Pass an array of integers to ASP.NET Web API?

Asked 7 years, 6 months ago Active 1 year, 2 months ago Viewed 240k times

384

87

I have an ASP.NET Web API (version 4) REST service where I need to pass an array of integers.

Here is my action method:

```
public IEnumerable<Category> GetCategories(int[] categoryIds){
// code to retrieve categories from database
}
```

And this is the URL that I have tried:

```
/Categories?categoryids=1,2,3,4
```

c# arrays rest asp.net-web-api

edited Jan 9 '18 at 17:47
DavidRR
10.4k 11 67 143

asked Apr 2 '12 at 17:58
Hemanshu Bhojak
8,379 14 41 57

1 I was getting a "Can't bind multiple parameters to the request's content" error when using a querystring like "/Categories?categoryids=1&categoryids=2&categoryids=3". Hope this brings people here who were getting this same error. – Josh Noe Apr 1 '14 at 18:33

1 @Josh Did you use [FromUri] though? public IEnumerable<Category> GetCategories([FromUri] int[] categoryids){...} – Anup Kattel Apr 14 '15 at 4:41

2 @FrankGorman No, I wasn't, which was my issue. – Josh Noe Apr 14 '15 at 19:49

15 Answers

561

```
GetCategories([FromUri] int[] categoryIds)
```

And send request:

```
/Categories?categoryids=1&categoryids=2&categoryids=3
```

edited Mar 14 '16 at 7:47
shA.t
13.6k 4 40 78

answered Jun 19 '12 at 11:57
Lavel
5,742 1 11 8

15 What if I don't know how much variables I have in the array? What if it's like 1000? The request shouldn't be like that. – Sahar Ch. May 30 '14 at 8:22

6 This gives me an error "An item with the same key has already been added.". It does however accept categoryids[0]=1&categoryids[1]=2& etc... – Doctor Jones Jul 11 '14 at 14:03

19 This should be the accepted answer - @Hemanshu Bhojak: isn't it about time to take your pick? – David Rettenbacher Mar 25 '15 at 14:21

10 This reason for this is due to the following statement from the ASP.NET Web API website talking about parameter binding: "If the parameter is a “simple” type, Web API tries to get the value from the URI. Simple types include the .NET primitive types (int, bool, double, and so forth), plus TimeSpan, DateTime, Guid, decimal, and string, plus any type with a type converter that can convert from a string." **an int[] is not a simple type.** – Tr1stan Aug 6 '15 at 13:03

3 This works well for me. One point. On the server code, the array parameter has to come first for it to work and any other parameters, after. When feeding in the parameters in the request, the order is unimportant. – Sparked Apr 23 '16 at 13:47

As Filip W points out, you might have to resort to a custom model binder like this (modified to bind to actual type of param):

94

```
public IEnumerable<Category>
GetCategories([ModelBinder(typeof(CommaDelimitedArrayModelBinder))]long[] categoryIds)
{
    // do your thing
}

public class CommaDelimitedArrayModelBinder : IModelBinder
{
    public bool BindModel(HttpActionContext actionContext, ModelBindingContext
bindingContext)
```

```
        if (val != null)
        {
            var s = val.AttemptedValue;
            if (s != null)
            {
                var elementType = bindingContext.ModelType.GetElementType();
                var converter = TypeDescriptor.GetConverter(elementType);
                var values = Array.ConvertAll(s.Split(new[] {
",",},StringSplitOptions.RemoveEmptyEntries),
                    x => { return converter.ConvertFromString(x != null ? x.Trim() : x);
});

                var typedValues = Array.CreateInstance(elementType, values.Length);

                values.CopyTo(typedValues, 0);

                bindingContext.Model = typedValues;
            }
            else
            {
                // change this line to null if you prefer nulls to empty arrays
                bindingContext.Model =
Array.CreateInstance(bindingContext.ModelType.GetElementType(), 0);
            }
            return true;
        }
        return false;
    }
}
```

And then you can say:

`/Categories?categoryids=1,2,3,4` and ASP.NET Web API will correctly bind your `categoryIds` array.

