mv.NET and REST

RESTful Service Developer Guide

A product from BlueFinity

Copyright Notices

Document ref: mvNET_RS_DG
Revision 4.5.0

Contacting Us

We are always very happy to be able to discuss all aspects of our products with our customers - prospective and current alike. You can contact us via the following means:

Website: www.bluefinity.com
Email: support@bluefinity.com
Address: 10260 SW Greenburg Road, Suite 700, Portland, OR 97223, USA
Address: Hamilton House, 111 Marlowes, Hemel Hempstead, Herts, HP1 1BB, UK

Trademark Acknowledgements

Contents

Welcome to mv.NET

Firstly, thank you for either purchasing one or more of the mv.NET products or for taking the time to explore the great functionality that they can provide to you and your fellow developers.

This chapter outlines the members of the mv.NET family of products and also summarizes the contents of this guide.

The mv.NET Family of Products

mv.NET is ***the*** essential tool for any MultiValue database developer wishing to create .NET based application interfaces to their current or new MultiValue database file system.

The design goal of mv.NET is to enable the MultiValue developer to combine the power and flexibility of proven MultiValue technology with the state-of-the art, feature rich .NET environment. Its design also enables and encourages the developer to leverage, wherever possible, previously acquired MultiValue skills.

BlueFinity's team of software engineers has huge knowledge and experience of using both MultiValue systems and the .NET environment. We proudly regard ourselves as being one of the foremost companies in providing this technology bridge and look forward to working with you to enable you to meet your software development goals.

Feature Overview

The REST integration components that are supplied as part of the standard mv.NET product include:

- Extensions to the Solution Objects entity modeling tool to allow the generated business access layer to be used inside a RESTful web service environment.
- A "REST wizard" to allow quick creation of:
  - Visual Studio RESTful service solutions
  - Client-side data access code.

Note, the RESTful services generated by mv.NET can be consumed by any type of client platform - .NET, Java, iOS, Android or any other of the many development environments able to execute standard HTTP requests.

The mv.NET Suite

The mv.NET suite of products comprises:

- **Core Objects** - object oriented native .NET access to MultiValue databases.
- **Solution Objects** - Strongly-typed class-based access to your MultiValue database.
- **Adapter Objects** - complete implementation of an ADO.NET managed data provider for MultiValue databases, offering a standardized interface to database access.

Developer Guide Contents

The contents of this guide are designed to provide a basis for learning about how mv.NET can be used to create RESTful web service interfaces to your MultiValued database using Microsoft's ASP.NET environment.

Assumptions

This guide, through necessity, makes some assumptions about your skill level and software install base. Specifically, it assumes that:

1. You have already installed and configured mv.NET to connect into your database server. Details on how to do this can be found in the accompanying Getting Started and Core Objects guides.
2. You have created the necessary extended dictionary definitions for your data files. Details on how to do this can be found in the accompanying Core Objects developer guide.
3. You have created an entity model using the Solution Objects component of mv.NET. Details on how to do this can be found in the accompanying Solution Objects developer guide.
4. You have installed Microsoft Visual Studio 2010 or 2012 and that you are reasonably familiar with its layout and general functioning.

REST Technical Overview

REST is an acronym with which most developers are now familiar. It stands for REpresentational State Transfer and is an architectural methodology for structuring the invocation of remote functionality hosted by a "server" by an end "client".

As Wikipedia puts it *"REST was initially described in the context of HTTP, but it is not limited to that protocol. RESTful architectures may be based on other Application Layer protocols if they already provide a rich and uniform vocabulary for applications based on the transfer of meaningful representational state. RESTful applications maximize the use of the existing, well-defined interface and other built-in capabilities provided by the chosen network protocol, and minimize the addition of new application-specific features on top of it."*

Thus, REST in the context of this manual (as in most other contexts) refers to REST over HTTP using the standard HTTP verbs.

There is a wealth of documentation on REST on the Web and thus this chapter provides a very brief overview of the technology in order to set the backdrop for the rest of this guide.

