ObjectEditor.NET

Design and Architecture

Koren Bar

Version 1.0

Table of Contents

---

1. Introduction

1.1 Purpose and Goals

ObjectEditor.NET provides developers with a dynamic, recursive editing tool for manipulating any .NET object at runtime. The tool is highly modular and designed with extensibility in mind, enabling developers to work seamlessly with nested and collection-based data structures.

1.2 High-Level Description

The project uses the MVC architecture to separate concerns, promote code reuse, and ensure flexibility. It dynamically adjusts its user interface based on the type and structure of the object being edited, leveraging reflection for metadata handling and recursive editing.

1.3 Target Audience

This tool is intended for developers and engineers who require a flexible, generic editor for data structures during debugging, development, or runtime customization.

---

2. System Overview

2.1 Functional Overview

ObjectEditor.NET enables:

- Recursive editing of .NET objects and their properties.
- Automatic adaptation to various data types and nested structures.
- Support for collections, dictionaries, and enumerable types.

---

3. System Architecture

3.1 Overview of the MVC Structure

The system is divided into three layers:

- **Model**: Represents data and provides metadata using reflection.
- **View**: Implements the dynamic, GUI-based user interface using WinForms.
- **Controller**: Bridges user input and data operations while managing recursive object handling.

3.2 Layer Responsibilities

- **Model**: Handles data representation and reflection-based metadata.
- **View**: Adapts dynamically to object structure and field types.
- **Controller**: Manages operations like editing, navigation, and synchronization.

---

4. Detailed Design

4.1 Model Layer

4.1.1 Data Representation

The model layer consists of:

- Metadata classes (`FieldInfo`, `Settings`) to store type and field details.
- Reflection-based utilities to inspect objects and their properties dynamically.

4.1.2 Reflection and Metadata Handling

Reflection provides:

- Identification of object properties and types.
- Recursive handling of nested objects and collections.

4.2 View Layer

4.2.1 Dynamic GUI Integration

The GUI dynamically adjusts controls based on the field type:

- **`TextBox`**: Default control for strings or undefined value types.
- **`CheckBox`**: For boolean fields.
- **`NumericUpDownBox`**: For numeric value fields.
- **`ComboBox`**: For enum value fields.

Control creation is managed by the `FieldControlFactory` static class, which maps types to controls.

4.2.2 Widgets and Controls Mapping

- **`ValueFieldControl`**: Connects value fields to appropriate `IValueControl`.
- **`ObjectFieldControl`**: Extends `ValueFieldControl` to manage references while supporting recursive editing.

4.3 Controller Layer

4.3.1 Editor Controllers

Specialized editors extend `ObjectEditorController` for specific scenarios:

- **EnumerableEditorController**: Extends `ObjectEditorController` to add support for iterating over `IEnumerable` fields, enabling viewing of collection items without modifying them.
- **CollectionEditorController**: Extends `EnumerableEditorController` to handle add/remove operations on `ICollection` fields, enabling modification of collections.
- **DictionaryEditorController**: Extends `CollectionEditorController` to manage `IDictionary` fields, enabling viewing and editing of key-value pairs. It provides specialized support for dictionaries by creating `KeyValuePairFieldController` instances for each dictionary entry.
- **ListEditorController**: Extends `CollectionEditorController` to manage `IList` fields, providing full support for adding, removing, and reordering items within lists.

4.3.2 Field Controllers

Controllers like `ObjectFieldController` provide recursive editing by delegating nested objects to sub-editors.

4.4 Supporting Components

4.4.1 Extensions and Utilities

- **Type Extensions**: GetDefaultValue, GetGenericType, GetInheritanceChain, ChangeType, etc.
- **Casting Extensions**: object.CastKeyValuePair, KeyValuePair.CastTo, etc.
- **Attributes Extensions**: Utilities related to custom attributes.
- **WinForms Extensions**: Utility functions for UI operations, including:
    - **InvokeUserAction**: Invoke the user action and handle exceptions, showing a message box with the error message(s) instead of throwing exceptions. Should be used for actions triggered from the UI only, as it shows a message box as a result.
      InvokeUI: Handles safe UI updates from background threads by using the dispatcher to ensure execution on the main thread.

