Mono

WinForms Designer

Installation

Mono

```
svn co svn://anonsvn.mono-project.com/source/trunk/mwf-designer
make; make run
```

The WinForms Designer **requires** ~~Mono SVN Head.~~ Mono 2.0 or newer.

However if you do indeed run it on Mono SVN Head you could benefit from any bugfixes and improvements added since the last official release, because 99% of the actual code is part of the class libraries.

Visual Studio

If you want to use, run, contribute or debug the code with Visual Studio, Microsoft.NET and Windows - no problem. Just follow these 3 easy steps:

```
svn co svn://anonsvn.mono-project.com/source/trunk/mwf-designer
```

1. Install Python - http://python.org/download/
2. Run "Prepare Visual Studio Build.bat"
3. Open mwf-designer.sln with Visual Studio and you are done!

What is happening automatically behind the scenes is the generation of a Mono.Design assembly - a subset of Mono's System.Design. The designer is then compiled with this assembly referenced.

The python script will:

- Download a subset of Mono System.Design assembly's source code from SVN
- Apply a set of patches to make it run against Microsoft .NET
- Change namespaces to "Mono.Design".

Mono with Mono.Design

```
svn co svn://anonsvn.mono-project.com/source/trunk/mwf-designer
make mono-design && make run
```

What is happening automatically behind the scenes is the generation of a Mono.Design assembly - a subset of Mono's System.Design. The designer is then compiled with this assembly referenced.

The python script will:

- Download a subset of Mono System.Design assembly's source code from SVN
- Apply a set of patches to make it run against Microsoft .NET
- Change namespaces to "Mono.Design".

Status

Unfortunately the designer is not ready yet for proper use. More information below.

99% of the functionality of the windows forms designer is in the System.Design assembly where the .NET 2.0 Design-Time Framework is and on where most of the development takes place.

The main stumble point is the lack of true Control transparency (WS_EX_TRANSPARENT), which is not yet supported by the MWF X11 backend, but is required for a proper Drag and Drop and other interaction (snap lines, etc) on the design service. There is WinForms bug in our bug tracker for this problem - http://bugzilla.novell.com/show_bug.cgi?id=323819.

Currently the designer can:

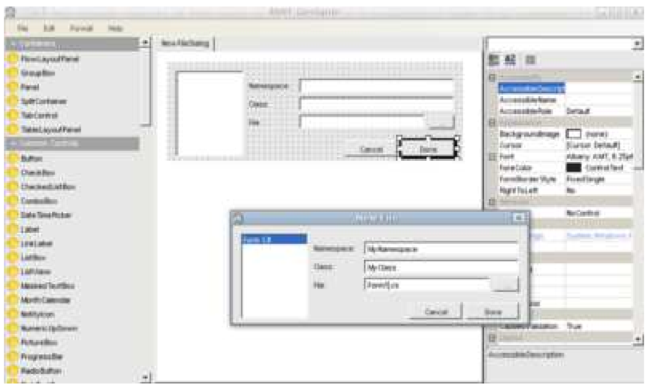
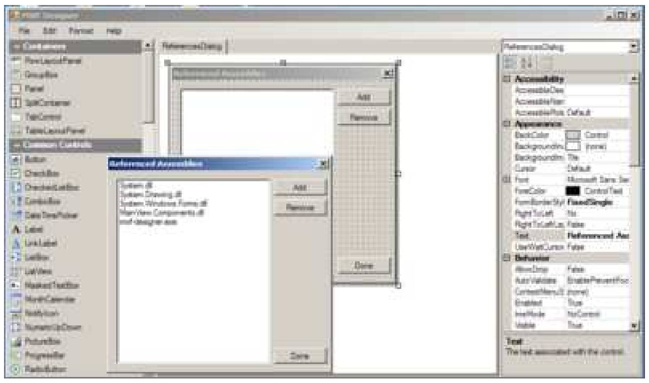
- Load C# Windows Forms form generated by Visual Studio/MS.NET
- Add simple controls, such as buttons and panels from the toolbox and modify their properties.
- Move and resize those controls.
- Undo/Redo/Cut/Copy/Paste those controls.
- Persist the form modifications back to source code compatible with Visual Studio/MS.NET.

