```
 1:  ***********************************************
 2:   maXbox Starter 48
 3:  ***********************************************
 4:
 5:
 6:  Work with Micro Services
 7:  ---------------------------------
 8:  Max Kleiner
 9:
10:  //Zwei Worte werden Dir im Leben viele Türen öffnen - "ziehen" und "stossen".
11:
12:  Essentially, microservice architecture is a method of developing software
     applications as a suite of independently deployable, small, modular services or
     building blocks in which each service runs a unique process and communicates
     through a well-defined, lightweight mechanism to serve a business goal.
13:
14:  Such a microservice can be
15:  - a socket server
16:  - a data logger
17:  - signal sensor
18:
19:  Then we add some business goal to the service:
20:
21:  - a socket server as a time and temp server
22:  - a data logger to store climate samples
23:  - signal sensor to get temperature and others
24:
25:  The idea behind microservices is that some types of applications become easier to
     build and maintain when they are broken down into smaller, composable pieces which
     work together. Each component is developed separately, and the application is then
     simply the sum of its constituent components.
26:  First example is the main of a http-server:
27:
28:  begin   //@main
29:   //TWebServerCreate;
30:   with TIdHTTPServer.Create(Nil) do begin
31:     sr:= GetIPfromHost(getHostName)  //'172.16.10.80';
32:     Bindings.Add.IP:= sr;
33:     Bindings.Add.Port:= 8080;
34:     OnCommandGet:= @HTTPServerCommandGet;
35:     Active:= True;
36:     try
37:       Writeln('Hello world/Web server start at: '+sr);
38:       ShowMessageBig('maXbox Hello WorldWeb server at: '+sr+#1310+
39:                      '  Press OK to quit webserver!'+#13);
40:     finally
41:       writeln('SocketServer stop: '+timetoStr(now)); //TWebServerDestroy;
42:       Active:= False;
43:       Free;
44:     end;
45:   end;
46:
47:   With Event-handlers or a delegate you do have the flexibility to act as a
     microservice. The OnCommandGet() event can be changed with a lot of use cases at
     design or at runtime as well.
48:  In this example like in Object Pascal or C#, you can think of a delegate as a
     pointer (or a reference) to a method. This is useful because the pointer can be
     passed around as a value like in above case @HTTPServerCommandGet;:
49:
50:  procedure HTTPServerCommandGet(AContext: TIdPeerThread;
51:               ARequestInfo: TIdHTTPRequestInfo; ARespInfo: TIdHTTPResponseInfo);
52:   begin
53:     ARespInfo.ResponseNo:= 200;
54:     ARespInfo.ContentType:= 'text/plain';
55:     ARespInfo.ContentText:= 'Hi IBZ TimeServe at: '
56:                                 +DateTimeToInternetStr(Now,true);
57:   end;
```

```
58:
59:  The central concept of a delegate is its signature, or shape:
60:
61:  HTTPServerCommandGet(AContext: TIdPeerThread;
62:               ARequestInfo: TIdHTTPRequestInfo; ARespInfo: TIdHTTPResponseInfo);
63:
64:  To do this, we create specific methods for the code we want to be executed. The
     glue between the event and the methods (event handlers) to be executed are the
     delegates.
65:  The common definition of microservices generally relies upon each microservice
     providing an API endpoint, often but not always a stateless REST API which can be
     accessed over HTTP(S) just like a standard webpage. This method for accessing
     microservices make them easy for developers to consume as they only require tools
     and methods many developers are already familiar with.
66:
67:  This is how get get the TMP36 sensor value from Arduino:
68:
69:  function connectAndGetValue: string;
70:  begin
71:    with TBlockSerial.Create do begin
72:      Config(9600,8,'N',1,true,false);
73:      Connect(COMPORT);
74:      result:= RecvString(1800)   //com timeout
75:      CloseSocket;
76:      Free;
77:    end;
78:  end;
79:
80:  Then the result is pushed to a web socket in a timer mode with another delegate:
81:
82:    arTimer:= TTimer.Create(Self);
83:    arTimer.Enabled:= true;
84:    arTimer.Interval:= 2000;
85:    arTimer.OnTimer:= @eventActTimer;
86:
87:   procedure eventActTimer(sender: TObject);
88:   begin
89:     tmpval:= connectAndGetValue;
90:     writeln(datetimetostr(now)+' C°: '+tmpval+'° >'+aremoteIP)
91:     aremoteIP:= '';
92:   end;
93:
94:  Be aware of the remoteIP:
95:  Exception: Could not bind socket. Address and port are already in use.
96:    PrintF('Command %s received: %s of temperature C°: %s',
97:            [RequestInfo.Command,thread.connection.Socket.binding.PeerIP,tmp2]);
98:
99:  This is a common newbie mistake. You are creating two bindings, one bound to
     127.0.0.1:DefaultPort, and one bound to 0.0.0.0:50001. You need one binding instead,
     that is bound to 127.0.0.1:50001 instead.