4.4.2 Attributes and Events

Attributes:

- **InfoAttribute:** Gives a name and description to a property field.
- **PermissionGroupAttribute**: Associates a property or class with a permission group.
- **EditorDisplayNameAttribute:** Uses the property value as a display name of an item in a collection.
- **EditorPasswordAttribute:** Masks the property value in the object editor.
- **EditorIgnoreAttribute:** Ignores the property in the object editor.
- **EditorIgnoreInheritedAttribute:** Ignores the class inherited properties in the object editor.

Editor Controller Events:

- **ValueChanged**: A field (or nested field) value was changed.
- **SaveRequiredChanged**: The source object was updated (true) or the data was saved (false).
- **ChangesApplied**: Changes on this controller were applied to the source object.
- **ChangesPendingChanged**: This occurs when the changes pending flag is changed.
- **DataSaved**: The data was saved to a file. (not supported yet)
- **FieldAdded**: This occurs when a field is added to this controller.
- **FieldRemoved**: This occurs when a field is removed from this controller.

Field Controller Events:

- **ValueChanged**: Occurs after the field value changes.
- **Removing**: Occurs when the user requires to remove this item from the collection.
- **StatusChanged**: Occurs when the status of the field changes.

---

5. Design Patterns

5.1 Factory Pattern

Simplifies the creation of controllers and GUI components based on types.

5.2 Observer Pattern

Facilitates real-time updates between the controller and view.

5.3 Inheritance Hierarchy

Promotes code reuse through a structured hierarchy for controllers and field handlers.

---

6. System Behavior

6.1 Interaction Flow

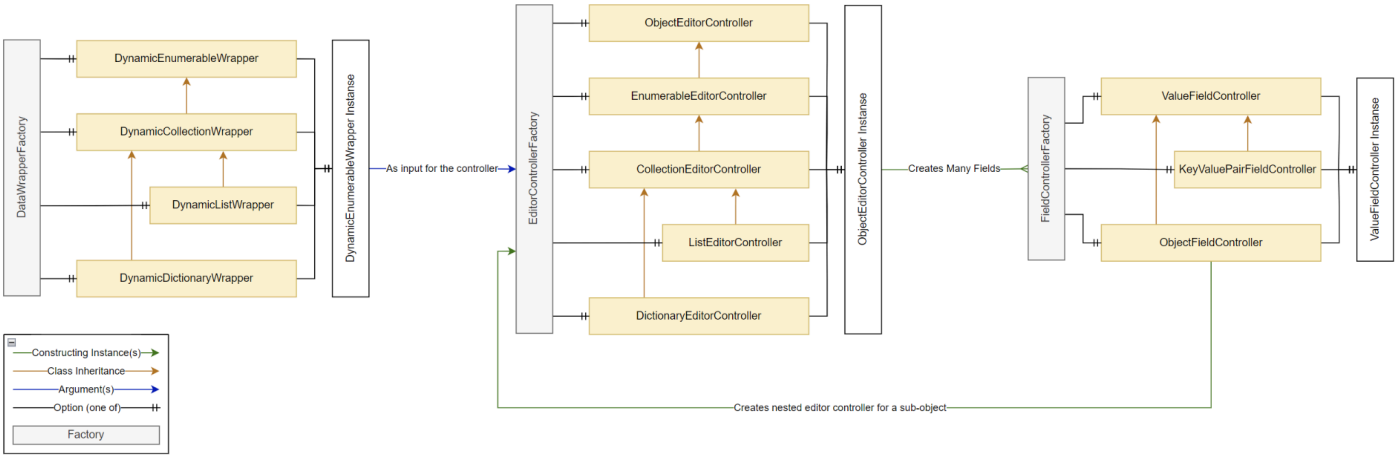
- User initializes an editor form (if the user gives no controller, a new one will be created by the form constructor).
- Editor controller instance dynamically creates field controllers (using factory).
- Value field controllers allow viewing and editing values.
- Object field controllers create child editor controllers for inner reference values.

