



maXbox Starter 27

Start with XML

1.1 From ASCII to XML

XML is not a replacement for HTML!

This provocation starts the Tutorial 27 with XML.

XML (Extensible Markup Language) is a flexible way to create common information formats and share both the format and the data on the World Wide Web, intranets, wikis, configuration and elsewhere in a cloud. For example, computer assemblers might agree on a standard or common way to describe the information about a computer product (processor speed, case mode, memory size, and so on) and then describe the product information format with XML.

Or suppose you are the engineer of a new TGV train type with 3 possible type settings:

```
= <SOAP-ENC:Array id="ref-35" SOAP-ENC:arrayType="xsd:anyType[16]">
  <item id="ref-46" xsi:type="SOAP-ENC:string">proto</item>
  <item id="ref-47" xsi:type="SOAP-ENC:string">TGV A2</item>
    <item xsi:type="xsd:int">304</item>
    <item xsi:type="xsd:int">230</item>
    <item xsi:type="xsd:int">490</item>
  </SOAP-ENC:Array>
```

That's how XML look likes and you imagine it's stored in a file.

I mentioned first XML it's not a replacement for HTML. The acronym HTML stands for Hyper Text Markup Language - the primary programming language used to write content or visual controls on the web. I'll explain that now:

XML and HTML were designed with different goals:

- XML was designed to transport and store data, with focus on what data is
- HTML was designed to display data, with focus on how data appears

HTML is about displaying information, while XML is about carrying information. This tutorial is based on the idea of carrying and transport information.

Next you see a so called interface description with message information:

```
= <binding name="IVCLScannerbinding" type="tns:IVCLScanner">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
    = <operation name="PostData">
      <soap:operation soapAction="urn:VCLScanIntf-IVCLScanner#PostData" style="rpc" />
    = <input message="tns:PostData0Request">
```

A SOAP or interface message is an ordinary XML document containing the following elements:

- An Envelope element that identifies XML documents as a SOAP message

- A Header element that contains header information
- A Body element that contains call and response data
- A Fault element containing errors and status data

SOAP provides a way to communicate between applications running on different operating systems, with different technologies and programming languages.

☞ XML data is stored in plain text format. This provides also a software- and hardware-independent way of storing data.

You see XML is a large topic but we won't go that far. Just one structured example and then we dive into 2 code examples to generate and measure some XML data.

This file you can get over the internet and it has the purpose of an ini-file:

<http://max.kleiner.com/myconfig.xml>

The ini-file format is still popular; many configuration files (such as desktop or persistence settings file) are in this format. This format is especially useful in cross-platform applications and mobile devices where you can't always count on a system registry for storing configuration data or save the user options.

It shows the implementation of 2 databases (Interbase and `mySQL`) which communicate with a client over TCP and sockets. I made it scriptable to demonstrate the fascination of 2 database boxes communicate with each other at the same time.

```
<?xml version="1.0" ?>
= <databases>
  = <database db="InterBase">
    = <connection>
      <name>TrueSoft</name>
      <path>D:\\SuperPath</path>
      <user>MaxMin</user>
    </connection>
  </database>
  = <database db="mySQL">
    = <connection>
      <name>maXml</name>
      <server>././kylix02</server>
      <user>carpeDiem</user>
    </connection>
  </database>
</databases>
```

The note of the ini-file as XML above is quite self descriptive. It has database and connection information, it also has a name and a user body.

But wait a second where's the function? Yeah you got it; this XML document (or most others) does not DO anything. It is just information wrapped in tags and structure. Someone must write a box of software to send, receive or display it.

When you read data, or write data, a function will not return until the operation of the software is complete. And once again:

☞ XML is used to transport and store data, while HTML is used to format and display the data. (XML became a W3C recommendation on February 10, 1998)

