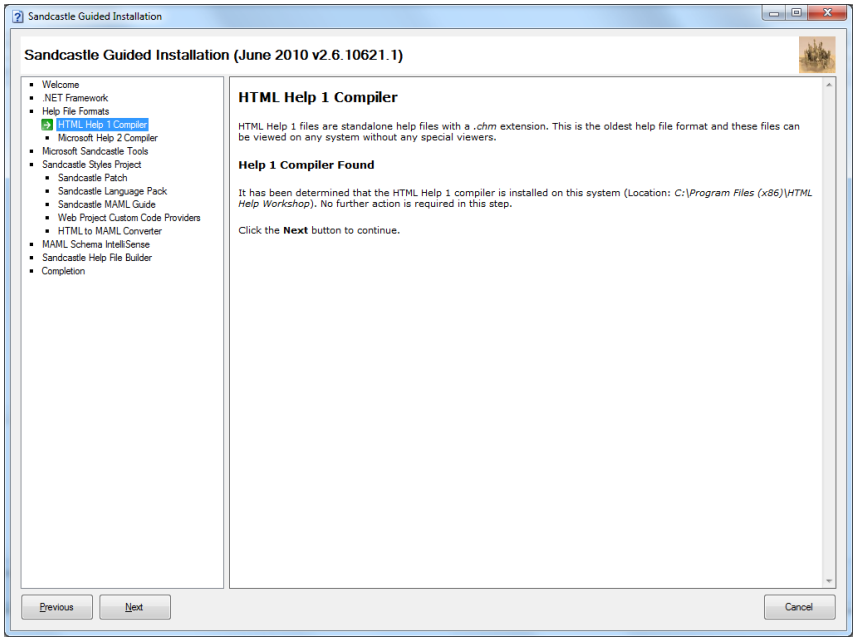
SANDCASTLE HELPFILE BUILDER

Sandcastle Helpfile Builder



Installation

- Guided Installer
- Takes you through all the necessary steps to run SHFB
  - Help Compilers
  - Sandcastle
  - SHFB
- Contains interesting and meaningful descriptions!
  - Read it!!



SHFB Tips & Tricks

- You can add multiple documentation sources at a time
  - Use e.g. solution or wildcards
  - See SHFB Online Help for details
- Central bibliography files
  - Generate a central bibliography XML file (see SHFB Online Help)
  - Use `<cite>` element to reference bibliography
- Use token files to add a central repository of tokens
  - Enhances maintainability of your documentation

Bedarf nach mehr?
→ Versteckte Slides

- C# Code Dokumentation
  - General Tips
  - Documentation Tags
  - C# Code Documentation – Advanced Topics

Documenting C# using XML

C# CODE DOCUMENTATION

General Tips

- All XML/HTML within an XML Documentation Comment must be well formed
- Use `<` and `>` for text in angle brackets
- You can use HTML markup to add additional formatting to your XML comments if needed
  - Avoid if possible

XML Documentation Comments Matrix

Tag	Defined By			Tag Type		Applies To											
	Microsoft	NDoc	Sandcastle	Top-Level	Inline	Namespace	Class	Structure	Interface	Enum	Delegate	Field	Constant	Property	Method	Event	Operator
<c>	X				X		X	X	X	X	X	X	X	X	X	X	X
<code>	X				X									X	X	X	X
<event>	X			X											X		X
<example>	X			X			X	X	X	X	X	X	X	X	X	X	X
<exception>	X			X										X	X	X	X
<exclude>		X	X	X			X	X	X	X	X	X	X	X	X	X	X
<include>	X			X			X	X	X	X	X	X	X	X	X	X	X
<list>	X				X		X	X	X	X	X	X	X	X	X	X	X
<note>		X	X		X		X	X	X	X	X	X	X	X	X	X	X
<overloads>		X	X	X										X	X	X	X
<para>	X				X												
<param>	X			X							X			X	X		
<paramref>	X				X		X	X	X	X	X	X	X	X	X	X	X
<permission>	X			X								X	X	X	X	X	X
<preliminary>		X	X	X			X	X	X	X	X	X	X	X	X	X	X
<remarks>	X			X			X	X	X	X	X	X	X	X	X	X	X
<returns>	X			X											X		
<see>	X			X	X		X	X	X	X	X	X	X	X	X	X	X
<seealso>	X			X	X		X	X	X	X	X	X	X	X	X	X	X
<summary>	X			X		X[1]	X	X	X	X	X	X	X	X	X	X	X
<threadsafety>		X	X	X			X	X									
<typeparam>	X			X											X		
<typeparmref>	X				X										X		
<value>	X			X										X			

