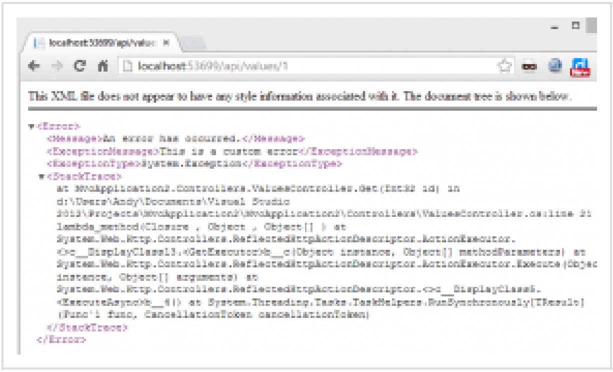
« Database driven Asp.net MVC Model Binding, Validation, and Metadata

New address in the cloud »

Web API, HttpError and the behavior of Exceptions – 'An error has occurred'

9 Feb 2013 | Web API
Tags: asp.net · exception · httperror · web API · WebAPI



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<Error>
   <Message>An error has occurred.</Message>
   <ExceptionMessage>This is a custom error</ExceptionMessage>
   <ExceptionType>System.Exception</ExceptionType>
 ▼<StackTrace>
    at MvcApplication2.Controllers.ValuesController.Get(Int32 id) in
    d:\Users\Andy\Documents\Visual Studio
    2012\Projects\MvcApplication2\MvcApplication2\Controllers\ValuesController.cs:line 21
    lambda_method(Closure , Object , Object[] ) at
    System.Web.Http.Controllers.ReflectedHttpActionDescriptor.ActionExecutor.
    <>c__DisplayClass13.<GetExecutor>b__c(Object instance, Object[] methodParameters) at
    System.Web.Http.Controllers.ReflectedHttpActionDescriptor.ActionExecutor.Execute(Objec
    instance, Object[] arguments) at
    System.Web.Http.Controllers.ReflectedHttpActionDescriptor.<>c__DisplayClass5.
    <ExecuteAsync>b__4() at System.Threading.Tasks.TaskHelpers.RunSynchronously[TResult]
    (Func`1 func, CancellationToken cancellationToken)
  </StackTrace>
</Error>
```

When you deploy an ASP.Net Web Api project to a server in RELEASE mode configuration and you have Custom Errors set to On, you'll likely notice that your once nicely formatted error responses are no longer so friendly.

During local development web api errors are formatted nicely with messages, stack trace, etc.

(XML)

<Error><Message>An error has occurred.</Message>
<ExceptionMessage>The method or operation is not implemented.
</ExceptionMessage>
<ExceptionType>System.NotImplementedException</ExceptionTyp
e><StackTrace> at

Recent Posts

- Using the C# generic cache: the Tester Doer pattern
- GooPh – a Google Voice client for Windows Phone 8
- New address in the cloud
- Web API, HttpError and the behavior of Exceptions – 'An error has occurred'
- Database driven Asp.net MVC Model Binding, Validation, and Metadata

Recent Comments

- Simple generic C# caching class (I use this all the time) ← Code is Hard on Using the C# generic cache: the Tester Doer pattern
- Using the C# generic cache: the Tester Doer pattern ← Code is Hard on Simple generic C# caching class (I use this all the time)
- Andy Cohen on Simple generic C# caching class (I use this all the time)
- akismet-889bf04a613f9c68792075d37a6f356eni on Simple generic C# caching class (I use this all the

AppCenter.Web.Controllers.ApplicantsController.&lt;Post&gt;d__a.MoveNext() in e:\Workspaces\AppCenter\DEV\Source\AppCenter\AppCenter.Web\Controllers\ApplicantsController.cs:line 86</StackTrace></Error>

(JSON)

```
{"Message":"An error has
occurred.","ExceptionMessage":"The method or
operation is not
implemented.","ExceptionType":"System.NotImplement
edException","StackTrace":" at
AppCenter.Web.Controllers.ApplicantsController.d__
a.MoveNext() in
e:\\Workspaces\\AppCenter\\DEV\\Source\\AppCenter\
\AppCenter.Web\\Controllers\\ApplicantsController.
cs:line 86"}
```

And here is the same thing when deployed:

(XML)

<Error><Message>An error has occurred.</Message></Error>

(JSON)

```
{"Message":"An error has occurred."}
```

As you can see, if you want your errors to flow to the consuming app, this is not ideal. You likely will (and should) want to return your errors in an object that has a friendly error message, and optionally, detailed message, error code, and even an error reference for lookup.

Here is an excerpt from the Apigee e-book "Web API Design – Crafting Interfaces that Developers Love":

> How to think about errors in a pragmatic way with REST? Let's take a look at how three top APIs approach it.
> Facebook
> HTTP Status Code: 200
> {"type" : "OauthException", "message":"(#803) Some of the aliases you requested do not exist: foo.bar"}
> Twilio
> HTTP Status Code: 401
> {"status" : "401", "message":"Authenticate","code": 20003, "more info": "http://www.twilio.com/docs/errors/20003"}
> SimpleGeo
> HTTP Status Code: 401
> {"code" : 401, "message": "Authentication Required"}

I like these patterns, but I especially like the following format:

```
{"developerMessage" : "Verbose, plain language
description of the problem for the app developer
```

time)

- Some Super Talented and Inspiring People | Software Blog on Await, Async, Mvc and Impersonation

Tags

ajax asp.net Async asynchronous await Azure binding type C# cache caching Cloud Editor Templates exception generic Google Voice HTML5 httperror impersonation JavaScript jQuery jQuery plugin jQueryUI lambda MEF model binding model state modelstate model type MVC plugin task TPL unobtrusive Validation WCF web API WebAPI Windows Phone Windows Phone 8

Archives

- May 2013
- April 2013
- March 2013
- February 2013
- November 2012
- September 2012
- April 2012
- February 2012
- January 2012
- December 2011
- November 2010

```
with hints about how to fix it.",
"userMessage":"Pass this message on to the app
user if needed.", "errorCode" : 12345, "more
info": "http://dev.teachdogrest.com/errors/12345"}
```

When dealing with web api and exceptions, there are a few things that you must realize:

ALL errors eventually are serialized into an HttpError object.

* manually thrown exceptions
* uncaught exceptions
* responses created using the Request.CreateErrorResponse extension method

HttpResponseException's are treated as "caught" or handled errors

That means that when you manually throw an HttpResponseException OR you use Request.CreateErrorResponse – the errors will not flow to any ExceptionFilterAttributes you may have created. That means, if you use a library like Elmah to handle your Exception reporting, these will NOT be reported.

The Ideal Developer Experience for Exceptions and Errors (at least this is my ideal)

I want to be able to consistently report my exceptions in a consistent and *friendly* format.
I don't want to have to worry about the different overloads of Request.CreateErrorResponse.
I want to be able to configure the way exceptions are dealt with. I don't want developers on my projects to have to worry about getting creative with their exception handling and reporting. I don't want it done one way here, one way there, etc.

My Solution

I created an ExceptionFilterAttribute that allows me to configure all my exceptions in one central place a static class in the App_Start folder (this is the preferred method these days it seems).

Here is my code:

```
public class MvcApplication : System.Web.HttpApplication
{

    protected void Application_Start()
    {
        AreaRegistration.RegisterAllAreas();
        WebApiConfig.Register(GlobalConfiguration.Configuration);
        WebApiExceptionConfig.RegisterExceptions(GlobalConfiguration.Configuration);
        FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
        RouteConfig.RegisterRoutes(RouteTable.Routes);
        BundleConfig.RegisterBundles(BundleTable.Bundles);
    }
}
```

Categories

› Async
› Azure
› C#
› HTML5
› JavaScript
› MEF
› Mobile
› MVC
› OData
› REST
› Uncategorized
› Web API
› Windows Phone 7
› Windows Phone 8

GitHub

omegaluz @ Github

Show my repositories

Me

Google+

@iamandycohen

LinkedIn

```
using System;
using System.Collections.Generic;
using System.Net;
using System.Web.Http;
using System.Web.Security;

  public static class WebApiExceptionConfig
  {
    public static void RegisterExceptions(HttpConfiguration config)
    {

      /* /// this is the easiest way to shove the exception messages into the httperror message property for ALL unhandled exceptions

      config.Filters.Add(new GlobalApiExceptionFilterAttribute(catchUnfilteredExceptions: true));

      */

      config.Filters.Add(new GlobalApiExceptionFilterAttribute(new List<GlobalApiExceptionDefinition>
      {

        /* /// Example 1 -- setting the error code and reference properties
        new GlobalApiExceptionDefinition(typeof(NotImplementedException)) { ErrorCode = "123456.cows", ErrorReference = "http://www.google.com?q=cows" },
        */

        /* /// Example 2 -- using the friendly message string overload
        new GlobalApiExceptionDefinition(typeof(NotImplementedException), "This method is really wonky", HttpStatusCode.NotAcceptable) { ErrorCode = "123456.cows", ErrorReference = "http://www.google.com?q=cows" },
        */

        /* /// Example 3 -- using the friendly message predicate overload
        new GlobalApiExceptionDefinition(typeof(MembershipCreateUserException), (ex) => MembershipHelper.MembershipCreateStatusToString((ex as MembershipCreateUserException).StatusCode), HttpStatusCode.Conflict)
        */

        new GlobalApiExceptionDefinition(typeof(MembershipCreateUserException))
        {
          Handle = (ex) => // we want to make sure the server error status codes are respected - we want to send back a 500
          {
            if (ex is MembershipCreateUserException)
            {
              var mex = ex as MembershipCreateUserException;
              switch
(mex.StatusCode)
                            {
                                case MembershipCreateStatus.DuplicateProviderUserKey:
                                case MembershipCreateStatus.InvalidProviderUserKey:
                                case MembershipCreateStatus.ProviderError:
                                    return true;
                                default:
                                        break;
                            }
                        }
                        return false;
                    }
                },
                new GlobalApiExceptionDefinition(typeof(MembershipCreateUserException), statusCode: HttpStatusCode.Conflict) // this will send back a 409, for all other types of membership create user exceptions
            }, catchUnfilteredExceptions: true));
        }
    }
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Web.Http;
using System.Web.Http.Filters;


    public class GlobalApiExceptionFilterAttribute : ExceptionFilterAttribute
    {

        const string ERROR_CODE_KEY = "ErrorCode";
        const string ERROR_REFERENCE_KEY = "ErrorReference";

