Tech Note

embarcadero

Mobile Application Development
Connecting with PHP REST Servers from Android

Daniele Teti

January 2011

Americas Headquarters
100 California Street, 12th Floor
San Francisco, California 94111

EMEA Headquarters
York House
18 York Road
Maidenhead, Berkshire
SL6 1SF, United Kingdom

Asia-Pacific Headquarters
L7. 313 La Trobe Street
Melbourne VIC 3000
Australia

INTRODUCTION

The need to have a mobile user interface for an existing application to add functionality or to ease the way a Client/server system or multi tier application is used, is growing at an increasing rate in IT. With many companies require their employees to access data from their management software even when out of the office. In this paper you'll learn how to write a REST service in PHP with RadPHP™ XE and subsequently interrogate it with an Android application. The two subjects (REST and Android) would require hundreds of pages to be fully explained. Fortunately, the basic concepts can be introduced in the few pages available here. The choice to use REST is not mandatory. You could also use JSON-RPC, but considering all the benefits that REST provides in terms of interoperability and ease of development, I have chosen to use REST for architectural services.

WHAT IS REST?

A few years ago, it seemed that web services SOAP was the solution to all the problems of B2B integration between heterogeneous systems. Today, however, although SOAP still maintains its own space and it is very suitable for use in certain scenarios, most web applications, including those developed by web service providers such as Google, Yahoo, Amazon, Twitter and Facebook, expose a REST API. The term REST stands for Representational State Transfer and was coined in 2000 by Roy Fielding in his Ph.D. dissertation.

More information about REST can be retrieved on Wikipedia (http://en.wikipedia.org/wiki/REST) and in the white paper written by Marco Cantù (http://www.embarcadero.com/rad-in-action/rest).

In a nutshell, REST provides an excellent architectural style to represent data and possible operations on that data.

REST is not a technology, and many tools available on the market might not support REST directly because of the lack of a specific standard. What makes REST the best choice in many scenarios, is its ease of implementation and the fact that REST is not tied to any particular system, language or tool. If you can send an HTTP request to a web server, you can use REST.

Although starting to use REST is very easy, a system that follows all the REST "principles", is not trivial to design and build[1]. One of the more complex concepts to be grasped by a

[1] In various sections of this white paper we develop a minimal REST system whose sole purpose is to illustrate the key concepts behind this architectural style. In many cases, the architectural purity has been

developer is the concept of "resource". REST focuses on the concept of remote resource and not on the method or remote object.

To define what should be done on a specific resource, REST uses the intrinsic meaning of the verbs of the http protocol. Here are the five main verbs that are commonly used in RESTful systems:

- GET Retrieve a resource
- PUT Create a resource
- POST Update a resource
- DELETE Delete a resource
- HEAD Retrieve the metadata that defines a resource

As has already been mentioned, REST does not provide standards, but provides a set of designing "principles" to follow. The REST principles can be summarized in five points:

- Identify the "things" (or resources) with an ID (aka URIs)
- Connect the "things" between them according to clear and understandable criteria (see also the principle http://en.wikipedia.org/wiki/HATEOAS)
- Use standard methods and interfaces
- Resources can have multiple representations (JSON, XML, YAML, CSV, etc.)
- The REST service must be stateless

To build a good REST service you need to answer the following questions:

- What are the URIs?
- What's the format?
- What methods are supported at each URI?
- What status codes could be returned?

A REST system that is not well designed might nullify or mitigate the benefits of using REST. To avoid bad design it could be helpful to read this article http://www.prescod.net/rest/mistakes/.

---

sacrificed to make the code easy to understand for those approaching this type of system for the first time. For real development I suggest you consider using one of the many PHP libraries on the market. Two of the most common are TONIC (http://peej.github.com/tonic/) and Zend REST (http://framework.zend.com/manual/en/zend.rest.html). If you decide to write a REST service from scratch, one of the most common patterns is FrontController (http://en.wikipedia.org/wiki/Front_Controller_pattern).

The above link is just a quick introduction to the REST architectural style. For the best information, please, refer to specific documentation.

WRITING A REST SERVER IN PHP

Writing a simple REST server in RadPHP XE is very easy. Figure 1 shows how to retrieve the current time using a simple REST Service.

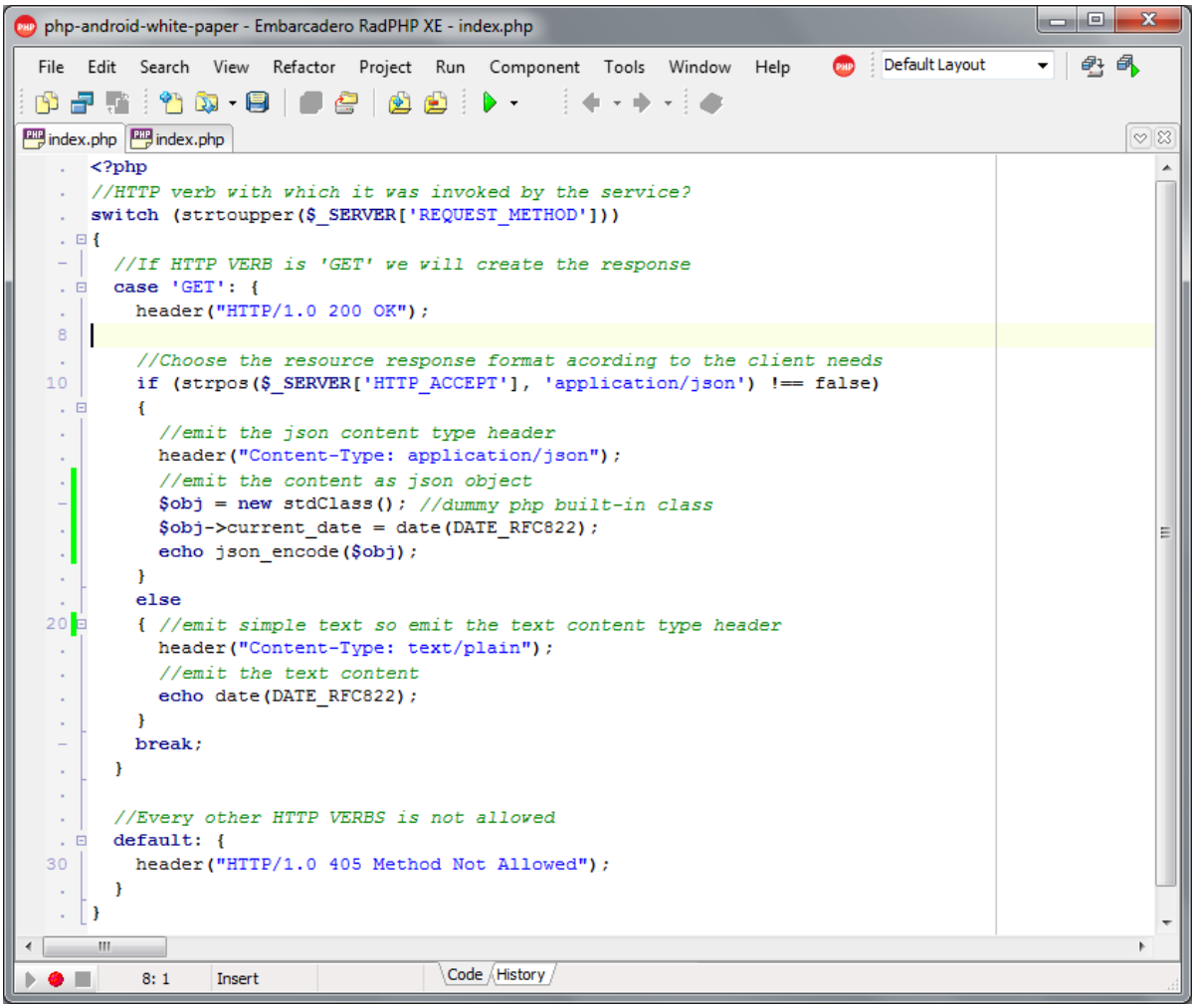


Figure 1: PHP REST server code to provide the current time

It is easy to see that all the routing logic is within the **switch** statement and the conditional **if** internal to a branch of the **switch**. After recognizing the http verb invoked by the URI, it proceeds to handle the request. The service, in the example, manages the GET verb only, which correctly returns an http code 200 as result. Others verbs (PUT, POST, DELETE, OPTIONS, HEAD) return a http return code 405 to the client. It is important that the

service follows HTTP protocol about returning codes as it is the standard way http communicates information to the client upon a request.

It is required in the request management that, the user's preferences for the response format are met. As mentioned above, a REST service may return representations of resources in various formats. In the example in Listing 1, we have provided JSON and plain text. If the client were to ask, usually via the ACCEPT header[2], for a representation that the service is unable to produce, the service would only provide a representation of default or return an http error code.

In PHP the ACCEPT header is made available in the global $_SERVER array, and identified by key 'HTTP_ACCEPT'. According to the http standard, the ACCEPT header contains a list of MIME formats that the client is able to interpret. The typical appearance of an ACCEPT header sent by a browser, looks like this:

```
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
```

The ACCEPT header is not the only header that the service can use to retrieve user preferences. Other headers useful for this purpose can be Accept-Language, Accept-Encoding and Accept-Charset. In addition, any other headers, including custom, can be used to send additional information to the service.

An in-depth discussion of the purpose and use of all the http headers is available at `http://www.w3.org/Protocols/rfc2616/rfc2616.html`

In this example we are using rather weak criteria to identify the mime type that the user prefers. In fact, we just check in $ _SERVER ['HTTP_ACCCEPT'] for the string "application / JSON", which is a JSON content MIME type.

If no type is specified, a plain text representation of the resource is returned.

Clicking "Run" from the menu RadPHP XE you can check the service response to a browser request [3].

---

[2] The Accept request-header field can be used to specify certain media types which are acceptable for the response. Accept headers can be used to indicate that the request is specifically limited to a small set of desired types, as in the case of a request for an in-line image.
(http://www.w3.org/Protocols/rfc2616/rfc2616.html)

[3] For testing REST services, we will use Firebug, this is a plug-in for Firefox and widely available on http://getfirebug.com/

