Rest DataWare Course

A fast way to serve and consume data with Delphi and Lazarus

PART I, II of V

Author: Ricardo da Rocha

2019

Abstract

This is a serie of many articles I wrote about Rest Dataware and Delphi. Rest Dataware is a Brazilian framework developed for Delphi and Lazarus, which help you to build and deploy server api. Rest Dataware can serve many kind of resources, such as datasets, files, streams and events. It is very popular in Brazil and its popularity is gaining world.

I have skipped the installation instructions, If you have some question have a look at installation guide, otherwise you can send to me an e-mail at ricardodarocha@outlook.com.

In development…

About the author

I'm Ricardo da Rocha, an industrial designer and programmer with focus in design of interfaces and development of software for Industry and for Business Intelligence. I have practice the volunteer teaching to improve my methodologies to solve Real World Challenges. I love to learn with the community and with my students.

I'm graduated as Industrial Designer, have been worked at UEMG as a Volunteer Professor, and as product designer at Estudio Miron. After, I went to Austria to study Master Course of Material Engineering, as part of UFOP and ThyssenKrupp partnership. I've experienced a large project of steel scrap classification, which inspired myself to keep studying and working with software, data structure and data visualization. Today my major is to help people to understand their business, and to help developers to do best software's.

Acknowledgment

This is a small contribution for Delphi-Lazarus Rest Data Ware Community, which doesn't measures efforts on the development of stable and secures version of Rest Dataware, and for the people around the world that want to learn and to use Rest Data Ware in their way.

Keep in mind Rest Dataware is an open source project.

Left your donation at restdataware site: https://www.restdw.com.br/ donation

ÉNóix

Sumário

Chapeter 1

Create your first server within just 3 steps

Just 3 Steps are enough to create your first rest dataware server

1. Control the port of server with RestServicePooler component
2. Represents an instance of uDWDatamodule.TServerMethodDataModule
3. Activate and Run RestServicePooler Server1

Step 1 → RestServicePooler

In a new application just grab the component RestServicePooler from the pallete Rest Dataware Service.

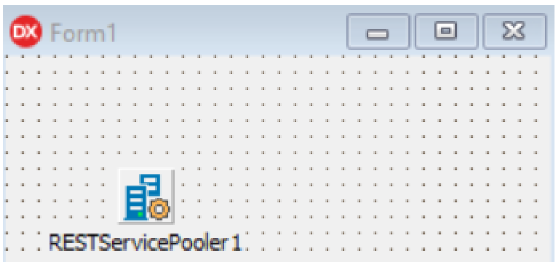
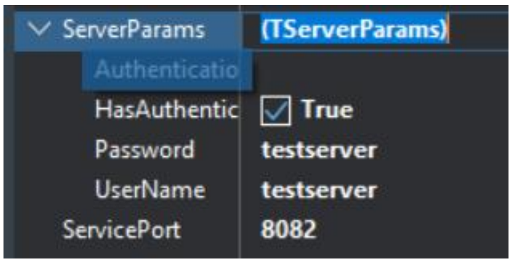


Image 1. Rest Service Pooler

This component will control a Port of your Server. To configure this component is very easy. Set the ServicePort you want to control.

If you want to create a secure connection just set True the parameter "ServerParams>HasAuthentication".

You can change the user and password at its respective properties.



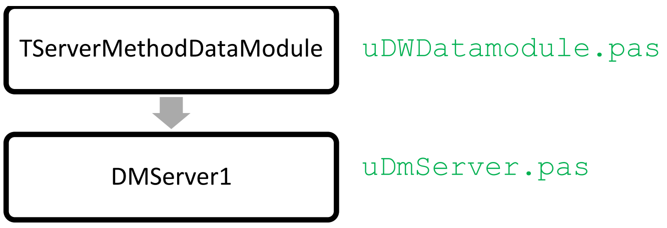
ServerParams	(TServerParams)
Authenticatio	
HasAuthentic	✓ True
Password	testserver
UserName	testserver
ServicePort	8082

Image 2 Server Params and Service Port

```
To test this step set RestServicePooler1.Active = True in design time.
Then go to the webbrowser and input "localhost:8082", you will see the browser ask for the
authentication. But remember set RestServicePooler1.Active = False again, otherwise we will not
be able to compile the application.
```

Step 2 → Configure your DWDatamodule Instance

I recommend you add the unit "uDWDatamodule.pas" into your project, but however you need to be shure you will never change the uDWDatamodule.pas, because when you update Restdw installation we will lose those changes.