Bugs, Patches, Questions

Bugs can be filed on the Mono bug tracker and should be assigned to me (Ivan N. Zlatev) - contact AT i-nz.net.. For more information on how to file bugs take a look at the Bugs wiki page.

Patches can be send to the mono-devel-list or mono-winforms-list Mailing Lists. That's also the place to ask questions. Please follow the Mono Code Style Guidelines.

Screenshots

TODOs

The best way to find out something you want to work on is to play with the designer and see what doesn't work or you don't like. Below are some related important tasks I have sorted in groups in semi-chronological order.

For background information on how the designer works check #.NET_Design-Time_Framework

Task Packs

~~New, Close~~

- ~~Implement New ->*~~
- ~~Implement Close Document~~
- ~~Doesn't support multi-component selection, because it only handles the ISelectionService.PrimarySelection and not the GetSelectedComponents.~~

~~Error Reporting~~

- ~~*Designer*~~
  - ~~Implement an IUIService. An easy to access place in the frontend where errors/warning will be shown, but also debug and trace information from the Design-Time code in debug mode.~~
- ~~*Serialization*~~

    - ~~Make use of the IUIService.~~
    - ~~Uncomment the try-catches in DesignerSerializationManager.~~
    - ~~Refactor current CWLs in the serializers to something actually useful to print more detailed information on the currently (de)serialized expression.~~
    - ~~May be create some sort of a TraceContext?~~

~~Undo/Redo (Cut, Copy, Paste)~~

- *~~Serialization~~*
    - ~~Implement: UndoEngine, ComponentSerializationService, CodeDomComponentSerializationService, CodeDomDesignerLoader.IDesignerSerializationService~~
- *~~Surface~~*
    - ~~Implement MenuCommandService : IMenuCommandService~~
    - ~~MenuCommand - update to 2.0 and review current code.~~
    - ~~Transactions must be created for:~~
        - ~~creating components~~
        - ~~destroying components~~
        - ~~moving controls~~

Quick Fixes Fun Pack

- *Serializers*
    - ~~ControlCodeDomSerializer - Resume/SuspendLayout.~~
    - ContainerCodeDomSerializer - to serialize the container for the components and its disposing.
- *~~MWF Designers~~*
    - ~~DocumentDesigner has "is Form" code (sets TopLevel to false) which should go in a FormDocumentDesigner.~~
- *Surface*
    - IDesignerHost.Activate is invoked after the root component is added to the host. This is just my guess. It has to be further investigated when, where and by what should that be invoked.
    - DesignerHost.Deactivated is never fired.
    - Currently there is a fallback DesignerHost.CreateDesigner because many designers are missing.
- *Designers*
    - Check ControlDesigner.OnContextMenu and related.

Resources (De)Serialization

Everything that has Localizable (true) attribute gets serialized to a resource.

VS.Net creates a global Properties.resx for the project, where it stores the resources and also Properties.Designer.cs, where it generates properties with the name of the resource. On this basis it just serializes property references when it comes to resource serialization. Properties.* probably get's compiled and loaded from a temporary assembly before the deserialization process is invoked.

~~PropertyGrid~~

- ~~Current designer's custom PropertyGrid.cs code is ugly and needs a clean rewrite.~~
- *~~Events Editing (MWF)~~*
    - ~~Implement EventsTab.~~
    - ~~Add EventsTab support in the PropertyGrid control.~~

New selection/DnD/resize service

- Implement the System.Windows.Forms.Design.Behavior namespace (BehaviorService, etc) to replace the current IUISelectionService.
- Requires true Control transparency (WS_EX_TRANSPARENT), which is not yet supported by the MWF X11 backend. MWF bug - http://bugzilla.novell.com/show_bug.cgi?id=323819 .

ComponentTray

- Depends on the Behavior-based selection service.

Menu editing

- *Designer*
    - Implement IMenuEditorService.
- *MWF Designers*
    - Menu has a designer in a Visual Studio assembly (Microsoft.VisualStudio.Windows.Forms.MenuDesigner) - fix that.

