FluentRestBuilder Documentation

Release 1.0.0-beta2

Lukas Spirig

Dec 09, 2017

Contents:

FluentRestBuilder is a library to easily build RESTful APIs on top of MVC Core.

It is based on the IObservable<T> and IObserver<T> interfaces and inspired by the Reactive Extensions (Rx.NET). It is however not fully compatible with Rx.NET, as FluentRestBuilder extends the IObservable<T> interface.

The motivation for this library is to reduce boilerplate code where possible, so instead of writing this:

```
[Route("posts")]
public class PostController : ControllerBase
{
    [HttpGet("{id}", Name = "PostResource")]
    public async Task<IActionResult> Get(int id)
    {
        var post = await this.dbContext.Posts
            .Include(p => p.Author)
            .SingleOrDefaultAsync(p => p.Id == id);
        if (post == null)
        {
            return this.NotFound();
        }

        var result = new PostResponse(post, this.Url);
        return this.Ok(result);
    }

    [HttpPost("{id}")]
    public async Task<IActionResult> Update([FromBody] PostRequest request, int id)
    {
        var post = await this.dbContext.Posts
            .Include(p => p.Author)
            .SingleOrDefaultAsync(p => p.Id == id);
        if (post == null)
        {
            return this.NotFound();
        }

        if (!this.ModelState.IsValid)
        {
            return this.BadRequest(this.ModelState);
        }

        post.Title = request.Title;
        post.Content = request.Content;
        await this.dbContext.SaveChangesAsync();

        var result = new PostResponse(post, this.Url);
        return this.Ok(post);
    }
}
```

you can simply write the following:

```
[Route("posts")]
public class PostController : ControllerBase
{
    [HttpGet("{id}", Name = "PostResource")]
    public async Task<IActionResult> Get(int id) =>
        await this.CreateQueryableSingle<Post>()
            .Include(p => p.Author)
```

```
            .SingleOrDefaultAsync(p => p.Id == id)
            .NotFoundWhenNull()
            .Map(p => new PostResponse(p, this.Url))
            .ToOkObjectResult();

    [HttpPost("{id}")]
    public async Task<IActionResult> Update([FromBody] PostRequest request, int id) =>
        await this.CreateEntitySingle<Post>(id)
            .NotFoundWhenNull()
            .BadRequestWhenModelStateIsInvalid(this.ModelState)
            .Do(p =>
            {
                p.Title = request.Title;
                p.Content = request.Content;
            })
            .SaveChangesAsync()
            .Map(p => new PostResponse(p, this.Url))
            .ToOkObjectResult();
}
```

CHAPTER 1

Getting Started

In order to get started, you first have to add one or more packages from the following list to your project, via nuget:

- FluentRestBuilder
- FluentRestBuilder.EntityFrameworkCore
- FluentRestBuilder.Caching
- FluentRestBuilder.HypertextApplicationLanguage

Register FluentRestBuilder in your Startup:

```
public void ConfigureServices(IServiceCollection services)
{
    ...

    services.AddMvc(options =>
    {
        // Add this if you want to use operators which depend
        // on the HttpContext.
        // Alternatively you can add this attribute to the controller
        // or the controller method where you want to use such an operator.
        options.Filters.Add(new HttpContextProviderAttribute());
    });

    services.AddFluentRestBuilder()
        // Optional. Only if you want to use EntityFramework operators.
        .AddEntityFrameworkCoreIntegration<ApplicationDbContext>()
        // Optional. Configures filters and order by expressions for pagination␣
↪operators.
        .ConfigureFiltersAndOrderByExpressionsForDbContextEntities
↪<ApplicationDbContext>();
}
```

Now you can use FluentRestBuilder in your controllers. (See sample for an example.)

```
using System.Threading.Tasks;
using FluentRestBuilder;
using Microsoft.AspNetCore.Mvc;
using Models;
using ViewModels;

[Route("posts")]
public class PostController : ControllerBase
{
    [HttpGet(Name = nameof(PostController))]
    public async Task<IActionResult> Get() =>
        await this.CreateQueryableSingle<Post>()
            .Include(p => p.Author)
            .OrderByDescending(p => p.CreatedAt)
            .ApplyFilterByClientRequest()
            .ApplyOrderByClientRequest()
            .ApplyPaginationByClientRequest()
            .ToListAsync()
            .MapToRestCollection(p => new PostResponse(p, this.Url))
            .ToOkObjectResult();

    [HttpGet("{id}", Name = "PostResource")]
    public async Task<IActionResult> Get(int id) =>
        await this.CreateQueryableSingle<Post>()
            .Include(p => p.Author)
            .SingleOrDefaultAsync(p => p.Id == id)
            .NotFoundWhenNull()
            .Map(p => new PostResponse(p, this.Url))
            .ToOkObjectResult();

    ...
}
```

CHAPTER 2

Observables

Observables can be subscribed to and can potentially emit values. They can be completed, after which no more values will be emitted. On error they will emit an Exception and afterwards can no longer emit a value or be completed.

Observables could emit multiple values, however the observables from FluentRestBuilder will only emit one value and complete afterwards or emit an exception in an error case.

Observables can be awaited. This will return the last emitted value on completion or throw the received exception in an error case.

2.1 SingleObservable<T>

This is an observable that emits the value that it was given on instancing and completes.

```
public SingleObservable(T value, IServiceProvider serviceProvider)
```

Examples

```
new SingleObservable<string>("example", serviceProvider)

// Alternatively
Observable.Single("example", serviceProvider)
```

2.2 AsyncSingleObservable<T>

This observable can be instantiated by providing a callback, which is only executed once the observable is subscribed to.

```
public AsyncSingleObservable(Func<Task<T>> valueTaskFactory, IServiceProvider␣
↪serviceProvider)
public AsyncSingleObservable(Func<T> valueFactory, IServiceProvider serviceProvider)
public AsyncSingleObservable(Lazy<T> lazyValue, IServiceProvider serviceProvider)
```

Examples

```
new AsyncSingleObservable<string>(() => "example", serviceProvider)
new AsyncSingleObservable<string>(async () => await AsynchronousTask(),␣
↪serviceProvider)

// Alternatively
Observable.AsyncSingle(() => "example", serviceProvider)
Observable.AsyncSingle(async () => await AsynchronousTask(), serviceProvider)
```

2.3 Controller Integration

FluentRestBuilder provides extension methods that can be used to create an observable in a controller. These use the service provider from the controller, so

FluentRestBuilder

```
public static IProviderObservable<TSource> CreateSingle<TSource>(
    this ControllerBase controller, TSource value)

public static IProviderObservable<TSource> CreateAsyncSingle<TSource>(
    this ControllerBase controller, Func<Task<TSource>> valueFactory)

public static IProviderObservable<TSource> CreateAsyncSingle<TSource>(
    this ControllerBase controller, Func<TSource> valueFactory)

public static IProviderObservable<TSource> CreateAsyncSingle<TSource>(
    this ControllerBase controller, Lazy<TSource> valueFactory)
```

FluentRestBuilder.EntityFrameworkCore

In order for these to work, the entity framework `DbContext` has to be registered for FluentRestBuilder.