100:
101:  with HTTPServer1.Bindings.Add do begin
102:    IP:= '127.0.0.1';
103:    Port:= 50001;
104:  end;
105:
106:  In its simplest forms, we can call now the service from a browser or a desktop app
     like a web- or win form. At least the client call:
107:
108:  procedure TDataFormbtnHTTPSendGetClick(Sender: TObject);
109:    var
110:      HTTPClient : TIdHTTP;
111:      responseStream : TMemoryStream;
112:    begin
113:      HTTPClient:= TIdHTTP.Create(Nil);
114:      responseStream:= TMemoryStream.Create;
115:      try
```

```
116:       try
117:         HTTPClient.Get1('http://127.0.0.1:8080',responseStream);
118:         responseStream.Seek(0, soFromBeginning);
119:         SetLength(Sr, responseStream.Size);
120:         responseStream.Read(Sr, responseStream.Size);
121:         writeln('response: '+sr)
122:       except
123:         //on e : Exception do begin
124:         Showmessage('Could not send get request to localhost, port 8080');
125:       end;
126:       //end;
127:     finally
128:       //@FreeAndNil(HTTPClient);
129:       HTTPClient.Free;
130:       HTTPClient:= Nil;
131:       responseStream.Free;
132:     end;
133:   end;
134:
135:
136:  Or take another old concept from cryptography RSA. Encode and decode can bee seen
     as two micro services with different use cases:
137:  We have public and private keys, each including of two values.
138:  For the public key the values are n of p*q, the so called "modulus", and E, a well
     known encrypting integer prime with the value: Const E = 65537;.
139:
140:  The private key values are also n, the same modulus that appears in the public key,
      and d, a big number which can decrypt any message encrypted using the public key.
141:
142:  There are obviously two cases:
143:
144:    1. Encrypting with the public key, and then decrypting with the private key.
145:       For a message or data
146:    2. Encrypting with the private key, and then decrypting with the public key.
147:       For a digital signature
148:
149:  Conclusion:
150:  The idea of separating applications into smaller parts is nothing new; there are
     other programming paradigms which address this same concept, such as Service
     Oriented Architecture (SOA). What may be new are some of the tools and techniques
     used to deliver on the promise of microservices like Docker, OpenStack or OpenShift.
151:
152:  Each service should be independently developed and deployed. No coordination
     should be needed with other service teams if no breaking API changes have been
     made. Each service is effectively it's own product with it's own codebase and
     lifecycle.
153:                      ----
154:                    /~@@~\,
155:  _______  . _\_\___/\ __ /\___|_|_ . _______
156: / ____  |=|       \  <_+>  /       |=|  ____ \
157: ~|    |\|=|======\\_______//======|=|/|    |~
158:  |_   |    \       |      |      /    |    |
159:   \==-|     \      |  mX4 |     /     |----|~~)
160:   |   |      |     |      |     |     |____/~/
161:   |   |       \____\____/____/      /    / /
162:   |   |         {----------}       /____/ /
163:   |___|        /~~~~~~~~~~~~\      |_/~|_|/
164:    \_/        [/~~~~~||~~~~~\]      /__|\
165:    | |         |    ||||    |      (/|[[\)
166:    [_]        |     |  |     |
167:               |_____|  |_____|
168:               (_____)  (_____)
169:               |     |  |     |
170:               |     |  |     |
171:               |/~~~\|  |/~~~\|
172:               /|___|\  /|___|\
173:              <_______><_______>
```

```
174:
175:  A microservice architecture shifts around complexity. Instead of a single complex
      system, you have a bunch of simple services with complex interactions.
176:
177:
178:  Ref: http://www.softwareschule.ch/maxbox.htm
179:
180:   ..\examples\210_RSA_crypto_complete8hybrid.txt
181:   ..\examples\750_ibz_cryptomem_RSA_proof_64.txt
182:   ..\examples\749_helloWebServer3_tempsensor3.txt
183:   ..\examples\749_helloWebServer3.txt
184:
185:
186:  Doc:
187:
188:   https://opensource.com/resources/what-are-microservices
189:
190:   http://www.delphiforfun.org/Programs/Math_Topics/RSA_KeyDemo.htm
191:
192:   http://stackoverflow.com/questions/803242/understanding-events-and-event-handlers-
      in-c-sharp
193:
194:   http://www.vinaysahni.com/best-practices-for-building-a-microservice-architecture
195:
196:   http://www.softwareschule.ch/download/maxbox_functions.txt
197:
198:   There are only 10 types of people: those who understand binary and those do not.
199:
200:
201:
202:                _od#HMM6&*MMMH::-_
203:            _dHMMMR??MMM? ""| `"'-?Hb_
204:         .~HMMMMMMMMHMMM#M?          `*HMb.
205:       ./?HMMMMMMMMMMM"*""""           &MHb.
206:      /'|MMMMMMMMMM'                -    `*MHM\
207:     /  |MMM'MMHHM''                      .MMMHb
208:    |   9HMMP   .Hq,                      TMM'MMH
209:   /     |MM\,H-""&&6\__                  `MMMMMMb
210:  |       `""HH#,       \                - MMMMMMM|
211:  |            `HoodHMM###.                 `9MMMMMH
212:  |               .MMMMMMM##\                 `*"?HM
213:  |          ..  ,HMMMMMMMMMMo\.                   |M
214:  |              |MMMMM'MMMMMMM'MNHo               |M
215:  |              ?MMMMMMM'MMMMMMMM*                |H
216:  |.               `#MMMMMMMM'MMMM'              .M|
217:   \                 `MMMMMMMMMMM*               |P
218:   `\                 MMMMMMMMT"'               ,H
219:    `\                `MMM'MMH?                 ./
220:      \.              |MMMH#"                  ,/
221:       `\.            |MMP'                 .,/'
222:         `~\          `HM:.-    .        ,/'
223:            "-\_        '_\ .      _.-"
224:                "-\-#odMM\_,oo==-"
```