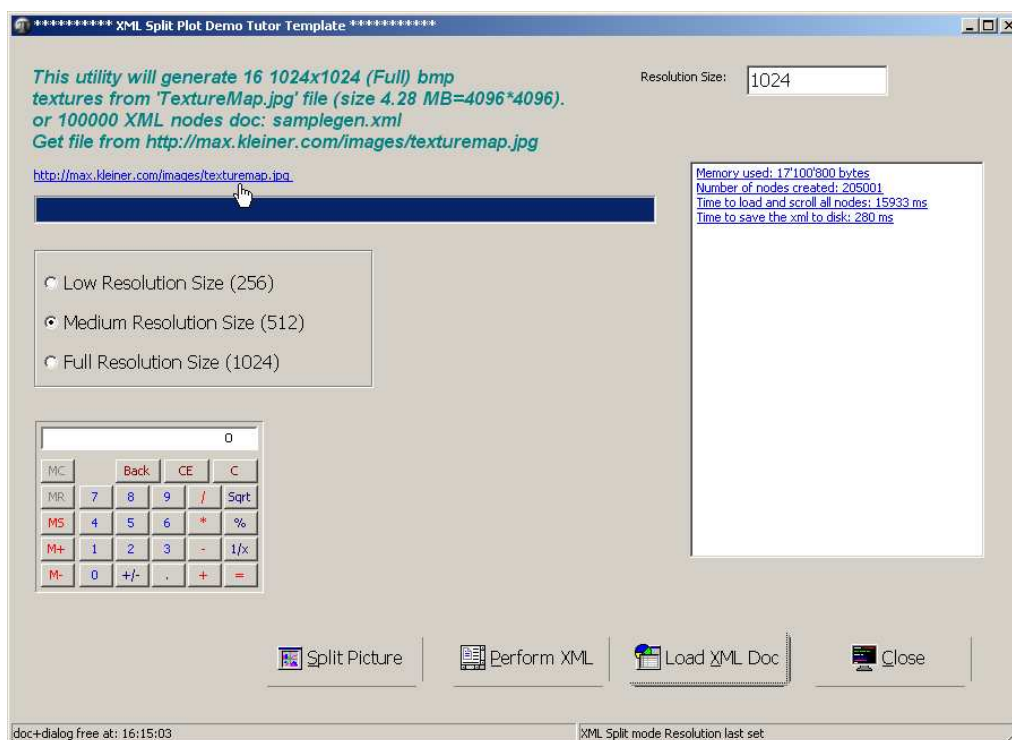
By the way: XML is the most common standard for data transmissions between all sorts of applications. For example, to connect simply to the above database call the connect path or server description from the XML file and wait for it to return. If it succeeds, it will return when it does. If it fails, it will raise an exception.

So far so dude, let's start scripting. If a maXbox script or app is programmed to assume to get a file, it is always started relative to the path where the `maXbox3.exe` as the host itself is:
So for example you want to store or load a file, all your external files (we need one file below mentioned) or resources in a script will be found relative to `ExePath()`:

```
E:\Program Files\maxbox\maxbox3\samplen3.xml
```

In this case `ExePath` is `> E:\Program Files\maxbox\maxbox3.`

`ExePath` is a useful function where you always get the path of `maXbox`.
If someone tries to start (install) the script or the app to or from a different drive for space or organizational reasons, it may fail to (install) or to run the script after installation¹.
You find all info concerning run environment of the app and script in menu
`../Program/Information/.....`
Next you see the XML App:



You need the following script file ready to download and start with compile:

http://www.softwareschule.ch/examples/440_XML_Tutor2.txt

Now we want to produce some huge data. In reality, an XML file can be large enough so the structure generates some overhead you don't have with a binary file. That's the price for transparency and common readability standard.

Such a standard way of describing data would enable a user to send an intelligent agent (a program) to each computer maker's web site, gather data, and then make a valid comparison. XML can be used by any person or group of individuals or companies that wants to share information in a consistent way.

¹ You don't have to install `maXbox` or a script anyway, just unzip and drop the script

So the app 440_XML_Tutor2.txt has the following functions:

- [Perform XML] Generate 100000 XML nodes to a file called `samplegen.xml`
- [Load XML Doc] Load and save the file as `samplegen_loadandsave.xml` and measure its performance
- [] Load the data for a database transaction.

And forget the [Split Picture] it's a function for forthcoming store picture in an XML database; but not topic of this tutorial. The third point needs further explanation.

XML describes the content in terms of what data is being described. For example, the word "phonenum" placed within markup tags (like `<server>`) could indicate that the data that followed was a phone number.

This means that an XML file can be processed purely as data by a program or it can be stored with similar data on another machine or, like an HTML file, that it can be displayed. For example, depending on how the application in the receiving computer wanted to handle the phone number, it could be stored, displayed, or dialled!