An internet search on the phrase "RESTful web services" will produce many excellent links to more in depth discussions of the topic.

REST Fundamentals

Put succinctly, the use of a RESTful style interface over HTTP means that the standard HTTP verbs are used to perform the action (and only the action) that each one was originally designed to do. Looking at the 4 main verbs:

- To create a resource on the server, use POST.
- To retrieve a resource, use GET.
- To change the state of a resource or to update it, use PUT.
- To remove or delete a resource, use DELETE.

A basic tenet of REST is that the same GET, DELETE and PUT action can be repeated multiple times with the same effect each time. A POST should only be invoked once and may produce different (presumably undesirable) effects if so repeated.

Should I Use REST?

REST is not necessarily the right choice for all situations. It has caught on as a way to design Web services with less dependence on proprietary middleware when compared with other alternatives such as SOAP.

In many ways, REST is a return back to the original basic principles of the Web; a return back to how it was before the advent of the large application servers able to store/restore state across HTTP calls in a desire to ease the task of writing non-trivial Web applications.

Many people are now adopting the principle of REST being the default implementation pattern for a web service. Only if there are overarching reasons for not using REST will other patterns be considered

mv.NET's REST-based Functionality

mv.NET contains a number of aspects that are centered on the task of creating a RESTful web service. This chapter takes you through each of these in turn.

Entity Model Integration Overview

mv.NET's REST functionality is ultimately based upon "Entity Models" created using the product's Solution Objects component.

Thus, a prerequisite for using the REST-based functionality within mv.NET is that you must use mv.NET's Solution Objects component (hosted within the product's Data Manager utility) to create an entity model of the data which you ultimately wish to expose via your RESTful web service. If you already have an entity model, you are already in a position to quickly and easily create a RESTful service.

If you are not familiar with mv.NET's Solution Objects and its entity modeling features, please refer to the Solution Objects Developer Guide for more information.

The ability to create custom projections of your entity model content through the use of multiple "Business Access Classes" (BACs) and "Business Access Layers" (BALs) within Solution Objects is a key aspect when looking at RESTful services. This is because the BACs and BALs provide a simple yet highly effective way to selectively isolate which parts of your database you wish to expose over a given RESTful service API.

Entity Model Integration Detail

Having got yourself familiar with Solution Objects and having created an entity model using the Data Manager, you are ready to use the REST-based features of mv.NET.

The basic principle is that you should create a Business Access Layer dedicated to exposing the sections of your entity model (and only those sections) that are required for each of your web services. You may well also want to create some additional Business Access Classes to restrict the properties supported by each entity included in your web service BAL.

The Business Access Layer definition form within the Entity Modeling section of the Data Manager allows you to specify the target runtime environment. You should select the "REST service" option. Please refer to the Business Access Layer chapter of Solution Objects Developer Guide for more details on this topic.

The "REST service" option will force the code generator to produce data access code that is capable of linking with a REST-based API.

Generating Entity Code

The entity modeling's code generation form allows you to generate VB.NET or C# code based on one or more of your BALs.

The code generator form also allows you to create a complete Visual Studio solution to host your generated code. It is the assembly built from this generated code that will be referenced by your Visual Studio web service project in order to link your web service to your database server.

Please refer to the Generating Code Modules chapter of Solution Objects Developer Guide for more details on this topic.

The Use of ASP.NET

The Visual Studio web service projects generated by the Data Manager are based on the standard ASP.NET MVC 3 project template, with all UI aspects removed.

MVC is used because of its powerful yet simple to use ability to route incoming URIs to the appropriate service logic – which is arguably one of the most important fundamental requirements of a RESTful web service.

There is, however, nothing to stop you from using a totally different approach to creating a .NET connected web service; in which case you will ultimately call into the relevant parts of the entity model data access assembly in exactly the same way as the ASP.NET MVC approach.