DesignerActions

- Surface
    - Implement DesignerActionService, DesignerActionUIService, DesignerAction* and DesignerActionUI*

Extender Providers

- *Surface*
  - Review ExtenderService : IExtenderProviderService, IExtenderListServiceIExtenderService implementation.
- *Serializers*
  - Support for (De)Serialization of "provided" properties.

AmbientValueAttribute - http://msdn2.microsoft.com/en-us/library/system.componentmodel.ambientvalueattribute.aspx

- *Serializers*
  - Just add a handling of AmbientValueAttribute in PropertyCodeDomSerializer.ShouldSerialize(...)

ISupportInitialize and ctor (IContainer container)

- *Serializers and Surface*
  - ISupportInitialize - http://msdn2.microsoft.com/en-us/library/system.componentmodel.isupportinitialize.aspx
  - Detection and handling of a constructor to accept the IContainer as a parameter: ctor (IContainer container) - http://books.google.com/books?id=CpgQthE6v_0C&pg=PA246&lpg=PA246&ots=VFVg1OvrMv&sig=LjPu15hzKWFFIJnIjBOqM8z-ts0#PPA248,M1

Visual Inheritance

- InheritanceService, InheritanceAttribute, "Modifier" Design-Time property. How is the Modifier property persisted for the project?

Support Classes

Serializers

- ~~ControlCodeDomSerializer~~
- ~~ControlCollectionCodeDomSerializer~~
- DataGridViewRowCollectionCodeDomSerializer
- ImageListCodeDomSerializer
- TableLayoutControlCollectionCodeDomSerializer
- TableLayoutPanelCodeDomSerializer
- ToolStripCodeDomSerializer
- ToolStripMenuItemCodeDomSerializer

Designers

- AxHostDesigner
- BindingNavigatorDesigner
- BindingSourceDesigner
- ButtonBaseDesigner
- ComboBoxDesigner
- DataGridDesigner
- DataGridViewColumnDesigner
- DataGridViewComboBoxColumnDesigner
- DataGridViewDesigner
- DateTimePickerDesigner
- FlowLayoutPanelDesigner
- FolderBrowserDialogDesigner
- GroupBoxDesigner
- ImageListDesigner
- LabelDesigner
- ListBoxDesigner
- ListViewDesigner
- MaskedTextBoxDesigner
- MonthCalendarDesigner
- NotifyIconDesigner
- OpenFileDialogDesigner
- ~~PanelDesigner~~
- PictureBoxDesigner
- PrintDialogDesigner
- PropertyGridDesigner
- RadioButtonDesigner
- SaveFileDialogDesigner
- ScrollableControlDesigner
- SplitContainerDesigner

- SplitterDesigner
- SplitterPanelDesigner
- StatusBarDesigner
- TabControlDesigner
- TableLayoutPanelDesigner
- TabPageDesigner
- TextBoxBaseDesigner
- TextBoxDesigner
- ToolBarButtonDesigner
- ToolBarDesigner
- ToolStripContainerDesigner
- ToolStripContentPanelDesigner
- ToolStripDesigner
- ToolStripDropDownDesigner
- ToolStripItemDesigner
- ToolStripMenuItemDesigner
- ToolStripPanelDesigner
- TrackBarDesigner
- TreeViewDesigner
- UpDownBaseDesigner
- WebBrowserBaseDesigner

