

# Internet Technologies in XBase++ 2.0

Frankfurter Xbase++ Tracks 2015

Roger Donnay



# Internet Technologies in XBase++ 2.0

Frankfurter Xbase++ Tracks 2015

Roger Donnay

## Contents

Internet Technologies in Xbase++ 2.0 .....	3
Web Server and Web Client Relationship .....	4
What are Compiled Pages? .....	5
Why use <CXP/> for Web Development? .....	6
Understanding the <CXP/> Runtime Environment.....	10
Installing <CXP/> with Internet Information Server (IIS) 7 or higher .....	11
Vista and later .....	11
Windows 2008 and later .....	11
Verify IIS/Web Server installation .....	12
Installing <CXP/> with the Apache Web Server.....	12
Known anomalies with Apache .....	14
Using your own libraries with <CXP/>.....	14
The basics of <CXP/> programming .....	17
The Language Reference.....	17
#code directive .....	19
#page directive .....	20
#region directive .....	20
<CXP/> Error Handling .....	22
<CXP/> Class Reference .....	22
CxpAbstractPage() .....	22
Configuration .....	23
Properties .....	23
Layout Management.....	23
Deferred Methods .....	23
Error-handling and tracing-support .....	24
CxpApplication().....	24
HttpManager() .....	24
Configuration .....	24
HttpRequest() .....	24
Properties .....	24

# Internet Technologies in XBase++ 2.0

Frankfurter Xbase++ Tracks 2015

Roger Donnay

Manipulation .....	24
HttpResponse() .....	25
Properties .....	25
HTTP Header Management.....	25
Cookie Management.....	25
Manipulation .....	25
HttpUploadedFile() .....	26
Properties .....	26
How to create your own Web Server with HttpEndPoint() .....	26
Class methods .....	27
Connection.....	27
Implementing a web endpoint.....	28
WebHandler() .....	29
Incoming .....	29
Properties .....	29
Implement a web endpoint .....	30
How to create a CXP Web Server with HttpEndPoint() .....	31
Sending email to a secure SMTP server .....	32
Using WebSocketClient() and WebSocketHandler() .....	35
WebSocketClient() .....	37
A Picture Server .....	38

# Internet Technologies in XBase++ 2.0

Frankfurter Xbase++ Tracks 2015

Roger Donnay

Xbase++ 2.0 has greatly expanded its internet capabilities and has introduced some exciting new technologies as well as improving on older technologies. This session will introduce users to techniques which will make it easier than ever to open your existing application data and reports to users via the web. Compiled Xbase++ Pages (<CXP/>) is a build and execution infrastructure that allows programmers to create powerful dynamic web sites.

Xbase++ 2.0 utilizes a new Socket layer, WebHandler() and HttpEndPoint() classes to provide the functionality for the creation of web servers and HTTP connections.

This session introduces the Xbase++ programmer to the following concepts:

- An introduction to the basics of the web server and web client (browser) relationship.
- What are compiled pages (<CXP/>) ?
- How to set up a server to use compiled pages. Installing and configuring APACHE or IIS for <CXP/>.
- Setting up an Xbase++ environment for use with <CXP/>.
- Sample <CXP/> programs that deliver dynamic HTML and Javascript.
- How to use your own function libraries with <CXP/>.
- The basics of <CXP/> programming.
- How <CXP/> compares to scripting languages such as PHP or ASP.
- <CXP/> error handling
- Creating a web server using HttpEndPoint() and WebHandler().
- Sending email to a secure SMTP server.
- How to create your own <CXP/> web server using HttpEndPoint().
- How to create a web client/server application using WebSocketClient(), WebSocketHandler() and HttpEndPoint().

## Web Server and Web Client Relationship

# Internet Technologies in XBase++ 2.0

Frankfurter Xbase++ Tracks 2015

Roger Donnay

The primary function of a web server is to store, process and deliver web pages to clients (the web browser). The communication between client and server takes place using the Hypertext Transfer Protocol (HTTP). Pages delivered are most frequently HTML documents, which may include images, style sheets and scripts in addition to text content.

A user agent, commonly a web browser, initiates communication by making a request for a specific resource using HTTP and the server responds with the content of that resource or an error message if unable to do so. The resource is typically a real file on the server, but this is not necessarily the case and depends on how the web server is implemented.

While the primary function is to serve content, a full implementation of HTTP also includes ways of receiving content from clients. This feature is used for submitting web forms, including uploading of files.