6.2 Key Use Cases

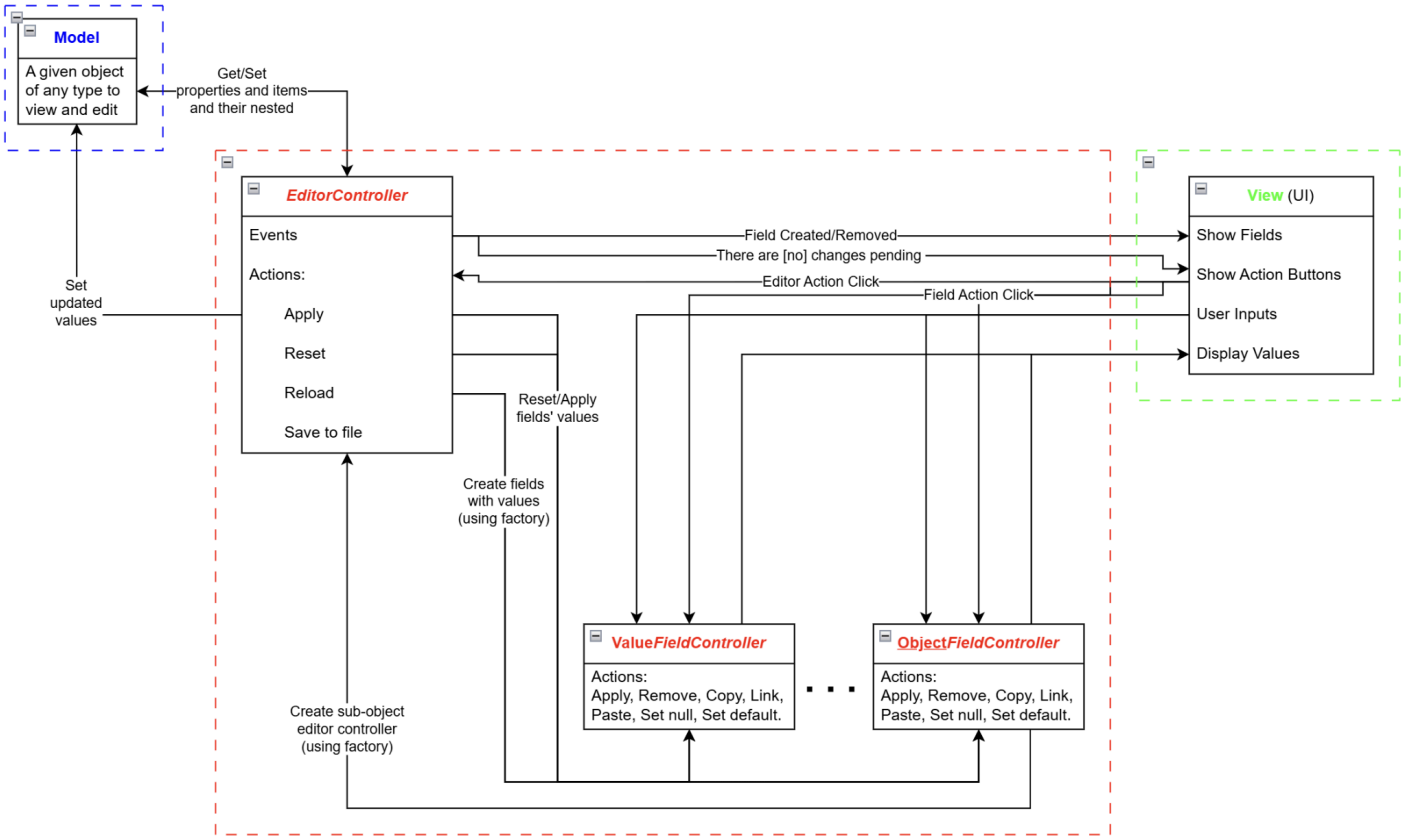
- Editing primitive properties.
- Navigating and editing nested structures.
- Handling collections and dictionaries.