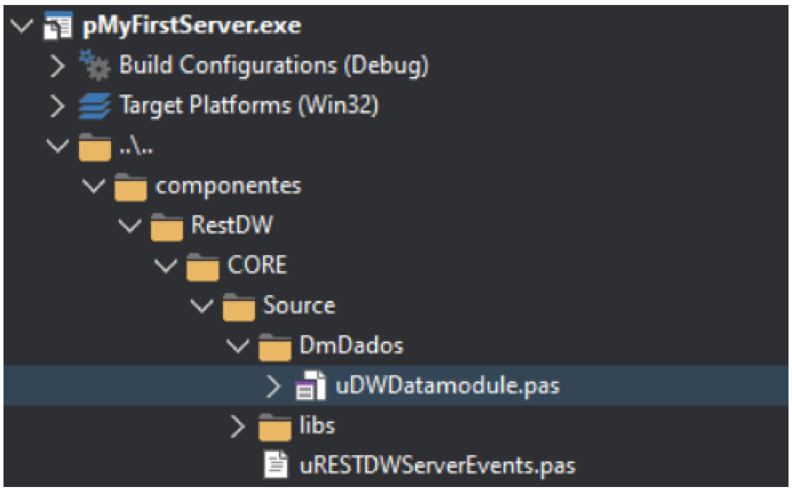


After Add uDWDatamodule , create an inherited Datamodule with descends from TServerMethodDataModule. I will show step by step.

Finding DWDatamodule.pas

Click Project/Add to project, or find the icon

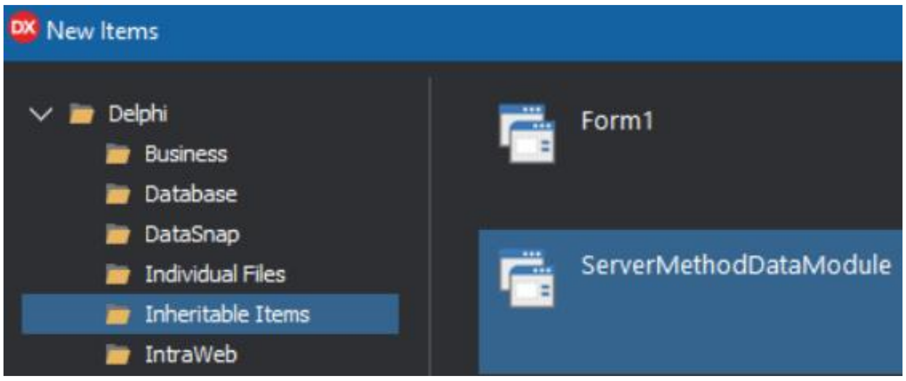
You can find DWDatamodule.pas at DMDados folder, in RestDW/Core/Source. For instance I installed the RDW Components at "Componentes" folder, and then I found the file in following path.



```
pMyFirstServer.exe
  Build Configurations (Debug)
  Target Platforms (Win32)
  ..\..
    componentes
      RestDW
        CORE
          Source
            DmDados
              uDWDatamodule.pas
            libs
            uRESTDWServerEvents.pas
```