PropertyGrid Editors

- System.Windows.Forms.Design.BorderSidesEditor
- Windows.Forms.Design.HelpNamespaceEditor
- Windows.Forms.Design.ImageCollectionEditor
- Windows.Forms.Design.ImageIndexEditor
- Windows.Forms.Design.LinkAreaEditor
- Windows.Forms.Design.ListViewSubItemCollectionEditor
- Windows.Forms.Design.MaskedTextBoxTextEditor
- Windows.Forms.Design.MaskPropertyEditor
- Windows.Forms.Design.SelectedPathEditor
- Windows.Forms.Design.ShortcutKeysEditor
- Windows.Forms.Design.StyleCollectionEditor
- Windows.Forms.Design.ToolStripCollectionEditor
- Windows.Forms.Design.ToolStripImageIndexEditor
- Windows.Forms.Design.TreeNodeCollectionEditor
- Windows.Forms.Design.DataGridColumnCollectionEditor
- Windows.Forms.Design.DataGridColumnStyleFormatEditor
- Windows.Forms.Design.DataGridColumnStyleMappingNameEditor
- Windows.Forms.Design.DataGridTableStyleMappingNameEditor
- Windows.Forms.Design.DataGridViewCellStyleEditor
- Windows.Forms.Design.DataGridViewColumnCollectionEditor
- Windows.Forms.Design.DataGridViewColumnDataPropertyNameEditor
- Windows.Forms.Design.DataGridViewComponentEditor
- ~~System.ComponentModel.Design.MultilineStringEditor~~ - stubbed
- ~~Windows.Forms.Design.DataMemberFieldEditor~~ - stubbed
- ~~Windows.Forms.Design.DataMemberListEditor~~ - stubbed
- ~~Windows.Forms.Design.FormatStringEditor~~ - stubbed
- ~~Windows.Forms.Design.ListControlStringCollectionEditor~~
- ~~Windows.Forms.Design.StringArrayEditor~~
- ~~Windows.Forms.Design.StringCollectionEditor~~
- ~~Windows.Forms.Design.TabPageCollectionEditor~~

Menu Commands

All of the commands below are added by DocumentDesigner.Initialize(...). This was found by the means of the following SpyMenuCommandService added to a DesignerSurface on MSNET.

```
public class SpyMenuCommandService : MenuCommandService
{
        public SpyMenuCommandService (IServiceProvider provider) : base (provider)
        {
        }

        public override void AddCommand (MenuCommand command)
        {
                Console.WriteLine ("[Command:] " + command.ToString () + Environment.NewLine);
                Console.WriteLine ("=== Stack Trace ===" + Environment.NewLine);
                Console.WriteLine (Environment.StackTrace);
                Console.WriteLine ("===================" + Environment.NewLine);
                base.AddCommand (command);
        }
}
```

- **StandardCommands**
  - AlignBottom
  - AlignHorizontalCenters
  - AlignLeft
  - AlignRight
  - AlignToGrid
  - AlignTop
  - AlignVerticalCenters
  - BringToFront
  - CenterHorizontally
  - CenterVertically
  - ~~Copy~~
  - ~~Cut~~
  - ~~Delete~~
  - HorizSpaceConcatenate
  - HorizSpaceDecrease
  - HorizSpaceIncrease
  - HorizSpaceMakeEqual
  - Paste
  - SelectAll
  - SendToBack
  - SizeToControl
  - SizeToControlHeight
  - SizeToControlWidth
  - SizeToGrid
  - SnapToGrid
  - TabOrder
  - VertSpaceConcatenate
  - VertSpaceDecrease
  - VertSpaceIncrease
  - VertSpaceMakeEqual
  - ShowGrid
  - LockControls

- **MenuCommands**
  - KeyDefaultAction
  - KeySelectNext
  - KeySelectPrevious
  - KeyMoveLeft
  - KeySizeWidthDecrease
  - KeyMoveRight
  - KeySizeWidthIncrease
  - KeyMoveUp
  - KeySizeHeightIncrease
  - KeyMoveDown
  - KeySizeHeightDecrease
  - KeyCancel
  - KeyNudgeLeft
  - KeyNudgeDown
  - KeyNudgeRight
  - KeyNudgeUp
  - KeyNudgeHeightIncrease
  - KeyNudgeHeightDecrease
  - KeyNudgeWidthDecrease
  - KeyNudgeWidthIncrease
  - DesignerProperties
  - KeyReverseCancel

Known Limitations

Not supported currently are, as follows.