Another example of the above showed ini-file concerning databases. The word `<connection>` and `<server>` within tags could be stored, configured or connected.

So I think its time for practice. Load the script and press the button [Perform XML] then you press [Load XML Doc] and chose the just produced file `samplegen.xml`. It will take some time running through the nodes and store the result as `samplegen_loadandsave.xml` but in the end you will get two new files on your store with the same size as proof of concept:

```
23.05.2013  19:54      <DIR>          examples
17.03.2013  13:14      <DIR>          web
21.01.2014  20:37          1'094 sample33.xml
30.01.2014  19:58          4'695'013 samplegen.xml
30.01.2014  19:43          4'695'013 samplegen3.xml
30.01.2014  19:59          4'695'013 samplegen_loadandsave.xml
          9 Datei(en)          14'238'681 Bytes
          0 Verzeichnis(se),  5'065'957'376 Bytes frei
```

This first project 440_XML_Tutor2 has been designed to be as simple as possible. But what does it mean running through the nodes?

☞ The XML DOM defines a standard way for accessing and manipulating XML documents.

A DOM (Document Object Model) defines a standard way for accessing, navigating and manipulating documents.

Therefore the XML DOM defines a standard way for accessing and manipulating XML documents. And the XML DOM views an XML document as a tree-structure.


All elements can be accessed through the DOM tree. Their content (element, text and attributes²) can be modified or deleted, and new elements can be created. The elements, their text, and their attributes are all known as **nodes**.

The first step in building a XML file is to understand the **DOM**. For standard files this is done by reading the appropriate tutorial RFC. I'm just kidding ;-).

☞ Most standards are conversational and plain text. Conversational means that a command or node is given, a status response follows, and possibly data. Protocols that are very limited are often not conversational, but are still plain text.

Plain text makes protocols much easier to debug and also using different programming languages and operating systems.

² Text, Attribute, Element, Comment, Document, CData Nodes

 The syntax rules of XML are very simple and logical. The well defined rules are easy to learn, and easy to use. Each opener should have a closure in a **well formed** way:

`<connection><path> attributes ./ elements </path></connection>`

(They are said to be "Well Formed" XML documents).

1.2 Get the Node

How about the code of our 100000 nodes function? The procedure will accept multiple commands, until a Quit or Count command is received; means the 2 for loops reach 100000 nodes.

First we create and configure some 3 objects of the classes TALXmlDocument and TALXmlNode:

```
procedure TForm1.BtnGenerate100000NodeWithALXmlDocumentClick(Sender:
TObject);
begin
    Try
        ProgBar.Position:= 0;
        aXMLDocument:= TALXmlDocument.Create;
        try
            //aXMLDocument:= CreateEmptyXMLDocument('root');
            aStartDate:= GetTickCount;
            aXMLDocument.Active:= true;
            aXMLDocument.AddChild('root');
            ProgBar.StepBy(4);
            For k:= 1 to 1000 do begin
                aNewRec:= aXMLDocument.DocumentElement.AddChild(alRandomStrU(8),0);
                aNewrec.Attributes[alRandomStrU(8)]:=alRandomStrU(25);
                aNewrec.Attributes[alRandomStrU(8)]:=alRandomStrU(25);
                For i:= 1 to 100 do begin
                    aValueRec:= aNewRec.AddChild(alRandomStrU(8),0);
                    aValueRec.Text:= alRandomStrU(25);
                end;
            end;

            ShowmessageBig('Close OK to shutdown TCP Server listen on:
                            '+intToStr(Bindings[0].Port));

        end;
        Writeln('Server Stopped at '+DateTimeToInternetStr(Now, true))
    end;
```

The only important parts that are XML specific are the `aXMLDocument.AddChild('root')` method, `aNewrec.Attributes` method, and the `aValueRec` method.

The `AddChild()` list represents a data Container as a key – value dictionary in a compound collection.

The following properties as the **root** node are our point of interest as an important rule:

- XML documents must contain one element that is the parent of all other elements. This element is called the **root** element.
- In XML, all elements must be properly nested within each other
- XML elements can have attributes in name/value pairs just like in HTML

We can say that an XML document forms a iterated tree structure that starts at "the root" and branches to "the leaves".

The elements in an XML document form a document well formed tree. The tree starts at the root and branches to the lowest level of the tree. And what about the children of the revolution?