See *Getting Started* for an example.

```
public static IProviderObservable<TSource> CreateEntitySingle<TSource>(
    this ControllerBase controller, Expression<Func<TSource, bool>> predicate)

public static IProviderObservable<TSource> CreateEntitySingle<TSource>(
    this ControllerBase controller, params object[] keyValues)

public static IProviderObservable<IQueryable<TSource>> CreateQueryableSingle<TSource>(
    this ControllerBase controller)
```

CHAPTER 3

Operators

Operators can be used with observables or other operators. They are a combination of observer and observable. This means that the description for *Observables* also applies.

3.1 ApplyFilterByClientRequest

Apply filter logic to the received `IQueryable<TSource>`.

Matches the query parameters with the keys of the given filter dictionary. Implement `IFilterByClientRequestInterpreter` for custom behavior.

Requires `HttpContextProviderAttribute` to be set.

Package: FluentRestBuilder

```
public static IProviderObservable<IQueryable<TSource>> ApplyFilterByClientRequest
↪<TSource>(
    this IProviderObservable<IQueryable<TSource>> observable,
    Func<FilterExpressionProviderDictionary<TSource>,
↪IFilterExpressionProviderDictionary<TSource>> factory)
```

Apply filter logic to the received `IQueryable<TSource>`. Tries to resolve `IFilterExpressionProviderDictionary<TSource>` via `IServiceProvider`.

Matches the query parameters with the keys of the given filter dictionary. Implement `IFilterByClientRequestInterpreter` for custom behavior.

Requires `HttpContextProviderAttribute` to be set.

Package: FluentRestBuilder

```
public static IProviderObservable<IQueryable<TSource>> ApplyFilterByClientRequest
↪<TSource>(
    this IProviderObservable<IQueryable<TSource>> observable)
```

Apply filter logic to the received `IQueryable<TSource>`.

Matches the query parameters with the keys of the given filter dictionary. Implement `IFilterByClientRequestInterpreter` for custom behavior.

Requires `HttpContextProviderAttribute` to be set.

Package: FluentRestBuilder

```
public static IProviderObservable<IQueryable<TSource>> ApplyFilterByClientRequest
↪<TSource>(
    this IProviderObservable<IQueryable<TSource>> observable,
    IDictionary<string,IFilterExpressionProvider<TSource>> filterDictionary)
```

3.2 ApplyOrderByClientRequest

Apply order by logic to the received `IQueryable<TSource>`.

The default query parameter key is "sort". A comma-separated list of properties is supported. Prefix the property with "-" to sort descending. Implement `IOrderByClientRequestInterpreter` for custom behavior.

Requires `HttpContextProviderAttribute` to be set.

Package: FluentRestBuilder

```
public static IProviderObservable<IQueryable<TSource>> ApplyOrderByClientRequest
↪<TSource>(
    this IProviderObservable<IQueryable<TSource>> observable,
    Func<OrderByExpressionDictionary<TSource>,IOrderByExpressionDictionary<TSource>>␣
↪factory)
```

Apply order by logic to the received `IQueryable<TSource>`. Tries to resolve `IOrderByExpressionDictionary<TSource>` via `IServiceProvider`.

The default query parameter key is "sort". A comma-separated list of properties is supported. Prefix the property with "-" to sort descending. Implement `IOrderByClientRequestInterpreter` for custom behavior.

Requires `HttpContextProviderAttribute` to be set.

Package: FluentRestBuilder

```
public static IProviderObservable<IQueryable<TSource>> ApplyOrderByClientRequest
↪<TSource>(
    this IProviderObservable<IQueryable<TSource>> observable)
```

Apply order by logic to the received `IQueryable<TSource>`. Provide a dictionary with provided order by expressions.

The default query parameter key is "sort". A comma-separated list of properties is supported. Prefix the property with "-" to sort descending. Implement `IOrderByClientRequestInterpreter` for custom behavior.

Requires `HttpContextProviderAttribute` to be set.

Package: FluentRestBuilder

```
public static IProviderObservable<IQueryable<TSource>> ApplyOrderByClientRequest
↪<TSource>(
    this IProviderObservable<IQueryable<TSource>> observable,
    IDictionary<string,IOrderByExpressionFactory<TSource>> orderByExpressions)
```

3.3 ApplyPaginationByClientRequest

Configure the pagination capabilities.

WARNING: Do not use this before FilterByClientRequest, SearchByClientRequest or OrderByClientRequest! This would result in erroneous pagination logic.

Requires `HttpContextProviderAttribute` to be set.

Package: FluentRestBuilder

```
public static IProviderObservable<IQueryable<TSource>> ApplyPaginationByClientRequest
↪<TSource>(
    this IProviderObservable<IQueryable<TSource>> observable,
    FluentRestBuilder.Operators.ClientRequest.PaginationOptions options)
```

3.4 ApplySearchByClientRequest

Apply a global search to the received `IQueryable<TSource>`.

The default query parameter key is "q". Implement `ISearchByClientRequestInterpreter` for custom behavior.

Requires `HttpContextProviderAttribute` to be set.

Package: FluentRestBuilder

```
public static IProviderObservable<IQueryable<TSource>> ApplySearchByClientRequest
↪<TSource>(
    this IProviderObservable<IQueryable<TSource>> observable,
    Func<string,Expression<Func<TSource,System.Boolean>>> searchExpression)
```

3.5 AsNoTracking

Returns a new query where the change tracker will not track any of the entities that are returned. If the entity instances are modified, this will not be detected by the change tracker and `SaveChanges` will not persist those changes to the database.

Disabling change tracking is useful for read-only scenarios because it avoids the overhead of setting up change tracking for each entity instance. You should not disable change tracking if you want to manipulate entity instances and persist those changes to the database using `SaveChanges`.

Identity resolution will still be performed to ensure that all occurrences of an entity with a given key in the result set are represented by the same entity instance.

The default tracking behavior for queries can be controlled by `QueryTrackingBehavior`.

Package: FluentRestBuilder.EntityFrameworkCore

```
public static IProviderObservable<IQueryable<TSource>> AsNoTracking<TSource>(
    this IProviderObservable<IQueryable<TSource>> observable)
```

3.6 BadRequestWhen

If the check returns `true`, `ValidationException` is emitted as an error with the status code 400 (Bad Request). Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> BadRequestWhen<TSource>(
    this IProviderObservable<TSource> observable,
    Func<TSource,System.Boolean> invalidCheck,
    Func<TSource,object> errorFactory)
```

If the check returns `true`, `ValidationException` is emitted as an error with the status code 400 (Bad Request). Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> BadRequestWhen<TSource>(
    this IProviderObservable<TSource> observable,
    Func<TSource,System.Boolean> invalidCheck,
    object error)
```

If the check returns `true`, `ValidationException` is emitted as an error with the status code 400 (Bad Request). Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> BadRequestWhen<TSource>(
    this IProviderObservable<TSource> observable,
    Func<System.Boolean> invalidCheck,
    Func<TSource,object> errorFactory)
```

If the check returns `true`, `ValidationException` is emitted as an error with the status code 400 (Bad Request). Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> BadRequestWhen<TSource>(
    this IProviderObservable<TSource> observable,
    Func<System.Boolean> invalidCheck,
    object error)
```