edited Apr 25 '16 at 11:32
Nkosi
138k 23 181 243

answered Oct 1 '13 at 4:02
Mrchief
63.4k 16 117 172

9 This may violate SRP and/or SoC, but you can easily make this also inherit from `ModelBinderAttribute` so it can be used directly instead of the laborious syntax using the `typeof()` argument. All you have to do is inherit like so: `CommaDelimitedArrayModelBinder : ModelBinderAttribute, IModelBinder` and then provide a default constructor that pushes the type definition down to the base class: `public CommaDelimitedArrayModelBinder() : base(typeof(CommaDelimitedArrayModelBinder)) { }` . – sliderhouserules Jan 11 '16 at 22:49

As a side note, this solution doesn't work with generics like `System.Collections.Generic.List<long>` as `bindingContext.ModelType.GetElementType()` only support `System.Array` types – ViRuSTriNiTy Jan 12 '16 at 10:02

@ViRuSTriNiTy: This question and the answer specifically talk about Arrays. If you need a generic list based solution, that's fairly trivial to implement. Feel free to raise a separate question if you're not sure how to go about that. – Mrchief Jan 12 '16 at 17:06

2 @codeMonkey: putting the array into the body makes good sense for a POST request, but what about GET requests? These usually have no content in the body. – stakx Aug 14 '16 at 19:17

39

I recently came across this requirement myself, and I decided to implement an `ActionFilter` to handle this.

```
public class ArrayInputAttribute : ActionFilterAttribute
{
    private readonly string _parameterName;

    public ArrayInputAttribute(string parameterName)
    {
        _parameterName = parameterName;
        Separator = ',';
    }

    public override void OnActionExecuting(HttpActionContext actionContext)
    {
        if (actionContext.ActionArguments.ContainsKey(_parameterName))
        {
            string parameters = string.Empty;
            if
(actionContext.ControllerContext.RouteData.Values.ContainsKey(_parameterName))
                parameters = (string)
actionContext.ControllerContext.RouteData.Values[_parameterName];
            else if
(actionContext.ControllerContext.Request.RequestUri.ParseQueryString()[_parameterName]
!= null)
                parameters =
actionContext.ControllerContext.Request.RequestUri.ParseQueryString()[_parameterName];

            actionContext.ActionArguments[_parameterName] =
parameters.Split(Separator).Select(int.Parse).ToArray();
        }
    }

    public char Separator { get; set; }
```

I am applying it like so (note that I used 'id', not 'ids', as that is how it is specified in my route):

```
[ArrayInput("id", Separator = ';')]
public IEnumerable<Measure> Get(int[] id)
{
    return id.Select(i => GetData(i));
}
```

And the public url would be:

```
/api/Data/1;2;3;4
```

You may have to refactor this to meet your specific needs.

edited Oct 8 '14 at 16:07
Mrchief
63.4k 16 117 172

answered Jul 9 '13 at 16:19
Steve Czetty
5,762 7 35 48

1 type int are hardcoded (int.Parse) in your solution . Imho, @Mrchief's solution is better – razon Jun 19 '15 at 10:21

24

In case someone would need - to achieve same or similar thing(like delete) via `POST` instead of `FromUri` , use `FromBody` and on client side(JS/jQuery) format param as `$.param({ '': categoryids }, true)`

c#:

```
public IHttpActionResult Remove([FromBody] int[] categoryIds)
```

jQuery:

```
$.ajax({
        type: 'POST',
        data: $.param({ '': categoryids }, true),
        url: url,
//...
```

The thing with `$.param({ '': categoryids }, true)` is that it .net will expect post body to contain urlencoded value like `=1&=2&=3` without parameter name, and without brackets.

edited Mar 14 '16 at 7:32
shA.t
13.6k 4 40 78

answered Mar 13 '15 at 8:46
Sofija
568 9 14

2 No need to resort to a POST. See @Lavel answer. – André Werlang Aug 5 '15 at 19:03

3 There is a limit in how much data you can send in a URI. And by standard, this should not be a GET request since it is actually modifying data. – Worthy7 Oct 3 '16 at 9:06

1 And where exactly did you see a GET here? :) – Sofija Oct 5 '16 at 6:33

3 @Sofija OP says `code to retrieve categories from database` , thus the method should be a GET method, not POST. – Azimuth May 30 '17 at 8:11