1. NDoc and Sandcastle Help File Builder support adding namespace comments using the GUI

Source: XML Documentation Comments Guide

<summary>

- Describe a type or a type member
- Important because used in
  - IntelliSense and
  - Object Browser
- Important StyleCop rules
  - SA1642: Constructor
    - Non-private, non-static:
      Initializes a new instance of the {class name} class.
    - Static:
      Initializes static members of the {class name} class.
    - Private:
      Prevents a default instance of the {class name} class from being created.
  - SA1623: Property
    - Has to start with *Gets or sets*, *Gets* or *Sets* according to read/write status
    - For boolean properties has to start with *Gets or sets a value indicating whether*

<summary> (continued)

- Recommendation for partial classes (SA1619)
  - Provide the official SDK documentation for the class on the main part of the partial class.
    - Use the standard `<summary>` tag.
  - All other parts of the partial class should omit the `<summary>` tag completely, and replace it with a `<content>` tag.

<remarks>

- Adds information about a type, supplementing the information specified with <summary>
- Important because displayed in Object Browser
- Sample:

```
/// <summary>
/// You may have some primary information about this class.
/// </summary>
/// <remarks>
/// You may have some additional information about this class.
/// </remarks>
public class TestClass
{ […] }
```

<value>

- Describes the value that a property represents, supplementing the information specified with <summary>
- Sample:

```
/// <summary>Gets or sets a value indicating whether this instance
     is dirty.</summary>
/// <value><c>true</c> if this instance is dirty>; otherwise,
     <c>false</c></value>
public bool Dirty
{ get {…} set {…} }
```

Important Inline Elements

- <c>
  - Indicate that text within a description should be marked as code
  - Sample:
    ```
    /// <summary><c>DoWork</c> is a method in the <c>TestClass</c>
    class.</summary>
    ```
- <see>
  - Reference to a member or field
  - More about code references later
  - Sample:
    ```
    This sample shows how to call the <see cref="GetZero"/> method.
    ```
- <para>
  - Used to define a paragraph
  - Sample:
    ```
    <para>Paragraph 1</para>
    <para>Paragraph 2</para>
    ```

Important Inline Elements (continued)

- <list>
  - Defines a bullet list, numbered list or a table
  - Sample (bullet list):

```
/// <summary>Here is an example of a bulleted list:
/// <list type="bullet">
///   <item><description>Item 1.</description></item>
///   <item><description>Item 2.</description></item>
/// </list>
/// </summary>
```

  - Sample (table):

```
/// <summary>Here is an example of a table:
/// <list type="table">
///   <listheader>
///     <term>Col1</term>
///     <description>Column 1</description>
///   </listheader>
///   <item>
///     <term>Col1</term>
///     <description>Column 1</description>
///   </item>
/// </list>
/// </summary>
```

Important Inline Elements (continued)

- <paramref> and <typeparamref>
  - Gives you a way to indicate that a word in the code comments refers to a (type) parameter
  - Sample `paramref`:

```
/// <summary>DoWork is a method in the TestClass class.
/// The <paramref name="Int1"/> parameter takes a number.
/// </summary>
public static void DoWork(int Int1)
{ […] }
```

  - Sample `typeparamref`:

```
/// <summary>
/// Creates a new array of arbitrary type <typeparamref name="T"/>
/// </summary>
/// <typeparam name="T">The element type of the array</typeparam>
public static T[] mkArray<T>(int n)
{ […] }
```

<example>

- Specifies an example of how to use a method or other library member
- Use <code> to specify sample code
- Sample:

```
/// <summary>
/// The GetZero method.
/// </summary>
/// <example>
/// Sample of how to call the <see cref="GetZero"/> method.
/// <code>
/// class TestClass
/// {
///     static int Main()
///     {
///         return GetZero();
///     }
/// }
/// </code>
/// </example>
```

<exception>

- Specifies which exceptions can be thrown
- Sample:

```
/// <exception cref="System.Exception">Thrown when...</exception>
```

<param>

- Describes one of the parameters for the method
- Sample:

```
/// <param name="Int1">Used to indicate status.</param>
/// <param name="Float1">Used to specify context.</param>
public static void DoWork(int Int1, float Float1)
{ […] }
```