3.7 BadRequestWhenAsync

If the check returns `true`, `ValidationException` is emitted as an error with the status code 400 (Bad Request). Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> BadRequestWhenAsync<TSource>(
    this IProviderObservable<TSource> observable,
    Func<TSource,Task<System.Boolean>> invalidCheck,
    Func<TSource,object> errorFactory)
```

If the check returns `true`, `ValidationException` is emitted as an error with the status code 400 (Bad Request). Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> BadRequestWhenAsync<TSource>(
    this IProviderObservable<TSource> observable,
    Func<TSource,Task<System.Boolean>> invalidCheck,
    object error)
```

If the check returns `true`, `ValidationException` is emitted as an error with the status code 400 (Bad Request). Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> BadRequestWhenAsync<TSource>(
    this IProviderObservable<TSource> observable,
    Func<Task<System.Boolean>> invalidCheck,
    Func<TSource,object> errorFactory)
```

If the check returns `true`, `ValidationException` is emitted as an error with the status code 400 (Bad Request). Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> BadRequestWhenAsync<TSource>(
    this IProviderObservable<TSource> observable,
    Func<Task<System.Boolean>> invalidCheck,
    object error)
```

3.8 BadRequestWhenModelStateIsInvalid

If the check returns `true`, `ValidationException` is emitted as an error with the status code 400 (Bad Request). Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> BadRequestWhenModelStateIsInvalid<TSource>(
    this IProviderObservable<TSource> observable,
    Microsoft.AspNetCore.Mvc.ModelBinding.ModelStateDictionary modelState)
```

3.9 CacheInDistributedCache

Cache the received value in `IDistributedCache` with the given key. If an entry with the given key is found in the cache, it will be emitted and the previous chain is skipped.

Package: FluentRestBuilder.Caching

```
public static IProviderObservable<TSource> CacheInDistributedCache<TSource>(
    this IProviderObservable<TSource> observable,
    string key)
```

Cache the received value in `IDistributedCache` with the given key and the defined absolute expiration moment. If an entry with the given key is found in the cache, it will be emitted and the previous chain is skipped.

Package: FluentRestBuilder.Caching

```
public static IProviderObservable<TSource> CacheInDistributedCache<TSource>(
    this IProviderObservable<TSource> observable,
    string key,
    System.DateTimeOffset absoluteExpiration)
```

Cache the received value in `IDistributedCache` with the given key and the defined absolute expiration moment. If an entry with the given key is found in the cache, it will be emitted and the previous chain is skipped.

Package: FluentRestBuilder.Caching

```
public static IProviderObservable<TSource> CacheInDistributedCache<TSource>(
    this IProviderObservable<TSource> observable,
    string key,
    System.TimeSpan absoluteExpirationRelativeToNow)
```

Cache the received value in `IDistributedCache` with the given key and the defined distributed cache options factory function. If an entry with the given key is found in the cache, it will be emitted and the previous chain is skipped.

Package: FluentRestBuilder.Caching

```
public static IProviderObservable<TSource> CacheInDistributedCache<TSource>(
    this IProviderObservable<TSource> observable,
    string key,
    Func<TSource,Microsoft.Extensions.Caching.Distributed.
↪DistributedCacheEntryOptions> optionsFactory)
```

3.10 CacheInMemoryCache

Cache the received value in `IMemoryCache` with the given key and the defined absolute expiration moment. If an entry with the given key is found in the cache, it will be emitted and the previous chain is skipped.

Package: FluentRestBuilder.Caching

```
public static IProviderObservable<TSource> CacheInMemoryCache<TSource>(
    this IProviderObservable<TSource> observable,
    object key,
    System.DateTimeOffset absoluteExpiration)
```

Cache the received value in `IMemoryCache` with the given key and the defined absolute expiration moment relative to now. If an entry with the given key is found in the cache, it will be emitted and the previous chain is skipped.

Package: FluentRestBuilder.Caching

```
public static IProviderObservable<TSource> CacheInMemoryCache<TSource>(
    this IProviderObservable<TSource> observable,
    object key,
    System.TimeSpan absoluteExpirationRelativeToNow)
```

Cache the received value in `IMemoryCache` with the given key and the given expiration token. If an entry with the given key is found in the cache, it will be emitted and the previous chain is skipped.

Package: FluentRestBuilder.Caching

```
public static IProviderObservable<TSource> CacheInMemoryCache<TSource>(
    this IProviderObservable<TSource> observable,
```

```
    object key,
    Microsoft.Extensions.Primitives.IChangeToken expirationToken)
```

Cache the received value in `IMemoryCache` with the given key and the defined memory cache options factory function. If an entry with the given key is found in the cache, it will be emitted and the previous chain is skipped.

Package: FluentRestBuilder.Caching

```
public static IProviderObservable<TSource> CacheInMemoryCache<TSource>(
    this IProviderObservable<TSource> observable,
    object key,
    Func<TSource,Microsoft.Extensions.Caching.Memory.MemoryCacheEntryOptions>␣
↪optionsFactory)
```

3.11 CurrentUserHas

If the check returns `false`, `ValidationException` is emitted as an error with the status code 403 (Forbidden). Otherwise the given value is emitted.

Requires usage of `HttpContextProviderAttribute`.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> CurrentUserHas<TSource>(
    this IProviderObservable<TSource> observable,
    Func<System.Security.Claims.ClaimsPrincipal,TSource,System.Boolean>␣
↪principalCheck,
    object error)
```

If the check returns `false`, `ValidationException` is emitted as an error with the status code 403 (Forbidden). Otherwise the given value is emitted.

Requires usage of `HttpContextProviderAttribute`.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> CurrentUserHas<TSource>(
    this IProviderObservable<TSource> observable,
    Func<System.Security.Claims.ClaimsPrincipal,System.Boolean> principalCheck,
    Func<TSource,object> errorFactory)
```

If the check returns `false`, `ValidationException` is emitted as an error with the status code 403 (Forbidden). Otherwise the given value is emitted.

Requires usage of `HttpContextProviderAttribute`.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> CurrentUserHas<TSource>(
    this IProviderObservable<TSource> observable,
    Func<System.Security.Claims.ClaimsPrincipal,System.Boolean> principalCheck,
    object error)
```

If the check returns `false`, `ValidationException` is emitted as an error with the status code 403 (Forbidden). Otherwise the given value is emitted.

Requires usage of `HttpContextProviderAttribute`.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> CurrentUserHas<TSource>(
    this IProviderObservable<TSource> observable,
    Func<System.Security.Claims.ClaimsPrincipal,TSource,System.Boolean>␣
↪principalCheck,
    Func<TSource,object> errorFactory)
```

3.12 CurrentUserHasClaim

If the check returns `false`, `ValidationException` is emitted as an error with the status code 403 (Forbidden). Otherwise the given value is emitted.

Requires usage of `HttpContextProviderAttribute`.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> CurrentUserHasClaim<TSource>(
    this IProviderObservable<TSource> observable,
    string claimType,
    string claim,
    Func<TSource,object> errorFactory)
```

If the check returns `false`, `ValidationException` is emitted as an error with the status code 403 (Forbidden). Otherwise the given value is emitted.