Many generic web servers also support server-side scripting using Active Server Pages (ASP), PHP, <CXP/> or other scripting languages. This means that the behavior of the web server can be scripted in separate files, while the actual server software remains unchanged. Usually, this function is used to generate HTML documents dynamically ("on-the-fly") as opposed to returning static documents. The former is primarily used for retrieving and/or modifying information from databases. The latter is typically much faster and more easily cached but cannot deliver dynamic content.

Here is a simple example (Figure 1). TEST.HTM contains the following content:

```
<!-- TEST.HTM -->

<body>
<H2>Hello from my Test Server</H2>
<IMG SRC="./images/donnay-logo.jpg" width=200>
</body>
```

Figure 1

The web browser requests TEST.HTM by connecting to the server via its IP address and port.

`http://localhost:8080/test.htm`

The contents of the file TEST.HTM are then sent to the browser. The browser must then interpret the HTML and render it. As the browser traverses the HTML content, it will encounter resources that exist on the server, such as the IMG tag that references **/images/donnay-logo.jpg**. The browser will request that file using the same connection as the original connection and the server will transfer that file. This will continue until all resources such as JPG files, GIF files, etc. are retrieved by the browser and rendered within the displayed page.

These kinds of files that are being transferred are referred to as "static documents".

## What are Compiled Pages?

# Internet Technologies in Xbase++ 2.0

Compiled pages, hereinafter referred to as <CXP/>, contain combinations of Xbase++ code and HTML that return content to the web browser. The content must always be a string of data that is understood by the web browser.

Web browsers understand HTML and Javascript, therefore the code that is executed when a <CXP/> page is loaded must always return such content.

<CXP/> utilizes a technique known as FastCGI to accomplish this task very quickly. The two most common web servers that support FastCGI are IIS (Internet Information Services) and Apache. <CXP/> works very well with both of these web servers.

A .CXP file would be requested instead of a .HTM file when the application needs "dynamic" content, such as information from a database. The .CXP page can also read information that is sent back to the server from a browser, such as a form of data or other data that is passed as a "query string".

.CXP pages are compiled similarly to the way .PRG code is compiled, except the <CXP/> language is much more flexible in its ability to embed the output of executed code within HTML and vice-versa.

Here is a simple example (Figure 2). TEST.CXP contains the following content:

```
<!-- TEST.CXP -->

<body>
<H2>Hello from my Test Server</H2>
The time on the server is @(Time()) and the date is @(Date()) <br>
<IMG SRC="./images/donnay-logo.jpg" width=200>
</body>
```

Figure 2

The web browser requests TEST.CXP by connecting to the server via its IP address and port.

`http://localhost:8080/test.cxp`

The contents of the file TEST.CXP are NOT sent to the browser. Instead, the web server calls a program named **cxp-worker.exe** and passes the name of the <CXP/> file to load. This relationship is previously set up when configuring IIS or Apache for <CXP/> during Xbase++ installation. Cxp-worker.exe will compile TEST.CXP and create TEST.CXP.DLL in a sub-folder named **cxp-application**. The code in the DLL will then be executed and will be responsible for returning dynamic HTML to the web browser.

## Why use <CXP/> for Web Development?

# Internet Technologies in XBase++ 2.0

Frankfurter Xbase++ Tracks 2015

Roger Donnay

Most Xbase++ developers have applications which have been deployed for many years and are often asked by customers if they can access data or reports from a web browser at home. Some developers are already doing this (and much more) by using a third-party library named Xb2.Net. Xb2.Net allows the Xbase++ developer to create his/her own web server which resides on the same server as the customer data and runtime binaries. This concept has its drawbacks and limitations.

Every time the program needs to be updated, the executable must be terminated on the server, the binaries need to be rebuilt, transferred to the server and then restarted on the server. This usually means testing everything on a local machine and then deploying later to the server. This is not unusual or undesirable if it is a large application that requires more rigorous testing. It can be a grueling and laborious process however if only a small modification is needed or if a new application is needed. Imagine going through this process while making dozens of changes and then testing each change via the web browser.

When using <CXP/> pages for the application code, only the .CXP file(s) need to be updated and then deployed to the server. Programmers who have been developing web applications that use static .HTM files are keenly aware of such advantages. They know that that don't need to stop the web server, recompile it and restart it just to deploy a quick change. They know that they only need to send the .HTM file to the server and then reload the page on the browser. This is also true for <CXP/> pages. Furthermore, by using a product like WebDrive to map a server drive to a local computer, the developer can edit the HTM or CXP files without even requiring a FTP or RDP connection to the server.