<typeparam>

- Used in the comment for a generic type or method declaration to describe a type parameter
- Sample:

```
/// <summary>
/// Creates a new array of arbitrary type <typeparamref name="T"/>
/// </summary>
/// <typeparam name="T">The element type of the array</typeparam>
public static T[] mkArray<T>(int n)
{ […] }
```

<seealso>

- Specifies the text that you might want to appear in a *See Also* section
- More about code references later
- Sample:

```
/// <summary>DoWork is a method in the TestClass class.
/// <para>Here's how you could make a second paragraph in a description.
<see cref="System.Console.WriteLine(System.String)"/> for information about
output statements.</para>
/// <seealso cref="TestClass.Main"/>
/// </summary>
```

<include>

- Refers to comments in another file that describe the types and members in your source code
- Uses the XML XPath syntax
- Problematic with StyleCop
- Sample:

```
/// <include file='xml_include_tag.doc'
       path='MyDocs/MyMembers[@name="test"]/*' />
class Test { […] }

/// <include file='xml_include_tag.doc'
       path='MyDocs/MyMembers[@name="test2"]/*' />
class Test2 { […] }
```

Code References (`cref`)

- Can reference a type, method, or property
- Use {} to reference generics
- Sample:

```
<see cref="GetGenericValue"/>
<see cref="GenericClass{T}"/>
<see cref="List{T}.RemoveAll(Predicate{T})"/>
```

Documenting C# using XML (the SHFB way)

C# CODE DOCUMENTATION ADVANCED TOPICS

External Code Samples

- Sandcastle supports enhanced code block
- Syntax:

```
<code [source="SourceFile"
  region="Region Name"
  lang="langId"
  numberLines="true|false"
  outlining="true|false"
  tabSize="###"
  title="Code Block Title"]>content</code>
```

- For details see Code Block Component in SHFB online help

API Filter

- Exclude certain parts of the API from the generated documentation
- Not defined by Microsoft, extension in *NDoc* and *SHFB*
- Sample:

```
/// <summary>
/// Gets the size of the file.
/// </summary>
/// <value>Size of the file.</value>
/// <exclude />
public int Size { get; private set; }
```

<overloads>

- Provides documentation that applies to all the overloads of a member.
- Not defined by Microsoft, extension in *NDoc* and *SHFB*
- Sample:

```
/// <summary>Decrements the number by 1.</summary>
/// <overloads>This method has two overloads.</overloads>
public void Dec()
{}

/// <summary>Decrements the number by amount.</summary>
/// <param name="amount">The amount to decrement it by.</param>
public void Dec(int amount)
{}
```

<see>

- NDoc and Sandcastle extend <see> element
- Improve readability with keyword expansion
- <see langword=„…“/>
  - null
    → “null reference (Nothing in Visual Basic)”
  - sealed
  - static
  - abstract
  - virtual