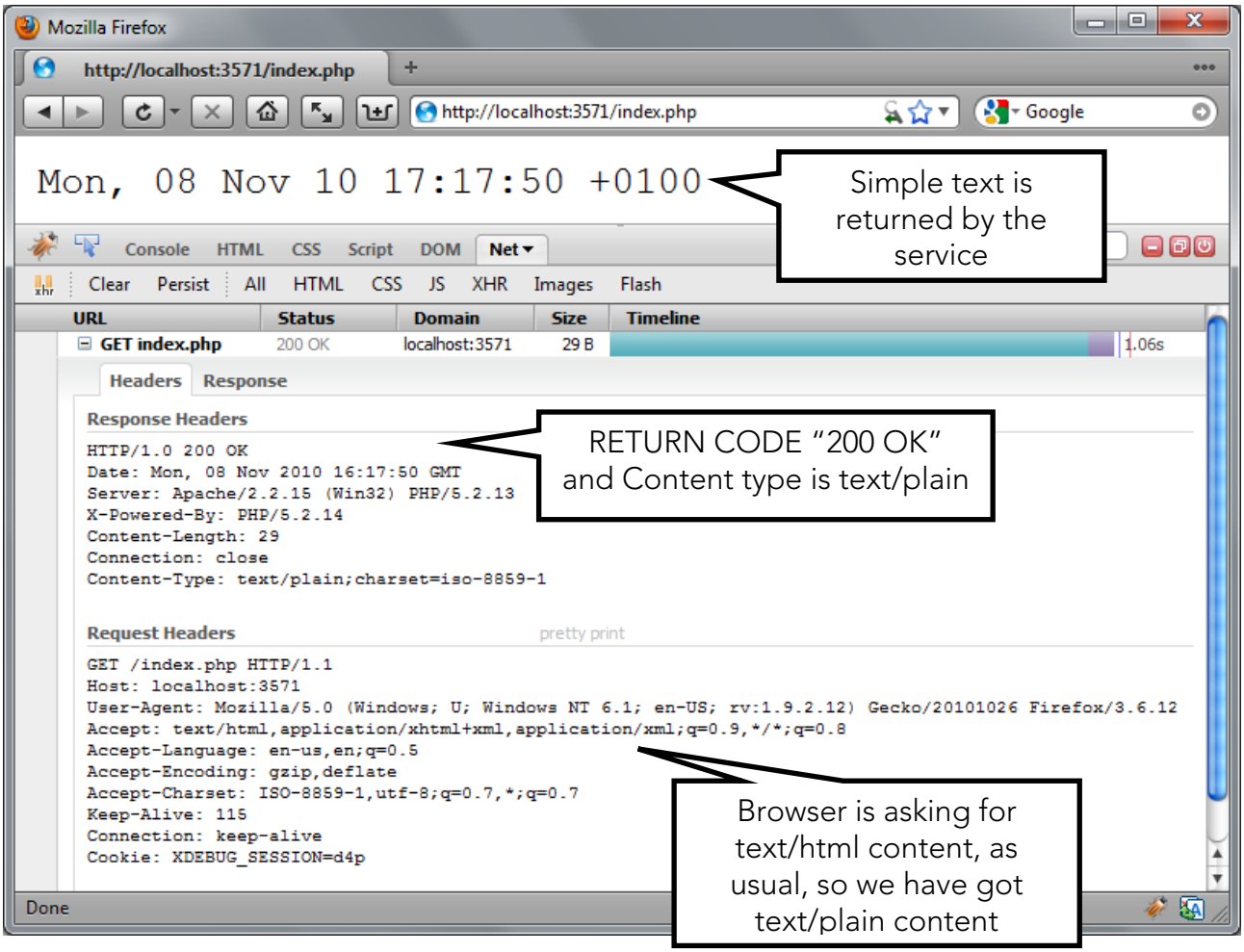


Figure 2: Details of the repsonse from the REST server

The response content-type generated is `text/plain`, because the browser has not included JSON in the headers.

Using REST services directly within the browser has some limitations:

- You cannot send requests that are not GET or POST[4] to the server
- It is not possible to change the ACCEPT header that the browser sends to the server without plug-ins or extensions.

[4] Using ECMAScript, better known as JavaScript, you can also send PUT and DELETE requests. There are open source libraries that make it easy to send these HTTP verbs in the context of an Ajax request. One of the most popular is definitely jQuery (http://jquery.org/).

There are several ways to solve these problems, including writing a simple REST client in Delphi XE or using "REST Client", a plug-in for Firefox. "REST Client" is made available by the Mozilla Foundation and can be downloaded from the plug-in page on Firefox site at (https://addons.mozilla.org/en-US/firefox/addon/9780/).

After installing "REST Client" in Firefox, let's ask our service to reply to us using the JSON format.

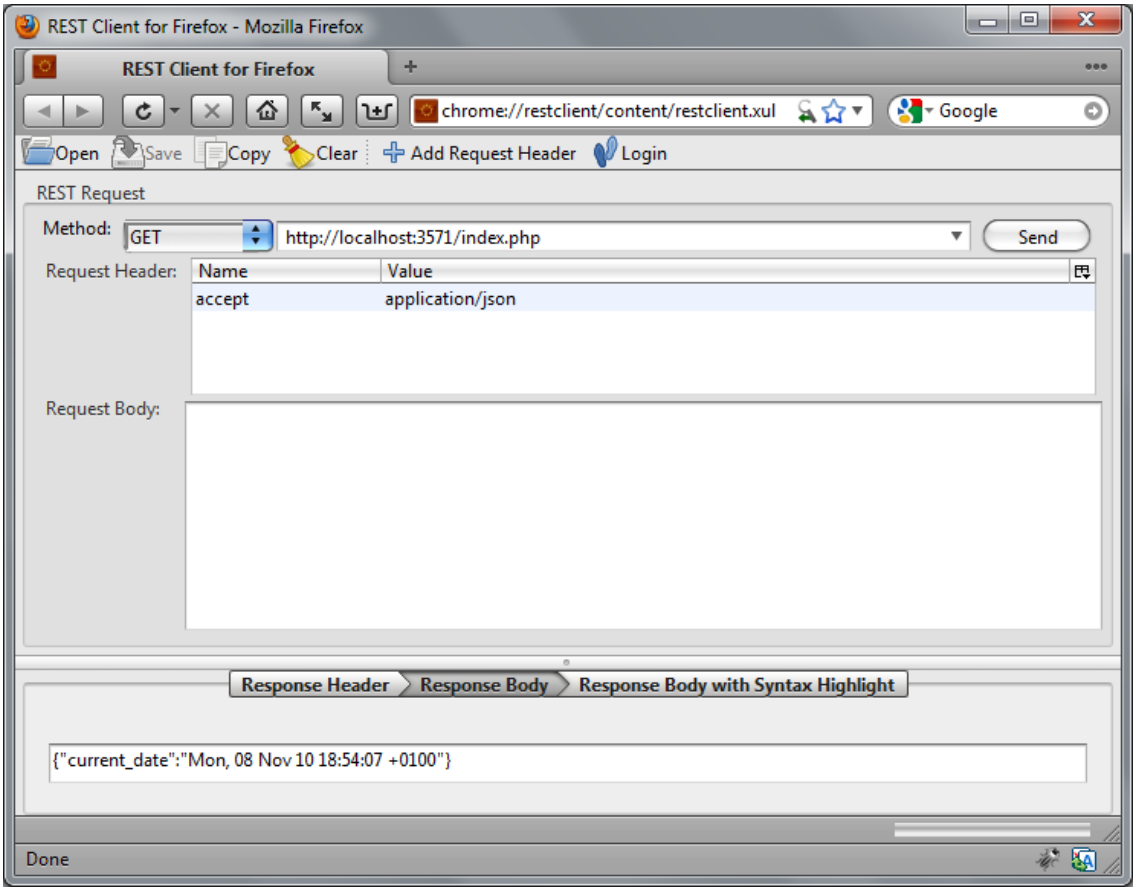


Figure 3: A RESTful reponse to Current Date

Having been able to read the ACCEPT header attached to the request, the service can generate a coherent response according to the client request. The server response this time is the representation of a JSON object.

Http GET verb does not have a BODY in its request, parameters and meta-information can be only be passed through the URI and the HTTP headers to the server. On the other hand, HTTP verbs POST, PUT and DELETE allow the sending of a message body in addition to the URI itself.

Requests involving a body, can be read using `php://input` stream. The code below shows how to build a POST http request.

```
POST /index.php HTTP/1.1
[…other headers…]

{"first_name":"Daniele", "last_name":"Teti",
"emails":["d.teti@bittime.it","daniele.teti@gmail.com"]}
```

The body of the request contains encapsulates the JSON that it is needed to be read from the server. To read the body on server side code, you must use specific PHP built-in streams. Referring to the example above `php://input` stream is used, as showed in figure 4.

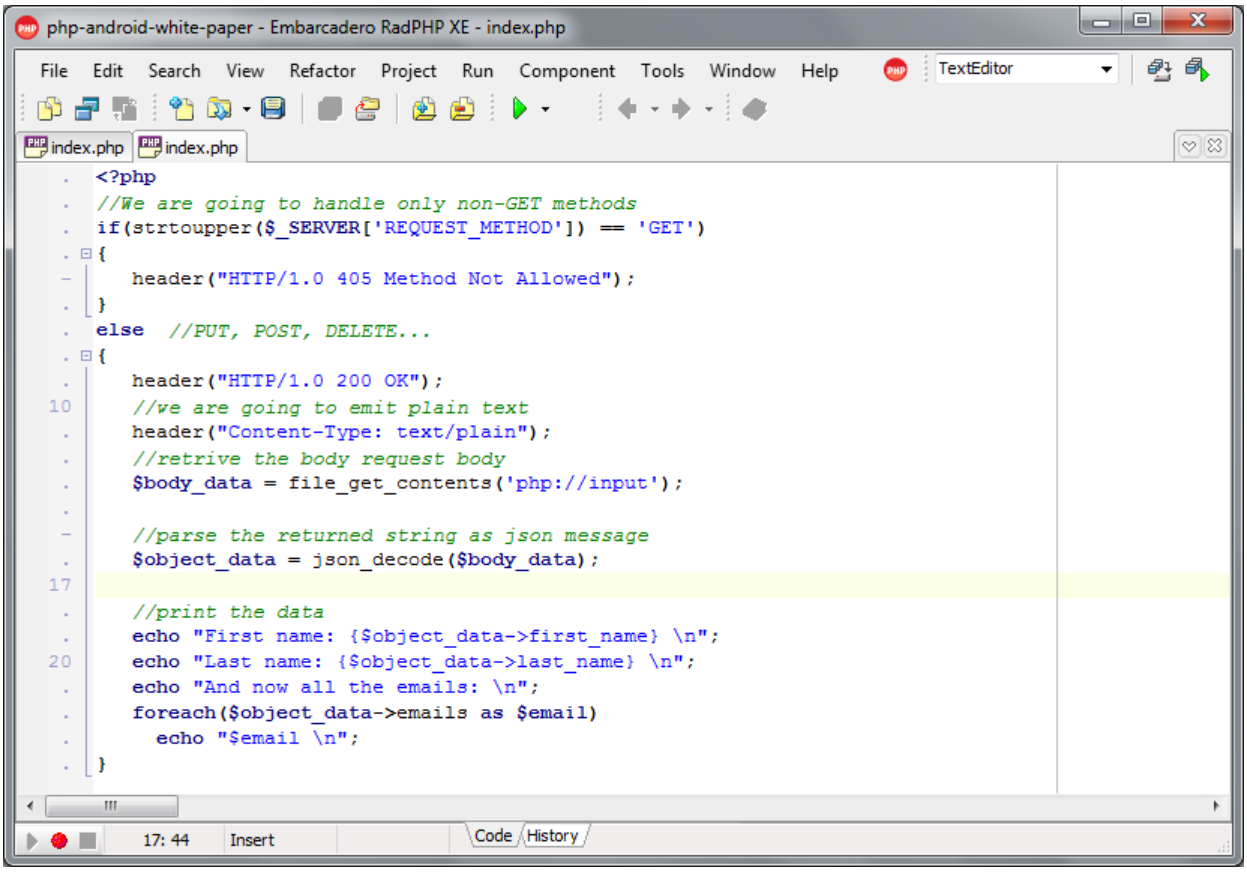


Figure 4: Code example to handle php://Input stream

WRITING A REST SERVICE USING RPCL DATAMODULES

Now that we have seen how to write a minimal REST service, we can move on to write a REST service that actually does something useful. In the next example we will use InterBase® and the todos.gdb database available for download with this paper. The service manages a to-do list using a single table defined as follow:

```
CREATE TABLE TODOS (
    ID           INTEGER NOT NULL,
    DESCRIPTION  VARCHAR(200) NOT NULL,
    COMPLETED    INTEGER NOT NULL
);

ALTER TABLE TODOS ADD CONSTRAINT PK_TODOS PRIMARY KEY (ID);

CREATE GENERATOR GEN_TODOS_ID;

SET TERM ^ ;
CREATE TRIGGER TODOS_BI FOR TODOS
ACTIVE BEFORE INSERT POSITION 0
as
begin
  if (new.id is null) then
    new.id = gen_id(gen_todos_id,1);
end
^
SET TERM ; ^
```

Create a new RadPHP XE console application, and add a datamodule. Create a connection to todos.gdb database and a table linked to the TODOS table. You can use DataExplorer or dragging each component individually.

If you use the downloadable code supplied, remember to edit the DatabaseName property of your Database component to make it compatible with your own specific environment.

Save datamodule as “main_datamodule.php” and the unit, name it according to the resources that it manages, for example `todos.php`.

The appearance of the designer should be similar as in Figure 5.

