Advanced ASP.NET MVC 2

Brad Wilson
http://bradwilson.typepad.com/
bradwils@microsoft.com

Topics

- Moving beyond GET and POST
- Dynamic scaffolding (“templates”) and customization
- Model metadata and custom providers
- Server-side validation and custom providers
- Client-side validation and JSON
- Asynchronous controller actions

Moving beyond GET and POST

- Want to support both APIs and browser w/ RESTful URLs
- REST support implies at least PUT and DELETE methods
- HTML forms only support GET and POST methods
- The solution? *POST + hidden input + updated [AcceptVerbs]*

```
<%= Html.HttpMethodOverride(HttpVerbs.Delete) %>
```

Generates:

```
<input name="X-HTTP-Method-Override"
       type="hidden"
       value="DELETE" />
```

Demo

Supporting REST-like URLs

Dynamic Scaffolding

```
<% using (Html.BeginForm()) { %>
    <%= Html.ValidationSummary(true, "Login was unsuccessful. Please correct the errors
    <fieldset>
        <legend>Account Information</legend>
        <%= Html.EditorForModel() %>
        <p><input type="submit" value="Log On" /></p>
    </fieldset>
<% } %>
```

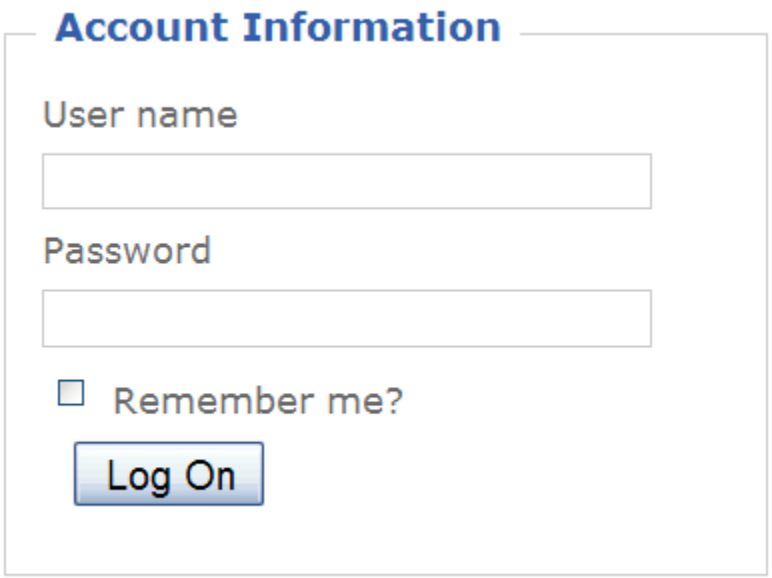
+

```
public class LogOnModel
{
    [Required]
    [DisplayName("User name")]
    public string UserName { get; set; }

    [Required]
    [DataType(DataType.Password)]
    [DisplayName("Password")]
    public string Password { get; set; }

    [DisplayName("Remember me?")]
    public bool RememberMe { get; set; }
}
```

=



Default Editor Templates

- `Boolean` – check-box or 3-state drop-down
- `Collection` – iterate and show editor (with name style "[n]")
- `Decimal` – text box (with formatting)
- `HiddenInput` – hidden input
- `MultilineText` – text area
- `Object` – edit all simple properties
- `Password` – text box (password)
- `String`, `Text` – text box

No deep diving by default
Default templates (as .ascx files) are in MvcFutures

Default Display Templates

- `Boolean` – disabled checkbox or 3-state drop-down
- `Collection` – iterate and display
- `Decimal` – encoded text (with formatting)
- `EmailAddress` – auto-linked (with mailto:)
- `HiddenInput` – encoded text (influenced by HideSurroundingHtml)
- `Html` – unencoded text
- `Object` – show all simple properties
- `String`, `Text` – encoded text
- `Url` – auto-linked

No deep diving by default
Default templates (as .ascx files) are in MvcFutures

Template Selection Locations

Display templates:

~/Areas/*AreaName*/Views/*ControllerName*/DisplayTemplates/

~/Areas/*AreaName*/Views/Shared/DisplayTemplates/

~/Views/*ControllerName*/DisplayTemplates/

~/Views/Shared/DisplayTemplates/

Editor templates:

~/Areas/*AreaName*/Views/*ControllerName*/EditorTemplates/

~/Areas/*AreaName*/Views/Shared/EditorTemplates/

~/Views/*ControllerName*/EditorTemplates/

~/Views/Shared/EditorTemplates/

Template Selection Algorithm

1. ModelMetadata.TemplateHint
2. ModelMetadata.DataTypeName
3. Concrete type name (stripping nullable)
4. Simple type? …
   a. `String`
5. … or interface? …
   a. `Collection` (if it implements IEnumerable)
   b. `Object`
6. … or complex type?
   a. Iterate over all base concrete types (stopping before Object)
   b. `Collection` (if it implements IEnumerable)
   c. `Object`