<threadsafety>

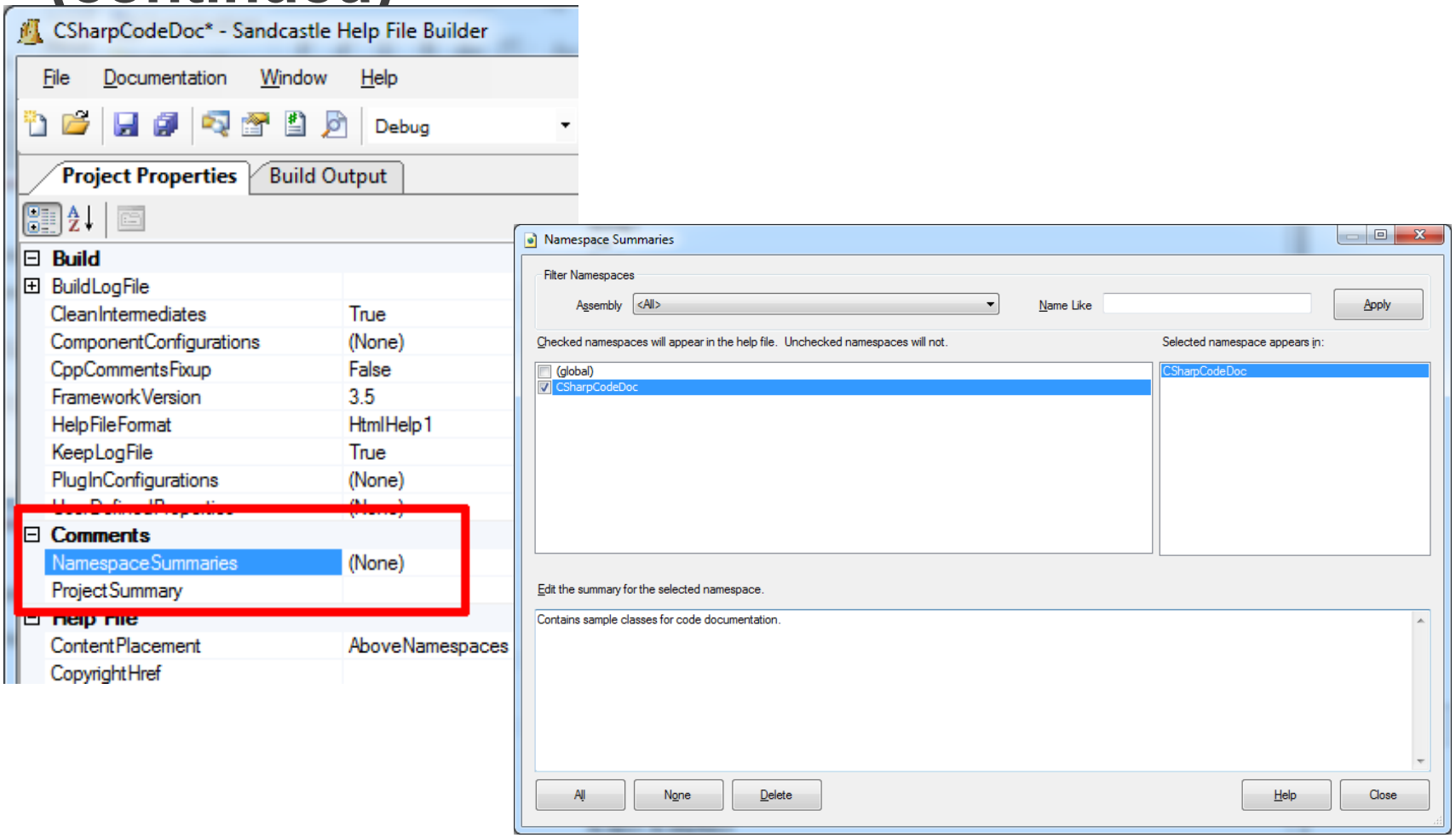
- Describes how a class or structure behaves in multi-threaded scenarios.
- Not defined by Microsoft, extension in *NDoc* and *SHFB*
- Syntax:

```
<threadsafety
  static="true|false"
  instance="true|false"/>
```

Namespace And Project Documentation

- Not defined by Microsoft, extension in SHFB
- Two options:
  - In project properties (*NamespaceSummaries* and *ProjectSummaries*)
  - Using a `NamespaceDoc` class

Namespace And Project Documentation (continued)



Namespace And Project Documentation (continued)

```
namespace CSharpCodeDoc
{
        using System.Runtime.CompilerServices;

        /// <summary>
        /// Contains sample classes for code documentation.
        /// </summary>
        [CompilerGenerated]
        internal class NamespaceDoc
        {
        }
}
```

Enhancing your documentation with conceptual content

CONCEPTUAL DOCUMENTATION

Introduction

- Add non-reference content to the help file
- Appears in the table of contents
- Written in *Microsoft Assistance Markup Language* (MAML)
  - Layout- and style-agnostic
  - XSL Transformations are used to generate the target format
- Currently no WYSIWYG editor in SHFB available
  - Use built-in editor or your favorite XML editor (VS, notepad++, etc.)
  - Use SHFB's "Debug" feature (F5) to preview result

Introduction (continued)

- Conceptual content file types
  - Content layout files
    - Defines TOC
    - Edit with content layout file editor
  - Topic files
    - Identified using a GUID
  - Image files
  - Token files
  - Code snippets files
- Tip: You can create your own template files (see SHFB Online Help)

Topic File Structure

```
<!-- The topic element contains the unique ID and revision number -->
<topic id="303c996a-2911-4c08-b492-6496c82b3edb" revisionNumber="1">

  <!-- This element name will change based on the document type -->
  <developerConceptualDocument
    xmlns="http://ddue.schemas.microsoft.com/authoring/2003/5"
    xmlns:xlink="http://www.w3.org/1999/xlink">

  <!-- The content goes here -->

  </developerConceptualDocument>

</topic>
```