20

Easy way to send array params to web api

API

```
public IEnumerable<Category> GetCategories([FromUri]int[] categoryIds){
 // code to retrieve categories from database
}
```

Jquery : send JSON object as request params

```
$.get('api/categories/GetCategories',{categoryIds:[1,2,3,4]}).done(function(response){
console.log(response);
//success response
});
```

It will generate your request URL like `../api/categories/GetCategories?categoryIds=1&categoryIds=2&categoryIds=3&categoryIds=4`

answered Apr 5 '17 at 10:28
Jignesh Variya
1,193 10 9

1 how is this different than the accepted answer? with the exception of implementing an ajax request via jquery which had nothing to do with the original post. – sksallaj Nov 14 '18 at 23:57

10

You may try this code for you to take comma separated values / an array of values to get back a JSON from webAPI

```
public class CategoryController : ApiController
{
    public List<Category> Get(String categoryIDs)
    {
        List<Category> categoryRepo = new List<Category>();

        String[] idRepo = categoryIDs.Split(',');

        foreach (var id in idRepo)
        {
            categoryRepo.Add(new Category()
            {
                CategoryID = id,
                CategoryName = String.Format("Category_{0}", id)
            });
        }
        return categoryRepo;
    }
}

public class Category
{
    public String CategoryID { get; set; }
    public String CategoryName { get; set; }
}
```

Output :

```
[
{"CategoryID":"4","CategoryName":"Category_4"},
{"CategoryID":"5","CategoryName":"Category_5"},
{"CategoryID":"3","CategoryName":"Category_3"}
]
```

3,119 25 38

12k 5 51 74

7

I originally used the solution that @Mrchief for years (it works great). But when when I added **Swagger** to my project for API documentation my end point was **NOT** showing up.

It took me a while, but this is what I came up with. It works with Swagger, and your API method signatures look cleaner:

In the end you can do:

```
    // GET: /api/values/1,2,3,4

    [Route("api/values/{ids}")]
    public IHttpActionResult GetIds(int[] ids)
    {
        return Ok(ids);
    }
```

WebApiConfig.cs

```
public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {
        // Allow WebApi to Use a Custom Parameter Binding
        config.ParameterBindingRules.Add(descriptor => descriptor.ParameterType ==
typeof(int[]) &&
descriptor.ActionDescriptor.SupportedHttpMethods.Contains(HttpMethod.Get)
                                                           ? new
CommaDelimitedArrayParameterBinder(descriptor)
                                                           : null);

        // Allow ApiExplorer to understand this type (Swagger uses ApiExplorer under the
hood)
        TypeDescriptor.AddAttributes(typeof(int[]), new
TypeConverterAttribute(typeof(StringToIntArrayConverter)));

        // Any existing Code ..

    }
}
```

```
public class CommaDelimitedArrayParameterBinder : HttpParameterBinding,
IValueProviderParameterBinding
{
    public CommaDelimitedArrayParameterBinder(HttpParameterDescriptor desc)
        : base(desc)
    {
    }

    /// <summary>
    /// Handles Binding (Converts a comma delimited string into an array of integers)
    /// </summary>
    public override Task ExecuteBindingAsync(ModelMetadataProvider metadataProvider,
                                             HttpActionContext actionContext,
                                             CancellationToken cancellationToken)
    {
        var queryString =
actionContext.ControllerContext.RouteData.Values[Descriptor.ParameterName] as string;

        var ints = queryString?.Split(',').Select(int.Parse).ToArray();

        SetValue(actionContext, ints);

        return Task.CompletedTask;
    }

    public IEnumerable<ValueProviderFactory> ValueProviderFactories { get; } = new[] {
new QueryStringValueProviderFactory() };
}
```

Create a new class: StringToIntArrayConverter.cs

```
public class StringToIntArrayConverter : TypeConverter
{
    public override bool CanConvertFrom(ITypeDescriptorContext context, Type sourceType)
    {
        return sourceType == typeof(string) || base.CanConvertFrom(context, sourceType);
    }
}
```

Notes:

- https://stackoverflow.com/a/47123965/862011 pointed me in the right direction edited May 11 '18 at 14:19

answered Mar 29 '18 at 19:48

crabCRUSHERclamCOLLECTOR
1,529 14 11

1 In-case anyone else needs info on the libraries this uses. Here is the using for "CommaDelimitedArrayParameterBinder ". using System.Collections.Generic; using System.Linq; using System.Threading; using System.Threading.Tasks; using System.Web.Http.Controllers; using System.Web.Http.Metadata; using System.Web.Http.ModelBinding; using System.Web.Http.ValueProviders; using System.Web.Http.ValueProviders.Providers; – SteckDEV Mar 15 at 19:45

7

ASP.NET Core 2.0 Solution (Swagger Ready)

Input

```
DELETE /api/items/1,2
DELETE /api/items/1
```

Code

Write the provider (how MVC knows what binder to use)

```
public class CustomBinderProvider : IModelBinderProvider
{
    public IModelBinder GetBinder(ModelBinderProviderContext context)
    {
        if (context == null)
        {
            throw new ArgumentNullException(nameof(context));
        }

        if (context.Metadata.ModelType == typeof(int[]) || context.Metadata.ModelType ==
typeof(List<int>))
        {
            return new
BinderTypeModelBinder(typeof(CommaDelimitedArrayParameterBinder));
```

```
    }
}
```

Write the actual binder (access all sorts of info about the request, action, models, types, whatever)

```
public class CommaDelimitedArrayParameterBinder : IModelBinder
{

    public Task BindModelAsync(ModelBindingContext bindingContext)
    {

        var value =
bindingContext.ActionContext.RouteData.Values[bindingContext.FieldName] as string;

        // Check if the argument value is null or empty
        if (string.IsNullOrEmpty(value))
        {
            return Task.CompletedTask;
        }

        var ints = value?.Split(',').Select(int.Parse).ToArray();

        bindingContext.Result = ModelBindingResult.Success(ints);

        if(bindingContext.ModelType == typeof(List<int>))
        {
            bindingContext.Result = ModelBindingResult.Success(ints.ToList());
        }

        return Task.CompletedTask;
    }
}
```

Register it with MVC

```
services.AddMvc(options =>
{
    // add custom binder to beginning of collection
    options.ModelBinderProviders.Insert(0, new CustomBinderProvider());
});
```

```
/// <summary>
/// Deletes a list of items.
/// </summary>
/// <param name="itemIds">The list of unique identifiers for the  items.</param>
/// <returns>The deleted item.</returns>
/// <response code="201">The item was successfully deleted.</response>
/// <response code="400">The item is invalid.</response>
[HttpDelete("{itemIds}", Name = ItemControllerRoute.DeleteItems)]
[ProducesResponseType(typeof(void), StatusCodes.Status204NoContent)]
[ProducesResponseType(typeof(void), StatusCodes.Status404NotFound)]
public async Task Delete(List<int> itemIds)
=> await _itemAppService.RemoveRangeAsync(itemIds);
```

EDIT: Microsoft recommends using a TypeConverter for these kids of operations over this approach. So follow the below posters advice and document your custom type with a SchemaFilter.

edited Jun 7 '18 at 18:09 answered Apr 5 '18 at 19:38

Victorio Berra
1,264 1 15 26

I think the MS recomendation you're talking about is satisfied by this answer: stackoverflow.com/a/49563970/4367683 – Machado Oct 15 '18 at 16:39

Did you see this? github.com/aspnet/Mvc/pull/7967 it looks as if they added a fix to start parsing List<whatever> in the query string without a need for a special binder. Also the post you linked is not ASPNET Core and I do not think helps with my situation. – Victorio Berra Oct 17 '18 at 20:38

The best, non-hacky answer. – Erik Philips Jun 10 at 23:16

6

```
public class ArrayInputAttribute : ActionFilterAttribute
{
    private readonly string[] _ParameterNames;
    /// <summary>
    ///
    /// </summary>
    public string Separator { get; set; }
    /// <summary>
    /// cons
    /// </summary>
    /// <param name="parameterName"></param>
```