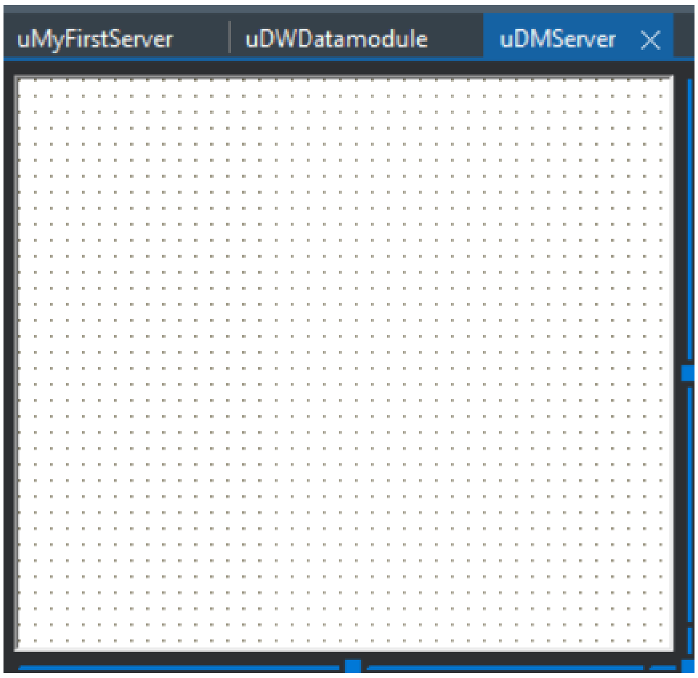
Creating inherited datamodule

Now go to File> New> Other and choose the option "Inherited Item", choose ServerMethodDatamodule and click OK.



We are creating a new datamodule you will put all resource we share with client: Connections, Events, WebPascal etc.

I suggest you to rename the Datamodule1 as DMServer1 and the unit as UDmServer.pas

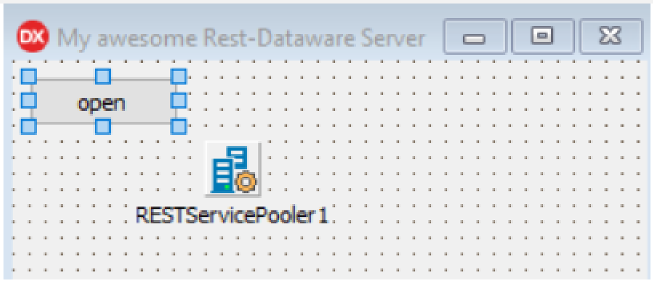


Step 3 → Activate and run

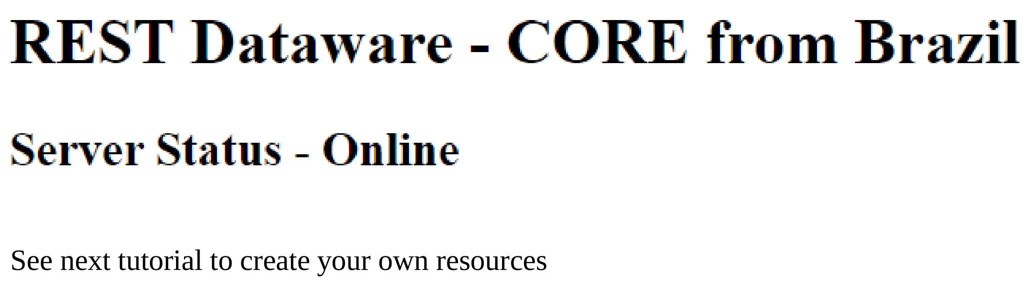
I changed the form 1 to frmFirstServer and created a button1 with instruction

Before active the RESTServicePooler1 set `RESTServicePooler1.ServerMethodClass  := TDmServer1` which you created at step2.

```
procedure TForm1. Button1Click (Sender: TObject);
begin
 RESTServicePooler1.ServerMethodClass := TDMServer1;
  RESTServicePooler1.Active := not RESTServicePooler1.Active;
  if RESTServicePooler1.Active then btnActivate.Caption := 'close' else btnActivate.Caption := 'open';
end;
```



That is all. You can compile and run your first server, and to prove you are a server running, go to Webbrowser and type "localhost:8082"



REST Dataware - CORE from Brazil

Server Status - Online

See next tutorial to create your own resources

Chapter 2

Chapter 2

Create your own resources

In my new tutorial you will learn more about how to serve data using resources in Rest-Dataware.