When a <CXP/> page is updated, cxp-worker.exe (on the server) compares the date/time of the CXP file against the last DLL that was created from that CXP source. If the file was updated, the DLL is unloaded, recompiled and reloaded before calling the page.

My interest in <CXP/> was due to the fact that I maintain 4 different web sites n my Apache server. Each is in its own sub folder under C:\wamp\www. I found that it took too much time to keep these websites updated with static information. I needed it to be more dynamic. For example, the USS James E. Kyes website (<http://ussjek.org>) supports association members who were aboard this U.S. Navy destroyer from 1946-1973. The website tries to list all shipmates who were aboard during those years. This list needs to be constantly maintained and updated on the website. The roster is in a Microsoft Excel file and it is then saved as a set of .HTM files which are loaded by the website. I wanted to give shipmates the ability to see the list in 5 different orders, ie State, Years On board, Rank/Rate, etc. This required saving 5 different sets of HTM files (one for each sort order). I found myself constantly being reminded that the roster on the website is not up to date. <CXP/> made it possible to make this completely dynamic and update the list directly from a database which resides on the server. Now, I no longer use Excel and publish the updated HTML, instead I just make changes to the database.

Here is the <CXP/> program that I created (Figure 3b) in only about 1/2 hour from a template (**databrowse\_template.cxp**). It was created for the purpose of browsing databases.

```
<!-- Roster.cxp -->

<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"/>
    <link rel="stylesheet" type="text/css" href="donnay.css" />
  </head>

  <body>

  <%

  IF PCount() = 0 .OR. Empty(::session:rows)
```

# Internet Technologies in XBase++ 2.0

Frankfurter Xbase++ Tracks 2015

Roger Donnay

```
cAction := oForm:navigate
cSetOrder := oForm:setorder
cSeekValue := oForm:seekValue

cSeekMessage := ''

BEGIN SEQUENCE
IF Empty(cOrder) .AND. Empty(cSetOrder)
  (cAlias)->(OrdSetFocus(0))
ELSEIF !Empty(cSetOrder)
  IF cSetOrder = 'Record'
    cSetorder := 0
  ENDIF
```



# Internet Technologies in XBase++ 2.0

Frankfurter Xbase++ Tracks 2015

Roger Donnay

```
<tr>
  @FOR i := 1 TO Len(aCols)
    <th @(aTHOption[i])>
      @IF !(aCols[i,3])=nil
        <input name='setorder' type='submit' value=@(aCols[i,3]) style="font-size:20px">
      @ELSE
        @(aCols[i,1])
      @ENDIF
    </th>
  @NEXT
</tr>
```

# Internet Technologies in XBase++ 2.0

Frankfurter Xbase++ Tracks 2015

Roger Donnay

If you already have a web solution that uses Xb2.Net or IIS or Apache, you can continue with that solution and yet still add functionality with <CXP/> at any time. <CXP/> pages run in their own environment and therefore do not require rebuilding any existing environment.

## Understanding the <CXP/> Runtime Environment

<CXP/> is designed to encapsulate or localize an entire runtime environment so as to insure that applications always run smoothly in the event that the greater environment has changed. <CXP/>

# Internet Technologies in XBase++ 2.0

Frankfurter Xbase++ Tracks 2015

Roger Donnay

applications are installed in a sub-directory under the web server's ROOT folder. In the case of IIS, this is **c:\inetpub\wwwroot**. In the case of Wamp Apache, this is **c:\wamp\www**. Every other file needed by the application will usually reside in a sub folder or a sub-sub folder.

The Xbase++ runtime will always reside in the same place, usually in the directory **c:\Program Files (x86)\Alaska Software\cxp20**.

It is not always practical, however, to encapsulate an entire environment in a sub folder. For example, the application may require access to databases that are on a different drive or even a different server. What should be localized is the source code (.CXP, .HTM, .PRG, .CH, .JPG) files, and binaries (.LIB, .DLL) files that are compiled and called by the application.