Requires usage of `HttpContextProviderAttribute`.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> CurrentUserHasClaim<TSource>(
    this IProviderObservable<TSource> observable,
    string claimType,
    string claim,
    object error)
```

If the check returns `false`, `ValidationException` is emitted as an error with the status code 403 (Forbidden). Otherwise the given value is emitted.

Requires usage of `HttpContextProviderAttribute`.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> CurrentUserHasClaim<TSource>(
    this IProviderObservable<TSource> observable,
    string claimType,
    Func<TSource,string> claimFactory,
    Func<TSource,object> errorFactory)
```

If the check returns `false`, `ValidationException` is emitted as an error with the status code 403 (Forbidden). Otherwise the given value is emitted.

Requires usage of `HttpContextProviderAttribute`.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> CurrentUserHasClaim<TSource>(
    this IProviderObservable<TSource> observable,
```

```
    string claimType,
    Func<TSource,string> claimFactory,
    object error)
```

3.13 DeleteEntity

Remove the received entity from the `DbContext` and save the change.

Package: FluentRestBuilder.EntityFrameworkCore

```
public static IProviderObservable<TSource> DeleteEntity<TSource>(
    this IProviderObservable<TSource> observable)
```

3.14 Do

Perform an action on the received value.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> Do<TSource>(
    this IProviderObservable<TSource> observable,
    Action<TSource> action)
```

3.15 DoAsync

Asynchronously perform an action on the received value.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> DoAsync<TSource>(
    this IProviderObservable<TSource> observable,
    Func<TSource,System.Threading.Tasks.Task> action)
```

3.16 First

Emits the first element of a sequence.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> First<TSource>(
    this IProviderObservable<IQueryable<TSource>> observable)
```

Emits the first element of a sequence that satisfies a specified condition.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> First<TSource>(
    this IProviderObservable<IQueryable<TSource>> observable,
    Expression<Func<TSource,System.Boolean>> predicate)
```

3.17 FirstAsync

Emits the first element of a sequence.

Package: FluentRestBuilder.EntityFrameworkCore

```
public static IProviderObservable<TSource> FirstAsync<TSource>(
    this IProviderObservable<IQueryable<TSource>> observable)
```

Emits the first element of a sequence that satisfies a specified condition.

Package: FluentRestBuilder.EntityFrameworkCore

```
public static IProviderObservable<TSource> FirstAsync<TSource>(
    this IProviderObservable<IQueryable<TSource>> observable,
    Expression<Func<TSource,System.Boolean>> predicate)
```

3.18 FirstOrDefault

Emits the first element of a sequence, or a default value if the sequence contains no elements.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> FirstOrDefault<TSource>(
    this IProviderObservable<IQueryable<TSource>> observable)
```

Emits the first element of a sequence that satisfies a specified condition or a default value if no such element is found.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> FirstOrDefault<TSource>(
    this IProviderObservable<IQueryable<TSource>> observable,
    Expression<Func<TSource,System.Boolean>> predicate)
```

3.19 FirstOrDefaultAsync

Emits the first element of a sequence, or a default value if the sequence contains no elements.

Package: FluentRestBuilder.EntityFrameworkCore

```
public static IProviderObservable<TSource> FirstOrDefaultAsync<TSource>(
    this IProviderObservable<IQueryable<TSource>> observable)
```

Emits the first element of a sequence that satisfies a specified condition or a default value if no such element is found.

Package: FluentRestBuilder.EntityFrameworkCore

```
public static IProviderObservable<TSource> FirstOrDefaultAsync<TSource>(
    this IProviderObservable<IQueryable<TSource>> observable,
    Expression<Func<TSource,System.Boolean>> predicate)
```

3.20 ForbiddenWhen

If the check returns `true`, `ValidationException` is emitted as an error with the status code 403 (Forbidden). Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> ForbiddenWhen<TSource>(
    this IProviderObservable<TSource> observable,
    Func<TSource,System.Boolean> invalidCheck,
    Func<TSource,object> errorFactory)
```

If the check returns `true`, `ValidationException` is emitted as an error with the status code 403 (Forbidden). Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> ForbiddenWhen<TSource>(
    this IProviderObservable<TSource> observable,
    Func<TSource,System.Boolean> invalidCheck,
    object error)
```

If the check returns `true`, `ValidationException` is emitted as an error with the status code 403 (Forbidden). Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> ForbiddenWhen<TSource>(
    this IProviderObservable<TSource> observable,
    Func<System.Boolean> invalidCheck,
    Func<TSource,object> errorFactory)
```

If the check returns `true`, `ValidationException` is emitted as an error with the status code 403 (Forbidden). Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> ForbiddenWhen<TSource>(
    this IProviderObservable<TSource> observable,
    Func<System.Boolean> invalidCheck,
    object error)
```

3.21 ForbiddenWhenAsync

If the check returns `true`, `ValidationException` is emitted as an error with the status code 403 (Forbidden). Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> ForbiddenWhenAsync<TSource>(
    this IProviderObservable<TSource> observable,
    Func<TSource,Task<System.Boolean>> invalidCheck,
    Func<TSource,object> errorFactory)
```

If the check returns `true`, `ValidationException` is emitted as an error with the status code 403 (Forbidden). Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> ForbiddenWhenAsync<TSource>(
    this IProviderObservable<TSource> observable,
    Func<TSource,Task<System.Boolean>> invalidCheck,
    object error)
```

If the check returns `true`, `ValidationException` is emitted as an error with the status code 403 (Forbidden). Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> ForbiddenWhenAsync<TSource>(
    this IProviderObservable<TSource> observable,
    Func<Task<System.Boolean>> invalidCheck,
    Func<TSource,object> errorFactory)
```

If the check returns `true`, `ValidationException` is emitted as an error with the status code 403 (Forbidden). Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> ForbiddenWhenAsync<TSource>(
    this IProviderObservable<TSource> observable,
    Func<Task<System.Boolean>> invalidCheck,
    object error)
```

3.22 GoneWhen

If the check returns `true`, `ValidationException` is emitted as an error with the status code 410 (Gone). Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> GoneWhen<TSource>(
    this IProviderObservable<TSource> observable,
    Func<TSource,System.Boolean> invalidCheck,
    Func<TSource,object> errorFactory)
```

If the check returns `true`, `ValidationException` is emitted as an error with the status code 410 (Gone). Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> GoneWhen<TSource>(
    this IProviderObservable<TSource> observable,
    Func<TSource,System.Boolean> invalidCheck,
    object error)
```

If the check returns `true`, `ValidationException` is emitted as an error with the status code 410 (Gone). Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> GoneWhen<TSource>(
    this IProviderObservable<TSource> observable,
```

```
    Func<System.Boolean> invalidCheck,
    Func<TSource,object> errorFactory)
```

If the check returns `true`, `ValidationException` is emitted as an error with the status code 410 (Gone). Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> GoneWhen<TSource>(
    this IProviderObservable<TSource> observable,
    Func<System.Boolean> invalidCheck,
    object error)
```

3.23 GoneWhenAsync

If the check returns `true`, `ValidationException` is emitted as an error with the status code 410 (Gone). Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> GoneWhenAsync<TSource>(
    this IProviderObservable<TSource> observable,
    Func<TSource,Task<System.Boolean>> invalidCheck,
    Func<TSource,object> errorFactory)
```