There are three components that will provide resources from the Server side to the clients.

It means there are three kinds of resources you can share, that you need to know.

4. RestDwPoolerDb →Provides connection to a database
5. DWServerEvents → Provides **json** response for the HTTP Verbs GET POST PUT DELETE
6. DWServerContext → Provides any kind of resource you want, as binary, html, stream and text. This is the engine which will send WebPascal pages.

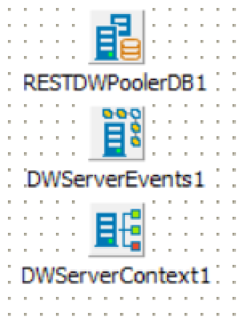


Image 3. RDW Resource Manager

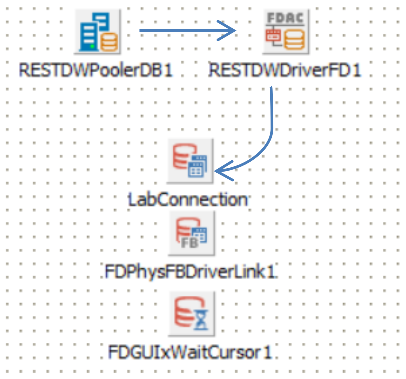
Resource 1 → RestDwPoolerDb

This component is pretty simple and will empower your Delphi Client to see the connection of other side of server. Also, it will allow you create your own SQL instructions in the CLIENT side, without the need of recompile the server.

Some information you need to remember from RestDwPoolerDb are:

a) The **client** `RESTDWDatabase` will communicate with **server** `RestDwPoolerDb`
b) Use one of those Drivers you installed. Look at pallete Rest Dataware – Core Drivers. In this example I'm using TRestDWDriverFD (Firedac Driver)

This is an example of the server architecture using `RestDwPoolerDb` and `Firedac`.



This component is visible only if you create a Delphi Client. For that reason I will cover this component in details next tutorial (4-Create your First Database Client)

Resource 2 → DWServer Events

In my opinion that is the most powerful component of Rest Dataware. With DWServerEvents you can create resources in PureJson, which communicate with any platform you want. Despite you can consume json with any language, it's pretty cool and pretty fast.

To understand the use of DW Events you need to know the basics of json (java script object notation). In abstract this is a way to transmit objects from different applications, usually thought HTTP protocol. The json looks like a structured database, which allows creating nested objects too.

```
JSON {"data": "I'm Json",
        "son": {"data": "I'm son"}
        }
```

For example, suppose you have an API schema that provides the following resources:

a) Date and Time of your server

b) Server IP

c) Version of your API

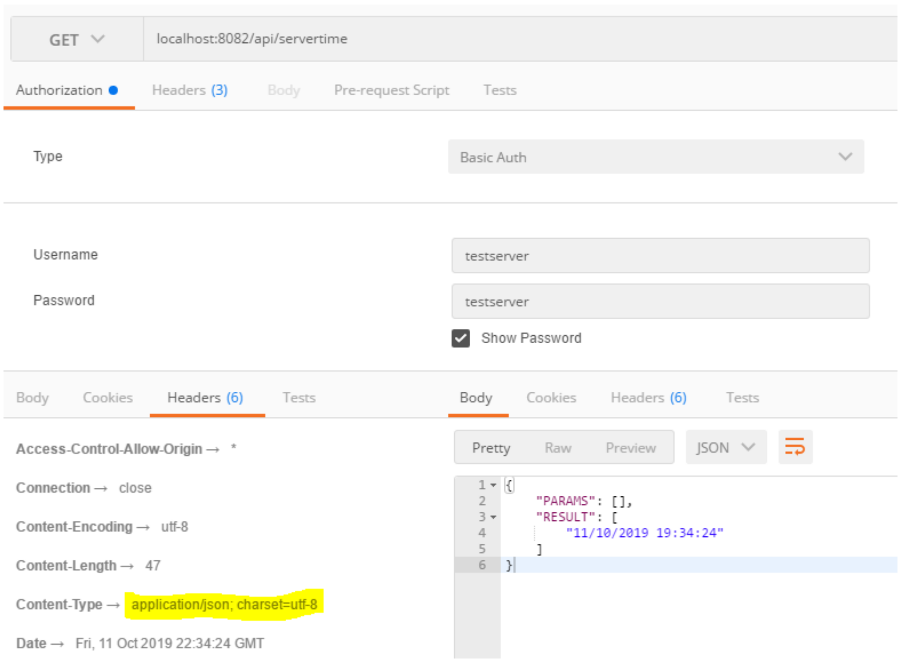
We can configure each resource to return its respective json, using variable **result**

```
{"result": {"version": "1.0"} }
```

I will cover the resource 2 in details below.

Resource 3 – DW Server Context

The resource 3 is very familiar with resource 2, and was implemented using the same structure. The main difference is that the Server Events ever returns json contents and the server context also returns text/HTML and other kind of resources. You can check the content type using PostMan to do the request.



GET localhost:8082/api/servertime

Authorization: Type: Basic Auth

Username: testserver

Password: testserver

Show Password ✓

Headers (6):

```
Access-Control-Allow-Origin → *
Connection → close
Content-Encoding → utf-8
Content-Length → 47
Content-Type → application/json; charset=utf-8
Date → Fri, 11 Oct 2019 22:34:24 GMT
```

Body:

```
{
    "PARAMS": [],
    "RESULT": [
        "11/10/2019 19:34:24"
    ]
}
```

I will cover DW Server Context in next chapters.

Understanding API

Request

Run your server at port 8082. Go to your web browser and type the route

```
localhost:8082/api/Version
```

The browser will return the value

```
Event not found...
```

As long as you created your resources on the server, the api localhost: 8082/ will responds to your request using routes that you give implemented: [1]

```
Ps. When you type /ServerTime in the browser you are calling the method GET
```

Response

The server would returns the following structure if you set up the variable result with the json string:

{"name": "RDW Server 1", "version": "1.0", "title": "tutorial 2", "student": "yourname"}

Let's implement our tutorial.

1. Add DWServerEvents

This component doesn't need any configuration. You just need to put the name of component you want to see at baseUrl. For example, I called this component "api"

Now we can create our resources

2. Returning the Version of API

You can click with right button over the component "api" and choose "events list", or select property "Events" at Object Inspector. Add new event, and rename to Version;

Now choose "events" and configure the event "OnReplyEvent"

```
Result := ‘1.0’;
```

Best practice is returning json string;

```
Result := ‘{“version”: “1.0”}’;
```

Compile and navigate to the address `localhost:8082/api/Version`

3. Create the resource ServerDateTime

The basis to create any method are the same. Just create the event name, and fill the **var** result with value you want using "OnReplyEvent".

```
Uses SysUtils;
Result := ‘{“result”: “’ + formatdatetime(‘dd/mm/yyyy hh:nn:ss.zzz’, now) + ‘”}’;
```

[1] We will use just the notation after the baseurl localhost:8082/api

```
Example:      /ServerTime
```

4. Create the resource ServerIP

Supose you have a most sophisticated method you want to treat some information, or process something, you can create other units and use the method “OnReplyEvent” to centralize the responses.

In this example I will keep the methods at the same reply. That reply uses INDY to access another API at internet

```
Procedure TDMServer1.APIEventsServerTimeReplyEvent(…
Var Response: IHTTPResponse;
Ip: String;
Begin
        Response := NETHttpClient1.Get(‘Http://checkup.dyndns.org/’);
        IP := Response.ContentAsString;
        Result := format(‘{“ip”: “’%s’”}’, [IP]);
End;
```

Furthermore

The techniques I have been shown to you despite RDW ServerEvents looks very simple. Indeed they are simple, however they are very powerful. For example, let me show how to easily serialize an object to json, using a Model (the class of the object) and one line of code.