## Installing <CXP/> with Internet Information Server (IIS) 7 or higher

Before you begin you must establish whether you are using a desktop or a server operating system. Vista, Windows 7 and later versions belong to the desktop operating systems group. Windows 2008 and later are server operating systems. The installation procedure differs slightly between these groups and is outlined in individual sections below. Further information on IIS configuration and administration can be found on <http://www.iis.net>. Once the IIS web server is properly installed, the installation package will properly install the <CXP/> technology and will also apply proper configuration on the IIS web server for <CXP/> usage.

### *Vista and later*

Open the Control Panel and select the item Programs. Under Programs and Features select Turn Windows features on or off. Then accomplish following steps. Please follow strictly the order below:

Ensure that the item Internet Information Service is selected. If this is not the case select the item Internet Information Service.

Also make sure that support for CGI executables is enabled which also enables the FastCGIModule. The CGI feature can be found at the following location in the feature tree:

Internet Information Services -> World Wide Web Services -> Application Development Features -> CGI

### *Windows 2008 and later*

Open the Server Manager and scroll down to Roles Summary. Click Add Roles. Select the role Web Server (IIS) and click Next. Ensure that the role service CGI is selected which also enables the FastCGIModule. The role service CGI can be found on the following location:

Web Server -> Application Development -> CGI

# Internet Technologies in Xbase++ 2.0

Frankfurter Xbase++ Tracks 2015

Roger Donnay

## *Verify IIS/Web Server installation*

To verify the IIS installation, open a web browser on your computer and navigate to **http://localhost**. The welcome page will be displayed if the web server was installed successfully.

To verify the <CXP/> installation, open a web browser and navigate to **http://localhost/cxpinfo.cxp**.

**HTTP Error 500.21 - Internal Server Error  
Handler "CxpWorker" has a bad module "FastCgiModule" in its module list**

If you get the above error when trying to load a .CXP page using IIS server, it is because you have not enabled the CGI role.

The below URL gives detailed instructions on how to do this.

**http://www.iis.net/configreference/system.webserver/fastcgi**

## *Installing <CXP/> with the Apache Web Server*

These instructions are specifically for Wamp Apache (Apache version 2.2.22). This is the version of Apache that installs with Wamp Server. Wamp Server includes Apache, PHP and MySQL because this is one of the most popular installations for a variety of applications, including PhpBB3, a popular web forum application. However, with minor modifications, these instructions apply to any installation of Apache. If you are using a version of Apache that is later than 2.2.22 then replace that version in the below instructions with the version you have installed.

If you follow these installation instructions step-by-step, you should get up and running very quickly.

- 1) Install Xbase++ 2.0 on a workstation or your server. It is not necessary to have the complete Xbase++ installation on your server. All that is needed on the server is the CXP20 folders.
- 2) After the Xbase++ 2.0 installation is complete, locate the following folder:  
**C:\Program Files (x86)\Alaska Software\cxp20**
- 3) If you already installed Xbase++ and <CXP/> on the same server or workstation as your Apache installation, then the environment will be ready for Apache. If you want to deploy only the <CXP/> runtime on your Apache server then do the following
  - a) Create a new folder on your server named **C:\wamp\cxp20**
  - b) Copy the entire **C:\Program Files (x86)\Alaska Software\cxp20** folder, including sub-directories to the new **C:\wamp\cxp20** folder on your server.
- 4) Locate the following folder on your workstation or server: **C:\inetpub\wwwroot**. If you installed <CXP/> when you installed Xbase++ 2.0, then the installation program will have copied some <CXP/> files to this folder. The file `cxpinfo.cxp` will be installed to `c:\inetpub\wwwroot` only if IIS is installed.

# Internet Technologies in XBase++ 2.0

Frankfurter Xbase++ Tracks 2015

Roger Donnay

- 5) Locate the following folder on your server: C:\wamp\www. This is the root folder for Apache. It is defined in the httpd.conf file of Apache as the root folder with the DocumentRoot entry. If you not using WAMP, then it will be defined as a different location.
- 6) a) Copy all \*.CXP files in the **C:\inetpub\wwwroot** folder to the C:\wamp\www folder on your server. If there is a **c:\inetpub\wwwroot\cxp-application** folder it does NOT need to be copied to your server. This folder is automatically maintained by <CXP/>. That folder does not even exist if the cxpinfo.cxp page was not executed previously.  
  