All elements can have sub elements called child elements.

As you will see in the following list, more classes in a DOM are needed for a full working with XML: This is the list for coding client requests.

- `procedure SIRegister_TALXMLDocument;`
- `procedure SIRegister_TALXmlNotationNode;`
- `procedure SIRegister_TALXmlDocFragmentNode;`
- `procedure SIRegister_TALXmlDocTypeNode;`
- `procedure SIRegister_TALXmlEntityNode;`
- `procedure SIRegister_TALXmlEntityRefNode;`
- `procedure SIRegister_TALXmlCDATADataNode;`
- `procedure SIRegister_TALXmlProcessingInstrNode;`
- `procedure SIRegister_TALXmlCommentNode;`
- `procedure SIRegister_TALXmlDocumentNode;`
- `procedure SIRegister_TALXmlTextNode;`
- `procedure SIRegister_TALXmlAttributeNode;`
- `procedure SIRegister_TALXmlElementNode;`
- `procedure SIRegister_TALXMLNode;`
- `procedure SIRegister_TALXMLNodeList;`
- `procedure SIRegister_EALXMLDocError;`
- `procedure SIRegister_ALXMLDoc;`

XML is also "extensible" because, unlike HTML, the markup symbols are unlimited and self-defining. XML is actually a simpler and easier-to-use subset of the Standard Generalized Markup Language (SGML), the standard for how to create a document structure.

It is expected that HTML and XML will be used together in many Web applications. XML markup, for example, may appear within an HTML page.



Early applications of XML include Microsoft's Channel Definition Format (CDF), which describes a channel, a portion of a web site that has been downloaded to your hard disk and is then is updated periodically as information changes.

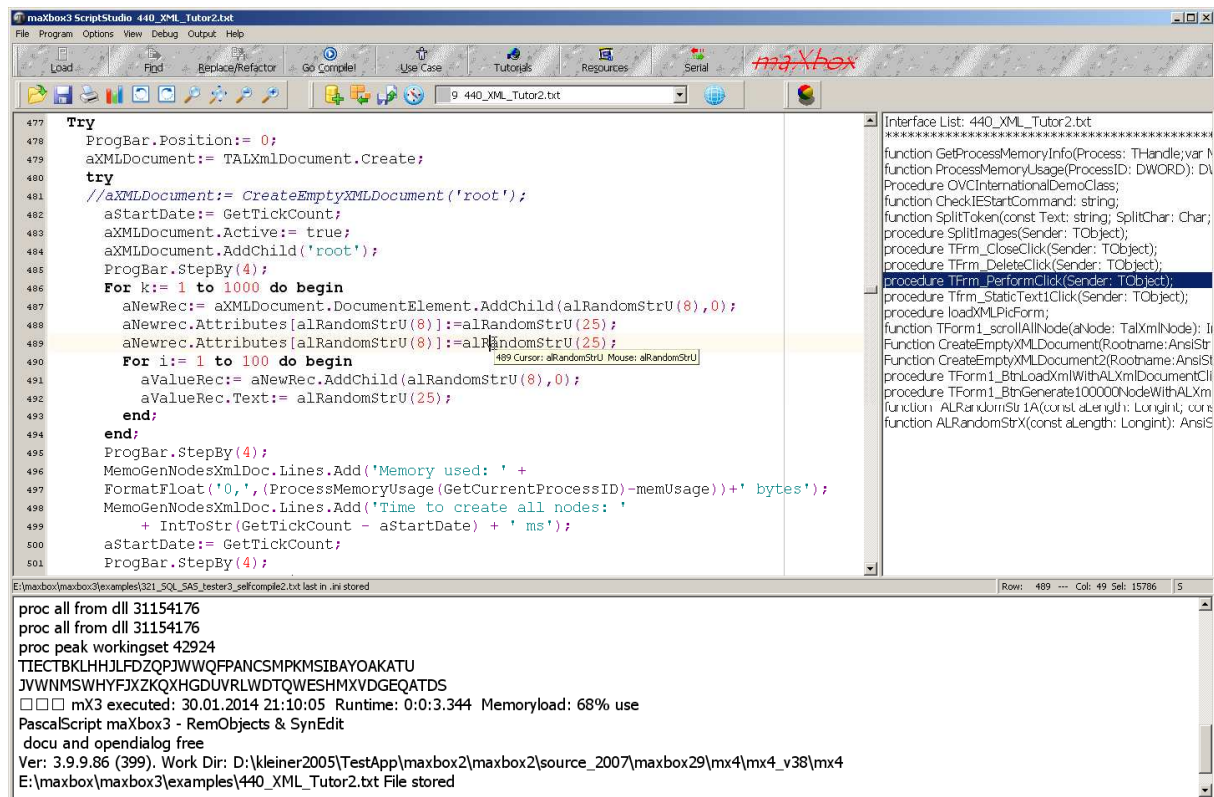
A specific CDF file contains data that specifies an initial Web page and how frequently it is updated.