        List<GlobalApiExceptionDefinition> exceptionHandlers;
        bool catchUnfilteredExceptions;

        public GlobalApiExceptionFilterAttribute(
            List<GlobalApiExceptionDefinition> exceptionHandlers = null, bool catchUnfilteredExceptions = false)
        {
            this.exceptionHandlers = exceptionHandlers ?? new List<GlobalApiExceptionDefinition>();
            this.catchUnfilteredExceptions = catchUnfilteredExceptions;
        }

        public override void OnException(HttpActionExecutedContext actionExecutedContext)
        {
            var exception = actionExecutedContext.Exception;
            GlobalApiExceptionDefinition globalExceptionDefinition = null;
```

```
      HttpStatusCode statusCode = HttpStatusCode.InternalServerError;

      if (LookupException(actionExecutedContext.Exception, out globalExceptionDefinition) || catchUnfilteredExceptions)
      {
        // set the friendly message
        string friendlyMessage = globalExceptionDefinition != null ? globalExceptionDefinition.FriendlyMessage(exception) : exception.Message;

        // create the friendly http error
        var friendlyHttpError = new HttpError(friendlyMessage);

        // if we found a globalExceptionDefinition then set properties of our friendly httpError object accordingly
        if (globalExceptionDefinition != null)
        {

          // set the status code
          statusCode = globalExceptionDefinition.StatusCode;

          // add optional error code
          if (!string.IsNullOrEmpty(globalExceptionDefinition.ErrorCode))
          {
            friendlyHttpError[ERROR_CODE_KEY] = globalExceptionDefinition.ErrorCode;
          }

          // add optional error reference
          if (!string.IsNullOrEmpty(globalExceptionDefinition.ErrorReference))
          {
            friendlyHttpError[ERROR_REFERENCE_KEY] = globalExceptionDefinition.ErrorReference;
          }

        }

        // set the response to our friendly http error
        actionExecutedContext.Response = actionExecutedContext.Request.CreateErrorResponse(statusCode, friendlyHttpError);

      }

      // flow through to the base
      base.OnException(actionExecutedContext);
    }

    private bool LookupException(Exception exception, out GlobalApiExceptionDefinition exceptionMatch)
    {
      exceptionMatch = null;

      var possibleMatches =
exceptionHandlers.Where(e => e.ExceptionType == exception.GetType());
      foreach (var possibleMatch in possibleMatches)
      {
        if (possibleMatch.Handle == null || possibleMatch.Handle(exception))
        {
          exceptionMatch = possibleMatch;

          return true;
        }
      }

      return false;
    }

  }

  public class GlobalApiExceptionDefinition
  {

    const string ARGUMENT_NULL_EXCEPTION_FMT = "Argument '{0}' cannot be null.";
    const string ARGUMENT_MUST_INHERIT_FROM_FMT = "Type must inherit from {0}.";

    public Type ExceptionType { get; private set; }
    public Func<Exception, string> FriendlyMessage { get; private set; }

    public Func<Exception, bool> Handle { get; set; }
    public HttpStatusCode StatusCode { get; set; }

    public string ErrorCode { get; set; }
    public string ErrorReference { get; set; }

    public GlobalApiExceptionDefinition(Type exceptionType, string friendlyMessage = null, HttpStatusCode statusCode = HttpStatusCode.InternalServerError) :
      this(exceptionType, (ex) => friendlyMessage ?? ex.Message, statusCode) { }

    public GlobalApiExceptionDefinition(Type exceptionType, Func<Exception, string> friendlyMessage, HttpStatusCode statusCode = HttpStatusCode.InternalServerError)
    {

      AssertParameterIsNotNull(friendlyMessage, "friendlyMessage");
      AssertParameterIsNotNull(exceptionType, "exceptionType");
      AssertParameterInheritsFrom(exceptionType, typeof(Exception), "exceptionType");

      ExceptionType = exceptionType;
      FriendlyMessage = friendlyMessage;
      StatusCode = statusCode;
    }
```

```
        #region "Argument Assertions"

        private static void AssertParameterInheritsFrom(Type type, Type inheritedType, string name)
        {
            if (!type.IsSubclassOf(inheritedType))
            {
                throw new ArgumentException(string.Format(ARGUMENT_MUST_INHERIT_FROM_FMT, inheritedType.Name), name);
            }
        }

        private static void AssertParameterIsNotNull(object parameter, string name)
        {
            if (parameter == null)
            {
                throw new ArgumentNullException(name, string.Format(ARGUMENT_NULL_EXCEPTION_FMT, name));
            }
        }

        #endregion

    }
```

You can download the source here.

Leave a Reply

Enter your comment here...

« Database driven Asp.net MVC Model Binding, Validation, and Metadata

New address in the cloud »