Design Surface

- DesignerOptions, WindowsFormsDesignerOptionService
- Extender Providers - http://msdn2.microsoft.com/en-us/library/ms171835.aspx
- LicenseContext

CodeDom (De)Serialization

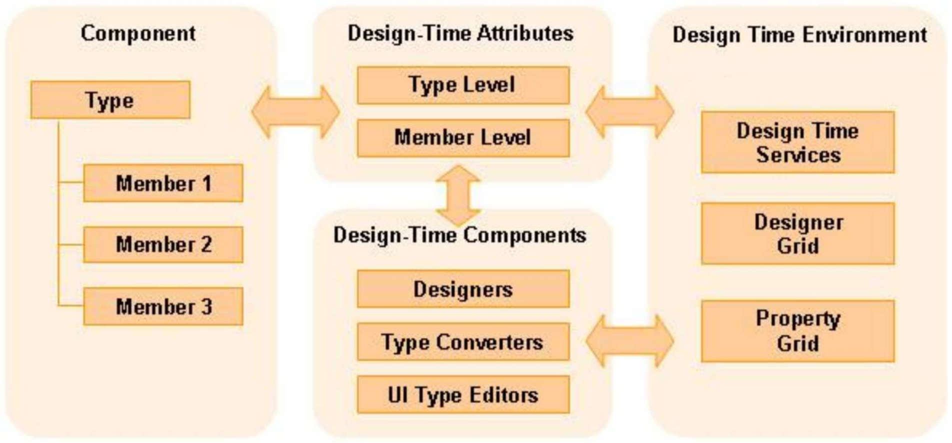
- No creation expression is serialized for nested components by the ComponentCodeDomSerializer. I can't think of a use case where this would be required.
- InheritanceService, InheritanceAttribute - what are those for?

.NET Design-Time Framework

The .NET Design-Time framework has a highly abstracted, extensible and complex service-oriented architecture which provides loose coupling between components, containers, services and the designer tool.

In the .NET Design-Time architecture components associate their Design-Time functionality independent of the design tool. A list of Design-Time services is provided, which communicate with the components, designers and between each other. The design surface is composed by those services, the designers and the components. The designer tool hosts the design surface and makes use of the available services as well as adds additional such.

The architecture makes use of attributes with metadata to describe behaviour for components and their members. Dynamic addition, removal and modification of the metadata of the components is possible.



Loose Coupling Concepts

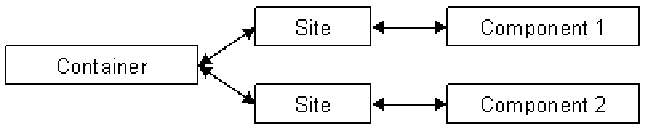
Loose coupling is a key concept which formed the design of the Design-Time framework due to the extensibility and modularity it provides.

Components, Containers, Sites

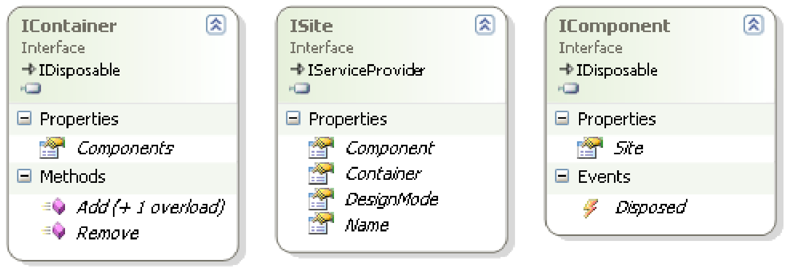
At the core of the Design-Time framework is an interface called *IComponent*. Anything that implements this interface is called a component. Components have three important characteristics:

1. They can be owned by/hosted in a container.
2. They can request services from the Site.
3. They can have a name assigned.

The first characteristic essentially allows the lifetime of the component to be controlled by the container. The second characteristic gives the component access to functionality provided by other parts of the framework and the designer tool. The third characteristics allows the container to assign unique identifier to each component. Components are design-time aware through their Site.



This is a fundamental concept as it provides full abstraction of visual and non visual components. Also the component can be a Windows Forms or an ASP.NET control and as long as it implements *IComponent* that will not be of significance to the design-time framework.



Services

A service is an instance of an object that implements a specific set of functionality. Services are stored in service containers and can be retrieved from service providers such as a component's Site and *IServiceProvider* implementations in general.

The power of services lies in their loose coupling: an application publishes the interface or base class that defines the service, but does not publish the class that implements the service. This design pattern allows the components to request the service and retrieve an instance they can use, but never have to know about the actual implementation nor where it comes from. In additional it

Service containers contain a table of services in the form of type-object key-value pair. To reduce the memory footprint service containers have been designed to support lazy instantiation of service objects.