If the check returns `true`, `ValidationException` is emitted as an error with the status code 410 (Gone). Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> GoneWhenAsync<TSource>(
    this IProviderObservable<TSource> observable,
    Func<TSource,Task<System.Boolean>> invalidCheck,
    object error)
```

If the check returns `true`, `ValidationException` is emitted as an error with the status code 410 (Gone). Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> GoneWhenAsync<TSource>(
    this IProviderObservable<TSource> observable,
    Func<Task<System.Boolean>> invalidCheck,
    Func<TSource,object> errorFactory)
```

If the check returns `true`, `ValidationException` is emitted as an error with the status code 410 (Gone). Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> GoneWhenAsync<TSource>(
    this IProviderObservable<TSource> observable,
    Func<Task<System.Boolean>> invalidCheck,
    object error)
```

3.24 Include

Specifies related entities to include in the query results. The navigation property to be included is specified starting with the type of entity being queried (<typeparamref name="TSource" />). If you wish to include additional types based on the navigation properties of the type being included, then chain a call to `Func{``<TSource>,``2}})` after this call.

Package: FluentRestBuilder.EntityFrameworkCore

```
public static IProviderObservable<IIncludableQueryable<TSource,TProperty>> Include
↪<TSource, TProperty>(
    this IProviderObservable<IQueryable<TSource>> observable,
    Expression<Func<TSource,TProperty>> navigationPropertyPath)
```

3.25 InsertEntity

Add the received entity from the `DbContext` and save the change.

Package: FluentRestBuilder.EntityFrameworkCore

```
public static IProviderObservable<TSource> InsertEntity<TSource>(
    this IProviderObservable<TSource> observable)
```

3.26 InvalidWhen

If the check returns `true`, `ValidationException` is emitted as an error with the given status code. Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> InvalidWhen<TSource>(
    this IProviderObservable<TSource> observable,
    Func<TSource,System.Boolean> invalidCheck,
    int statusCode,
    object error)
```

If the check returns `true`, `ValidationException` is emitted as an error with the given status code. Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> InvalidWhen<TSource>(
    this IProviderObservable<TSource> observable,
    Func<System.Boolean> invalidCheck,
    int statusCode,
    Func<TSource,object> errorFactory)
```

If the check returns `true`, `ValidationException` is emitted as an error with the given status code. Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> InvalidWhen<TSource>(
    this IProviderObservable<TSource> observable,
    Func<System.Boolean> invalidCheck,
    int statusCode,
    object error)
```

If the check returns `true`, `ValidationException` is emitted as an error with the given status code. Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> InvalidWhen<TSource>(
    this IProviderObservable<TSource> observable,
    Func<TSource,System.Boolean> invalidCheck,
    int statusCode,
    Func<TSource,object> errorFactory)
```

3.27 InvalidWhenAsync

If the check returns `true`, `ValidationException` is emitted as an error with the given status code. Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> InvalidWhenAsync<TSource>(
    this IProviderObservable<TSource> observable,
    Func<TSource,Task<System.Boolean>> invalidCheck,
    int statusCode,
    object error)
```

If the check returns `true`, `ValidationException` is emitted as an error with the given status code. Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> InvalidWhenAsync<TSource>(
    this IProviderObservable<TSource> observable,
    Func<Task<System.Boolean>> invalidCheck,
    int statusCode,
    Func<TSource,object> errorFactory)
```

If the check returns `true`, `ValidationException` is emitted as an error with the given status code. Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> InvalidWhenAsync<TSource>(
    this IProviderObservable<TSource> observable,
    Func<Task<System.Boolean>> invalidCheck,
    int statusCode,
    object error)
```

If the check returns `true`, `ValidationException` is emitted as an error with the given status code. Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> InvalidWhenAsync<TSource>(
    this IProviderObservable<TSource> observable,
    Func<TSource,Task<System.Boolean>> invalidCheck,
    int statusCode,
    Func<TSource,object> errorFactory)
```

3.28 LoadCollection

Load a reference collection from the database.

Package: FluentRestBuilder.EntityFrameworkCore

```
public static IProviderObservable<TSource> LoadCollection<TSource, TProperty>(
    this IProviderObservable<TSource> observable,
    Expression<Func<TSource,IEnumerable<TProperty>>> propertyExpression)
```

3.29 LoadReference

Load a single reference from the database.

Package: FluentRestBuilder.EntityFrameworkCore

```
public static IProviderObservable<TSource> LoadReference<TSource, TProperty>(
    this IProviderObservable<TSource> observable,
    Expression<Func<TSource,TProperty>> propertyExpression)
```

3.30 Map

Map the received value to the desired output.

Package: FluentRestBuilder

```
public static IProviderObservable<TTarget> Map<TSource, TTarget>(
    this IProviderObservable<TSource> observable,
    Func<TSource,TTarget> mapping)
```

3.31 MapAsync

Asynchronously map the received value to the desired output.

Package: FluentRestBuilder

```
public static IProviderObservable<TTarget> MapAsync<TSource, TTarget>(
    this IProviderObservable<TSource> observable,
    Func<TSource,Task<TTarget>> mapping)
```

3.32 MapToQueryable

Map to a `IQueryable<TSource>` from the received `DbContext`. Use the `Set`<TSource>` method to select the appropriate `IQueryable<TSource>`.

Package: FluentRestBuilder.EntityFrameworkCore

```
public static IProviderObservable<IQueryable<TTarget>> MapToQueryable<TSource,␣
↪TTarget>(
    this IProviderObservable<TSource> observable,
    Func<Microsoft.EntityFrameworkCore.DbContext,IQueryable<TTarget>> mapping)
```

Map to a `IQueryable<TSource>` from the received `DbContext`. Use the `Set`<TSource>` method to select the appropriate `IQueryable<TSource>`.

Package: FluentRestBuilder.EntityFrameworkCore

```
public static IProviderObservable<IQueryable<TTarget>> MapToQueryable<TSource,␣
↪TTarget>(
    this IProviderObservable<TSource> observable,
    Func<TSource,Microsoft.EntityFrameworkCore.DbContext,IQueryable<TTarget>> mapping)
```

3.33 MapToRestCollection

Maps the entries of the received `IEnumerable<TSource>` according to the given mapping function and wraps the result in an `IRestEntity`.

Requires `HttpContextProviderAttribute` to be set.

Package: FluentRestBuilder.HypertextApplicationLanguage

```
public static IProviderObservable<FluentRestBuilder.HypertextApplicationLanguage.
↪IRestEntity> MapToRestCollection<TSource, TTarget>(
    this IProviderObservable<IEnumerable<TSource>> observable,
    Func<TSource,TTarget> mapping)
```

3.34 NotFoundWhen

If the check returns `true`, `ValidationException` is emitted as an error with the status code 404 (Not Found). Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> NotFoundWhen<TSource>(
    this IProviderObservable<TSource> observable,
    Func<TSource,System.Boolean> invalidCheck,
    Func<TSource,object> errorFactory)
```