Also Indy server components create a listener thread that is separate from the main thread of the program to use such XML agents. The listener thread listens for incoming client requests. For each client that it answers, it then spawns a new thread to service that client. The appropriate events are then fired within the context of that thread.

We should note that beside DOM also SAX exists. Wiki says:

SAX (Simple API for XML) is an event-based sequential access parser API developed by the XML-DEV mailing list for XML documents. SAX provides a mechanism for reading data from an

XML document that is an alternative to that provided by the Document Object Model (DOM). Where the DOM operates on the document as a whole, SAX parsers operate on each piece of the XML document sequentially. SAX parsers have some benefits over DOM-style parsers.



1. Many Lines to push XML

At least take a look about measuring the nodes performance. The below lines read the nodes with 3 specific optimisations of a stack (see appendix) and puts the input into the local push method and get it back with a pop method variable i: integer.

```
Function TForm1_scrollAllNode(aNode: TalXmlNode): Integer;
Var aStack: TStack;
    i: integer;
begin
    Result := 0;
    aStack := TStack.Create;
    try
        {with TSTVMatrix.Create(1,2,3,4,'',6) do begin
        flushcache; free; end;}
        For i := 0 to aNode.ChildNodes.Count - 1 do
            aStack.Push((ANode.ChildNodes[i]));
        While astack.Count > 0 do begin
            inc(result);
            aNode := TalXmlNode(astack.Pop);
            If assigned(ANode.ChildNodes) then
                For i := 0 to ANode.ChildNodes.Count - 1 do
                    aStack.Push((ANode.ChildNodes[i]));
            If assigned(ANode.AttributeNodes) then
                For i := 0 to ANode.attributeNodes.Count - 1 do
```



```

aStack.Push((ANode.AttributeNodes[i]));
end;
finally
aStack.Free;
end;
end;
end;