Demo

Default Templates

jQuery Date Picker Custom Template

Model Metadata

- Describes information used by the templates and validators
  - Template name
  - Display name (long and short)
  - Data formats
  - Visibility (display and/or edit)
  - Null display text
  - Watermark, etc.
- Models can be stand-alone objects or properties of objects
- “Simple” model == convertible to/from a string
- String- and lambda-based lookup
- Single provider system (default is DataAnnotations attributes)

Model Metadata Providers

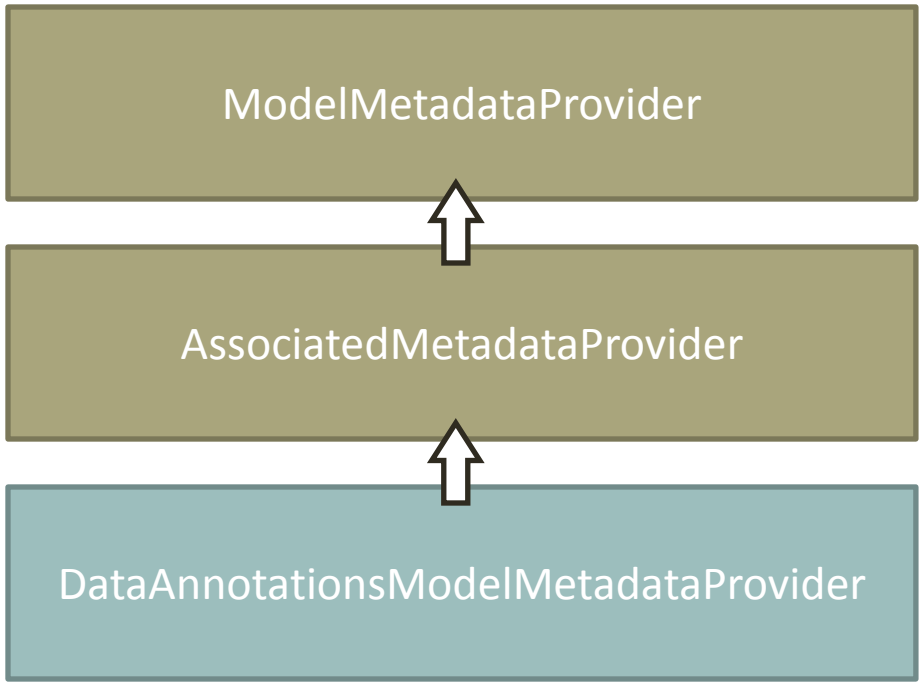
ModelMetadataProvider has three methods:

```
abstract ModelMetadata
    GetMetadataForType
        (Func<object> modelAccessor,
         Type         modelType)

abstract ModelMetadata
    GetMetadataForProperty
        (Func<object> modelAccessor,
         Type         containerType,
         string       propertyName)

abstract IEnumerable<ModelMetadata>
    GetMetadataForProperties(
        (object container,
         Type   containerType)
```

In-Box Metadata Providers



Reflection- and attribute-based, includes “buddy class” support

Attributes from DataAnnotations

Key: Abstract Concrete

Validation

- Server-side validators
  - Run validation code on the server
  - Provide information on how to write up client-side validation
  - Specific to the server-side validation technology
  - Agnostic to the client-side validation technology
- Client-side validators
  - Wire up based on validation rules from the server
  - Specific to the client-side validation technology
  - Agnostic to the server-side validation technology
- JSON is the glue
- Multiple provider architecture

Server-Side Architecture

- Automatically performed during model binding
  - Top-level complex objects only
  - Set all values (bind) before any validation is run
  - Full model validation, not just input properties
- Depth-first validation
  - Model binding errors first (wrong type, non-nullable)
  - Skip property if it already has a model binding error
  - Skip container if any of its properties are invalid
- “IsRequired” validators are special
  - Setters which throw are a problem (like EF)
  - If value is null, run “IsRequired” validators before we set it

Client-Side Architecture

- Written against ASP.NET AJAX 4
- Built-in validators
  - `number`, `required`, `stringLength`, `range`, `regularExpression`
- Supported events
  - `change`, `blur`, `submit`
- FormContext
  - `formElement`, `fields`, `validationSummaryElement`, `replaceValidationSummary`, `addError(msg)`, `addErrors(msgs)`, `clearErrors()`, `validate(eventName)`
- FieldContext
  - `elements`, `validations`, `formContext`, `addError(msg)`, `addErrors(msgs)`, `clearErrors()`, `validate(eventName)`