If the check returns `true`, `ValidationException` is emitted as an error with the status code 404 (Not Found). Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> NotFoundWhen<TSource>(
    this IProviderObservable<TSource> observable,
    Func<TSource,System.Boolean> invalidCheck,
    object error)
```

If the check returns `true`, `ValidationException` is emitted as an error with the status code 404 (Not Found). Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> NotFoundWhen<TSource>(
    this IProviderObservable<TSource> observable,
    Func<System.Boolean> invalidCheck,
    Func<TSource,object> errorFactory)
```

If the check returns `true`, `ValidationException` is emitted as an error with the status code 404 (Not Found). Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> NotFoundWhen<TSource>(
    this IProviderObservable<TSource> observable,
    Func<System.Boolean> invalidCheck,
    object error)
```

3.35 NotFoundWhenAsync

If the check returns `true`, `ValidationException` is emitted as an error with the status code 404 (Not Found). Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> NotFoundWhenAsync<TSource>(
    this IProviderObservable<TSource> observable,
    Func<TSource,Task<System.Boolean>> invalidCheck,
    Func<TSource,object> errorFactory)
```

If the check returns `true`, `ValidationException` is emitted as an error with the status code 404 (Not Found). Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> NotFoundWhenAsync<TSource>(
    this IProviderObservable<TSource> observable,
    Func<TSource,Task<System.Boolean>> invalidCheck,
    object error)
```

If the check returns `true`, `ValidationException` is emitted as an error with the status code 404 (Not Found). Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> NotFoundWhenAsync<TSource>(
    this IProviderObservable<TSource> observable,
    Func<Task<System.Boolean>> invalidCheck,
    Func<TSource,object> errorFactory)
```

If the check returns `true`, `ValidationException` is emitted as an error with the status code 404 (Not Found). Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> NotFoundWhenAsync<TSource>(
    this IProviderObservable<TSource> observable,
    Func<Task<System.Boolean>> invalidCheck,
    object error)
```

3.36 NotFoundWhenNull

If the received value is null, `ValidationException` is emitted as an error with the status code 404 (Not Found). Otherwise the given value is emitted.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> NotFoundWhenNull<TSource>(
    this IProviderObservable<TSource> observable,
    Func<TSource,object> errorFactory)
```

3.37 OrderBy

Sorts the elements of a sequence in ascending order according to a key.

Package: FluentRestBuilder

```
public static IProviderObservable<IOrderedQueryable<TSource>> OrderBy<TSource, TKey>(
    this IProviderObservable<IQueryable<TSource>> observable,
    Expression<Func<TSource,TKey>> keySelector)
```

Sorts the elements of a sequence in ascending order by using a specified comparer.

Package: FluentRestBuilder

```
public static IProviderObservable<IOrderedQueryable<TSource>> OrderBy<TSource, TKey>(
    this IProviderObservable<IQueryable<TSource>> observable,
    Expression<Func<TSource,TKey>> keySelector,
    IComparer<TKey> comparer)
```

3.38 OrderByDescending

Sorts the elements of a sequence in descending order according to a key.

Package: FluentRestBuilder

```
public static IProviderObservable<IOrderedQueryable<TSource>> OrderByDescending
↪<TSource, TKey>(
    this IProviderObservable<IQueryable<TSource>> observable,
    Expression<Func<TSource,TKey>> keySelector)
```

Sorts the elements of a sequence in descending order by using a specified comparer.

Package: FluentRestBuilder

```
public static IProviderObservable<IOrderedQueryable<TSource>> OrderByDescending
↪<TSource, TKey>(
    this IProviderObservable<IQueryable<TSource>> observable,
    Expression<Func<TSource,TKey>> keySelector,
    IComparer<TKey> comparer)
```

3.39 ReloadEntity

Reloads the entity from the database overwriting any property values with values from the database.

The entity will be in the `Unchanged` state after calling this method, unless the entity does not exist in the database, in which case the entity will be `Detached`. Finally, calling Reload on an `Added` entity that does not exist in the database is a no-op. Note, however, that an Added entity may not yet have had its permanent key value created.

Package: FluentRestBuilder.EntityFrameworkCore

```
public static IProviderObservable<TSource> ReloadEntity<TSource>(
    this IProviderObservable<TSource> observable)
```

3.40 RemoveDistributedCacheEntry

Remove a cache entry from the `IDistributedCache` with the key generated by the key factory function.

Package: FluentRestBuilder.Caching

```
public static IProviderObservable<TSource> RemoveDistributedCacheEntry<TSource>(
    this IProviderObservable<TSource> observable,
    Func<TSource,string> keyFactory)
```

Remove a cache entry from the `IDistributedCache` with the key generated by the key factory function.

Package: FluentRestBuilder.Caching

```
public static IProviderObservable<TSource> RemoveDistributedCacheEntry<TSource>(
    this IProviderObservable<TSource> observable,
    string key)
```

3.41 RemoveMemoryCacheEntry

Remove a cache entry from the `IMemoryCache` with the key generated by the key factory function.

Package: FluentRestBuilder.Caching

```
public static IProviderObservable<TSource> RemoveMemoryCacheEntry<TSource>(
    this IProviderObservable<TSource> observable,
    Func<TSource,object> keyFactory)
```

Remove a cache entry from the `IMemoryCache` with the given key.

Package: FluentRestBuilder.Caching

```
public static IProviderObservable<TSource> RemoveMemoryCacheEntry<TSource>(
    this IProviderObservable<TSource> observable,
    object key)
```

3.42 SaveChangesAsync

Save changes to the `DbContext` asynchronously.

Package: FluentRestBuilder.EntityFrameworkCore

```
public static IProviderObservable<TSource> SaveChangesAsync<TSource>(
    this IProviderObservable<TSource> observable)
```

3.43 Single

Emits the only element of a sequence, and throws an exception if there is not exactly one element in the sequence.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> Single<TSource>(
    this IProviderObservable<IQueryable<TSource>> observable)
```

Emits the only element of a sequence that satisfies a specified condition, and throws an exception if more than one such element exists.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> Single<TSource>(
    this IProviderObservable<IQueryable<TSource>> observable,
    Expression<Func<TSource,System.Boolean>> predicate)
```

3.44 SingleAsync

Emits the only element of a sequence, and throws an exception if there is not exactly one element in the sequence.

Package: FluentRestBuilder.EntityFrameworkCore

```
public static IProviderObservable<TSource> SingleAsync<TSource>(
    this IProviderObservable<IQueryable<TSource>> observable)
```

Emits the only element of a sequence that satisfies a specified condition, and throws an exception if more than one such element exists.

