Microsoft® tech·ed North America | 2010

JUNE 7-10, 2010 | NEW ORLEANS, LA

Microsoft®

SESSION CODE: WEB02-INT

Extending Microsoft ASP.NET MVC 2

Brad Wilson
Senior Software Developer
Microsoft Corporation

Action Filter

Scenarios

- Intercept actions and perform pre- and post-processing
- *Easier to test and apply orthogonal concerns*

How To

- Base class: `FilterAttribute`, plus:
  - Interface: `IActionFilter`
  - Interface: `IResultFilter`
  - Interface: `IAuthorizationFilter`
  - Interface: `IExceptionFilter`
- Base class: `ActionFilterAttribute`
  - Implements `IActionFilter` and `IResultFilter`

Action Timing Filter

Demo

Action Result

Scenarios

- Convert data into response (new rendering types)
- Perform server-side activities (redirect, return HTTP errors, etc.)
- *Helps facilitate better testing (separation of concerns)*

How To

- Base class: `ActionResult`
  - Implement `ExecuteResult`
- *Optional:* Helper method(s) to make it easier to use
  - Controller extension method
  - New base class for controllers with helpers methods in it

Serialize an object into XML

Demo

Controller Factory

Scenario

- Dependency injection containers

How To

- Interface: `IControllerFactory`
  - Create controller from `RequestContext` + controller name
  - Base class: `DefaultControllerFactory`
    - Create controller from `RequestContext` + controller `Type`
  - Register with `ControllerBuilder`

Unity 2 Controller Factory

http://bit.ly/unity2

Demo

HTTP Encoder *(requires .NET 4)*

Scenario

- Customize encoding of HTML, URLs, and attributes

How To

- Base class: `HttpEncoder`
  - HTML encode/decode
  - Attribute encode/decode
  - URL path encode
  - URL query string encode
- Register with `HttpEncoder.Current` or in Web.config

Microsoft Anti-XSS Library

http://bit.ly/antixss

Demo

Custom Validation

Scenarios

- Complex, composite, or business-specific validation
- *Client-side validation for better usability*

How To

- Base class: `ValidationAttribute`
  - Implement `IsValid`
- Base class: `DataAnnotationsModelValidator`
  - Register with `DataAnnotationsModelValidatorProvider.RegisterAdapter`
- Write client-side validation logic in JavaScript
  - Respond to events: `input`, `blur`, `submit`

Price Validator

Demo

View Engine

Scenarios

- New view rendering systems
- Customize existing view systems with app-specific rules

How To

- Interface: `IView`
- Interface: `IViewEngine`
  - Base class: `VirtualPathProviderViewEngine`
  - Register with `ViewEngines.Engines`
  - Multiple view engines; first one to say “yes” wins

Custom View Engine

Demo

Validator Provider

Scenario

- Provide and/or enhance validation rules

How To

- Base class: `ModelValidatorProvider`
  - Return validators for `Type`
  - Base class: `AssociatedValidatorProvider`
    - Return validators for `Type` + `IEnumerable<Attribute>`
  - Register with `ModelValidatorProviders.Providers`
    - Multiple validator providers
    - Validators run during model binding

Fluent Validator Provider

Demo

Metadata Provider

Scenario

- Provide and/or enhance model metadata
- *Consumed by HTML template helpers and validation*

How To

- Base class: `ModelMetadataProvider`
  - Return metadata for: `Type`, property of `Type`, all properties of `Type`
  - Base class: `AssociatedMetadataProvider`
    - Return metadata for: model `Type` + model value + container `Type` + property name + `IEnumerable<Attribute>`
  - Register with `ModelMetadataProviders.Current`
    - Single metadata provider

Fluent Metadata Provider

Demo

Model Binder

Scenarios

- Turn request values (form, query string, etc.) into objects
- *Only use when the default binder can't make your object properly*

How To

- Interface: `IModelBinder`
  - Base class: `DefaultModelBinder`
  - Register with `ModelBinders.Binders` (type-specific and default)
  - Register with `[ModelBinder]` attribute on class/property
- Responsible for running validation after binding

Immutable Object Model Binder

Demo

Value Provider

Scenario

- Provide new sources of data for model binders

How To

- Interface: `IValueProvider`
  - Base class: `DictionaryValueProvider<TValue>`
  - Base class: `NameValueCollectionValueProvider`
- Base class: `ValueProviderFactory`
  - Register with `ValueProviderFactories.Factories`

TempData Value Provider

http://bit.ly/mvc2futures

Demo

TempData provider

Scenario

- Change the storage location of `TempData`

How To

- `TempData` lives through the *end of the request in which its read*
  - Change from MVC 1.0 (where it lived for a single request only)
  - The MVC infrastructure takes care of lifetime management
- Interface: `ITempDataProvider`
  - Load and save `ControllerContext` + `IDictionary<String,Object>`
  - Creation owned by the controller (can make your own controller class and override `CreateTempDataProvider`, or set the `TempDataProvider` property on the controller; controller factory injection possibilities)

Cookie TempData Provider

http://bit.ly/mvc2futures

Demo

Routing

Scenario

- Customized route registration
- Customized routing constraints

How To

- Extension method on `RouteCollection`
  - MVC's `MapRoute` method is an extension method, too!
- Interface: `IRouteConstraint`
  - Register during route creation

MVC 2-aware HTTP method constraint

Demo

Resources

MVC information	http://www.asp.net/mvc
MVC news & source	http://aspnet.codeplex.com
MVC forums	http://bit.ly/mvcforums
Web Platform Installer	http://bit.ly/webpi
MVC 2 Futures	http://bit.ly/mvc2futures
Anti-XSS Library	http://bit.ly/antixss
P&P Unity 2.0	http://bit.ly/unity2
Today’s code & slides	http://bit.ly/bradstalks

Microsoft®