```

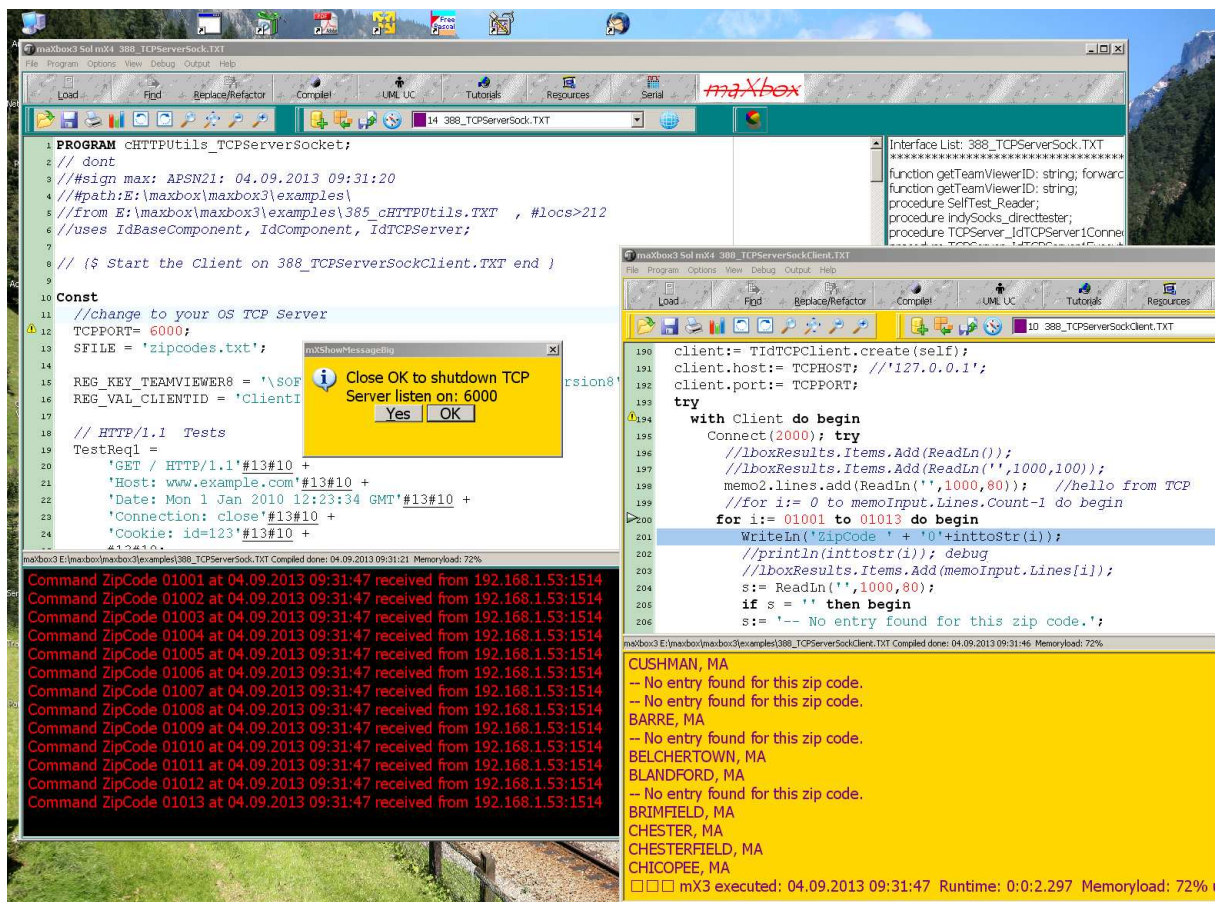
Next the call of TForm1_scrollAllNode() which is stack parsed to see which memory and time consumption the client issued.

```

aStartDate:= GetTickCount;
aALXMLDocument.LoadFromFile(mOpenDialog.FileName, false);
aNodeCount:= TForm1_scrollAllNode(aALXMLDocument.Node);
MemoLoadXmlDocu.Lines.Add('Memory used: ' +
FormatFloat('0,', (ProcessMemoryUsage(GetCurrentProcessID) -
memUsage)) + ' bytes');
MemoLoadXmlDocu.Lines.Add('Number of nodes created: ' +
IntToStr(aNodeCount));

```

If the command index top is "Count" the loop stops at the end of the list. No more reading or writing of the node is permitted after that call. The result is the overall node count. At the very end a topic to remember to exchange XML data:



2: TCP Client/Server Script with an XML exchange

Client is a TIdTCPClient and is a component in the app. The following properties were altered from the default:

- Host = 127.0.0.1 - Host was set to contact a server on the same machine as the client (or set by 192.168.1.53).
- Port = 6000 - An arbitrary number for this demo. This is the port that the client will contact the server with.

```

procedure TCPMain_Connect1Click(Sender: TObject);
begin
    //btnLookup.Enabled:= true;
    client:= TIdTCPClient.create(self);
    client.host:= TCPHOST; //'127.0.0.1';
    client.port:= TCPPORT;
    try
        with Client do begin
            Connect(2000); try
                //lboxResults.Items.Add(ReadLn());
                memo2.lines.add(ReadLn('',1000,80)); //hello from TCP
                //for i:= 0 to memoInput.Lines.Count-1 do begin
                for i:= 01001 to 01013 do begin
                    WriteLn('ZipCode ' + '0'+intToStr(i));
                end;
            end;
        end;
    finally
        writeln('infile sysdata2: '+boot_conf+':'+intToStr(ip_port));
        Free;
    end;
end;

```

This process is handled directly, through an object so each time it changes ReadLn of the connection also and not on demand.