Document Types

- Conceptual
- Error Message
- Glossary
- How-To
- Orientation
- Reference
- Reference With Syntax
- Reference Without Syntax
- Sample
- SDK Technology Architecture
- SDK Technology Code Directory
- SDK Technology Orientation
- SDK Technology Scenarios
- SDK Technology Summary
- Troubleshooting
- User Interface Reference
- Walkthrough
- Whitepaper
- XML Reference

Typical Conceptual MAML File

```
<?xml version="1.0" encoding="utf-8"?>
<topic id="00000000-0000-0000-0000-000000000000" revisionNumber="1">
  <developerConceptualDocument xmlns="…" xmlns:xlink="…">
    <summary>
      <para>Optional summary abstract</para>
    </summary>

    <introduction>
      <para>Required introduction</para>
      <autoOutline />
    </introduction>

    <section address="Section1">
      <title>Optional section title</title>
      <content>
        <autoOutline />
        <para>Add one or more sections with content</para>
      </content>
      <!-- optional sub sections -->
    </section>

    <relatedTopics>
      <link xlink:href="Other Topic's ID">Link inner text</link>
    </relatedTopics>
  </developerConceptualDocument>
</topic>
```

Typical Snippets File

```
<?xml version="1.0" encoding="utf-8" ?>
<examples>
  <item id="ClassDefinition#Define">
    <sampleCode language="CSharp">
      public class CSharpClass()
      {
          // Members go here
      }
    </sampleCode>
    <sampleCode language="VisualBasic">
      Public Class VBClass
          ' Members go here
      End Class
    </sampleCode>
  </item>

  […]

  </item>
</examples>
```

Bedarf nach mehr?
→ Versteckte Slides

- Tipps für MAML
  - Block Elements
  - Inline Elements
  - Medien (Bilder, Videos)
  - Links
    - Externe Links
    - Interne Links
    - Links zum SDK

Important Block Elements

- `alert`
  - Creates a note-like section to draw attention to some important information
  - General notes, cautionary, security, language-specific notes
  - Sample:

```
<alert class="note">
  <para>This is a note</para>
</alert>
```

- `code`
  - Like `<code>` element in C# code documentation
  - Tip: Use `<![CDATA[…]]>` to avoid encoding of sample code
- `codeReference`
  - Inserts a code snippet from an external code snippets file
  - Sample:

```
<codeReference>ClassDefinition#Define</codeReference>
```

Important Block Elements (continued)

- `definitionTable, definedTerm, definition`
  - Defines a list of terms and their definitions (two-column table without a header or footer)
  - Sample:

```
<definitionTable>
  <definedTerm>Term 1</definedTerm>
  <definition>Definition 1</definition>
  …
</definitionTable>
```

- `list, listItem`
  - Bullet, ordered or nobullet
  - Sample:

```
<list class="bullet">
  <listItem>Item 1</listItem>
  <listItem>Item 2</listItem>
  …
</list>
```

Important Block Elements (continued)

- `relatedTopics`
  - List of links to other topics that may be of interest to the reader
  - Link types (details see later)
    - `codeEntityReference`
    - `externalLink`
    - `link`
  - Link grouping
    - Secify target group using optional `topicType_id` attribute to elements
    - Group GUID values see MAML guide
  - Sample:

```
<relatedTopics>
  <link topicType_id="1FE70836-AA7D-4515-B54B-E10C4B516E50"
    xlink:href="b32a73b8-fc26-4c98-912c-d595fc1a17c2" />
  <externalLink>
    <linkText>Sandcastle Styles</linkText>
    <linkUri>http://www.codeplex.com/SandcastleStyles</linkUri>
    <linkTarget>_blank</linkTarget>
  </externalLink>
  <codeEntityReference qualifyHint="true">
    T:System.IO.FileStream</codeEntityReference>
</relatedTopics>
```

Important Block Elements (continued)

- `table, tableHeader, row, entry`
  - Arrange data in a table format with rows and columns
  - Sample:

```
<table>
  <title>A Simple Table with Title and Headers</title>
  <tableHeader>
      <row>
        <entry>Header 1</entry>
        <entry>Header 2</entry>
        <entry>Header 3</entry>
      </row>
  </tableHeader>
  <row>
        <entry>Row 1, Cell 1</entry>
        <entry>Row 1, Cell 2</entry>
        <entry>Row 1, Cell 3</entry>
  </row>
</table>
```

