Method 1:- Simple way

The simplest way is to use the traditional .NET exception handling style i.e. try and catch block. Now when exception happens catch block gets executed and it redirects to the error view.

But if we use this method then we will not be utilizing MVC exception mechanism properly and completely. In the further sections we will discuss five important ways by which we can utilize MVC provided features for exception handling.

public ActionResult SomeError()

{

try

{

//código ejecutable

}

catch(Exception ex)

{

return View("Error");}

}

Method 2:- Override “OnException” method

In this method we can override the “OnException” event of the controller and set the “Result” to the view name. This view gets invoked when error occurs in this controller. In the below code you can see we have set the “Result” to a view named as “Error”.

We have also set the exception so that it can be displayed inside the view.

public class HomeController : Controller

{

protected override void OnException(ExceptionContext filterContext)

{

Exception ex = filterContext.Exception;

filterContext.ExceptionHandled = true;

var model = new HandleErrorInfo(filterContext.Exception, "Controller","Action");

filterContext.Result = new ViewResult()

{

ViewName = "Error",

ViewData = new ViewDataDictionary(model)

};

}

}

To display the above error in view we can use the below code:-

@Model.Exception;

The problem with this approach is we cannot reuse the error handling logic across multiple controllers.

Method 3:- Using “HandleError” Attribute

The other way of handling error is my using “HandleError” attribute. Implementing “HandleError” attribute is a two-step process:-

Step 1 :- We need to first decorate the action method with “HandleError” attribute as shown in the below code.

public class HomeController : Controller

{

**[HandleError()]**

public ActionResult SomeError()

{

throw new Exception("test");

}

}

Step 2:- In the “Web.config” file you need to add the “customErrors” tag and point to the “Error” view as shown in the below “Web.config” code snippet.

<system.web>

<customErrors defaultRedirect="Error.cshtml" mode="On">

</customErrors>

</system.web>

In case you want different error views for different exception types you can decorate action method with multiple “HandleError” attribute point to multiple views as per exception types.

public class HomeController : Controller

{

[HandleError(ExceptionType=typeof(ArithmeticException),View="Arthimetic")]

[HandleError(ExceptionType = typeof(NotImplementedException),View ="Error1")]

public ActionResult SomeError()

{

}

}

Method 4:- Inheriting from “HandleErrorAttribute” (Buena práctica)

One of the biggest drawbacks of all the previous method was reusability. Error handling logic cannot be reused across other controllers.

In order to reuse error handling logic across controller we can inherit from “HandleErrorAttribute”class anddecorate this class as attribute across controller.

public class Err : HandleErrorAttribute

{

public override void OnException(ExceptionContext filterContext)

{

Exception ex = filterContext.Exception;

filterContext.ExceptionHandled = true;

var model = new HandleErrorInfo(filterContext.Exception, "Controller", "Action");

filterContext.Result = new ViewResult()

{

ViewName = "Error1",

ViewData = new ViewDataDictionary(model)

};

}

}

Method 5:- Handling HTTP errors

All MVC exception handling techniques discussed till now do not handle HTTP errors like file not found, HTTP 500 error’s etc. For that we need to make an entry of the error action and the error status code as shown in the below config file.

<system.web>

<customErrors mode="On" defaultRedirect="Error1">

<error statusCode="404" redirect="~/Testing/NoPageFound"/>

<error statusCode="403" redirect="~/Testing/NoPageFound1"/>

<error statusCode="500"

redirect="InternalError.htm"/>

</customErrors>

</system.web>

Method 6:- Global Error handling in MVC (Buena práctica)

If you wish to do global error handling across your application you can override the “Application\_Error” event and do a response.redirect from the global error event. So if the error handling is not done at the controller level it will get propagated to “**Global.asax**” file.

public class MvcApplication : System.Web.HttpApplication

{

protected void Application\_Error(object sender, EventArgs e)

{

Exception exception = Server.GetLastError();

Server.ClearError();

Response.Redirect("/Home/Error");

}

}

What’s the best practice ?

The best is combination of “Method 4” and “Method 6”. Create error handling classeswhich inherit from “HandleErrorAttribute” class and decorate them respectively on controllers and action methods. So this takes care of errors happening on controllers and actions.

As a safety enable Global error handling as a fallback for any unexpected and unhandled errors byusing “Application\_Error” event as described in “Method 6”.