Design-Time Attributes

Design-Time attributes associate a type or type member with a class that extends its design-time behaviour without creating a functional dependency for the type on the class. This framework makes use of the following attributes:

Attribute	**Description**
DesignerAttribute	Associates a type with a designer.
TypeConverterAttribute	Associates a type or type member with a type converter.
EditorAttribute	Associates a type or type member with a type editor.
DesignerSerializerAttribute	Associates a type with a serializer

Design Surface

The *Design Surface* is a thin front-end to the .Net Design-Time framework and is designed to be instantiated and used by the designer tool directly.

The designer surface has to be initialized by the designer tool with a root component type, such as a System Windows Form or an ASP.NET Web Page and is responsible for:

- Allowing the designer tool to request the surface to be loaded and provide access to the errors if any during the loading process.
- Provide the designer tool with a visual representation of the design surface. This is the View that will be presented to the user.
- Provide a service container for the underlying components of the framework.
- Allow the designer tool to request the persistence of the design surface.

State Persistence

The design surface delegates its loading and persisting to a *Designer Loader*, which is to be supplied by the designer tool during initialization. The Designer loader implementation of the tool is responsible for feeding the Design-Time Serialization system with an object graph recovered from the persistent storage as well as for persisting to the desired by the tool format an object graph resulted from the serialization of the surface. That way system maintains its format neutrality.

At load-time the Design-Time Serialization system will process the object graph to create component instances, which it will then add to the Design-Time container.

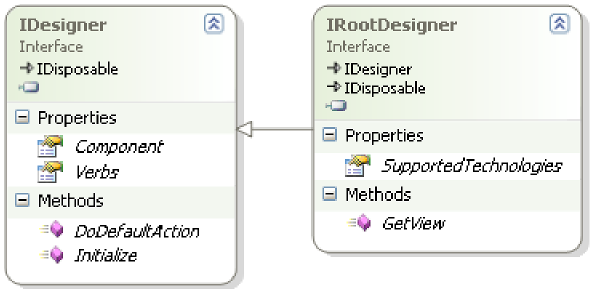
Design-Time Container

The Design-Time container is designed to be the central storage of components in design mode. They are added by type, instantiated dynamically using reflection together with their associated designer. Each component has an unique identifier - its name, which it gets assigned after being added to the container.

Designers

Designers are classes that are associated with a component type via the *DesignerAttribute and* implement the *IDesigner* interface. Designers are initialized with an instance of a component, which they manage.

The root component, such as a Windows Forms form or an ASP.NET web page, has to implement the *IRootDesigner* interface, which is special in that it has to provide the visual representation of the component - the View. It is very important to note that the *GetView* method (Illustration 4: IDesigner and IRootDesigner ,Page 5) is explicitly designed not to return a specific type (such as a Windows Forms *Control* or an ASP.NET web page), but a generic *Object*, which can be casted to the appropriate type by the designer tool. The framework implements two Windows Forms root designers - for forms and for custom controls (*UserControl*).



The designers are responsible for:

- Performing custom initialization for a component in design mode.
- Alter and extend the behaviour or appearance of components in design mode.
- Adjust the attributes, events, and properties exposed by a component with which the designer is associated through the powerful type description functionality of .NET.
- Add menu items to the shortcut menu of a component.

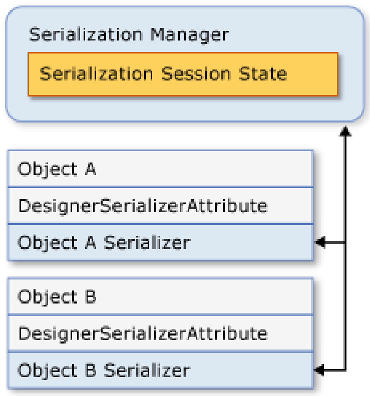
Design-Time Services

The design surface provides a service container to the rest of the components of the design-time framework and populates it with a default implementation of several important design-time services, including the following.

Service	**Description**
INameCreationService	Generates or validates names for a components.
ISelectionService	Defines an interface for programmatic component selection and selection tracking, but not a visual one.
IMenuCommandService	Allows designers to add actions to the right click menu of a component during design time.
IServiceContainer	The design-time service container, where other parts of the framework can add their provided services.
IComponentChangeService	Notifies for when components are added/removed from the design surface and also when they get modified.