Important Inline Elements

- For a complete list see MAML Guide
- `codeInline`
  - String of code or a single keyword
- `command`
  - Name of an executable or other software application than can be run
- `literal`
  - Literal value
- `ui`
  - Describes a user interface element
- `token`
  - References a token in the token file

Links To Media

- Insert an image within the text of a conceptual topic
- Tip: Use `medialLinkInline` (details see MAML Guide) to display an image inline with text.
- Sample:

```
<!-- No caption, default image placement -->
<mediaLink>
<image xlink:href="6be7079d-a9d8-4189-9021-0f72d1642beb"/>
</mediaLink>

<!-- Caption before, centered image -->
<mediaLink>
<caption>Caption Before</caption>
<image placement="center" xlink:href="6be7079d-a9d8-4189-9021-0f72d1642beb"/>
</mediaLink>

<!-- Caption after with lead-in text, far image alignment -->
<mediaLink>
<caption placement="after" lead="Figure 1">
  Caption after with lead-in</caption>
<image placement="far" xlink:href="6be7079d-a9d8-4189-9021-0f72d1642beb"/>
</mediaLink>
```

Links

- `link`
  - Link to another conceptual topic using its ID value
  - Links to elements within the same page that have an `address` attribute (e.g. `section`)
  - Sample:

```
<link xlink:href="cfd9dabf-22f3-4742-8b54-
d84404610db1" />
<link xlink:href="cfd9dabf-22f3-4742-8b54-
d84404610db1">
companion file</link>
<link xlink:href="#intro">Back to Intro</link>
<link xlink:href="dc4fcc96-283e-4202-9ecc-
08a65e0c9313#BuildComps" />
```

Links (continued)

- `externalLink`
  - Link to an external URL
  - Attributes
    - `linkText`
    - `linkAlternateText` (for mouse hovering over it)
    - `linkUri`
    - `linkTarget` (default: `_blank`)
  - Sample:

```
<externalLink>
    <linkText>Sandcastle Styles</linkText>
    <linkAlternateText>
      Visit Sandcastle Styles</linkAlternateText>
    <linkUri>http://www.codeplex.com/SandcastleStyles</linkUri>
</externalLink>
```

Links (continued)

- `codeEntityReference`
  - Reference to a code entity such as a type, method, property, event, field, etc.
  - Can be a reference to a member in one of your classes or one in the .NET Framework.
  - Sample:

```
<codeEntityReference qualifyHint="true">
  T:System.IO.FileStream</codeEntityReference>
<codeEntityReference qualifyHint="true" autoUpgrade="true">
  M:System.IO.FileStream.#ctor(
    System.String,System.IO.FileMode)
</codeEntityReference>
<codeEntityReference qualifyHint="true" autoUpgrade="true">
  M:System.IO.FileStream.Write(
    System.Byte[],System.Int32,System.Int32)
</codeEntityReference>
<codeEntityReference qualifyHint="false">
  P:System.IO.FileStream.Length
</codeEntityReference>
<codeEntityReference qualifyHint="false" autoUpgrade="true">
  M:System.IO.FileStream.Flush
</codeEntityReference>
```

Bibliographie
Help und Sandcastle

- Hands-On Lab Sandcastle
  - Blogartikel von Rainer Stropek
- Collection of help links by The Helpware Group
- Windows Help on MSDN
  - Covers MS Help 1.4 and Assisted Platform 1.0 Client
- Sandcastle Blog on MSDN
- Wikipedia articles on Online help and MS Compiled HTML Help

Bibliographie
Help und Sandcastle

- The Story of Help in Visual Studio 2010
- MSHelpWiki.com
- XML Documentation Comments on MSDN
- Sandcastle Help File Builder installation instructions
- Online Help on Sandcastle Help File Builder

Tool Reference

- Sandcastle
  - Documentation Compiler for Managed Class Libraries
- MshcMigrate.exe
  - Convert older projects over to MS Help Viewer
- H3Viewer.exe
  - Alternative VS 2010 help viewer
- Sandcastle Styles Project
  - Style Patches and resources about MAML

Tool Reference (continued)

- GhostDoc
  - Generates documentation based on naming conventions
- StyleCop
  - Analyzes C# source code to enforce a set of style and consistency rules
- Sandcastle Help File Builder
  - Provides graphical and command line based tools to build a help file in an automated fashion

Tool Reference (continued)

- Sandcastle MAML Guide
  - Help about Microsoft Assistance Markup Language (MAML)