Writing a Client-Side Validator

```
Sys.Mvc.ValidatorRegistry.validators.validatorName =
function(rule) {

  // Do any one-time work against the rule

  return function(value, context) {

    // NOOP on null or empty values
    if (!value || !value.length) { return true; }

    // Do any validation work here
    // Return true is valid, false if invalid
    // Return a string for a custom error message

  }

}
```

Validation Providers

ModelValidatorProvider has one method:

```
abstract IEnumerable<ModelValidator>
    GetValidators
        (ModelMetadata     metadata,
         ControllerContext context)
```

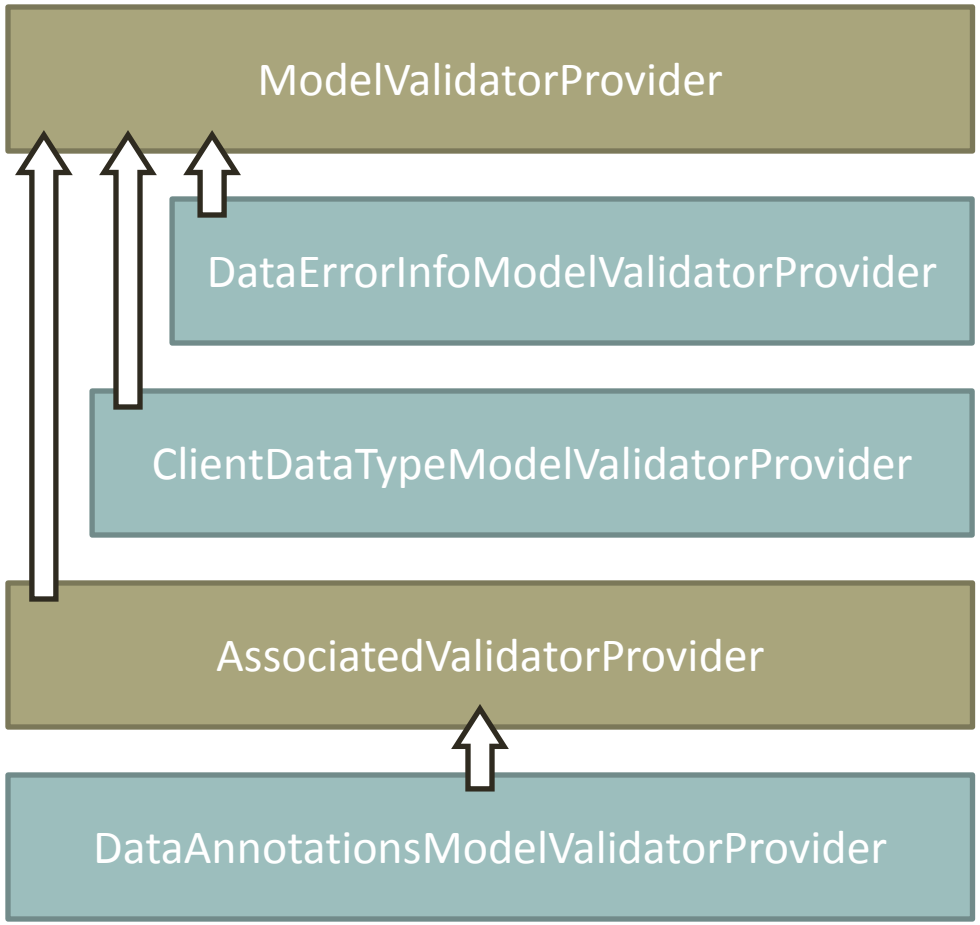
ModelValidator has two methods and one property:

```
abstract IEnumerable<ModelValidationResult>
    Validate(object container)

virtual IEnumerable<ModelClientValidationRule>
    GetClientValidationRules()

virtual bool IsRequired { get; }
```

In-Box Validation Providers



Supports IDataErrorInfo

Client-side only validators for numeric types

Reflection- and attribute-based, includes “buddy class” support

Attributes which derive from ValidationAttribute

Key: Abstract Concrete

Demo

Custom Server- and Client-Side Validator

See Also

http://bradwilson.typepad.com/blog/2009/10/enterprise-library-validation-example-for-aspnet-mvc-2.html

(also http://bit.ly/HpNRT)

Asynchronous Actions

- Derive from AsyncController instead of Controller

```
public void ActionNameAsync(...)
public ActionResult ActionNameCompleted(...)
```

- AsyncManager object
  - `Finished` – event for when action is finished
  - `OutstandingOperations` – counter of operations in progress
  - `Parameters` – output parameters to Complete method
  - `Timeout` – overall timeout for async action

Demo

Async Twitter Search Results

Q&A
and Thank You!

http://bradwilson.typepad.com/
bradwils@microsoft.com