RealThinClient SDK - Sending Dynamically Generated Content

Now we're going to create an application to send Dynamically Generated Content. We'll use as base the RealThinClient SDK – Your First Web Server project. You can download the source code to work with.

In this lesson, you'll see how two data providers can work together and how you can use multiple **Write** calls to send complex content out.

We are going to:

- Open Web Server's project.
- Add one **RtcDataProvider** component to our project.
- Connect **RtcDataProvider** to our server.
- Configure **RtcDataProvider** events watching for a particular request and adding content to be sent.
- Check that our server is running and returning the requested data.

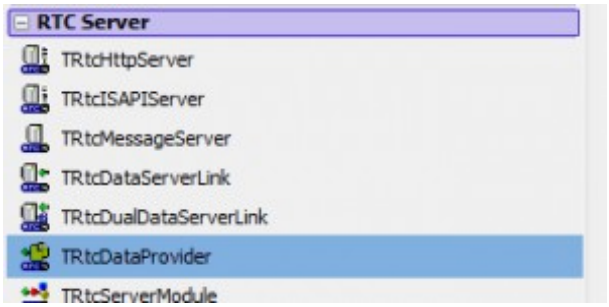
Steps.

1. Open our WebServer project.

First, we open our WebServer project.

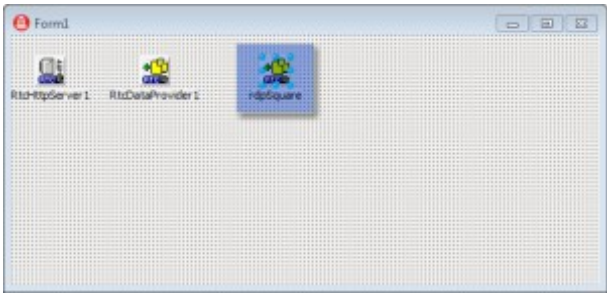
2. Add one RtcDataProvider component to our project

We're adding a new **RtcDataProvider** to our project that will produce a page with a table with square values from 1 to 100. We already have another **RtcDataProvider** component that is running in this project and is serving content for "/TIME" requests. Now we add another **RtcDataProvider** component that will serve content for "/SQUARE" requests.



RtcDataProvider component on RTC Server Group of components

We take an **TrtcDataProvider** component from the **RTC Server** components group and place it in our **Form1**.

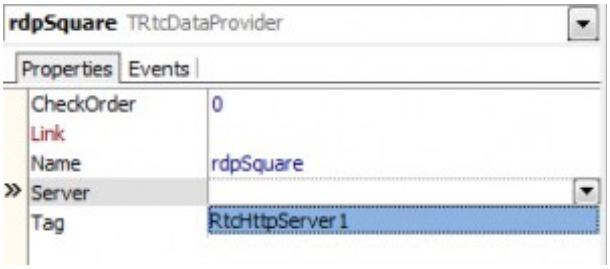


RtcDataProvider on Form1

In this example, the **RtcDataProvider** component has been renamed to **rdpSquare** to improve the readability of our project.

3. Connect our new RtcDataProvider component to our server.

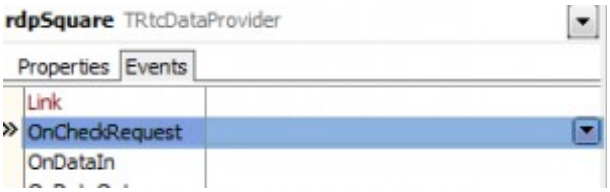
As we did in our first project, we need to tell our **RtcDataProvider** which server (**RtcHttpServer** component) it will use, Remember that we can have several server components listening in several ports at the same time.



Server Property for RtcDataProvider component

4. Define the OnCheckRequest event for our new component.

Now that we have a **RtcDataProvider** connected to a server, we need to define what type of request our server will listen for. In this case, this **RtcDataProvider** component will listen for "/SQUARE" requests.