7. Diagrams

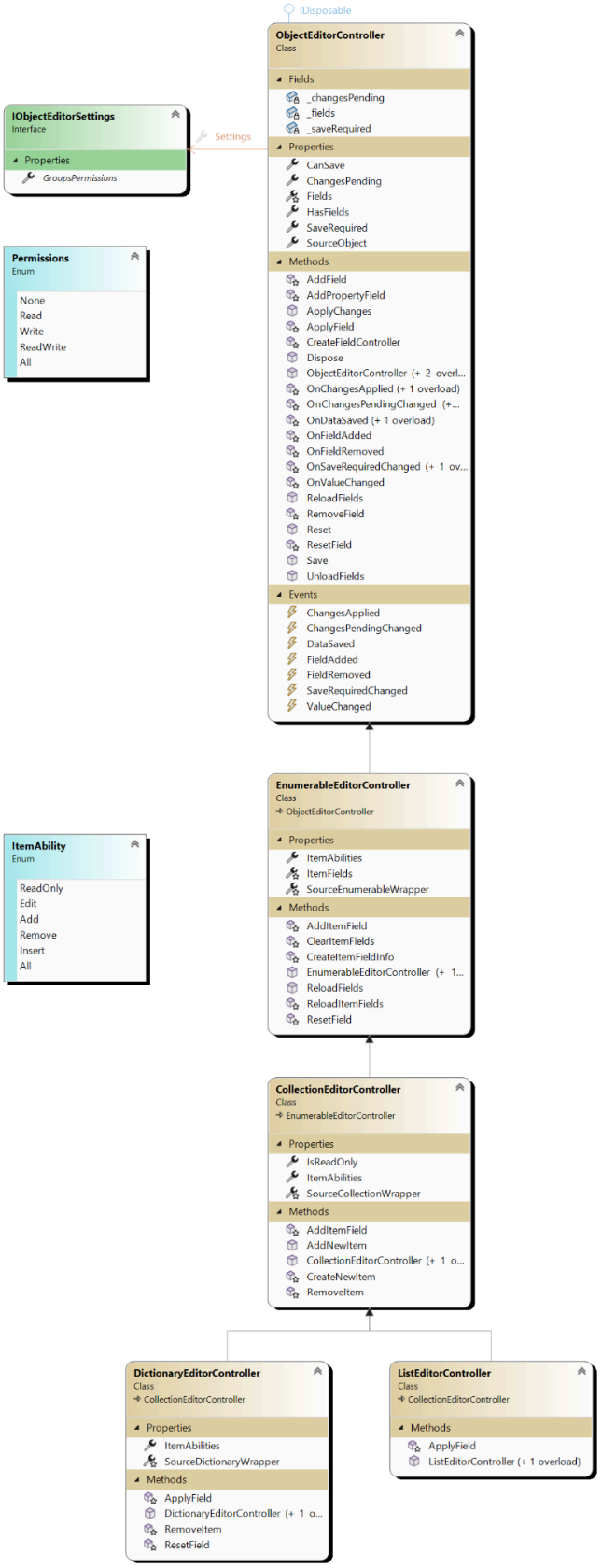
7.1 Class Factories



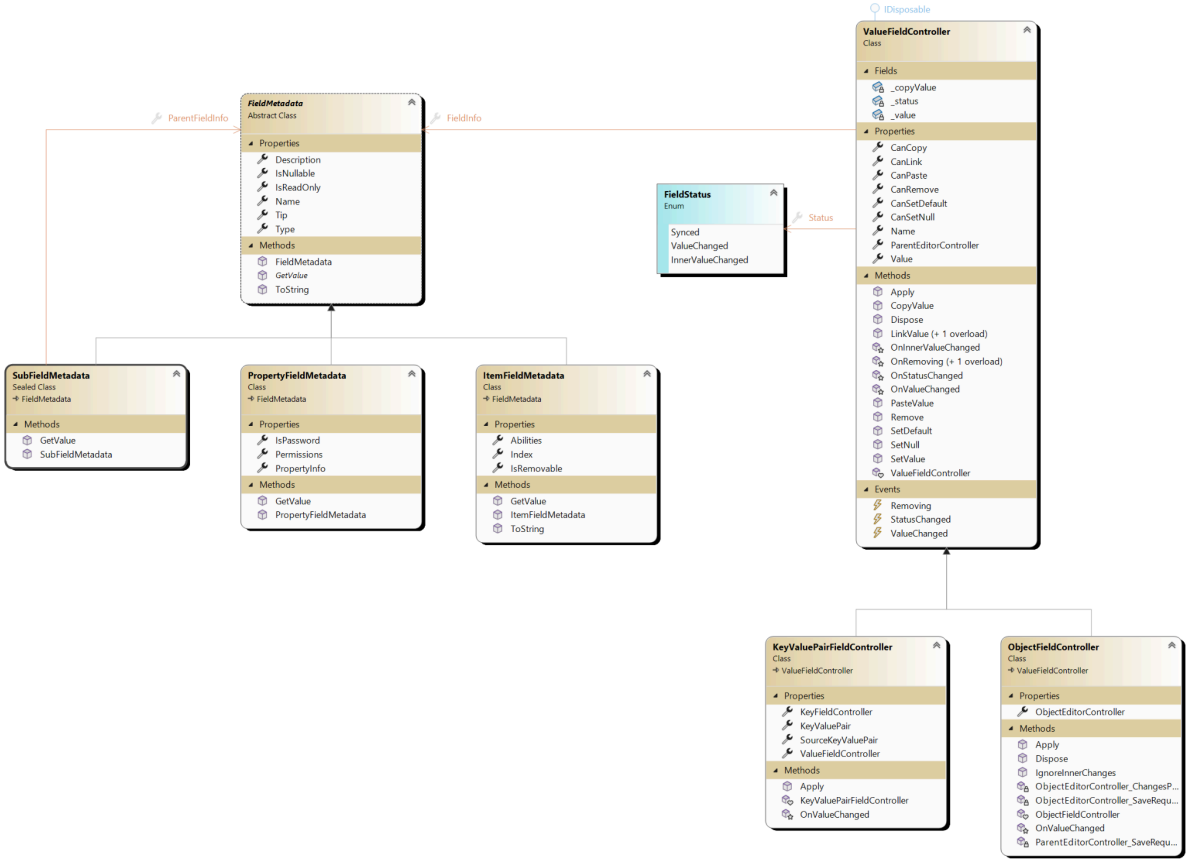
7.2 Components Interaction



7.3 Object Editor Controller

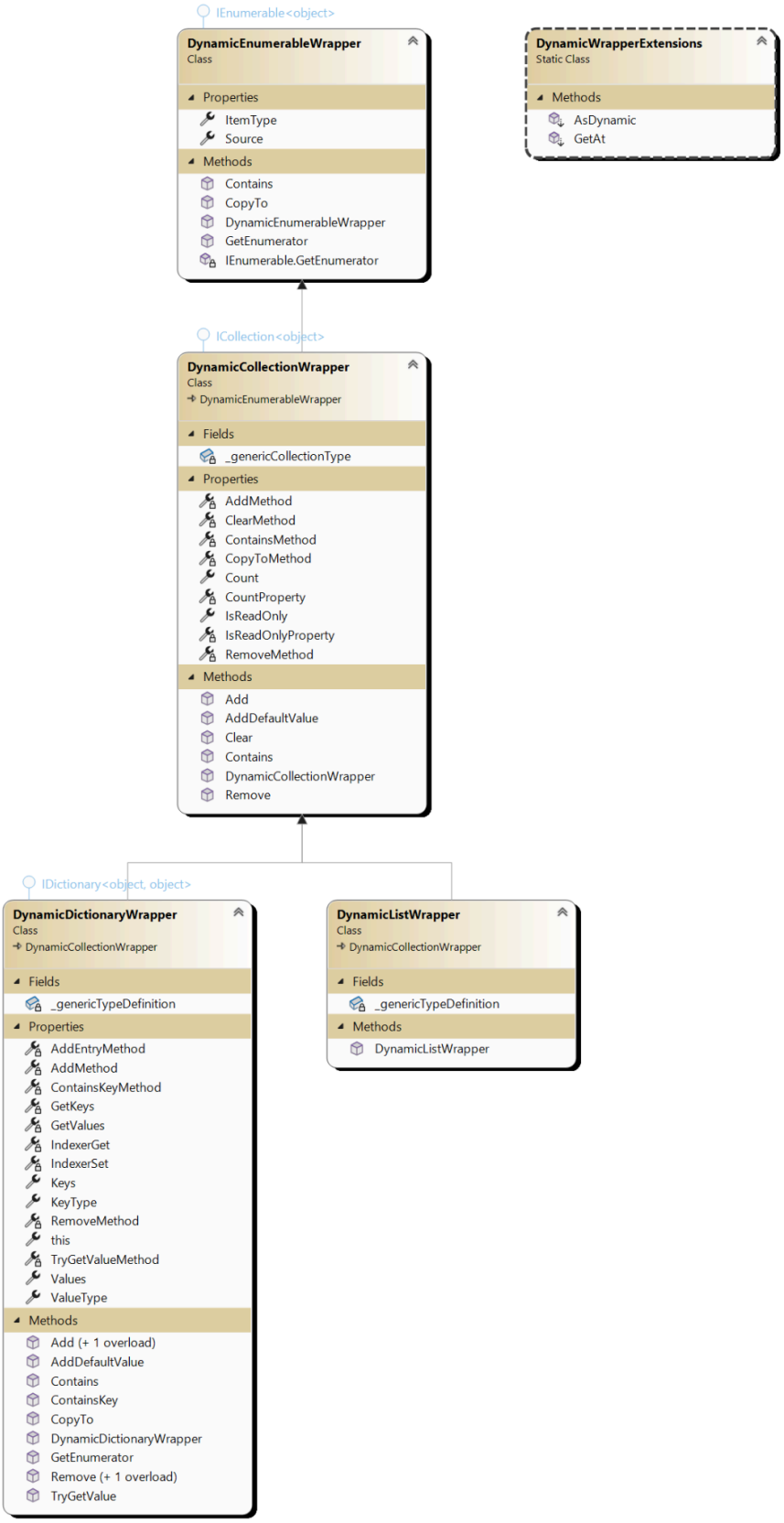


7.4 Field Controller and Metadata

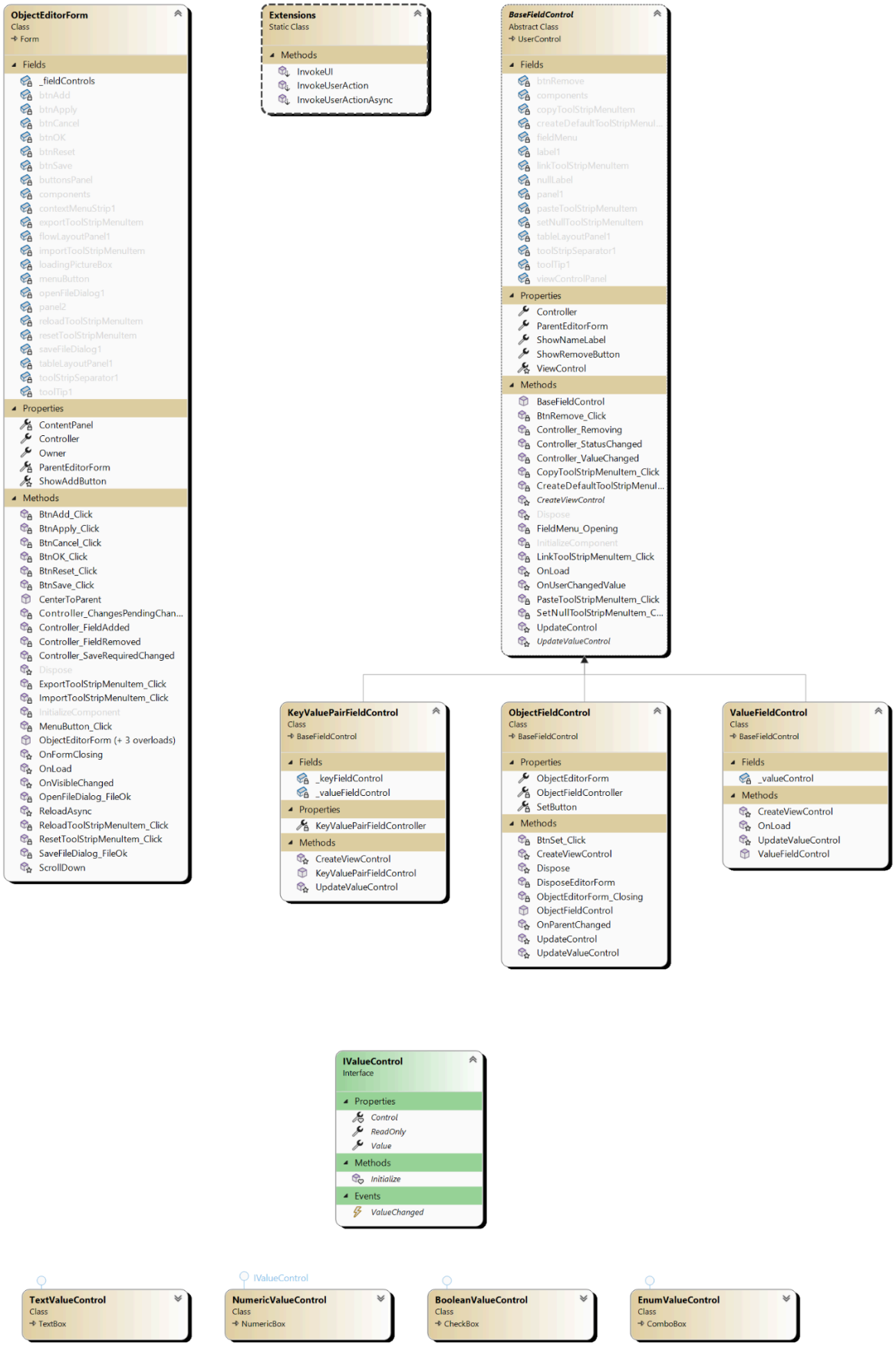


7.5 Event Arguments

7.6 Dynamic Enumerable Wrapper



7.7 View (UI) Class Diagram



---

8. Coding Standards and Naming Conventions

8.1 File and Directory Organization

- Files are grouped by functionality (e.g., Controllers, Extensions).
- Namespaces are set according to the directory organization.

8.2 Naming Rules

- **Classes, Methods, Functions, and Properties** are named in `PascalCase`.
- **Private variable members** are named in `camelCase` and begin with an underscore (`_`), e.g., `_privateClassVar`.
- **Local variables** are named in `camelCase`, e.g., `localFuncVar`.

---

9. Testing

9.1 Unit Test Coverage

Includes tests for controllers, utilities, and extensions.

9.2 Example Test Scenarios

- Verify dynamic control creation for various types.
- Test recursive editor initialization.

---

10. Future Extensions

- Support for exporting/importing JSON and XML.
- Enhanced permissions system.
- Web-based editor integration.

---

11. Conclusion

ObjectEditor.NET combines reflection, MVC principles, and dynamic GUI generation to create a flexible and powerful object editing tool. Its modular architecture ensures adaptability for future use cases and extensions.