The REST Wizard

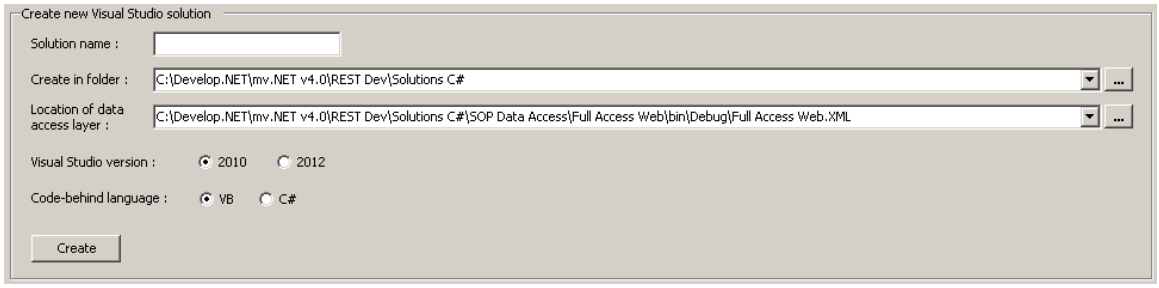
The top menu bar of the Data Manager contains a "REST Wizard" option which launches the screen responsible for allowing you to easily create ASP.NET-based RESTful web services.

At the top of the REST Wizard form are 3 radio buttons which allow you to:

- Generate a complete ASP.NET Visual Studio solution hosting a web service
- Update a previous generated ASP.NET solution
- Generate source code for use in various client-side environments

Generating a New RESTful Visual Studio Solution

Selecting this option results in the following being displayed:



In this screen you are able to specify the name of your solution along with the location in which the top-level solution folder will be created. You are also able to select the xml file generated by Visual Studio when you generated the data access assembly that you wish to use inside this web service.

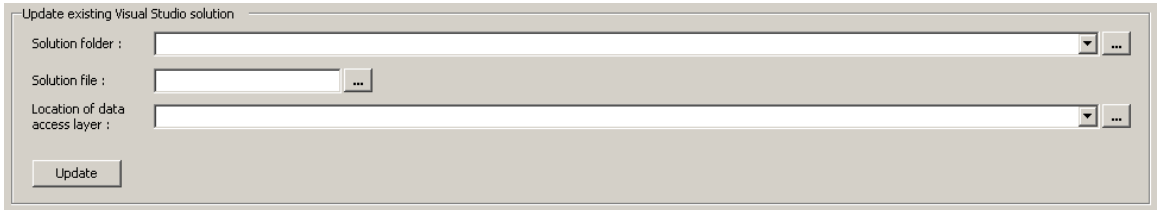
Additionally you are able to select the version of Visual Studio that you intend to use as well as your programming language of choice.

On clicking the Create button the Visual Studio solution will be created and you will be offered the option of invoking Visual Studio on the newly generated solution.

Updating a Previously Generated Visual Studio Solution

If you add or remove entities from the entity model upon which your web service is based, you will need to regenerate the previously generated MVC controller code used within the Visual Studio project. This option allows you to do this.

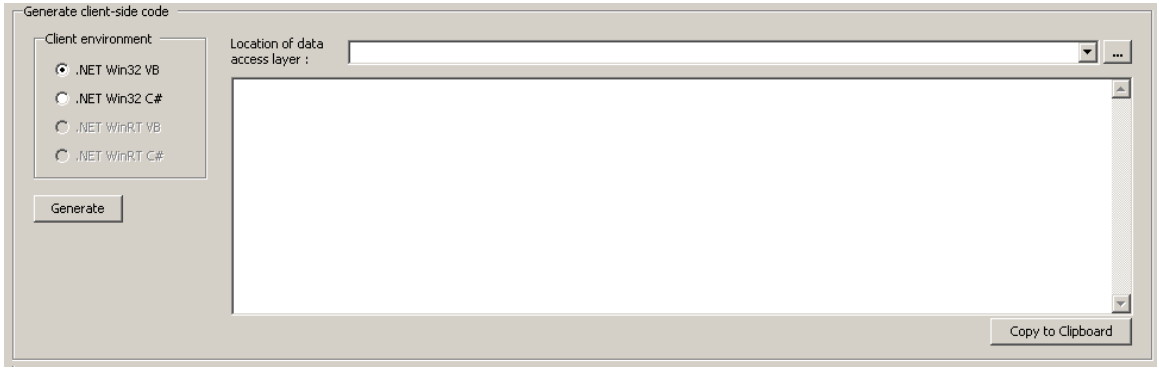
Selecting the update option results in the following being displayed:



This screen allows you to identify the location of your previously generated Visual Studio solution. You may select an alternative data access xml file if necessary.

Generating Client-side Code

If you area creating a client application (consumer) of a RESTful web service in any of the environments listed in the left hand side of the panel displayed when this option is selected, you are able to generate the client-side class definition code based upon the underlying entity model:



Once the Generate button has been clicked, the generated code will appear in the center of the form and you can then grab this content onto your clipboard by clicking the Copy to Clipboard button in the bottom right corner of the form.

The Anatomy of a RESTful Web Service

The Visual Studio solutions generated by the REST Wizard follow a predefined structure. This chapter takes you through this structure and explains how you can safely customize or extend it as required.

Generated Solution Structure

The REST Wizard generates a Visual Studio solution based on the standard ASP.NET MVC 3 project template. The UI related aspects of this template are removed to leave just the controller part of the project.

The reason for using MVC is purely to take advantage of its excellent URI routing capabilities. This allows an intuitive URI API syntax to be used to invoke the REST service functionality.

Route Registration

Within the Global.asax code file there is a MapRoute call within the RegisterRoutes subroutine. This tells the MVC routing mechanism about the syntax of the REST service's URI scheme. The URI structure is covered in a later section in this chapter.

Controller Code

Initially in the Controllers folder of the project are 2 code files - APIController and APIControllerCustom. These files contain the class definitions that will be used by the MVC URI routing mechanism and it is by the use of these classes that a connection to the entity model classes is established.

The content of the APIController file is "owned" by the REST Wizard in that its entire content will be regenerated if you update the project (using the REST Wizard) at some future point in time.

The content of the APIControllerCustom file is owned by you. Its content (after initial creation) will not be altered by the REST Wizard's code generator. The purpose of the custom code file is discussed in the following section.

The APIController code file contains one class per entity contained within the source Business Access Layer of the source entity model. Each of these classes subclasses the MVC "Controller" class and contains a function method for each of the 4 HTTP verbs supported by the REST service (GET, PUT POST and DELETE). Each of the methods calls into the code generated by the Solution Objects code generator in order to interact with the back-end database.

Customizing/Extending Functionality

Each of the 4 "HTTP verb functions" in each of the classes within the APIController code file contains 2 lines of code, the first of which is a call into the APIControllerCustom code file.

This structure allows you to pre-parse the incoming request in order to perform any checks, such as inspection of header information for security information. For this reason the entire incoming request is passed into the custom routine.

If you do not wish the request to be processed in the normal manner an exception should be thrown within the custom code routine.

Data Format

The code generated by REST Wizard is based around the creation of a REST service that supplies and receives data in JSON format. This is fast becoming the de-facto standard for data interchange.

If you do not wish to use JSON you can alter the REST Wizard's code generator template to utilize a different serialization component (e.g. XML). This template can be found in location:

```
C:\Program Files (x86)\BlueFinity\mv.NET\Version4.0\Code Templates\Solutions\VS2012\C#{or VB}\REST Web Service\Controllers\ Controller Template.txt
```

Basic URI Composition

The line of code in the Global.asax code file which performs the MapRoute call within the RegisterRoutes subroutine is the place where the basic structure of the URI of the REST service is established.

The default is based on the pattern of:

http://{domain}/restapi/{entity}

If you wish to use a different pattern, the MapRoute call can be changed as required.