b) Copy the **c:\inetpub\wwwroot\websamples** folder to the **c:\wamp\www** folder on your server. The websamples provide a source of information illustrating usage and filling gaps where the documentation is weak.
- 7) Download the following file from the Donnay Software website:  
[http://bb.donnay-software.com/download/mod\\_fcgid-2.3.6-win32-x86.zip](http://bb.donnay-software.com/download/mod_fcgid-2.3.6-win32-x86.zip) (32 bit)  
or  
[http://bb.donnay-software.com/download/mod\\_fcgid-2.3.6-win64.zip](http://bb.donnay-software.com/download/mod_fcgid-2.3.6-win64.zip) (64 bit)  
then extract the mod\_fcgid.so file into the **C:\wamp\bin\apache\apache2.2.22\modules** folder on the server. This is the module that handles FastCGI.
- 8) Run the C:\cpx20\bin\apachecfg.exe with the following parameters:  
**-i -c=C:\wamp\bin\apache\apache2.2.22 -n=c:\wamp\cpx20\bin\cxp-worker.exe**. This will create a backup of your **c:\wamp\bin\apache\apache2.2.22\conf\httpd.conf** file and will add new entries that will enable the <CXP/> system and FastCGI.  
  
Hint: apachecfg.exe -h prints the usage of the program.
- 9) Stop and restart your wampApache service. Note: this may not be necessary if Apache is already running because apachecfg.exe normally restarts the service. It will only be necessary if you had to make additional modifications to your httpd.conf file (which is not recommended).
- 10) Open a web browser on your server and navigate to **http://localhost/cxpinfo.cxp**. This should show your first <CXP/> page. After you confirm that everything is running ok, then navigate to **http://localhost/websamples/index.cxp** to run the sample selection.

NOTE: If you did not get Apache from the WAMP installation, then the Apache system will have a different installation folder on your server and may also be a different version. The above instructions are for WAMP Apache 2.22.22. Just modify the instructions to accommodate your directory structure.

After configuration, your Apache HTTPD.CONF file will have the following additions (Figure 4):

```
#CXP Start Installer automatically modified
#Do not manually modify here! This will be removed on deinstallation.
LoadModule fcgid_module modules/mod_fcgid.so
#CXP End Installer automatically modified

<Directory "c:/wamp/www/">
  #CXP Start Installer automatically modified
  #Do not manually modify here! This will be removed on deinstallation.
  <IfModule fcgid_module>
    <Files *.cpx>
```

## Known anomalies with Apache

When calling the CXP page from the web browser, Apache is case sensitive.

This will NOT work: **`http://localhost/dd787.Cxp`**

This WILL work: **`http://localhost/dd787.cxp`**

The CXP page cannot be given the same name as the root directory.

This will cause an Internal Error: **`http://localhost/dd787/dd787.cxp`**

This will work: **`http://localhost/dd787/rsvp787.cxp`**

A CXP layout file cannot be given the same name as a CXP file.

This will cause an Internal Error

## Using your own libraries with <CXP/>

<CXP/>, by default, is only aware of the Xbase++ runtime libraries (DLLs), the compile-time libraries (LIBs) and the compile-time includes (CHs). If your

# Internet Technologies in XBase++ 2.0

Frankfurter Xbase++ Tracks 2015

Roger Donnay

web application needs access to your own DLL, LIB and CH files, then you must make them available to cxp-worker.

Here are instructions for setting up <CXP/> to use your own libraries that have been compiled in Xbase++.

Under the WWW root directory of your <CXP/> application, you will need to create a new folder named Helpers. For example, below is the <CXP/> source code for an application that lists customers from a database. The application is in a file named CUSTOMER.CXP under **C:\wamp\www\ds\customers** (This example is for Apache). If it were using IIS then it would be in **c:\inetpub\wwwroot\ds\customers**.

In the below example, I created a folder named C:\wamp\www\ds\customers\Helpers and then copied DCLIPX.LIB and DCLIPX.DLL into the Helpers folder.

I then created a file named **C:\wamp\www\ds\customers\application.config**. It has the following content:

A sample application.config file (Figure 5):

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<info author    = "Roger Donnay"
      version   = "1.0"
      copyright= "Copyright (C) $YEAR$ Donnay Software Designs. No rights deserved"
      title     = "Customers" header    = "Program for listing
      Sample Customer Database" />
<!-- List the libraries to be linked with the CXP page -->
<helpers
  lib="dclipx.lib"
/>
```

Figure 5

Unfortunately, CH files are a different matter. They must reside in the **C:\Alaska Software\Xbase++\Cxp20\Include** folder. This will be changed in a future <CXP/> update in which CH files can also be added to the Helpers.

A sample CUSTOMER.CXP file that uses your library (Figure 6).