OnCheckRequest Event for RtcDataProvider component

Using **with**:

```
procedure TForm1.rdpSquareCheckRequest(Sender: TRtcConnection);
begin
 with TRtcDataServer(Sender) do
 if UpperCase(Request.FileName) = '/SQUARE' then
 Accept;
end;
```

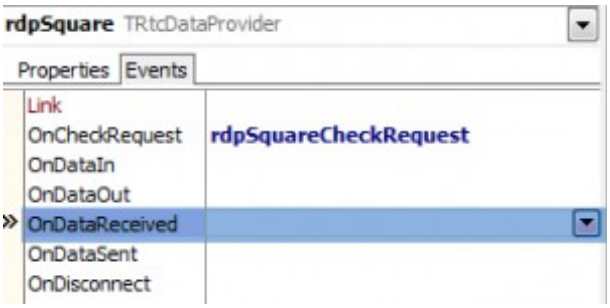
Without using **with**

```
procedure TForm1.rdpSquareCheckRequest(Sender: TRtcConnection);
 var
 rdsServer : TRtcDataServer absolute Sender;
begin
 if UpperCase(rdsServer.Request.FileName) = '/SQUARE' then
 rdsServer.Accept;
end;
```

5. Define the OnDataReceived event to send response out.

For small files or files where you feel can fit safely in your server's memory (for example, up to 32K) or to prepare the website output, there's no need to split the transfer. You can write the file out directly from the **OnDataReceived** event of your **RtcDataProvider** component. You can also use the **Write** method consecutively.

We have to define what content will be sent when a request for our **RtcDataProvider** component arrives.



OnCheckRequest Event for RtcDataProvider component

For this, as we previously did, we have to code the **OnDataReceived** event of our **RtcDataProvider** component.

Remember that we have to wait until the Request is complete to send a response back to the client requesting the data.

Using **with**:

```
procedure TForm1.rdpSquareDataReceived(Sender: TRtcConnection);
  var
  viLine : integer;
begin
  with TRtcDataServer(Sender) do
  begin
    if Request.Complete then
    begin
      Write('<html><body>');
      Write('<table border="1"><caption>Square Values</caption>');
      Write('<tr><td>Number</td><td>Square</td></tr>');
      for viLine := 1 to 100 do
      begin
        Write('<tr><td>' + IntToStr(viLine) + '</td>');
        Write('<td>' + IntToStr(viLine * viLine) + '</td></tr>');
      end;
      Write('</table></body></html>');
    end;
  end;
end;
```

Without using **with**:

```
procedure TForm1.rdpSquareDataReceived(Sender: TRtcConnection);
  var
    viLine : integer;
    rdsServer : TRtcDataServer absolute Sender;
begin
  if rdsServer.Request.Complete then
  begin
    rdsServer.Write('<html><body>');
    rdsServer.Write('<table border="1"><caption>Square
Values</caption>');
    rdsServer.Write('<tr><td>Number</td><td>Square</td></tr>');
    for viLine := 1 to 100 do
    begin
      rdsServer.Write('<tr><td>' + IntToStr(viLine) + '</td>');
      rdsServer.Write('<td>' + IntToStr(viLine * viLine) +
'</td></tr>');
    end;
    rdsServer.Write('</table></body></html>');
  end;
end;
```

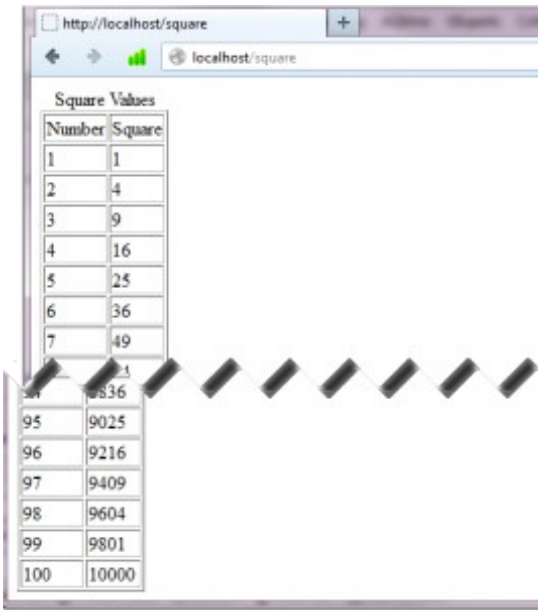
6. Save and run our project to check Server's response.

Once we have coded our **RtcDataProvider** component events, we compile and run the project.

If everything is right, we should see our **Form1** on screen.

Project Running

If we open a web browser and go to **http://localhost/square** we should get a table with the square values for numbers from 1 to 100.



Browser Output

We can also check that our previous **RtcDataProvider** component is sending content when we request /TIME. Go to **http://localhost/time** and you should get back the time in your machine.

Browser Showing Server´s Response

With this we finish our second Demo.