

# 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.

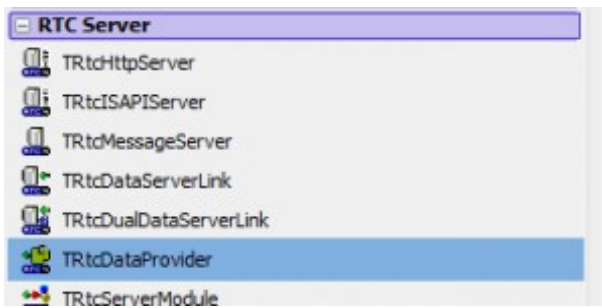
Steps.

## 1. Open our WebServer project.

First, we open our WebServer project.

## 2. Add one RtcDataProvider component to our project

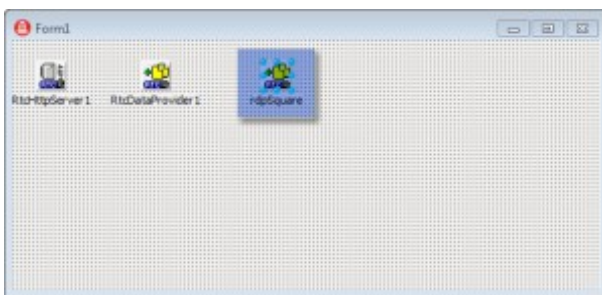
We're adding a new **RtcDataProvider** to our project that will produce a page with a table with square values from 1 to 100. We already have another **RtcDataProvider** component that is running in this project and is serving content for `"/TIME"` requests. Now we add another **RtcDataProvider** component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

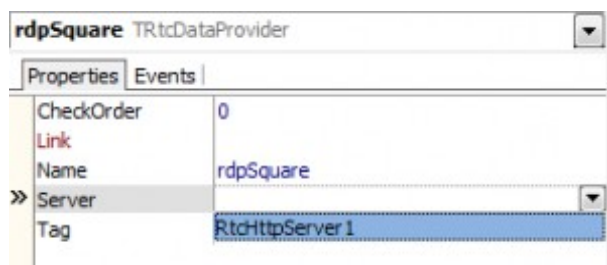


*RtcDataProvider on Form1*

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.

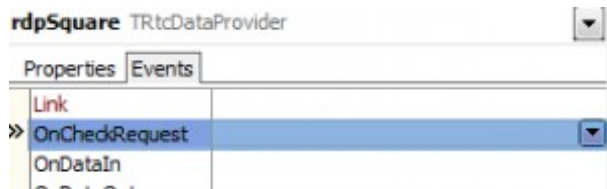
As we did in our first project, we need to tell our **RtcDataProvider** which server (**RtcHttpServer** component) it will use, Remember that we can have several server components listening in several ports at the same time.



*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**:

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

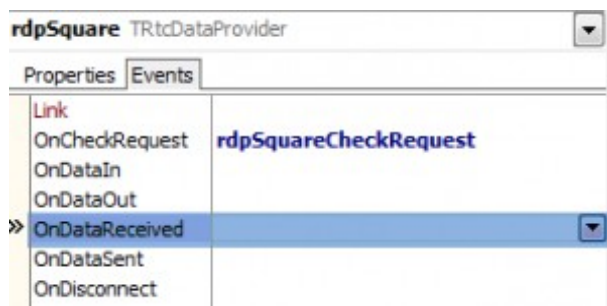
Without using **with**

```
1 procedure TForm1.rdpSquareCheckRequest(Sender: TRtcConnection);
2 var
3   rdsServer : TRtcDataServer absolute Sender;
4 begin
5   if UpperCase(rdsServer.Request.FileName) = '/SQUARE' then
6     rdsServer.Accept;
7 end;
8
9
```

#### 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**:

```
1 procedure TForm1.rdpSquareDataReceived(Sender: TRtcConnection);
2   var
3     viLine : integer;
4 begin
5   with TRtcDataServer(Sender) do
6     begin
7       if Request.Complete then
8         begin
9           Write('<html><body>');
10          Write('<table border="1"><caption>Square Values</caption>');
11          Write('<tr><td>Number</td><td>Square</td></tr>');
12          for viLine := 1 to 100 do
13            begin
14              Write('<tr><td>' + IntToStr(viLine) + '</td>');
15              Write('<td>' + IntToStr(viLine * viLine) + '</td></tr>');
16            end;
17          Write('</table></body></html>');
18        end;
19      end;
20 end;
```