Design-Time Serialization

Design-Time serialization is the process of converting an object graph into a source file (code, markup or other format) that can later be used to recover the object graph. It is based on reflection and type transformation.


Serialization Manager
Serialization Session State
Object A
DesignerSerializerAttribute
Object A Serializer
Object B
DesignerSerializerAttribute
Object B Serializer

The Design-Time serialization differs from the standard object serialization in the following ways:

- It separates the object that is performing the serialization from the one that is being serialized. Again, the keyword here is loose coupling.
- It serializes only properties that have been modified in order to minimize the output as well as provide only meaningful such.
- It ignores objects that aren't convertible to instance descriptors instead of throwing exceptions and interrupting the serialization process.

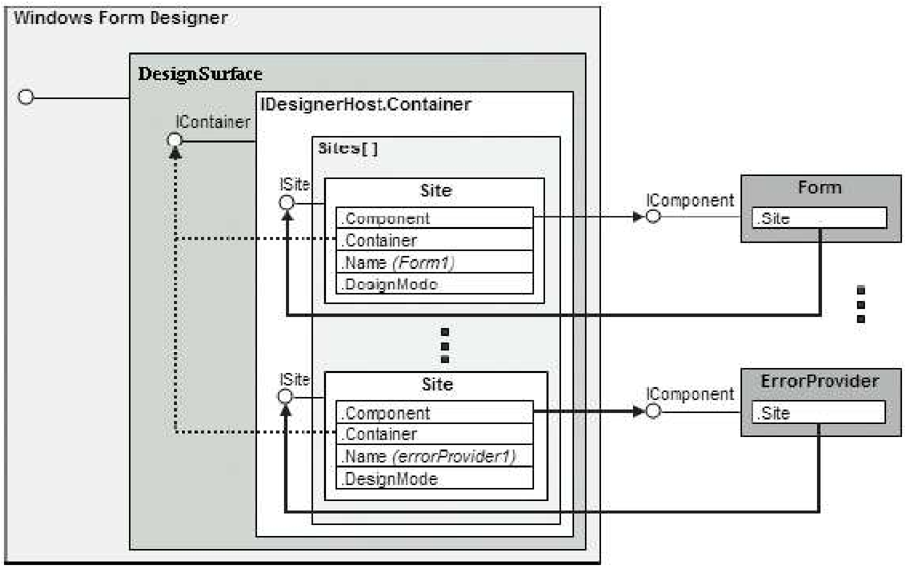
The design aims at providing a serialization system that is:

- Modular - Types are loosely associated with serializers via the *DesignerSerializationAttribute*.
- Format neutral - Each serializer handles the serialization for a specific type and it's up to the serializer to decide in what format the data is to be stored.
- Extensible - The system is designed to give priority to objects defined as serializer providers before checking for *DesignerSerializationAttribute*. Those object feed the serialization system with serializers based on an internally stored type-serializer table. This allows default serialization associations via attributes to be overriden or association of serializers with types that lack a *DesignerSerializationAttribute*.
- Context Sensitive - The serialization process is managed by a serialization manager an instance of which is passed to all serializers to allow them to communicate.

The Designer Tool

Based on the proposed design the designer tool can be said to be just a thin front end to the whole Design-Time framework (Illustration 6: A designer tool hosting a design surface, Page 8), with only a few responsibilities:

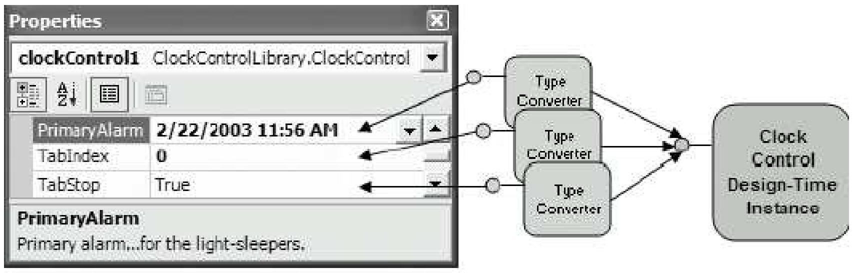
1. Instantiate the design surface with an arbitrary root component type and host the design surface view.
2. Provide a two way bridge between the Design-Time serialization system object graph and the data persistence format of choice.
3. Provide means for editing the metadata of the components in the surface.
4. Provide a way to add components to to the surface.