```
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"/>
</head>

<body>
<h1>My Customer List</h1>
<h3>Rendering as simple text</h3>
<pre>
```

# Internet Technologies in XBase++ 2.0

Frankfurter Xbase++ Tracks 2015

Roger Donnay

## Caveat:

It is possible to create an Xbase++ library that is linked to Xbase++ runtime DLLs that are not included in the **C:\Program Files (x86)\Alaska Software\cxp20\bin** folder. This would cause a failure of a .CXP program to load if that library is used in your .CXP program. For example, I found that one of my .CXP programs would not load when using a .DLL that required the Xbase++ DLLs **ASUED10.dll** and **ASUED10C.dll**. These DLLs exist in the **C:\Program Files (x86)\Alaska Software\cpp20\runtime** folder and must be copied to the **C:\Program Files (x86)\Alaska Software\cxp20\bin** folder. A load error will not be obvious because there will be no CXP error other than something similar to the following displayed in the web browser: `\inetpub\wwwroot\medalion\cxp-application\drivers.cxp`.

If you experience such a problem, then there is a simple method for determining the cause. Use the **CHK4DLL.EXE** utility to test your DLLs to make sure that they will load when your path is set to the CXP bin folder.



## The basics of <CXP/> programming

### The Language Reference

#### <%...%> code section

A <CXP/> file normally contains HTML tags, just like an HTML file. However, a <CXP/> file can also contain code sections surrounded by the delimiters <% and %>. These code sections are executed on the server and can be made up of any expression, statement, procedure, or class valid in the Xbase++ language.

Inside a code section, the ? command is used to write to the current output section. The following example (Figure 7) writes the string "Hello World" into the body element of the document.

```
<html>
<body>
<%
  ? "Hello World!"
%>
</body>
</html>
```

Figure 7

The command ? internally uses the method ::HttpResponse:WriteStr(), which performs HTML-encoding and appends a CRLF-pair to the string. Data without a line break can be output using the command ??, which uses the ::HttpResponse:Write() method instead. To output the data as-is (Figure 8), without any encoding or line breaks, the method ::HttpResponse:WriteRaw() must be called directly.

```
<html>
<head>
<!-- Output the version data inside the title -->
<!-- without a crlf pair -->
<title>Greetings from <??Version()%></title>
<body>
<%

  // This iteration repeats the output, adding 16 lines
  // of text to the resulting page
  FOR n:=1 TO 16
    ? "<p style='background-color:#f5'+StrZero(n*6,2)+'20'>Hello world v2!</p>"
  NEXT n
%>
</body>
</html>
```

Figure 8

# Internet Technologies in Xbase++ 2.0

Frankfurter Xbase++ Tracks 2015

Roger Donnay

## @(...) inline expression

@(<expression>)

The @(<expression>) syntax allows to embed expressions directly inside your declarative HTML markup in a simple and distinct way. The following sample code shows how the expression is transformed by the CxcBuilder to output the expression result in the current section.

```
<h2>Today is @(Date())!</h2>
```

Code:

```
::HttpResponse:writeStr("<h2>Today is ",Date(),"!</h2>")
```

Note: Note that the output is automatically HTML-encoded! Therefore, inline expressions can not be used to write raw data to the output section.

## @... single-line code

@<expression>|<command>|<statement>

When using the @ command, the entire line is declared as code. The line continuation character (;) can be used to break a single statement into multiple lines. The @ command allows writing code without the open/close markers normally required to identify code sections. In sections containing alternating lines of Xbase++ code and HTML markup, using the @ command increases readability and avoids unnecessary noise. This is demonstrated in the code fragment below. Please note that it is not allowed to have multiple @ commands in the same line (Figure 9).

```
<%  
x := 125  
aDir := Directory()  
%>  
  
@IF x>10  
  <h1>x is larger than 10</h1>  
@ELSE  
  <h1>y is smaller or equal to 10</h1>  
@ENDIF  
  
<h2>Directory Listing</h2>  
<ul>  
@FOR n:=1 TO Len(aDir)  
  <li><pre>@(aDir[n,1]), @(aDir[n,2]), @(aDir[n,3]), @(aDir[n,4])</pre></li>  
@NEXT n  
</ul>
```

Figure 9

## #code directive

Controls where code is emitted by the CxcBuilder

