

# #Develop – Design & Architecture

## VO DevCon Europe 2004

Mike Krüger  
Senior Code Wrangler  
[mike@icsharpcode.net](mailto:mike@icsharpcode.net)

# Agenda

- Basic Ideas behind the #Develop Architecture
  - Vision, Scope, History, Development Process
- #Develop Add-In Structure
- Components of the Architecture
  - Codons, Conditions and Services
- Backend Bindings
- Pads
- Display Bindings

# Vision and Scope of the Architecture

- Component oriented Application
  - As extensible as possible
  - Avoid Side Effects between AddIns
  - Loose Coupling
- Permanent Refactoring
  - No immutable Interfaces
- All Data Files based on XML-Basis
- Use .NET Features wherever possible (Attributes, Reflection etc.)

# The #Develop Story

- Under Development since September 2001
- Open Source Project with a Core Team + Contributors
- Currently 2 active Main Contributors + a 'Cloud' of irregular Contributors
- Central issues:
  - Forms Designer
  - VB.NET Parser
  - Current: Debugger

# The Development Process

## ■ Team Structure:

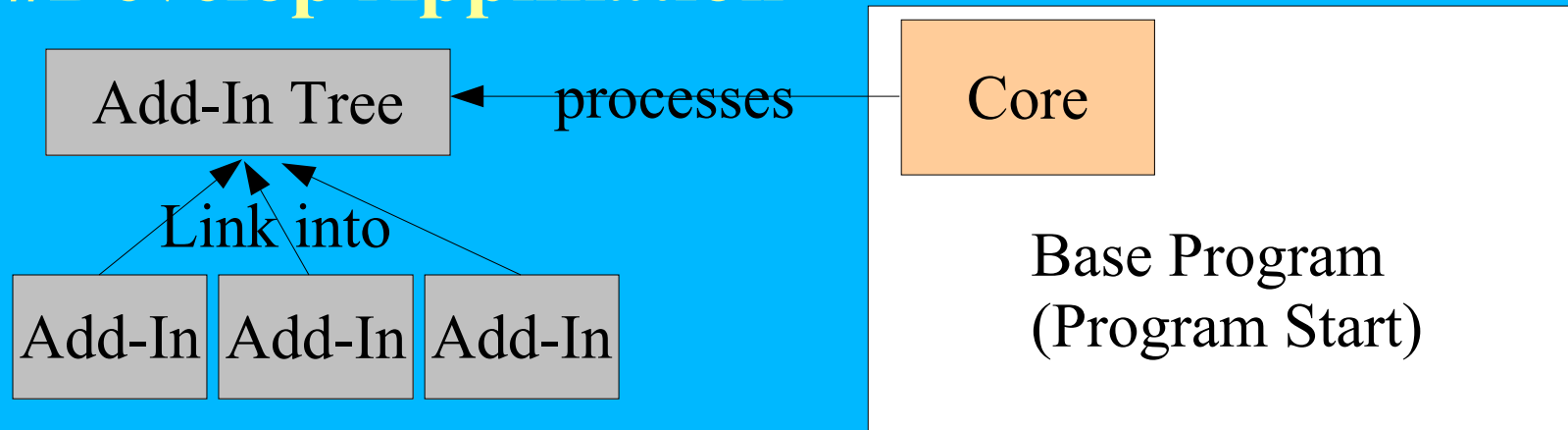
- Core Team (Wranglera)
  - Project, Code, Word, Bug, Parser
- Contributors
  - Code, Localization, Documentation

## ■ Process:

- Lightweight
- Flexible due to everything being AddIns
- Refactoring oriented
- Team Coordination via Code, IM, Email and Wiki

# #Develop Add-In Structure

## #Develop Applikation



- Every Component of #Develop is an Add-In
- Add-Ins may extend other Add-Ins
- However: Extensibility must be designed in by the the Developer of the Add-In
- Base Program is independent of #Develop

# Components of the Architecture

- Core
- Codons
- Conditionals
- Services
  - Properties
  - Resources

# Application Core

- Builds a tree in memory based on the XML Definition of the Add-In Tree
- Add-Ins may be removed at Run Time
- Attempts to determine the 'correct' Loading Sequence of the Add-Ins
- From a Sub-Tree an Array of Objects may be created
- Very Stable: nearly unmodified since 3 Years