Accessing Multi and Sub Valued Data

If nested data items are exposed by your entity model, you can read individual multivalue and subvalue positions via REST calls.

For example, in the sample SOP database there is an entity called SalesOrderLine, this represents a multivalue associated group in the SALESORDER file.

In order to retrieve a single SalesOrderLine instance, a URI of the form show below should be used:

```
http://localhost/SOPData/restapi/SalesOrderLine/4?Line=2
```

In this example the second order line of sales order 4 will be retrieved. It can be seen that the required multivalue position is specified via a standard query string parameter in the URI. The word “Line” has been used in the above example – although, in fact, any word could have been used, the important piece of data is that which appears after the equal sign. Thus, the following URI would render the same result:

```
http://localhost/SOPData/restapi/SalesOrderLine/4?row=2
```

In order to retrieve an individual subvalue set of data using an entity that exposes a subvalue associated group, the same principle is used:

```
http://localhost/SOPData/restapi/SalesDelivery/4?Line=1&Delivery=2
```

In this example the second delivery of order line#1 of sales order 4 will be retrieved. Again, the words “Line” and “Delivery” can be replaced with any word.

Selecting Items

As well as performing single record-based CRUD operations, the RESTful interface generated by mv.NET allows multiple records to be selected. These selected records will be returned in the form of an array of JSON objects.

The URI used for selecting multiple entity instances uses the collective name of the entity and also references the name of a selection method defined within your entity model. For example:

```
http://localhost/SOPData/restapi/SalesOrders/SelectByCustomer?Key=1
```

In this example the SelectByCustomer selection method defined for the SalesOrder entity is being accessed. Currently, only selection methods with a single argument can be accessed via the REST interface, although there are ways of allowing multiple selection details to be handled (see below).

In the above example, the SelectByCustomer selection method accepts the item ID of the required customer ("Key=1"). As in single record reading, the word preceding the equal sign is not relevant, only the value following the equal sign will be used. Thus, the above URI will retrieve all sales orders for customer 1.

Complex Dynamic Selection Criteria Support

If you wish to provide a mechanism for consumers of your REST service to provide multiple (and possibly a variable number of) selection criteria elements via the URI, you will need to create a selection method with a single general purpose argument. This argument is then used to pass a complete selection criteria clause into the selection process.

Also, using the "BeforeSelection" method contained in the custom code file associated with the code generated from your entity model, you are able to parse the incoming selection criteria to ensure that any single request is not going to overload your database server with an overly large selection task.

Using the above technique, it is easy to safely support URI's of the kind:

```
http://localhost/SOPData/restapi/SalesOrders?Customer EQ "1" And
DatePlaced GT "1/1/2010"
```

Tips and Recommendations

This section contains a number of tips and recommendations for when you are creating a RESTful service.

Object Nesting

With REST, there is no such thing as "lazy data loading". This means that all of the data for a request will be assembled and delivered in a single logical roundtrip to the service.

It is very common for entity model classes to contain properties that reference other entities. This fact needs careful consideration in the context of a RESTful service, in that:

- you need to ensure that there are no cyclic references via properties within your classes
- you need to consider and control exactly how "deep" the recursive nature of data serialization goes when processing a request

The good news here is that Solution Objects' BAC and BAL give you all of the necessary control to make sure that the above 2 pitfalls are avoided.

The main mechanism is the BAC. This allows you to create custom projections of your entities specifically for your REST service and then within these custom projections allows you both omit object reference properties and also non-required properties. This, thus, allows you to avoid cyclic entity references and also to prevent unwanted related entity data to be inadvertently retrieved as part of a request.

The BAL allows you to gather together these custom projections into a single assembly for use by the REST service

Date Handling

Unfortunately, there is currently no standard for how date values are serialized within a JSON payload. Therefore, we recommend that you pass date values as strings in order to avoid parsing mismatches by the client and server JSON parsers.