Package: FluentRestBuilder.EntityFrameworkCore

```
public static IProviderObservable<TSource> SingleAsync<TSource>(
    this IProviderObservable<IQueryable<TSource>> observable,
    Expression<Func<TSource,System.Boolean>> predicate)
```

3.45 SingleOrDefault

Emits the only element of a sequence, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> SingleOrDefault<TSource>(
    this IProviderObservable<IQueryable<TSource>> observable)
```

Emits the only element of a sequence that satisfies a specified condition or a default value if no such element exists; this method throws an exception if more than one element satisfies the condition.

Package: FluentRestBuilder

```
public static IProviderObservable<TSource> SingleOrDefault<TSource>(
    this IProviderObservable<IQueryable<TSource>> observable,
    Expression<Func<TSource,System.Boolean>> predicate)
```

3.46 SingleOrDefaultAsync

Emits the only element of a sequence, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.

Package: FluentRestBuilder.EntityFrameworkCore

```
public static IProviderObservable<TSource> SingleOrDefaultAsync<TSource>(
    this IProviderObservable<IQueryable<TSource>> observable)
```

Emits the only element of a sequence that satisfies a specified condition or a default value if no such element exists; this method throws an exception if more than one element satisfies the condition.

Package: FluentRestBuilder.EntityFrameworkCore

```
public static IProviderObservable<TSource> SingleOrDefaultAsync<TSource>(
    this IProviderObservable<IQueryable<TSource>> observable,
    Expression<Func<TSource,System.Boolean>> predicate)
```

3.47 ThenBy

Performs a subsequent ordering of the elements in a sequence in ascending order according to a key.

Package: FluentRestBuilder

```
public static IProviderObservable<IOrderedQueryable<TSource>> ThenBy<TSource, TKey>(
    this IProviderObservable<IOrderedQueryable<TSource>> observable,
    Expression<Func<TSource,TKey>> keySelector)
```

Performs a subsequent ordering of the elements in a sequence in ascending order by using a specified comparer.

Package: FluentRestBuilder

```
public static IProviderObservable<IOrderedQueryable<TSource>> ThenBy<TSource, TKey>(
    this IProviderObservable<IOrderedQueryable<TSource>> observable,
    Expression<Func<TSource,TKey>> keySelector,
    IComparer<TKey> comparer)
```

3.48 ThenByDescending

Performs a subsequent ordering of the elements in a sequence in descending order, according to a key.

Package: FluentRestBuilder

```
public static IProviderObservable<IOrderedQueryable<TSource>> ThenByDescending
↪<TSource, TKey>(
    this IProviderObservable<IOrderedQueryable<TSource>> observable,
    Expression<Func<TSource,TKey>> keySelector)
```

Performs a subsequent ordering of the elements in a sequence in descending order by using a specified comparer.

Package: FluentRestBuilder

```
public static IProviderObservable<IOrderedQueryable<TSource>> ThenByDescending
↪<TSource, TKey>(
    this IProviderObservable<IOrderedQueryable<TSource>> observable,
    Expression<Func<TSource,TKey>> keySelector,
    IComparer<TKey> comparer)
```

3.49 ThenInclude

Specifies additional related data to be further included based on a related type that was just included.

Package: FluentRestBuilder.EntityFrameworkCore

```
public static IProviderObservable<IIncludableQueryable<TSource,TProperty>> ThenInclude
↪<TSource, TPreviousProperty, TProperty>(
    this IProviderObservable<IIncludableQueryable<TSource,TPreviousProperty>>␣
↪observable,
    Expression<Func<TPreviousProperty,TProperty>> navigationPropertyPath)
```

3.50 ToAcceptedObjectResult

Wrap the received value in an `ObjectResult` with status code 202 (Accepted).

Catches `ValidationException` and converts it to an appropriate `IActionResult`.

Package: FluentRestBuilder

```
public static IProviderObservable<Microsoft.AspNetCore.Mvc.IActionResult>␣
↪ToAcceptedObjectResult<TSource>(
    this IProviderObservable<TSource> observable)
```

3.51 ToActionResult

Convert the received value into an `IActionResult`.

Catches `ValidationException` and converts it to an appropriate `IActionResult`.

Package: FluentRestBuilder

```
public static IProviderObservable<Microsoft.AspNetCore.Mvc.IActionResult>␣
↪ToActionResult<TSource>(
    this IProviderObservable<TSource> observable,
    Func<TSource,Microsoft.AspNetCore.Mvc.IActionResult> mapping)
```

3.52 ToCreatedAtRouteResult

Wrap the received value in an `CreatedAtRouteResult` with status code 201 (Created).

Catches `ValidationException` and converts it to an appropriate `IActionResult`.

Package: FluentRestBuilder

```
public static IProviderObservable<Microsoft.AspNetCore.Mvc.IActionResult>␣
↪ToCreatedAtRouteResult<TSource>(
    this IProviderObservable<TSource> observable,
    string routeName,
    Func<TSource,object> routeValuesFactory)
```

Wrap the received value in an `CreatedAtRouteResult` with status code 201 (Created).

Catches `ValidationException` and converts it to an appropriate `IActionResult`.

Package: FluentRestBuilder

```
public static IProviderObservable<Microsoft.AspNetCore.Mvc.IActionResult>␣
↪ToCreatedAtRouteResult<TSource>(
    this IProviderObservable<TSource> observable,
    Func<TSource,object> routeValuesFactory)
```

3.53 ToList

Creates and emits a <see cref="T:System.Collections.Generic.List`1"></see> from an <see cref="T:System.Collections.Generic.IEnumerable`1"></see>.

Package: FluentRestBuilder

```
public static IProviderObservable<List<TSource>> ToList<TSource>(
    this IProviderObservable<IEnumerable<TSource>> observable)
```

3.54 ToListAsync

Asynchronously creates a `List<TSource>` from an `IQueryable<TSource>` by enumerating it asynchronously.

Package: FluentRestBuilder.EntityFrameworkCore

```
public static IProviderObservable<List<TSource>> ToListAsync<TSource>(
    this IProviderObservable<IQueryable<TSource>> observable)
```

3.55 ToNoContentResult

Emits `NoContentResult` on receiving a value. Does not contain the value.

Catches `ValidationException` and converts it to an appropriate `IActionResult`.

Package: FluentRestBuilder

```
public static IProviderObservable<Microsoft.AspNetCore.Mvc.IActionResult>␣
↪ToNoContentResult<TSource>(
    this IProviderObservable<TSource> observable)
```

3.56 ToOkObjectResult

Wrap the received value in an `OkObjectResult`.

Catches `ValidationException` and converts it to an appropriate `IActionResult`.

Package: FluentRestBuilder

```
public static IProviderObservable<Microsoft.AspNetCore.Mvc.IActionResult>␣
↪ToOkObjectResult<TSource>(
    this IProviderObservable<TSource> observable)
```

3.57 ToOptionsResult

Emits an `OptionsResult` which lists the allowed HTTP verbs.

Catches `ValidationException` and converts it to an appropriate `IActionResult`.

Package: FluentRestBuilder

```
public static IProviderObservable<Microsoft.AspNetCore.Mvc.IActionResult>␣
↪ToOptionsResult<TSource>(
    this IProviderObservable<TSource> observable,
    Func<TSource,IEnumerable<FluentRestBuilder.HttpVerb>> verbsFactory)
```

Emits an `OptionsResult` which lists the allowed HTTP verbs.

Catches `ValidationException` and converts it to an appropriate `IActionResult`.

Requires usage of `HttpContextProviderAttribute`.

Package: FluentRestBuilder

```
public static IProviderObservable<Microsoft.AspNetCore.Mvc.IActionResult>␣
↪ToOptionsResult<TSource>(
    this IProviderObservable<TSource> observable,
    Func<AllowedOptionsBuilder<TSource>,AllowedOptionsBuilder<TSource>> factory)
```

3.58 Where

Filters a sequence of values based on a predicate.

Package: FluentRestBuilder