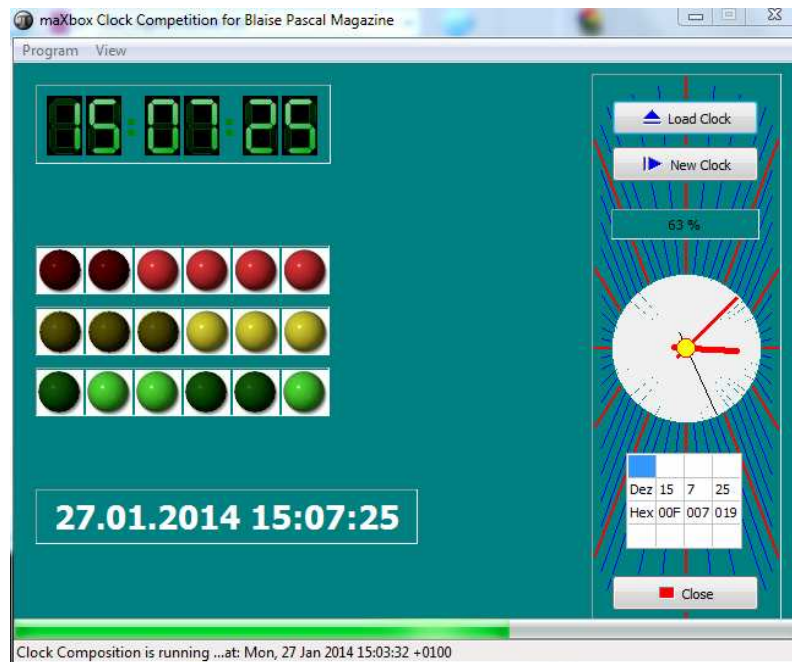


In other words within the operating system and the application that created a socket, a socket is referred to by a unique integer number called socket identifier or socket number.

The operating system forwards the payload of incoming IP packets to the corresponding application by extracting the socket address information from the IP and transport protocol headers and stripping the headers from the application data.

Related glossary terms:

- Web Services Description Language (WSDL),
- Java Message Service (JMS), appliance computing,
- JNDI (Java Naming and Directory Interface), table, redirection,
- WS-AtomicTransaction (WS-AT), Portal Markup Language (PML),
- URI (Uniform Resource Identifier), Internet map



3: XML Time in Action

Telnet is a client-server protocol, based on a reliable connection-oriented transport. Typically this protocol is used to establish a connection to Transmission Control Protocol (TCP) port number 23, where a Telnet server application (`telnetd`) is listening. Telnet, however, predates TCP/IP and was originally run over Network Control Program (NCP) protocols.

~~maXbox~~

Links of maXbox and XML Programming:

<http://www.softwareschule.ch/maxbox.htm>

<http://sourceforge.net/projects/maxbox>

<http://sourceforge.net/apps/mediawiki/maxbox/>

<http://www.w3schools.com/xml/default.asp>

http://en.wikipedia.org/wiki/Simple_API_for_XML

1.3 Appendix

EXAMPLE: List Objects

Working with Lists

The VCL/RTL includes many classes that represents lists or collections of items. They vary depending on the types

of items they contain, what operations they support, and whether they are persistent.

The following table lists various list classes, and indicates the types of items they contain:

Object Maintains

TList A list of pointers

TThreadList A thread-safe list of pointers

TBucketList A hashed list of pointers

TObjectBucketList A hashed list of object instances

TObjectList A memory-managed list of object instances

TComponentList A memory-managed list of components (that is, instances of classes descended from *TComponent*)

TClassList A list of class references

TInterfaceList A list of interface pointers.

TQueue A first-in first-out list of pointers

TStack A last-in first-out list of pointers or objects

TObjectQueue A first-in first-out list of objects

TObjectStack A last-in first-out list of objects

TCollection Base class for many specialized classes of typed items.

TStringList A list of strings

THashedStringList A list of strings with the form Name=Value, hashed for performance.

The ProgID for each of the Shell objects is shown in the following table.

Object	Function
TList	A list of pointers
TThreadList	A thread-safe list of pointers
TBucketList	A hashed list of pointers
TObjectBucketList	A hashed list of object instances
TObjectList	A memory-managed list of object instances
TComponentList	A memory-managed list of components
TClassList	A list of class references
TInterfaceList	A list of interface pointers
TQueue	A first-in first-out list of pointers
TStack	A last-in first-out list of pointers/objects
TObjectQueue	A first-in first-out list of objects
TObjectStack	A last-in first-out list of objects
TCollection	Base class for many specialized classes of typed items
TStringList	A list of strings
THashedStringList	A list of strings with the form Name=Value, hashed for performance.