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

```
            Separator = ",";
        }

        /// <summary>
        ///
        /// </summary>
        public void ProcessArrayInput(HttpActionContext actionContext, string parameterName)
        {
            if (actionContext.ActionArguments.ContainsKey(parameterName))
            {
                var parameterDescriptor =
    actionContext.ActionDescriptor.GetParameters().FirstOrDefault(p => p.ParameterName ==
    parameterName);
                if (parameterDescriptor != null &&
    parameterDescriptor.ParameterType.IsArray)
                {
                    var type = parameterDescriptor.ParameterType.GetElementType();
                    var parameters = String.Empty;
                    if
    (actionContext.ControllerContext.RouteData.Values.ContainsKey(parameterName))
                    {
                        parameters =
    (string)actionContext.ControllerContext.RouteData.Values[parameterName];
                    }
                    else
                    {
                        var queryString =
    actionContext.ControllerContext.Request.RequestUri.ParseQueryString();
                        if (queryString[parameterName] != null)
                        {
                            parameters = queryString[parameterName];
                        }
                    }

                    var values = parameters.Split(new[] { Separator },
    StringSplitOptions.RemoveEmptyEntries)

    .Select(TypeDescriptor.GetConverter(type).ConvertFromString).ToArray();
                    var typedValues = Array.CreateInstance(type, values.Length);
                    values.CopyTo(typedValues, 0);
                    actionContext.ActionArguments[parameterName] = typedValues;
                }
            }
        }

        public override void OnActionExecuting(HttpActionContext actionContext)
        }
    }
```

Usage:

```
        [HttpDelete]
        [ArrayInput("tagIDs")]
        [Route("api/v1/files/{fileID}/tags/{tagIDs}")]
        public HttpResponseMessage RemoveFileTags(Guid fileID, Guid[] tagIDs)
        {
            _FileRepository.RemoveFileTags(fileID, tagIDs);
            return Request.CreateResponse(HttpStatusCode.OK);
        }
```

Request uri

```
http://localhost/api/v1/files/2a9937c7-8201-59b7-bc8d-11a9178895d0/tags/BBA5CD5D-F07D-
47A9-8DEE-D19F5FA65F63,BBA5CD5D-F07D-47A9-8DEE-D19F5FA65F63
```

edited Jul 18 '14 at 9:51

answered Jul 15 '14 at 5:04
Waninlezu
69 1 2

can you explain your code? – Sahar Ch. Jul 16 '14 at 8:31

@Elsa Could you please point out which piece you can't understand? I think the code is quite clear to explanation it self. It's hard for me to explain this all in English, sorry. – Waninlezu Jul 17 '14 at 1:41

@Steve Czetty here's my reconstructed version, thanks for your idea – Waninlezu Jul 17 '14 at 1:46

Will it work with `/` as the seperator? Then you could have: dns/root/mystuff/path/to/some/resource mapped to `public string GetMyStuff(params string[] pathBits)` – RoboJ1M Oct 20 '14 at 9:01

5

If you want to list/ array of integers easiest way to do this is accept the comma(,) separated list of string and convert it to list of integers.Do not forgot to mention [FromUri] attriubte.your url look like:

```
public HttpResponseMessage test([FromUri]int ID, [FromUri]string accountID)
{
    List<int> accountIdList = new List<int>();
    string[] arrAccountId = accountId.Split(new char[] { ',' });
    for (var i = 0; i < arrAccountId.Length; i++)
    {
        try
        {
           accountIdList.Add(Int32.Parse(arrAccountId[i]));
        }
        catch (Exception)
        {
        }
    }
}
```

edited May 20 '16 at 15:50
Daniël Tulp
927 2 13 45

answered Aug 4 '15 at 8:30
Vaibhav
67 1 2

why do you use `List<string>` instead of just `string` ? it will only have one string in it which is `1,2,3,289,56` in your example. I will suggest an edit. – Daniël Tulp May 20 '16 at 14:20

Worked for me. I was surprised my controller wouldn't bind to a `List<Guid>` automatically though. Note in Asp.net Core the annotation is `[FromQuery]` , and it is not needed. – kitsu.eb Jun 28 '16 at 17:56

2 For a one-line Linq version: int[] accountIdArray = accountId.Split(',').Select(i => int.Parse(i)).ToArray(); I'd avoid the catch since it will mask somebody passing in bad data. – Steve In CO Oct 13 '17 at 14:27