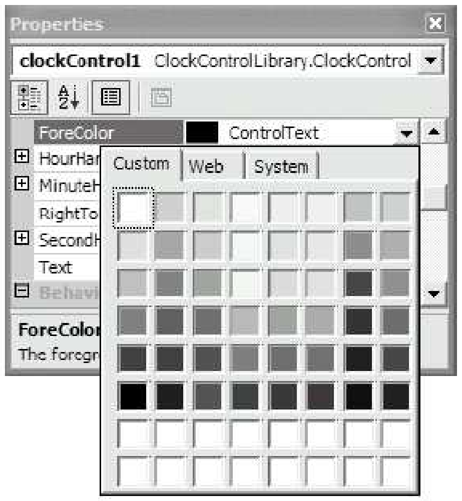


Metadata Editing

With the proposed design the designer tool has access to the *ISeletionService,* offered by the design surface, which provides information about the currently selected component as well as a mechanism for notification when the selection changes.

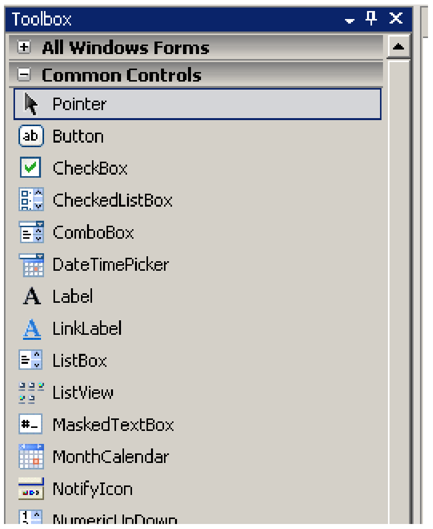
The designer tool will make use of that service to monitor the surface and populate a property grid with the currently selected object's properties using reflection. It will use type transformation (Illustration 7: Type Transformation in action, Page 9) and type editing (Illustration 8: Type Editing in action, Page 9) to provide user friendly facilities for metadata customization.


Properties
clockControl1 ClockControlLibrary.ClockControl
PrimaryAlarm 2/22/2003 11:56 AM
TabIndex 0
TabStop True
PrimaryAlarm
Primary alarm...for the light-sleepers.
Type Converter
Type Converter
Type Converter
Clock Control Design-Time Instance



Component Toolbox

Based on the powerful reflection functionality of .NET the designer tool can populate a component toolbox for a particular GUI toolkit by filtering all types that derive from the toolkit's foundation control type (e.g. *Control* for Windows Forms). The user will be able to drag and drop a component toolbox item onto the design surface which will be handled by the designer for the control under cursor to request the instantiation of the component.



Documentation Resources

- Building User Interface Design Tools (*http://www.i-nz.net/files /docs/Building%20User%20Interface%20Design%20Tools%20-%20Ivan%20N.%20Zlatev.pdf*)
- Create And Host Custom Designers With The .NET Framework 2.0 (*http://www.i-nz.net/files/docs/designerhosting/create-and-host-custom-designers-dot-net.html*)
- Extending Design-Time Support (*http://msdn2.microsoft.com/en-us/library/37899azc.aspx*)

Table of Contents

-
    -
    -
    - Legal Notices
    - Editor Login
- Mono
    - About
    - Roadmap
    - Technologies
    - Screenshots
    - FAQ
    - Contact
- Download
    - Latest Release
    - MonoDevelop
    - Migration Analyzer
    - Daily Snapshots
    - Previous Releases
- Documentation
    - Getting Started
    - API Reference
    - Articles
- Community
    - Mailing Lists
    - Forums
    - Chat/IRC
    - Blogs
- Contribute
    - Contributing Guide
    - Report Bugs
    - SVN
    - Build Status
    - Class Status