For example, I’d created the resource “DishOfDay”

```
Type
TFood = class
        Category: String;
        Name: String;
        Description: String;
        Price: String;
        Constructor Create(Category, Name, Description: String; Price: Currency);
End;

Implementation

Uses System.Json, Rest.Json, MyRestaurant.Models;

Procedure TDMServer1.APIDisOfDayReplyEvent(…
Var Dish: TFood;
Begin
        Dish := TFood.Create(‘Indonesian’, ‘Ayam goreng’, ‘Indonesian fried chicken’, 10000.00);
        Result := TJson.ObjectToJsonObject(Dish).ToString;
        FreeAndNil(Dish);
End;
```

Look this amazing 1:N Collection and its behavior;

```
Type
TFood = class
        Category: String;
        Name: String;
        Description: String;
        Price: String;
        Constructor Create(Category, Name, Description: String; Price: Currency);
End;


Implementation

Uses System.Json, Rest.Json, MyRestaurant.Models, Generics.Collections;

Procedure TDMServer1.APIDisOfDayReplyEvent(…
Var
  Dish1, Dish2, Dish3: TFood;
  Menu:TArray<TFood>;
Begin
        Dish1 := TFood.Create(‘Indonesian’, ‘Ayam goreng’, ‘Indonesian fried chicken’, 10000.0);
        Dish2 := TFood.Create(‘Italian, ‘Pizza’, ‘Sicilian Ingredients’, 30000.0);
        Dish3 := TFood.Create(‘Japonese’, ‘Sushi’, ‘Fresh Tuna’, 22000.0);
        Menu := TArray<TDish>.Create;
        Menu.Add(Dish1);
        Menu.Add(Dish2);
        Menu.Add(Dish3);
        Result := TJson.ObjectToJsonObject(Menu).ToString;
        FreeAndNil(Dish);
        FreeAndNil(Menu);
End;
```

Chapter 3
Server Events in Deep

I told you before that the Server events responses the HTTP verbs GET PUT POST DELETE, and now we will see how it works.

Usually I use the method `onReplyEvent` to respond no complex requests, however I suggest you to learn the most flexible `onReplyEventByType` whose **var** RequestType give you the chance to return different results depending the kind of the request.

```
procedure TServerMethodDM.apiDishOfDayReplyEventByType(…
begin
  case RequestType of
    rtGet: Result := ‘{“Ayam goring”:1000.00}’;

    rtPost: SaveInDatabase(Params.ItemsString['promotiontoday'].Asstring);

    rtPut: RegisterNewFood(Params.ItemsString['newfood'].Asstring, Price);

    rtDelete: DeleteFood(Params.ItemsString['ID'].Asstring);
  end;
end;
```

Understanding HTTP

Suppose you want to get some information from Server, it's pretty clear that you will use the method GET. Actually this is the default method of HTTP. Because that is the default you just need to call the route directly in the browser, and you don't need to specify the verb you want for.

However if you debug the request header at the server you will see the browser sent the addres GET localhost:8082/api/ServerTime.

Which situation will you need to choose another verb?