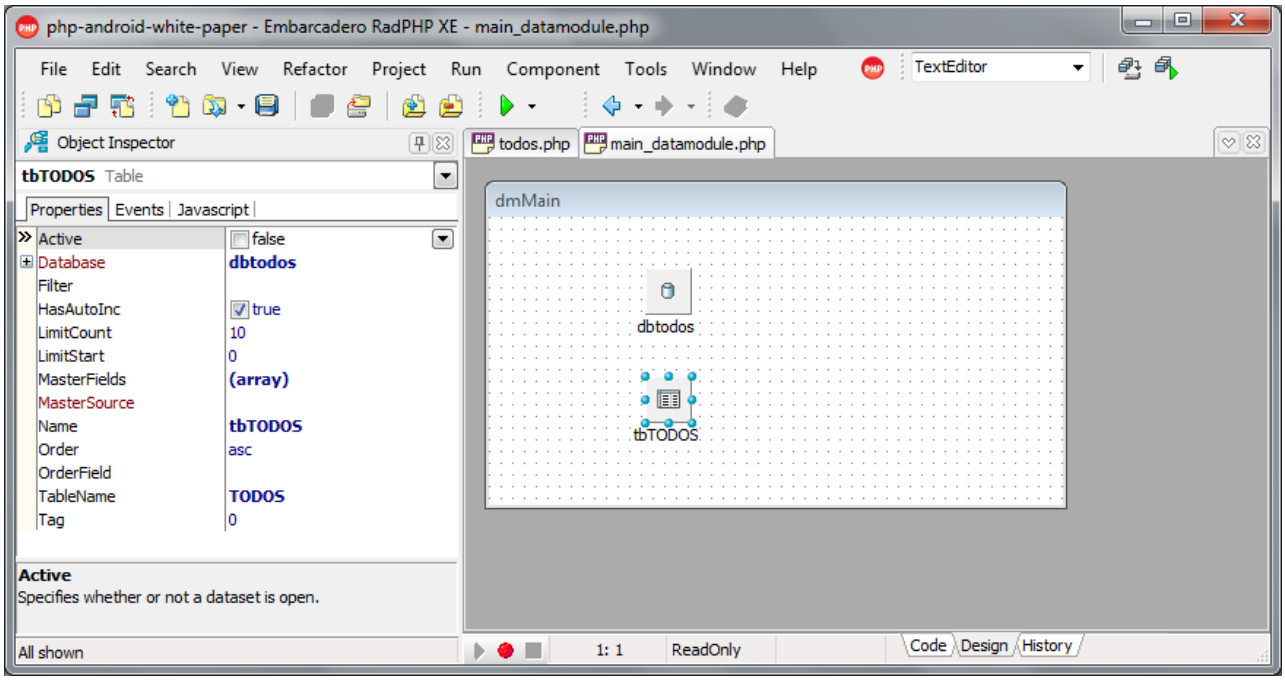


Figure 5: The RadPHP designer with database components placed

Do not leave the component connected to the database at design time. Ensure that the Connected property is not set to `true`.

Now that we have defined what our system will do (ie CRUD on a database table), we need to define which interfaces our service will expose. RESTful services allow the user to "discover" all available URIs through one resource that contains the list of available resources with the relative URI. For simplicity, in this example, we will not implement this resource. For a more accurate discussion of the problem see http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven.

One of the REST principles requires that the REST system should not expose URL that indirectly provides details about its internal implementation. For example, a REST URL should never refer to a specific file or real extension on the webserver.

What the request does?	What are the URI/Resource?	What is the format?	What methods are supported?	What status code could be returned?
Create a todo	/todos.php	JSON	PUT	201
Delete a todo	/todos.php/{id}	JSON	DELETE	200, 204
Update a todo	/todos.php/{id}	JSON	POST	200, 400, 410
Retrieve a single todo	/todos.php/{id}	JSON	GET	200, 404
Retrieve all todos	/todos.php	JSON	GET	200

In a real world system that aspires to be considered RESTful, ".php" extension should not be seen as it exposes details of the service that the end client should not know. Normally in PHP you use the Apache's mod_rewrite[5] module to make URLs independent from the script below.

Now let's discuss the most important parts of our application. Complete code can be found in the folder 02_REST_DB_Service.

We will start with our usual switch that allows us to choose the action that the user wants to perform on the resource. In this case we are running the HTTP verbs for CRUD actions. To access the data we have chosen to change our datamodule making it a "Table Data Gateway." A Table Data Gateway (TDG) is a simple design pattern for accessing data as described by Martin Fowler, who first formalized it in the following way:

"An object that acts as a Gateway to a database table. One instance handles all the rows in the table."

TDG methods are shown in Figure 6.

[5] http://httpd.apache.org/docs/2.2/mod/mod_rewrite.html

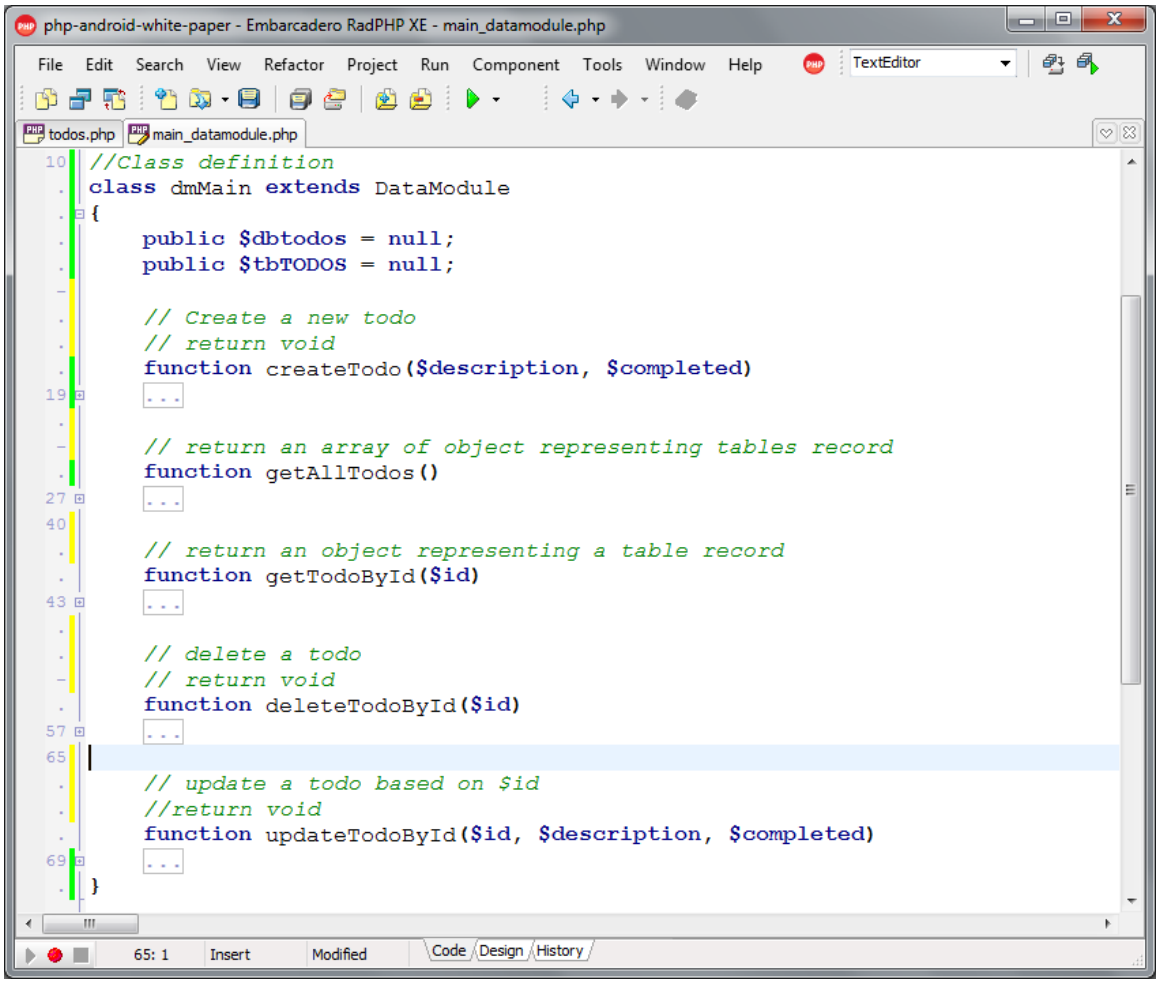


Figure 6: TDG methods implemented in the RadPHP IDE

Our todos.php file, which represents the resource to the remote user, contains the code that allows actions on our TDG. These are in the context of the HTTP request, parameters in URI and in the body.

Let us analyze the case of the GET verb.

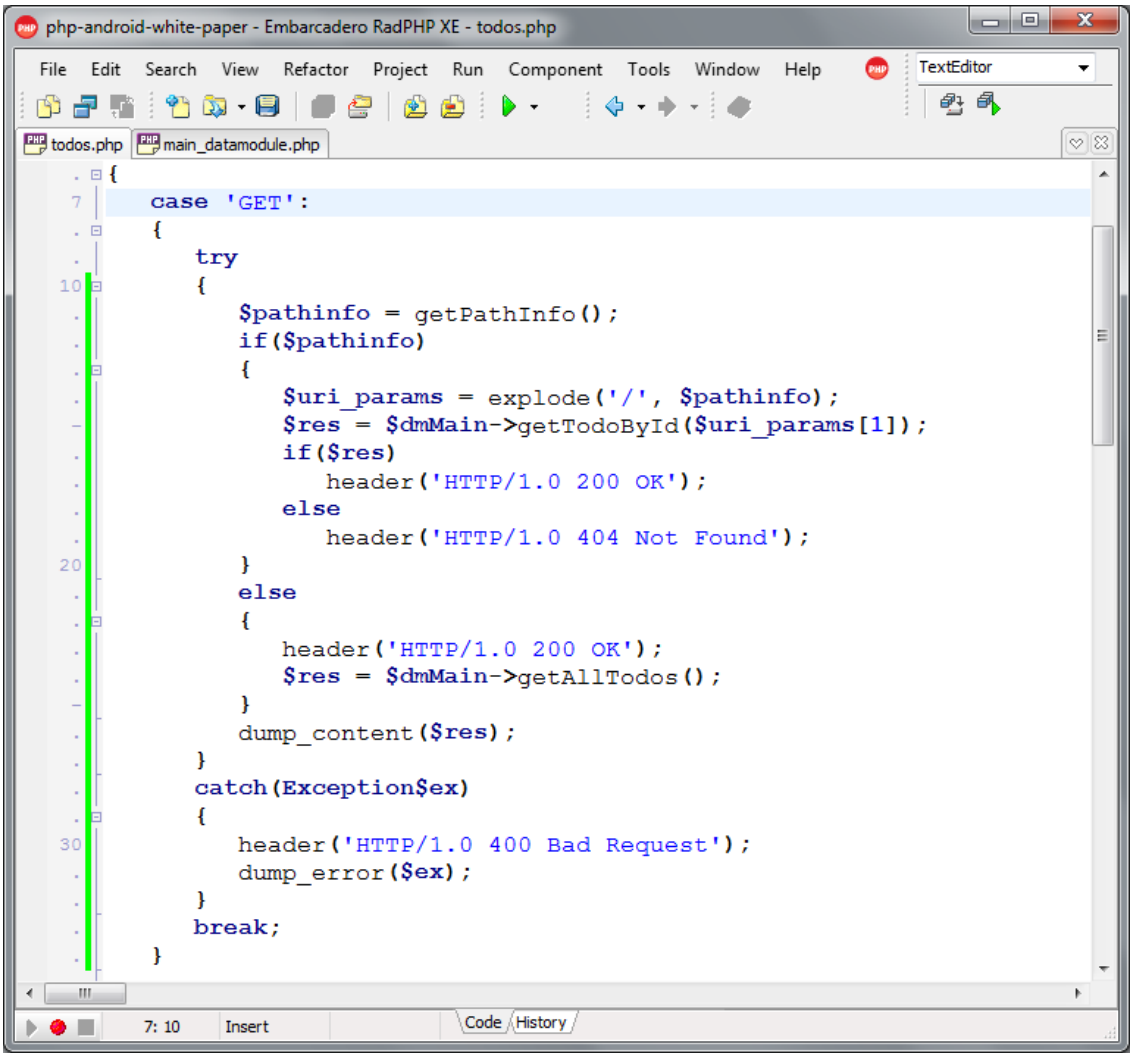


Figure 7: The "case" code for the GET verb

This code handles the following two types of requests:

1. "GET www.mywebsite.com/todos.php"
    - Returns a list of all the todo items as a JSON Array
2. "GET www.mywebsite.com/todos.php/25"
    - Returns a single todo item as a JSON Object

The code that generates the response is deliberately kept separate from the application logic. In `commons.php` file, there are three utility functions that help to separate responsibilities among different parts of the application.