5

Instead of using a custom ModelBinder, you can also use a custom type with a TypeConverter.

```
[TypeConverter(typeof(StrListConverter))]
public class StrList : List<string>
{
    public StrList(IEnumerable<string> collection) : base(collection) {}
}

public class StrListConverter : TypeConverter
```

```
    }

    public override object ConvertFrom(ITypeDescriptorContext context, CultureInfo
culture, object value)
    {
        if (value == null)
            return null;

        if (value is string s)
        {
            if (string.IsNullOrEmpty(s))
                return null;
            return new StrList(s.Split(','));
        }
        return base.ConvertFrom(context, culture, value);
    }
}
```

The advantage is that it makes the Web API method's parameters very simple. You dont't even need to specify [FromUri].

```
public IEnumerable<Category> GetCategories(StrList categoryIds) {
  // code to retrieve categories from database
}
```

This example is for a List of strings, but you could do `categoryIds.Select(int.Parse)` or simply write an IntList instead.

answered Feb 5 '18 at 11:26
PhillipM
104 1 2

Don't understand why this solution didn't get much votes. It is nice and clean and works with swagger without adding custom binders and stuff. – Thieme Aug 21 '18 at 19:06

The best/cleanest answer in my opinion. Thanks PhillipM! – Leigh Bowers Mar 18 at 12:00

2

Make the method type [HttpPost], create a model that has one int[] parameter, and post with json:

```
/* Model */
```

```
    public int[] Categories { get; set; }
}

/* WebApi */
[HttpPost]
public HttpResponseMessage GetCategories(CategoryRequestModel model)
{
    HttpResponseMessage resp = null;

    try
    {
        var categories = //your code to get categories

        resp = Request.CreateResponse(HttpStatusCode.OK, categories);

    }
    catch(Exception ex)
    {
        resp = Request.CreateErrorResponse(HttpStatusCode.InternalServerError, ex);
    }

    return resp;
}

/* jQuery */
var ajaxSettings = {
    type: 'POST',
    url: '/Categories',
    data: JSON.serialize({Categories: [1,2,3,4]}),
    contentType: 'application/json',
    success: function(data, textStatus, jqXHR)
    {
        //get categories from data
    }
};

$.ajax(ajaxSettings);
```

answered Jun 10 '16 at 15:50

codeMonkey
1,761 1 16 29

You're wrapping your array in a class - that will work fine (MVC/WebAPI notwithstanding). The OP was about binding to array without a wrapper class. –

go down that path too far you'll get to a point where you need the API to pick up a really complex js object, and query param's will fail you. Might as well learn to do it the way that will work every time. – codeMonkey Jun 10 '16 at 16:05

`public IEnumerable<Category> GetCategories(int[] categoryIds){` - yeah you could interpret in different ways I suppose. But many a times, I do not want to create wrapper classes for the sake of creating wrappers. If you have complex objects, then that will just work. Supporting these simpler cases is what doesn't work out of the box, hence the OP. – Mrchief Jun 10 '16 at 16:08

2 Doing this via `POST` is actually against the REST paradigm. Thus such API wouldn't be a REST API. – Azimuth May 30 '17 at 8:08

1 @Azimuth give me a paradigm in one hand, what works with .NET in the other – codeMonkey May 30 '17 at 16:38

3

Or you could just pass a string of delimited items and put it into an array or list on the receiving end.

answered Aug 30 '17 at 16:03

Sirentec
609 6 15

1 People like to reinvent the wheel... – Carlos ABS Nov 14 '18 at 12:36

2

I addressed this issue this way.

I used a post message to the api to send the list of integers as data.

Then I returned the data as an ienumerable.