```
public static IProviderObservable<IQueryable<TSource>> Where<TSource>(
    this IProviderObservable<IQueryable<TSource>> observable,
    Expression<Func<TSource,System.Boolean>> predicate)
```

Filters a sequence of values based on a predicate. Each element's index is used in the logic of the predicate function.

Package: FluentRestBuilder

```
public static IProviderObservable<IQueryable<TSource>> Where<TSource>(
    this IProviderObservable<IQueryable<TSource>> observable,
    Expression<Func<TSource,int,System.Boolean>> predicate)
```

3.59 WithEntityEntry

Perform an action with the `EntityEntry<TSource>` of the received value.

Package: FluentRestBuilder.EntityFrameworkCore

```
public static IProviderObservable<TSource> WithEntityEntry<TSource>(
    this IProviderObservable<TSource> observable,
    Action<EntityEntry<TSource>> action)
```

3.60 WithEntityEntryAsync

Perform an async action with the `EntityEntry<TSource>` of the received value.

Package: FluentRestBuilder.EntityFrameworkCore

```
public static IProviderObservable<TSource> WithEntityEntryAsync<TSource>(
    this IProviderObservable<TSource> observable,
    Func<EntityEntry<TSource>,System.Threading.Tasks.Task> action)
```

CHAPTER 4

Pagination

FluentRestBuilder provides operators to implement pagination (filtering, sorting, offset and limit) for `IQueryable<TSource>`.

There are four pagination operators for different purposes.

Operator	Effect
ApplyFilterByClientRequest	Applies Where clauses to the `IQueryable<TSource>`
ApplyOrderByClientRequest	Applies OrderBy clauses to the `IQueryable<TSource>`
ApplyPaginationByClientRequest	Applies Skip and Take to the `IQueryable<TSource>`
ApplySearchByClientRequest	Applies a single Where clause to the `IQueryable<TSource>`

4.1 Naming Convention

FluentRestBuilder tries to figure out the naming convention from the MVC JSON configuration. By default (of MVC) this follows camelCasing.

Implement `FluentRestBuilder.Json.IJsonPropertyNameResolver` and register it as a singleton in order to provide your own name convention logic.

4.2 Interpreters

Each operator has its own interpreter for interpreting a client request.

4.2.1 Filter Interpreter

The default filter interpreter parses the query paramters of a request. The interpreter (not the operator) supports the following filter types:

Prefix (The filter value must start with this)	Type	Example
No prefix	Default (Depends on configuration)	field=search
~	Contains	field=~search
<=	LessThanOrEqual	field=<=5
>=	GreaterThanOrEqual	field=>=5
<	LessThan	field=<5
>	GreaterThan	field=>5
=	Equals	field==search
^=	StartsWith	field=^=search
$=	EndsWith	field=$=search
!=	NotEqual	field=!=search

This means in order to search for a field value equal to =, the query parameter value must contain `=%3D` (...&field==%3D&...), as the actual search value (=) must be encoded. The reason for this is that MVC Core cannot interpret `...&field===&...`, but it can interpret `...&field==%3D` or `...&field===a&...`.

In order to implement your own filter interpreter implement `FluentRestBuilder.Operators.ClientRequest.Interpreters.IFilterByClientRequestInterpreter`.

Example:

...&fieldA=a&fieldC=~jo&fieldE=<5&fieldG=>=3&...

4.2.2 Order By Interpreter

The default order by interpreter parses the query paramter "sort" (or "Sort" depending on the naming convention). The value is split by comma. To order by descending prepend the field with "-".

In order to implement your own order by interpreter implement `FluentRestBuilder.Operators.ClientRequest.Interpreters.IOrderByClientRequestInterpreter`.

Example:

...&sort=fieldA,-fieldC,fieldF&...

4.2.3 Pagination Interpreter

The default pagination interpreter parses the query parameters "limit" and "offset" (or "Limit" and "Offset" depending on the naming convention) or the Range header.

Range Header: `Range: items=10-29` is the same as ...&offset=10&limit=20&...

In order to implement your own pagination interpreter implement `FluentRestBuilder.Operators.ClientRequest.Interpreters.IPaginationByClientRequestInterpreter`.

4.2.4 Search Interpreter

The default search interpreter uses the query parameter "q" (or "Q" depending on the naming convention).

In order to implement your own search interpreter implement `FluentRestBuilder.Operators.ClientRequest.Interpreters.ISearchByClientRequestInterpreter`.

4.3 Filter Providers

The `ApplyFilterByClientRequest` operator requires an instance of `IDictionary<string, IFilterExpressionProvider<TSource>>` or `IFilterExpressionProviderDictionary<TSource>` (which is a wrapper for the first).

This can be achieved by either providing it to the operator directly as a parameter or by adding a service as `IFilterExpressionProviderDictionary<TSource>` to the dependency injection container.

There are two extension methods on the `IFluentRestBuilderConfiguration` to configure filter providers (and order by expressions) via reflection.

```
public void ConfigureServices(IServiceCollection services)
{
    ...

    services.AddFluentRestBuilder()
        // Configures filter provider (and order by expressions) for an entity
        .ConfigureFiltersAndOrderByExpressionsForEntity<ExampleEntity>()
        // Configures filter providers (and order by expressions) for all
        // entities in the specified DbContext
        // Requires the FluentRestBuilder.EntityFrameworkCore package
        .AddEntityFrameworkCoreIntegration<ApplicationDbContext>() // Required for EF␣
↪Core operators to work
        .ConfigureFiltersAndOrderByExpressionsForDbContextEntities
↪<ApplicationDbContext>();
}
```

Alternatively the `FilterExpressionProviderDictionary<TSource>` class can be used, either in the `ApplyFilterByClientRequest` overload or when registering it as a service in the dependency injection container.

4.4 Order By Expressions

The `ApplyOrderByClientRequest` operator requires an instance of `IDictionary<string, IOrderByExpressionFactory<TSource>>` or `IOrderByExpressionDictionary<TSource>` (which is a wrapper for the first).

This can be achieved by either providing it to the operator directly as a parameter or by adding a service as `IOrderByExpressionDictionary<TSource>` to the dependency injection container.

There are two extension methods on the `IFluentRestBuilderConfiguration` to configure order by expressions (and filter providers) via reflection.

```
public void ConfigureServices(IServiceCollection services)
{
    ...

    services.AddFluentRestBuilder()
        // Configures order by expressions (and filter providers) for an entity
        .ConfigureFiltersAndOrderByExpressionsForEntity<ExampleEntity>()
        // Configures order by expressions (and filter providers) for all
        // entities in the specified DbContext
        // Requires the FluentRestBuilder.EntityFrameworkCore package
        .AddEntityFrameworkCoreIntegration<ApplicationDbContext>() // Required for EF␣
↪Core operators to work
```

```
        .ConfigureFiltersAndOrderByExpressionsForDbContextEntities
↪<ApplicationDbContext>();
}
```

Alternatively the `OrderByExpressionDictionary<TSource>` class can be used, either in the `ApplyOrderByClientRequest` overload or when registering it as a service in the dependency injection container.