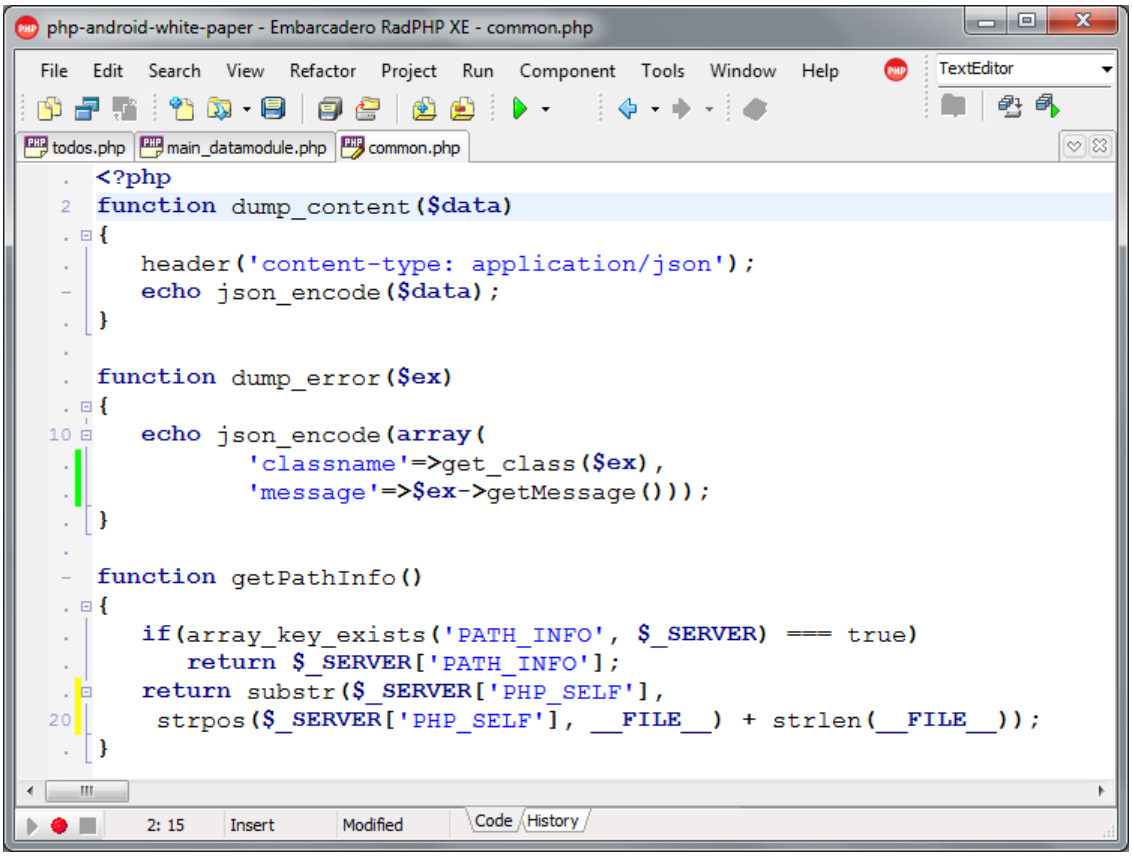


Figure 8: Examples of resource representation formats

To replace or add a new resource representation format, you need to add another function, very similar to `dump_content`. This architecture is only a hint of how a RESTful system should be built, but still it shows the basic concepts.

About error handling, we have chosen to return the appropriate http return code and a JSON body containing a detailed description of the error. Function `dump_error` shows how to implement it.

Let's look at the code that handles a request of type PUT. On analyzing this code, we can understand how to handle the HTTP methods that require a body of the request.

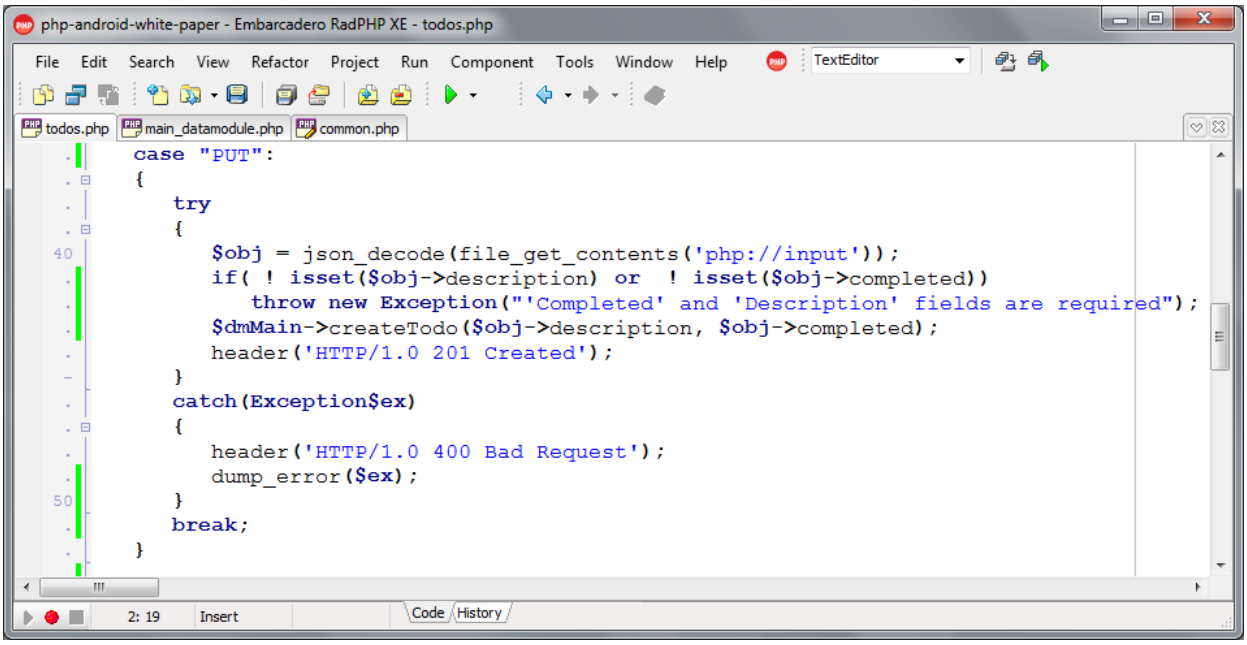


Figure 9: Code to handle the PUT request

The code is trivial. After retrieving the body of the request from `php://input` stream, we try to interpret the string as JSON, then we check that the required data is present and call the appropriate method from TDG. To test this code, we use "REST Client" in Firefox. This time, in addition to the URL, we also enter the body of the request as shown in Figure 10.

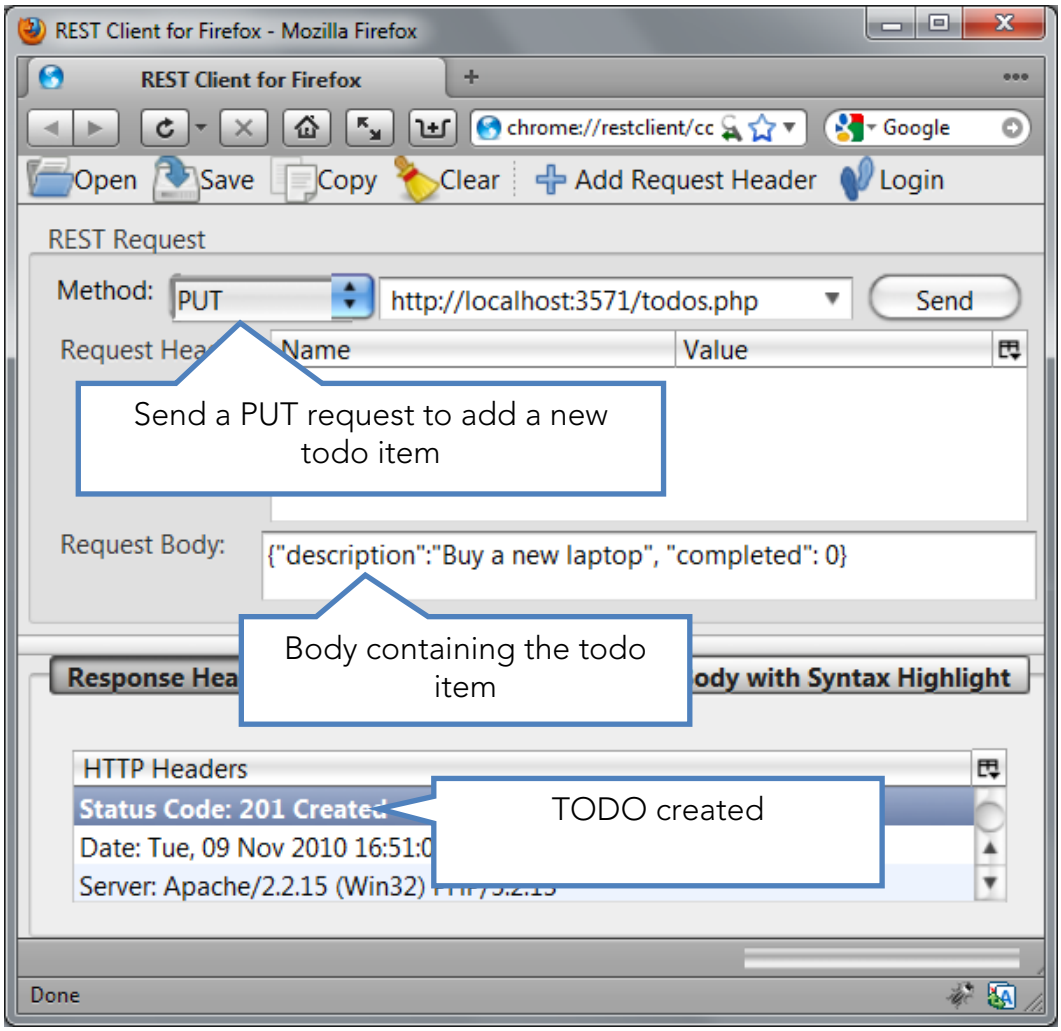


Figure 10: Using the REST Client to add the body of the request for the PUT

To verify that errors have been properly managed, it is necessary to change the body request to make it incoherent with the requirements of the service, and then resubmit the request. The test for error handling in the PUT verb is shown in Figure 11.