The sending code is as follows:

```
public override IEnumerable<Contact> Fill(IEnumerable<int> ids)
{
    IEnumerable<Contact> result = null;
    if (ids!=null&&ids.Count()>0)
    {
        try
        {
            using (var client = new HttpClient())
            {
```

```
MediaTypeWithQualityHeaderValue("application/json"));

                String _endPoint = "api/" + typeof(Contact).Name + "/ListArray";

                HttpResponseMessage response = client.PostAsJsonAsync<IEnumerable<int>>
(_endPoint, ids).Result;
                response.EnsureSuccessStatusCode();
                if (response.IsSuccessStatusCode)
                {
                    result = JsonConvert.DeserializeObject<IEnumerable<Contact>>
(response.Content.ReadAsStringAsync().Result);
                }

            }

        }
        catch (Exception)
        {

        }
    }
    return result;
}
```

The receiving code is as follows:

```
// POST api/<controller>
[HttpPost]
[ActionName("ListArray")]
public IEnumerable<Contact> Post([FromBody]IEnumerable<int> ids)
{
    IEnumerable<Contact> result = null;
    if (ids != null && ids.Count() > 0)
    {
        return contactRepository.Fill(ids);
    }
    return result;
}
```

It works just fine for one record or many records. The fill is an overloaded method using DapperExtensions:

```
public override IEnumerable<Contact> Fill(IEnumerable<int> ids)
```

```
            using (IDbConnection dbConnection = ConnectionProvider.OpenConnection())
            {
                dbConnection.Open();
                var predicate = Predicates.Field<Contact>(f => f.id, Operator.Eq, ids);
                result = dbConnection.GetList<Contact>(predicate);
                dbConnection.Close();
            }
        }
        return result;
    }
```

This allows you to fetch data from a composite table (the id list), and then return the records you are really interested in from the target table.

You could do the same with a view, but this gives you a little more control and flexibility.

In addition, the details of what you are seeking from the database are not shown in the query string. You also do not have to convert from a csv file.

You have to keep in mind when using any tool like the web api 2.x interface is that the get, put, post, delete, head, etc., functions have a general use, but are not restricted to that use.

So, while post is generally used in a create context in the web api interface, it is not restricted to that use. It is a *regular* html call that can be used for any purpose permitted by html practice.

In addition, the details of what is going on are hidden from those "prying eyes" we hear so much about these days.

The flexibility in naming conventions in the web api 2.x interface and use of regular web calling means you send a call to the web api that misleads snoopers into thinking you are really doing something else. You can use "POST" to really retrieve data, for example.

edited Oct 27 '15 at 14:19

Vivek Jain
3,360 6 23 45

answered Oct 27 '15 at 13:45

Timothy Dooling
297 3 14

0

My solution was to create an attribute to validate strings, it does a bunch of extra common features, including regex validation that you can use to check for numbers only and then later I convert to integers as needed...

This is how you use:

```
public class MustBeListAndContainAttribute : ValidationAttribute
{
    private Regex regex = null;
    public bool RemoveDuplicates { get; }
    public string Separator { get; }
    public int MinimumItems { get; }
    public int MaximumItems { get; }

    public MustBeListAndContainAttribute(string regexEachItem,
        int minimumItems = 1,
        int maximumItems = 0,
        string separator = ",",
        bool removeDuplicates = false) : base()
    {
        this.MinimumItems = minimumItems;
        this.MaximumItems = maximumItems;
        this.Separator = separator;
        this.RemoveDuplicates = removeDuplicates;

        if (!string.IsNullOrEmpty(regexEachItem))
            regex = new Regex(regexEachItem, RegexOptions.Compiled |
RegexOptions.Singleline | RegexOptions.IgnoreCase);
    }

    protected override ValidationResult IsValid(object value, ValidationContext
validationContext)
    {
        var listOfdValues = (value as List<string>)?[0];

        if (string.IsNullOrWhiteSpace(listOfdValues))
        {
            if (MinimumItems > 0)
                return new ValidationResult(this.ErrorMessage);
            else
                return null;
        };

        var list = new List<string>();

        list.AddRange(listOfdValues.Split(new[] { Separator },
System.StringSplitOptions.RemoveEmptyEntries));

        if (RemoveDuplicates) list = list.Distinct().ToList();

        var prop =
        if (regex != null)
            if (list.Any(c => string.IsNullOrWhiteSpace(c) || !regex.IsMatch(c)))
                return new ValidationResult(this.ErrorMessage);

        return null;
    }
}
```

edited Jul 29 '18 at 6:19

answered Jun 21 '18 at 10:03

Alan Cardoso
58 5

protected by Pavneet_Singh Aug 11 '18 at 21:09

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, posting an answer now requires 10 reputation on this site (the association bonus does not count).

Would you like to answer one of these unanswered questions instead?