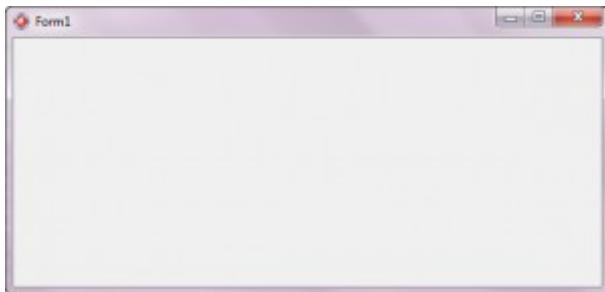
Without using **with**:

```
procedure TForm1.rdpSquareDataReceived(Sender: TRtcConnection);
1  var
2    viLine : integer;
3    rdsServer : TRtcDataServer absolute Sender;
4 begin
5   if rdsServer.Request.Complete then
6     begin
7       rdsServer.Write('<html><body>');
8       rdsServer.Write('<table border="1"><caption>Square
9Values</caption>');
10      rdsServer.Write('<tr><td>Number</td><td>Square</td></tr>');
11      for viLine := 1 to 100 do
12        begin
13          rdsServer.Write('<tr><td>' + IntToStr(viLine) + '</td>');
14          rdsServer.Write('<td>' + IntToStr(viLine * viLine) +
15'</td></tr>');
16        end;
17      rdsServer.Write('</table></body></html>');
18    end;
  end;
```

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

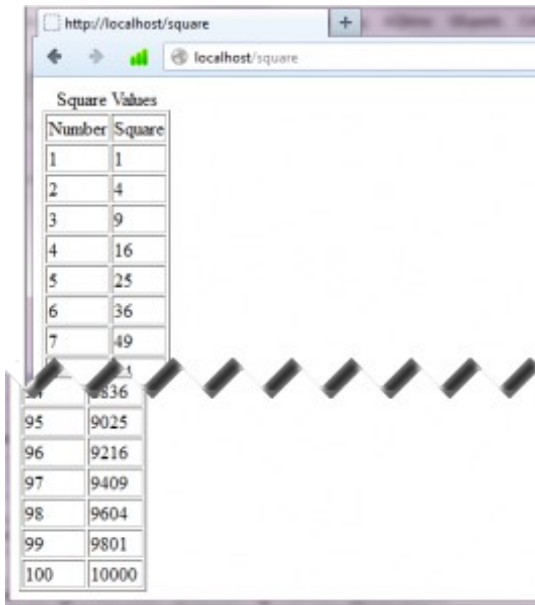
Once we have coded our **RtcDataProvider** component events, we compile and run the project.

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.

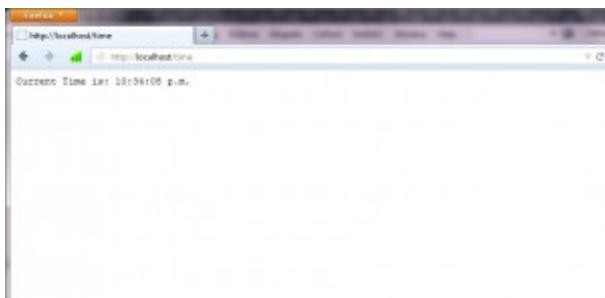


The screenshot shows a web browser window with the address bar set to <http://localhost/square>. The page content is a table titled "Square Values". The table has two columns: "Number" and "Square". The rows contain the numbers 1 through 100 and their corresponding squares. The table is partially obscured by a redacted area in the middle.

Number	Square
1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81
10	100
11	121
12	144
13	169
14	196
15	225
16	256
17	289
18	324
19	361
20	400
21	441
22	484
23	529
24	576
25	625
26	676
27	729
28	784
29	841
30	900
31	961
32	1024
33	1089
34	1156
35	1225
36	1296
37	1369
38	1444
39	1521
40	1600
41	1681
42	1764
43	1849
44	1936
45	2025
46	2116
47	2209
48	2304
49	2401
50	2500
51	2601
52	2704
53	2809
54	2916
55	3025
56	3136
57	3249
58	3364
59	3481
60	3600
61	3721
62	3844
63	3969
64	4096
65	4225
66	4356
67	4489
68	4624
69	4761
70	4900
71	5041
72	5184
73	5329
74	5476
75	5625
76	5776
77	5929
78	6084
79	6241
80	6400
81	6561
82	6724
83	6889
84	7056
85	7225
86	7396
87	7569
88	7744
89	7921
90	8100
91	8281
92	8464
93	8649
94	8836
95	9025
96	9216
97	9409
98	9604
99	9801
100	10000

*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.