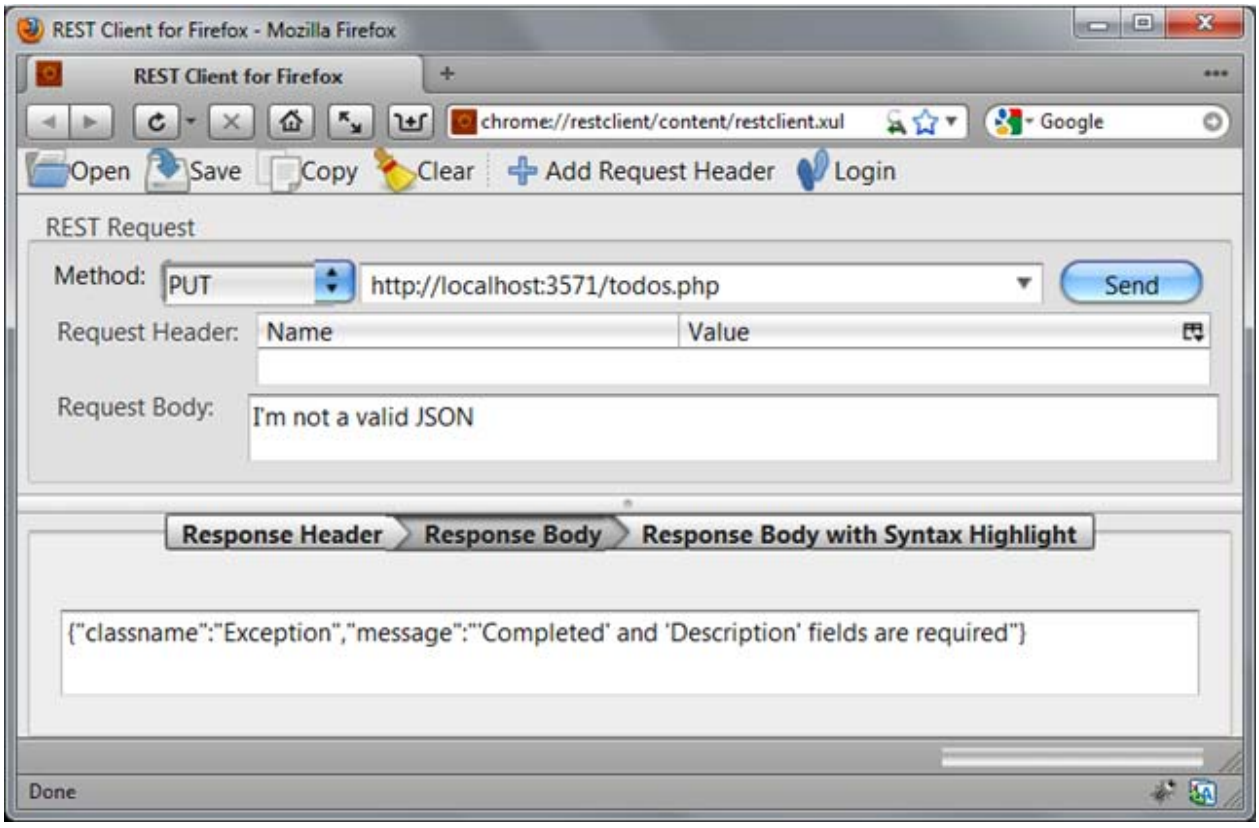


Figure 11: The test for error handling in the PUT verb

The response body contains a JSON array that provides the class name and description of the exception. In addition, the client receives HTTP return code "400 Bad Request".

Now that we know how to build a REST service in PHP that uses JSON as transport format, we can start to talk about Android.

Introducing Android Development

As Google says…

"Android is a software stack for mobile devices that includes an operating system, middleware and key applications. The Android SDK provides the tools and APIs necessary to begin developing applications on the Android platform using the Java programming language." (http://developer.android.com/guide/basics/what-is-android.html)

A lot has been written on the web on the pros and cons of Android, so we will not repeat that here. To get an introduction to Android and the motivations that led Google to

undertake the development of Android, you can consult Wikipedia at the following URL http://en.wikipedia.org/wiki/Android_%28operating_system%29.

Before starting to develop for Android, you must set up the development environment. Although it is not mandatory to use a specific development environment, Google has written a very useful plug-in for Eclipse called ADT (Android Development Tools), and later in this article we will use Eclipse with ADT. Google provides an excellent guide on how to configure everything you need to develop on Android. The rest of this paper will assume that your development environment is configured as recommended in this guide (http://developer.android.com/sdk/installing.html) and that you have successfully tried the "HelloWorld" tutorial (http://developer.android.com/resources/tutorials/hello-world.html).

In the rest of the paper, I will refer to some key components of the architecture of Android. To understand this , and avoid lengthy copy-paste, I suggest you refer to "Application Components". You can find more on this here.

http://developer.android.com/guide/topics/fundamentals.html#appcomp

LAYOUT

The most common way to define the appearance of an activity is using the XML layout file. The structure of the XML used by Android for layout, is reminiscent of a HTML web page. Each element within the XML can be either a `View` or a `ViewGroup` (including their descendants). The name of the XML elements of the layout file correspond to JAVA class that represents them. So a `<Button/>` element creates an instance of the `Button` within the GUI and an element `<LinearLayout/>` creates an instance of the ViewGroup `LinearLayout`. When loading resources, Android initializes the objects defined in the XML file layout creating functional objects. For a Delphi programmer, the closest concept to an Android layout file is the DFM file.

VIEW AND EVENT HANDLING

An activity is the Android counterpart to the Delphi Form. So, whatever the user sees is contained in an Activity. Every visible element within Android is called a `View` and is drawn on an activity. Specifically, graphical controls are called widgets and are defined in the package `android.widget`.

One difference for a Delphi developer is the event handling mechanism, but it is very similar to that of the graphics libraries available on Java SE.

The application that we are going to build, will allow us to understand the listener mechanism and how to parse a string in JSON format.

We will start by creating a new "Android Project" and configuring it as shown in Figure 12. From Eclipse you can create a project related to the unit test, but on this occasion we will not create it but just click "Finish".

Now modify the XML code of the layout "main" (that is found in `res/layout/main.xml`) to create a `Button`, an `EditView` and a `TextView`. To define the layout there is also a visual editor, not particularly advanced, but is is fit for purpose.

The XML code should be as follows[6]. You can either write it by hand or use the designer.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
<EditText
  android:id="@+id/edt"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"/>
<Button
  android:id="@+id/btn_click_me"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:text="Click ME"/>
<TextView
  android:id="@+id/txt"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"/>
</LinearLayout>
```

As soon as the XML file is saved, the ADT generates a static class called R that contains references to each resource in the directory `res`. Among these references you will also find the IDs of the controls defined in the XML file layout. To refer the ExitText, for example, we will use a constant `R.id.edt`, while for the Button there is `R.id.btn_click_me`.

[6]It is good practice to use constants defined in different XML files that Android provides, rather than hard coded values in your code.

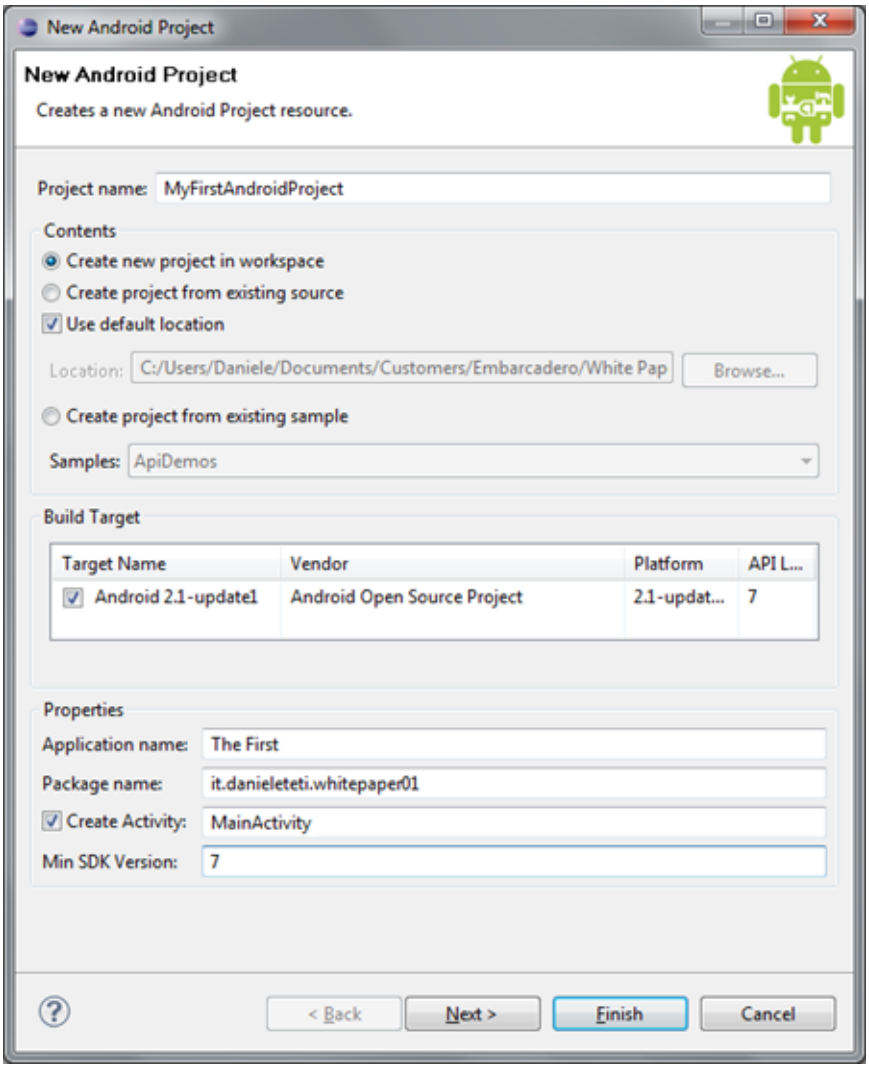


Figure 12 Configuring an Andriod project

After defining the graphical interface, you must link the event handler to all the views it contains.

Each `View` can respond to user inputs and be be connected to a `Listener`. In our particular case, we need to handle the listener for the button click.

Unlike Delphi, we don't have references between the Activity and the controls, as its aspect depends on the layout it holds. For this reason there is a method `findViewById(int id)` that receives a View identifier and returns its current activity as a generic reference to the `View`.

In our case, for example, to retrieve the reference to the button and link it to the event handler, we write:

```
Button btn = (Button)findViewById(R.id.btn_click_me);
```

In this way, the local variable *btn* refers to the instance of the Button class identified by ID `btn_click_me` currently drawn on the activity.

On `onCreate`[7], you need to retrieve the references to all the objects that we will refer in the code.

Then connect the listener for the button OnClick event. At the click of the button, we are going to write the TextView (like a TLabel) the text from the EditText (similar to a TEdit).

When you write the code, Eclipse might warn that some imports, regarding classes Button, TextView EditText are missing. As already mentioned, all the widgets, or nearly so, are contained in the package `android.widget.*`. Import it as shown in the code, and that will placate Eclipse.

---

[7] Despite the name, as it can be misleading, onCreate is not technically an event as in Delphi. onCreate is a simple method override. This can be considered as an implementation of the design pattern "Template Method" (http://it.wikipedia.org/wiki/Template_method)

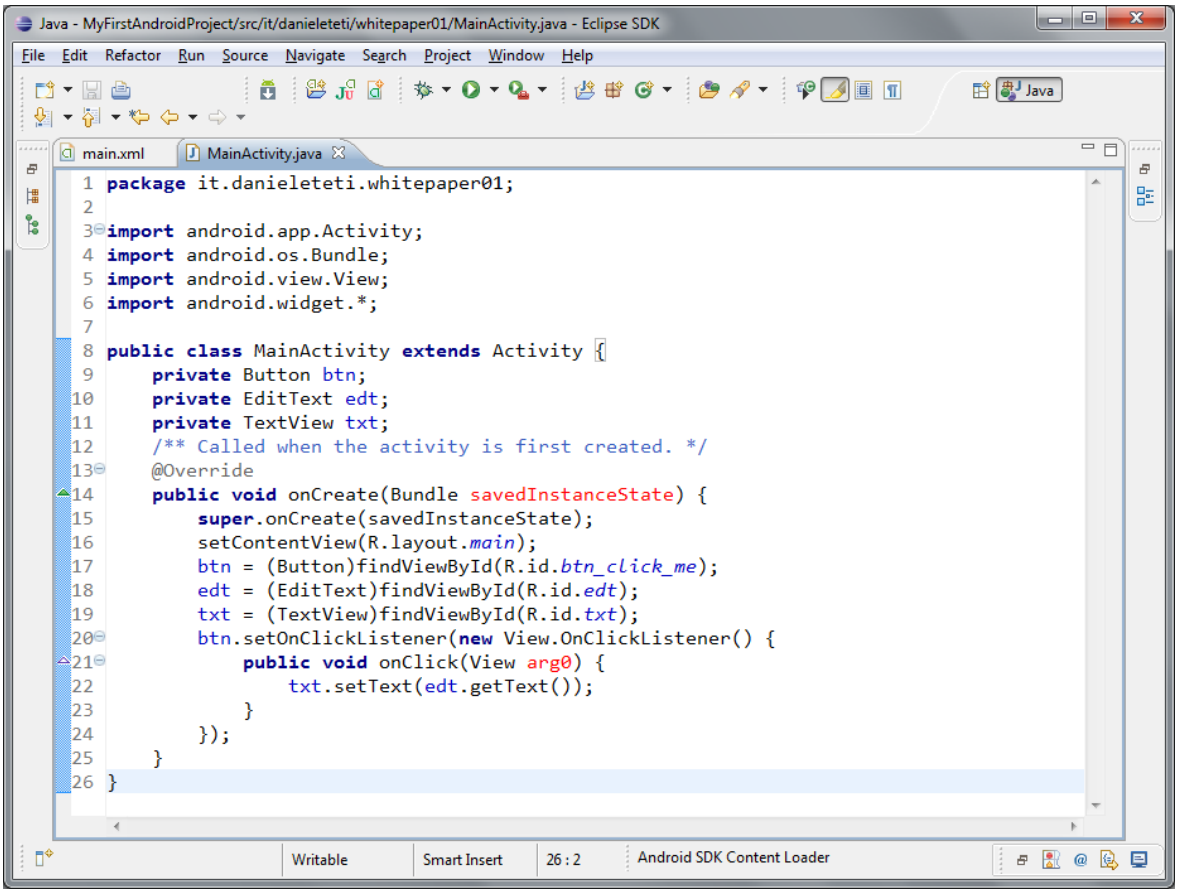


Figure 13: The MainActivity.java file within the IDE

On line 20 we linked an OnClickListener to a button using a Java Anonymous Class (http://java.sun.com/new2java/divelog/part5/page5.jsp). The `onClick` method of the listener is invoked on button click and acts as the event handler for the `onClick` event. In the body of the method we are using references to `EditText` and `TextView` to copy the text. In figure 14 you can see the running application on my Android phone.

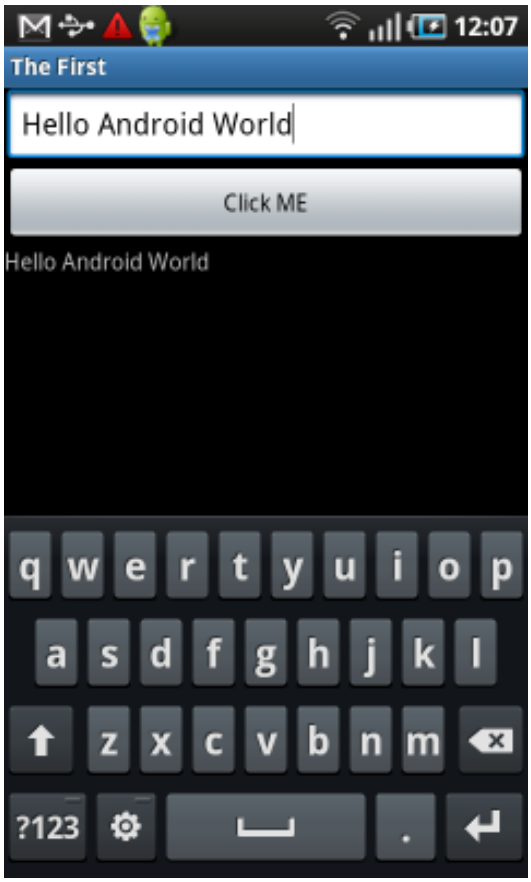


Figure 14: Running the application on an Android device

Now we can move on to interrogate our service PHP REST.

ACCESSING THE WEB SERVER

To send a request to a web server from an Android application is not complicated. For this purpose, Android provides the well-known Java `HttpClient` library from the Apache Software Foundation. The documentation on this is really great and well done. To understand how `HttpClient` works, it is useful to use the official documentation of the project and you can find it here http://hc.apache.org/httpcomponents-client-ga/tutorial/html/index.html

What we want to do now is to modify the application so that it can send a GET request to an arbitrary URL. The user can enter an URL within the edit and press the button. The application makes a request to the web site and prints the received text in the TextView. To accomplish this, we are going to write an activity private method that receives the url of the website as an input parameter, and returns the body of the response as output.

```
private String execHttpRequest(String url)
{
try {
            //Create default http client
            HttpClient httpclient = new DefaultHttpClient();
            //We want send a GET request, so Create an HttpGet object
            HttpGet httpget = new HttpGet(url);
            //Execute GET request over the httpclient
            HttpResponse response = httpclient.execute(httpget);
            //Retrieve the response body or "entity"
            HttpEntity entity = response.getEntity();
            //Return the entity as String
            return EntityUtils.toString(entity);
    } catch (Exception e) {
            //If something goes wrong, print the stack trace
            //and return the Exception message
            e.printStackTrace();
            return e.getMessage();
    }
}
```

If you try to run this application on the emulator or a device at the click of a button you get the following error message:

```
Permission denied (maybe missing INTERNET permission)
```

This is because, in Android, every application that uses a network connection, must inform the operating system [8].

To inform Android that our application needs to access to the Internet, open the file AndroidManifest.xml and add the following line just before the closing tag `</manifest>`:

```
<uses-permission android:name="android.permission.INTERNET"></uses-permission>
```

You can use the visual editor if you prefer. Now when running the application it should be possible to access the Internet.

Add an URL to the EditView and then click the Button. The result will be similar to Figure 15.

---

[8] Permissions are an application-level security mechanism that lets you restrict access to application components.

Permissions are used to prevent malicious applications from corrupting data, gaining access

to sensitive information, or making excessive (or unauthorized) use of hardware resources or external

communication channels.

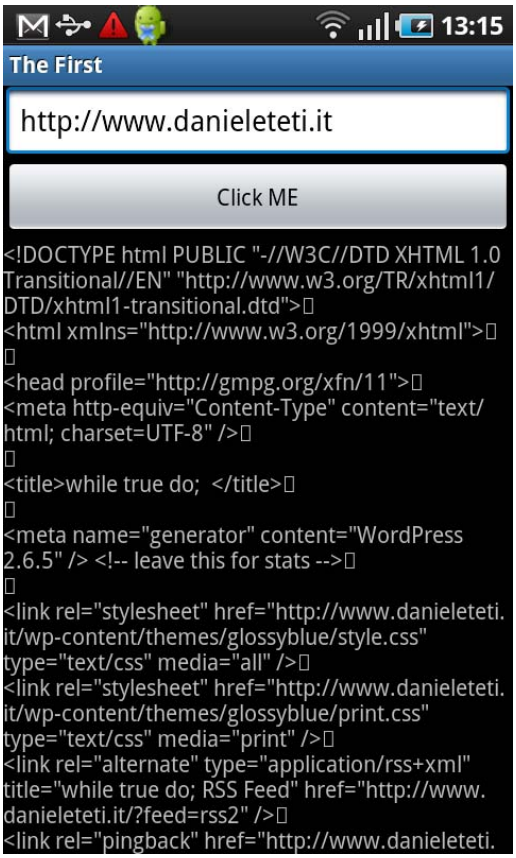


Figure 15: The reponse on the Andriod client from accessing the web server

Parse the JSON in the http response

In standard Android SDK, there is also one of the most popular parser in the JAVA world. The package is in `org.json`.

As a first example we will create a simple Android application that asks the current time from the first REST service that we have written. The complete code for the service can be found in the folder 00_First_REST_PHP_Server.

On the button click , we send a GET request to the service by setting `ACCEPT: application/json` in the headers. The service, we will return the JSON that we saw in Figure 3. At this point we have to parse it. The detail of the code that makes the request and parses the JSON is in the figure 16.

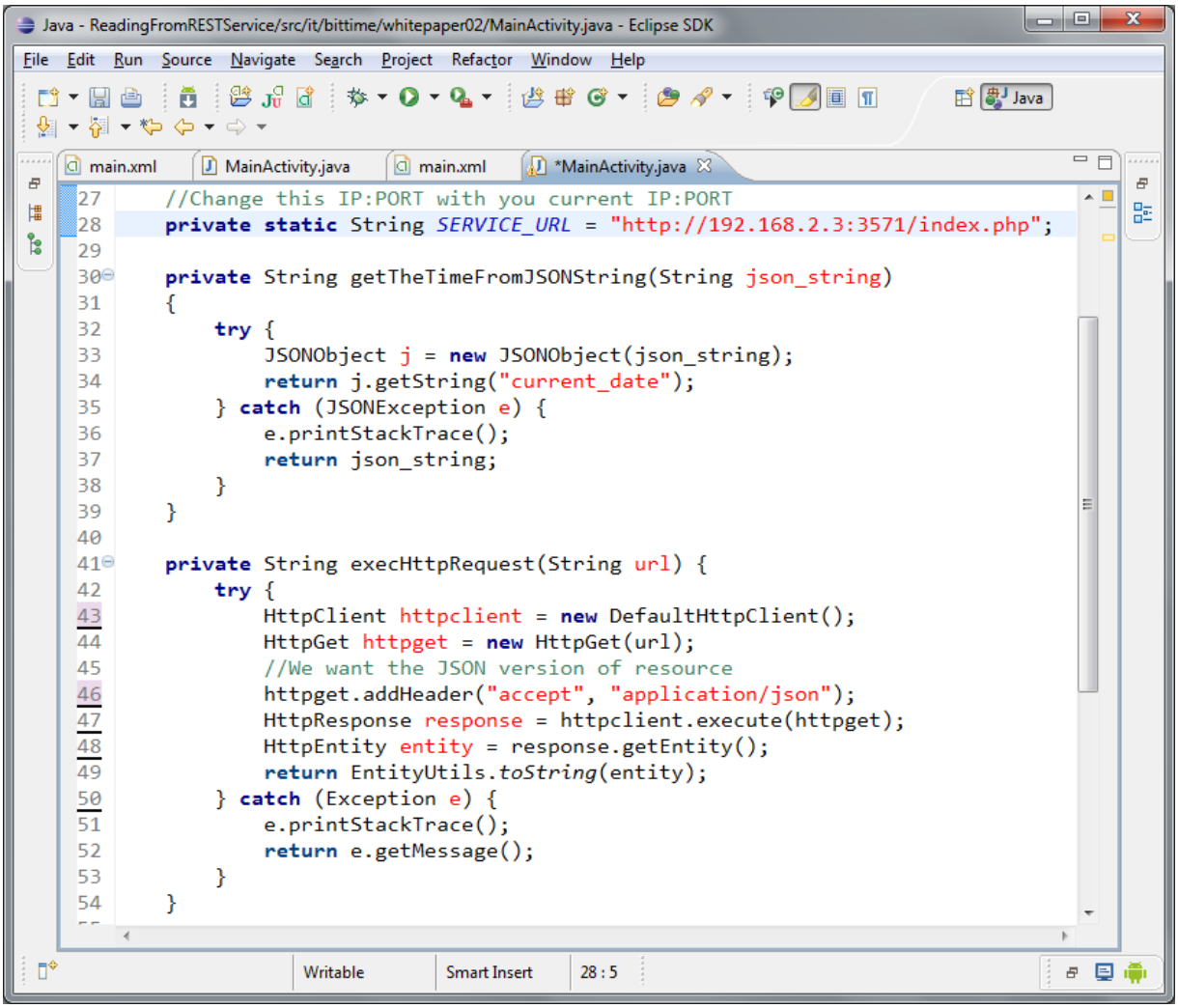


Figure 16: Making the request and parsing the JSON response

Please notice at line 46, the `accept` header has been added in order to get a JSON representation of the current time. In figure 17 is shown the running application.

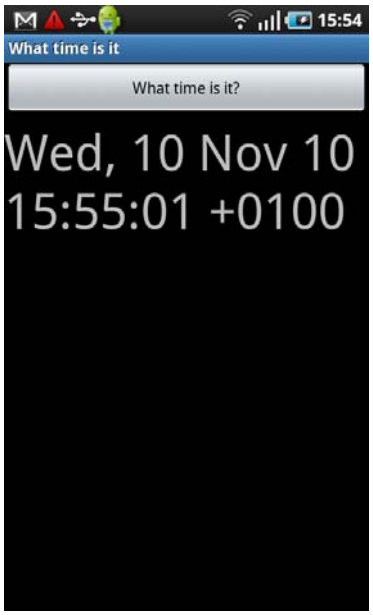


Figure 17: The Andriod client application running with the date/time response

DEVELOPING A TO DO LIST

Now that we understand how to interface an Android application with a REST service using JSON, we will develop a simple ToDo list. A ToDo list is simply a list of things to do, without a specific date on that they must be done. As the server we are using the service that we built in the previous example "Writing a REST service using RPCL DataModules" and this can be found in folder 02_REST_DB_Service.

The interface is minimal but effective. The main activity shows a list of all the 'todo' items, and whether they have been completed or not yet completed[9]. Clicking on a TODO item, using an Intent, we will open a new activity that will then allow us to edit it. In addition, you can also insert new 'todo' items directly from the main activity using the "New ToDo" button.

During the development we will learn how to connect a ToDo list retrieved from the server to a ListView, how to open a secondary activity, how to create menus for the activity, how to manage user preferences and how to use Toast. It is not in the scope of this paper to be a course on Android, but understanding these simple mechanisms will enable you to develop a simple Android application that does something really functional.

[9] An excellent extension of the ToDo application is that it can decide to only show only that the method completed or already completed. This extension is left as an exercise to the reader.

WRITING THE PROXY

From the code perspective, the most interesting part is the proxy to access the REST service (the proxy file is RESTProxy.java). Let's take a look at the most significant methods, starting with the method that returns the complete ToDo list:

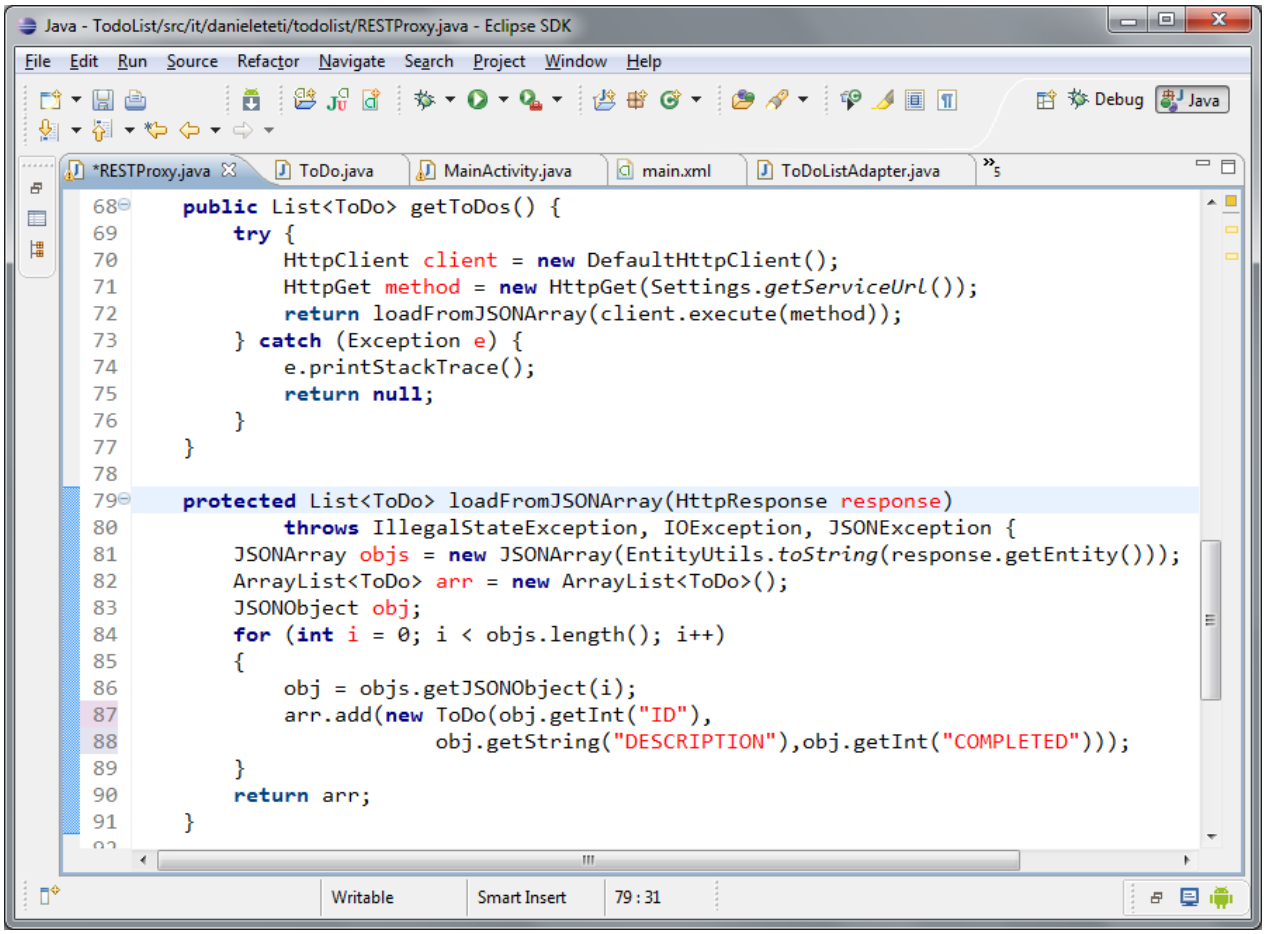
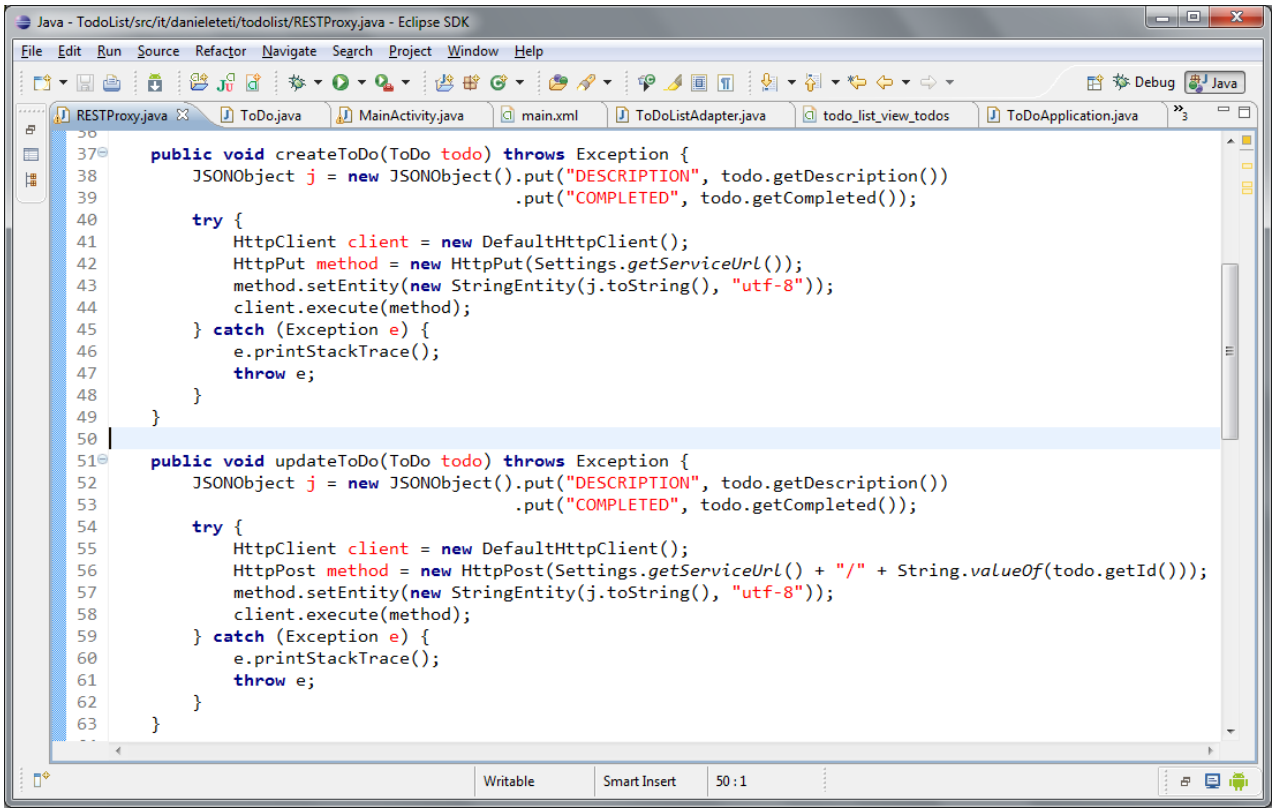


Figure 18: The code to convert the JSON array ToDo items into an ArrayList

As can be seen in Figure 18, the TODO list is returned from the service as an array of JSON objects. We have written a method that takes care of converting the JSON representation of ToDo into a real `List<ToDo>`. At this point the code should be clear. Creating and editing a ToDo item, is just a little bit more complex, as we have to handle the body of the request.




```
37    public void createToDo(ToDo todo) throws Exception {
38        JSONObject j = new JSONObject().put("DESCRIPTION", todo.getDescription())
39                                       .put("COMPLETED", todo.getCompleted());
40        try {
41            HttpClient client = new DefaultHttpClient();
42            HttpPut method = new HttpPut(Settings.getServiceUrl());
43            method.setEntity(new StringEntity(j.toString(), "utf-8"));
44            client.execute(method);
45        } catch (Exception e) {
46            e.printStackTrace();
47            throw e;
48        }
49    }
50
51    public void updateToDo(ToDo todo) throws Exception {
52        JSONObject j = new JSONObject().put("DESCRIPTION", todo.getDescription())
53                                       .put("COMPLETED", todo.getCompleted());
54        try {
55            HttpClient client = new DefaultHttpClient();
56            HttpPost method = new HttpPost(Settings.getServiceUrl() + "/" + String.valueOf(todo.getId()));
57            method.setEntity(new StringEntity(j.toString(), "utf-8"));
58            client.execute(method);
59        } catch (Exception e) {
60            e.printStackTrace();
61            throw e;
62        }
63    }
```

Figure 19: The create and update methods

Figure 19 shows more on the PUT and POST. In this case, in addition to sending the request with the appropriate method (PUT to create and POST to update) we are required to also send the body of the request in the expected format. The creation of JSON, necessary to represent the JAVA object in JSON, is the responsibility of the proxy method. In an application in the Real World, it is correct to ensure that every object can serialize its internal state independently via Reflection or by implementing an interface such as the following:

```
public interface JSONSerializable {
    public String toJSONString() throws JSONException;
    public void fromJSONString(String json) throws JSONException;
}
```

Please notice at line 56 of figure 19 the url is built to identify which ToDo item to edit, using parameters. The same criterion is used by the method `deleteToDo`.

THE APPLICATION

Let's start with the definition of the main activity. We want to have a ToDo list and a button to add a new item. Here is the content of the file layout `main.xml`.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:weightSum="50">
           <ListView
             android:id="@+id/lv_todo"
             android:layout_width="fill_parent"
             android:layout_height="wrap_content"
             android:layout_weight="49"
             android:layout_gravity="left"/>
           <Button
             android:id="@+id/btn_new_todo"
             android:text="New ToDo"
             android:layout_width="fill_parent"
             android:layout_height="wrap_content"
             android:layout_weight="1"/>
</LinearLayout>
```

In the onCreate method of the main activity we upload the TODO list from the server. But be careful, as you can read in the documentation (http://developer.android.com/reference/android/app/Activity.html) the activity requires precise LifeCycle handling. The OnCreate method is invoked whenever the activity is re-created. Unfortunately, when you change the orientation of your Android phone or emulator, the activity will be destroyed and recreated with the new layout to suit the new layout of the display. As a result, every time you turn the display on the phone, the activity will require a new ToDo list from the server. For this type of example it is not a problem, but for real world applications, it could potentially have implications. However, there are several techniques to avoid requesting the data from when there is a change in display orientation.