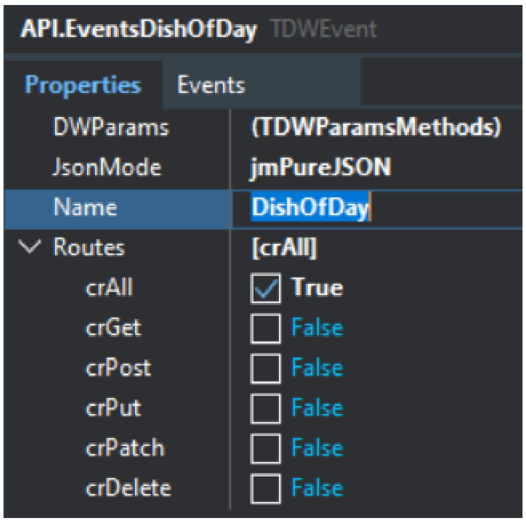
For example, if you create a web app page a Restaurant, that shows the customers the foods menu, you can call a GET. Otherwise if you want to create a WEB Application for the restaurant owner, to himself register the foods, and update the prices, and inform the dish of the day for example, then you will call the method POST/PUT. You need to create a route to the customer register the order, and also to show the cost of dinner.

Creating parameters

PUT LocalHost:8082/api/newOrder?**foodid**=614&**persons**=2&**table**=12

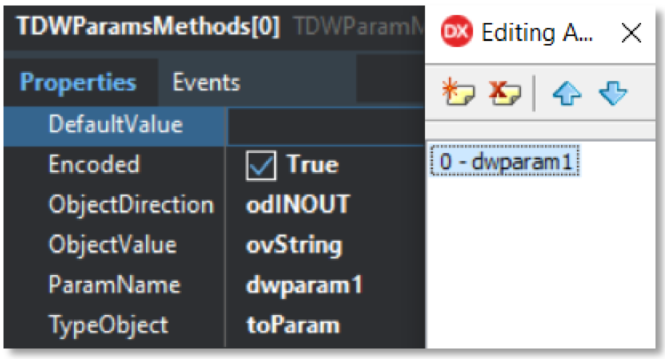
For example, to inform the kitchen that the customer chooses a food, you need to specify in the request which food it chooses, and which table the customer is sitting at. The parameters are attached in the header of request, such as the verb and the route.

To create a parameter choose Events List at the 'api' component, and then choose the event you want to configure. For example, I choose the event DishOfDay we created before. You will see the DishOfDay Event has a name, jmPureJson as a mode, and DWParams which avoid you to create some parameters. Let's click on the DWParams and choose [...] to edit.


API.EventsDishOfDay TDWEvent

Properties | Events

DWParams	(TDWParamsMethods)
JsonMode	jmPureJSON
Name	DishOfDay
Routes	[crAll]
crAll	✓ True
crGet	False
crPost	False
crPut	False
crPatch	False
crDelete	False


TDWParamsMethods[0] TDWParamM

Properties | Events

DefaultValue	
Encoded	✓ True
ObjectDirection	odINOUT
ObjectValue	ovString
ParamName	dwparam1
TypeObject	toParam

You can create many parameters as you want.

Properties of ParamMethod

ParamName	The name you want to pass during request	Example: 'promotiontoday'
Encoded	Default behavior	True
ObjectDirection	In-Out direction. Also accept both	odIN, just accept from client
ObjectValue	Any Delphi type you want	ovString
TypeObject	Default type	toParam

Planning your api

Before you create your api in Rest Dataware you need to draw a project, to understand the routes and how it will works. Which kind of behavior do you expects for your application?

I will create a model about how you can do that.

API Restaurant BaseURL: localhost:8082/restaurantapi

get	/food
get	/food?**category**=pizza
put	/food?**name**= Ayam%20goring&**category**=Indonesian&**price**=1000
put	/order? **foodid**=614&**persons**=2&**table**=12
get	/order? **table**=12
post	/cancelorder? **table**=12&r**eason**=too late
delete	/order?**item**=614&**reason**=unavailable&**choosenewitem**=true

See you next tutorial **Ricardo da Rocha**

Contact

If you have some question about RestDW have a look at installation guide, otherwise you can send to me an e-mail at ricardodarocha@outlook.com.

Visit official web-site https://www.restdw.com.br/

Join our community using Telegran, Skype or Facebook.

You will find me at RestDW Skype Group 1

You can read my profile in Linked'in

Visit my github to get some Delphi example. I will publish exclusive contents about RestDw library.