# Codons

- Serve as basic Building Blocks (=Nodes in the Tree)
- Codons have Attributes
  - Some Attributes are compulsory (e.g. ID)
  - There are optional Attributes
- Codon Declarations are Classes having a CodonName Attribute
  - TheCodonName is used for Identification in the XML File

# XML Sample

```
<AddIn>
```

```
<Runtime>
```

```
<Import assembly="Sample.dll" />
```

```
</Runtime>
```

```
<Extension path =
```

```
" /SharpDevelop/AddIns/MyAddIn">
```

```
<Class id      = "MyTestClass"
```

```
  class = "MyNamespace.MyTestClass" />
```

```
</Extension>
```

```
</AddIn>
```

# Codon Definition Sample

```
[CodonName ("TestCodon")]
public class MyCodon : AbstractCodon
{
    [XmlAttribute ("firstAttribute")]
    string attribute1 = null;

    [XmlAttribute ("count", IsRequired=true)]
    int count = null;

    public override object BuildItem(object owner,
                                     ArrayList subItems,
                                     ConditionCollection conditions)
    {
        // TODO: implement Builditem
    }
}
```

# Conditionals

- A Condition can modify the Behaviour of a Codon at Run Time
  - Codon can be virtually 'removed' from the tree
  - Codon may be disabled as well
- Any Codon may be assigned any Number of Conditions
- Conditions may be joined with logical Operators AND/OR/NOT

# DEMO

# #Develop Services

- In Principle, #Develop Services are Services that:
  - Define a unified Interface for the Use of a Subsystem
  - Simplify the Use of a Subsystem
  - Are accessible by any Add-In at any Time
- Services are defined in the Add-In Tree
- !Most Important Concept after the Add-In Tree!

# Properties

- Is implemented as a Service
- A Service managing Key-Value Pairs
  - Keys are Strings
  - Values are Base Types or implement `IXmlConvertible`
- These Properties are serialized as XML
- The XML Serialization can be manipulated
- Used to manage ALL IDE Options

# Resources

- The Resource Service manages Key-String Pairs for Language Localization
- Loads Resource Files on Startup
  - Must load the correct File for the current Language
  - The current IDE Language is a Property
- The Resource Files are generated from a Database
- Every visual #Develop Component directly or indirectly accesses this Service

# Backend Bindings

- Are used to add new Programming Languages to #Develop
- In Principle, a Backend Binding consists of
  - Project Representation (Project Files)
  - Compile & Run Capability
  - Project Option Panels for Configuration dialogs
  - Syntax Highlighting Definition
- Optional: Parser, Smart Indenting Engine, Folding Engine etc.

# Example: BrainFuck

- Very small Turing complete Programming Language
- Invented in 1993 by Urban Müller with the goal of writing the smallest possible Compiler for a Programming Language (1<sup>st</sup> Version 240 Bytes)

'Hello World' in Brainfuck:

```
+++++++[[>++++++>++++++>+++>+<<<<.]
>+,>+.,+++++.,++.,>+.,<<+++++.,
>.,+++.
```

# Language Specification

- Works with a „Pointer“ on a 30.000 Byte „Tape“.
- < Increment Pointer
- > Decrement Pointer
- + Increment Byte at Pointer Position
- Decrement Byte at Pointer Position
- . Output Byte at Pointer Position (ASCII)
- , Input Byte at Pointer Position
- [ Jump to corresponding ] , if Byte at Pointer Position = 0
- ] Jump to corresponding [ , if Byte at Pointer Position != 0

# DEMO

# Pads

- Pads are Tool Windows
  - Project Browser, Output Window etc.
- Pads implement the abstract Interface `IPadContent` (provides the Pad Control)
- Abstract Class `AbstractPadContent` ought to be used
  - Note: There are NO immutable Interfaces in `#Develop`

# DEMO

# Display Bindings

- Define new Document windows for #Develop
  - Editors, Display Windows etc.
- Must define IDisplayBinding
  - 'Intelligent' Factory for the Document
- The actual Control is provided by IViewContent
  - Created by IDisplayBinding
- Problem: Several Displaybindings may want to 'display' the same File

# DEMO

# Questions?

develop

ic#code