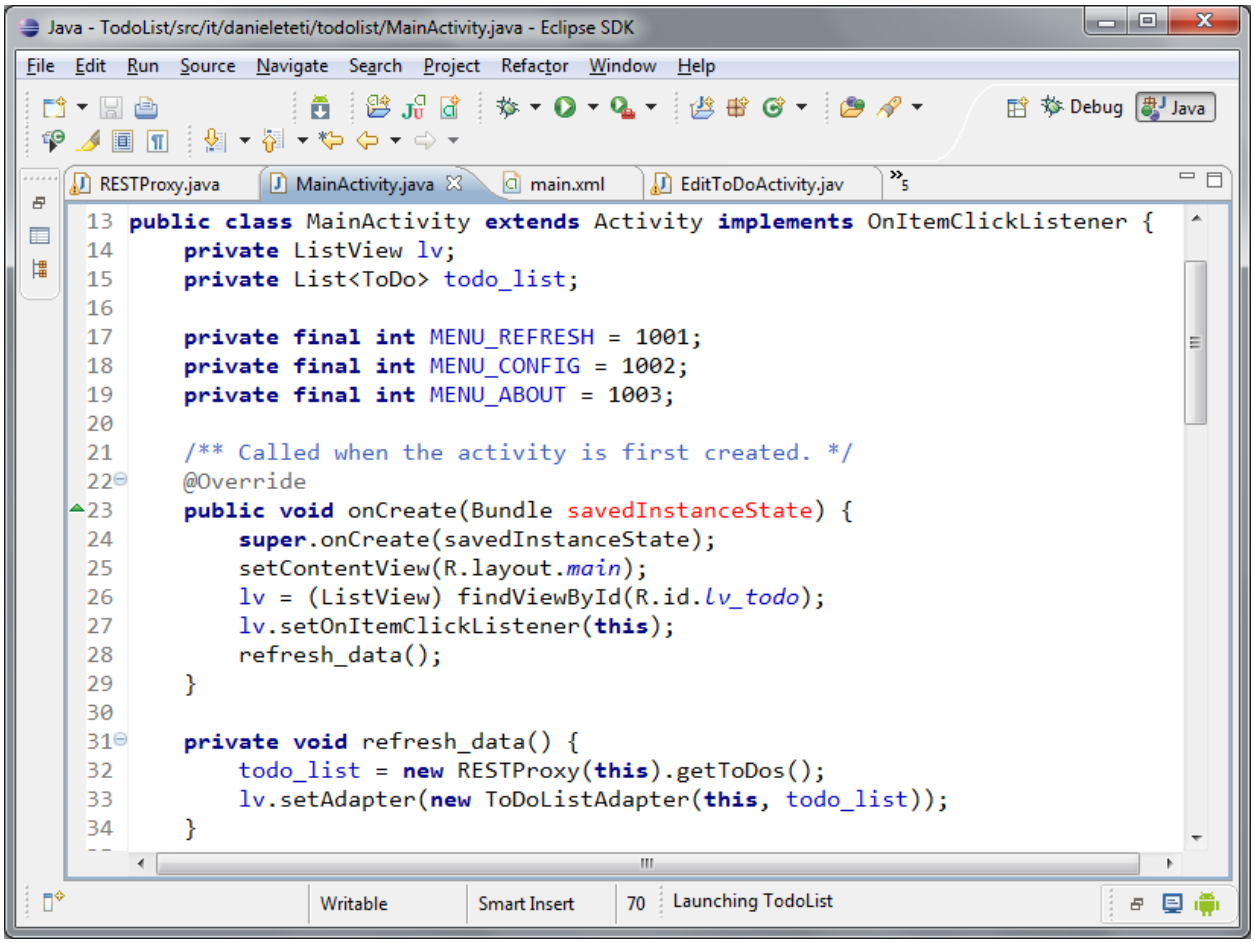


Figure 20: The onCreate methods in the code editor

As evident from the previous code, after setting the layout, as read from XML resources, we request the data from the remote service using the proxy. This then returns a simple list of items. This list is attached to the `ListView` through an adapter. To fully explain the purpose and implementation of a custom adapter is outside the scope of this paper. For now, consider it to be an object that handles the binding from a list of objects and the `ListView.` For more information see the following link http://developer.android.com/resources/tutorials/views/hello-listview.html .

On clicking "New ToDo" we need to create a new activity to edit the ToDo item. The mechanism used is that of Intent.

```
	public void newTodoClick(View btn) {
		Intent i = new Intent(this, EditToDoActivity.class);
		// start the other activity
		startActivity(i);
	}
```

The activity EditToDoActivity is used both to create a new ToDo item and to to edit an already existing one. In the demo code supplied, you can also check how to create menus for the activity. Each activity must be recorded in the file AndroidManifest.xml otherwise Android will fail to instantiate it. In AndroidManifest.xml file enter, the following line, just before the closing tag `</application>`:

```
<activity android:name="EditToDoActivity"></activity>
```

The code that accesses the service to update the data is contained in the activity `EditToDoActivity` and is trivial and is shown below in Figure 21.

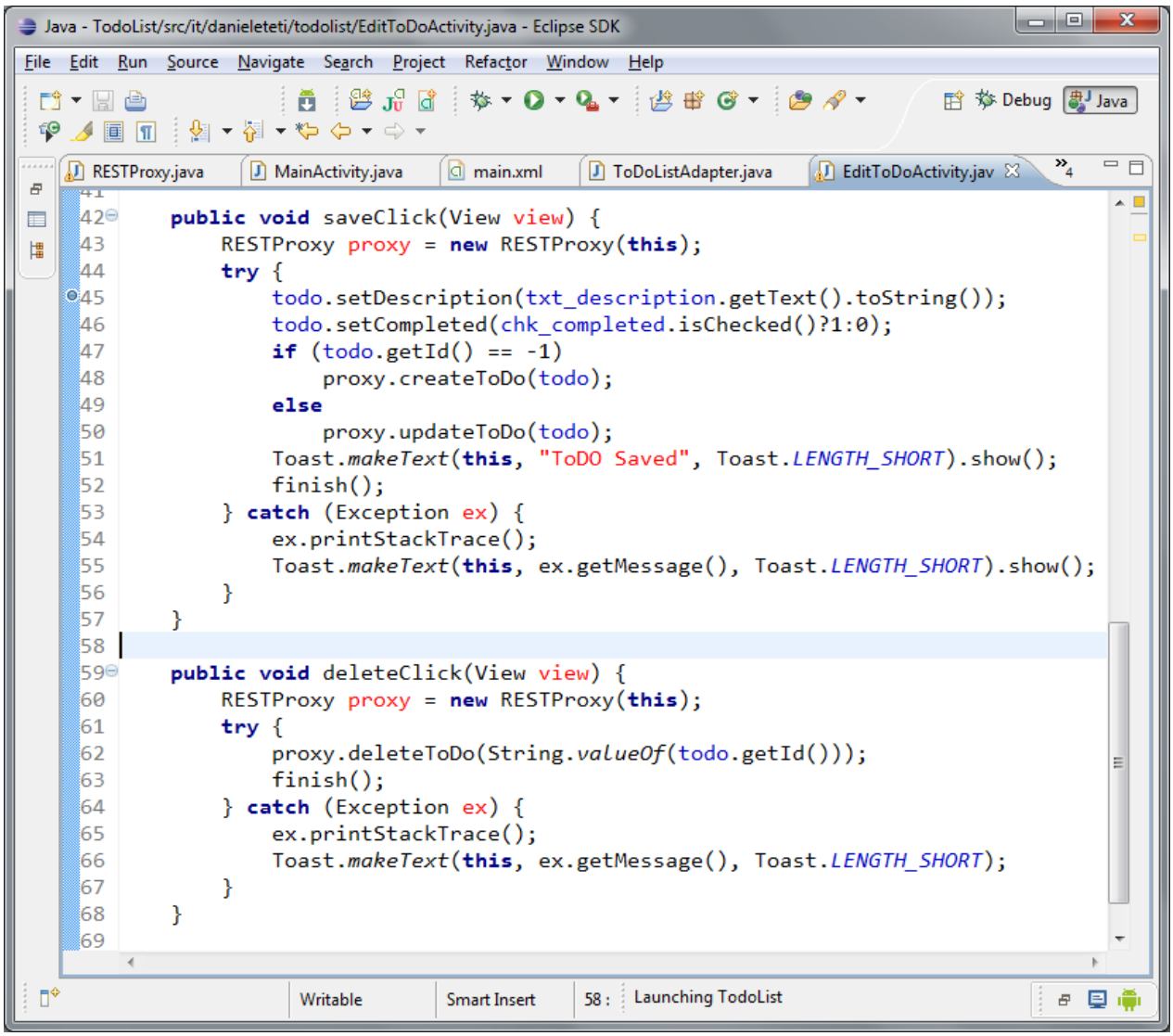


Figure 21 : The code to Save or Delete a ToDo

In this activity we have used a new way to link the OnClick event to the buttons. Instead of recovering the reference and attaching a listener, we have directly defined the name of the method to be invoked when the button is clicked in the layout file. This mechanism makes it even more like the layout file to the Delphi dfm. For details, see the file `res\layout\edit_todo.xml`.

SUMMARY

In this quick introduction to the Android world, we have only scratched the surface of the subject. We have nevertheless been introduced to several key concepts that allow the reader to start developing REST services in PHP, and enable an Android application to manage remote data while enjoying all the features of the mobile device.

If you would like to learn more, I would recommend that you study the samples in the SDK provided by Google. They can be found under the directory samples \<your-platform-number> of the SDK.

ABOUT RADPHP XE

Embarcadero® RadPHP™ XE revolutionizes PHP web development with a completely integrated, rapid visual development approach and component framework. RadPHP XE provides a powerful editor, debugger, visual development tools and connectivity with leading databases. The integrated reusable class library includes components for everything from UI design to building applications for Facebook. Learn more and download a free trial at http://www.embarcadero.com/products/radphp.

About the Author

Daniele Teti is R&D Director at bit Time Software, Embarcadero representative in Italy. He has many years of experience in the ICT world. He works on (and with) several open source projects for Delphi, PHP and Android communities.

He is speaker at Italian and international Delphi, PHP and Android conferences. In his spare time, Daniele likes writing articles for websites and Italian magazines.

He's also a staunch supporter of agile methodologies and design patterns and in its educational activity he spends lot of time on concepts related to design patterns, integration patterns and SOA/ROA.

You can read Daniele's blog at http://www.danieleteti.it, follow him on Twitter at http://twitter.com/danieleteti, and contact him on d.teti@bittime.it.

Embarcadero Technologies, Inc. is the leading provider of software tools that empower application developers and data management professionals to design, build, and run applications and databases more efficiently in heterogeneous IT environments. Over 90 of the Fortune 100 and an active community of more than three million users worldwide rely on Embarcadero's award-winning products to optimize costs, streamline compliance, and accelerate development and innovation. Founded in 1993, Embarcadero is headquartered in San Francisco with offices located around the world. Embarcadero is online at www.embarcadero.com.