```
< %#code locality=<locality-name>%>
```

The #code directive is used to specify the locality of the code section following the directive. The supported locality tokens are described in the table below. The default locality is "page-render". The CxcBuilder automatically switches back to that locality after it has emitted the code section immediately following a #code directive.

### Locality Tokens

Token	Description
-------	-------------

\*) Default locality

page-init	Code section is injected at the end of the :Init() method of the page class
page-load	Code section is injected into the :Load() method of the page class. :Load() is executed when the page binary is loaded into the worker process.
page-unload	Code section is injected into the :Unload() method of the page class. :Unload() is executed when page binary is unloaded from worker process. This usually occurs before worker process gets recycled. The exact time is not determined.
page-render*)	Code sections is injected into the :Render() method of the page class. :Render() is where the HTML output (view content) is created.
page-global	Code section is injected into module header of the page class implementation file. Use this locality for defining your own functions, classes or static declarations. #include statements should also have be placed in this locality.

**Note:** page-load/page-unload is a good place for data base connection such as DacSession() connect / disconnect or table USE operations, as this will increase performance if worker process is used to handle multiple requests (Figure 10).

```
< %#code locality="page-global"%>
<%
// This section is injected into the module header
STATIC aData := { 3, 5, 7, 11 }

FUNCTION MyHelper(cInput)
RETURN(cOutput)
%>
<!-- this comment and the following code goes into -->
<!-- the page-render method. -->
<h1>
<%
? Version()
%>
</h1>
```

Figure 10

## #page directive

Controls page behavior ()Figure 11

```
<%#page layout=<filename>
      trace="yes|no"
      implements=<classname>
      inherits=<baseclass> %>
```

Figure 11

Using the #page directive the behaviour of a page can be customized. The table below shows the available page token/value pairs.

page tokens

Token	Description
layout	Defines the layout master file for this page
trace	Enables/disables tracing output created via the :Trace() method for the debugger or the DebugView utility. The value must be specified as a string literal.
implements	Allows to change the name of the page class. By default, the class name is derived from the filename of the cpx page.
inherits	Allows to override the base class of the page class. By default, CxpAbstractPage is used.

## #region directive

Controls display of code in the source code editor

```
<##region <your text>%>
<##end%
```

The #region directive allows to specify a block of code that can be expanded or collapsed in the source code editor. By default, content embedded into a #region/#end directive is displayed collapsed and is not visible in the editor. Instead, only the text defined in <your text> is shown. This feature called "code folding" can contribute significantly to source code readability when using the Workbench.

## Using Layout files, the RENDER command and the SECTION command

The RENDER command is used in layout files to specify the location in which a specific page section is rendered. If a layout is referenced in a page, page composition is defined in the layout file. In this case, the RENDER command must be used to selectively render the output defined in the content page. To do this, the name of the section must be specified.

# Internet Technologies in XBase++ 2.0

Frankfurter Xbase++ Tracks 2015

Roger Donnay

By default output of a content page is written to the BODY section. Therefore at least the RENDER BODY command has to be used in the layout file. Otherwise the output of the content page is lost.

Figure 12 shows a sample layout file ("hello1.layout")

```
<html>
<head>
  <title>MySite by @(Version())</title>
</head>
<body>

  <div style="font-size:18pt;background-color:#ff0000;color:#efefef;">
  MyApplication
  </div>

  <!-- output from content page is rendered here -->
  @RENDER Body
  <div style="font-size:20pt;background-color:#0000ff;color:#efefef;">
    @RENDER Footer
  </div>

  @RENDER Hello

</body>
</html>
```

Figure 12

Figure 13 demonstrates a CXP Page with layout reference ("hello.cxp")

```
<%#page layout="./hello1.layout" %>

@SECTION Body
<h2>This text and the text in the footer are defined in page hello.cxp</h2>
<h3>The heading, colors and the overall page structure are defined in layout
file simplesite.layout</h3>

@SECTION Footer
<p>Nothing special today</p>

@SECTION Hello
<H2>Hello from my Test Server</H2>
The time on the server is @(Time()) and the date is @(Date()) <br>
<IMG SRC="./images/donnay-logo.jpg" width=200>
<p>
```

Figure 13

## <CXP/> Error Handling

The <CXP/> infrastructure knows about different error types. Each error is demonstrated by the following CXP page (Figure 14):

```
// compile error
SD SD
FOR x:=1 TO 10
  ? "TaTa"
ENDIF

// link error
TaTa()

// runtime error
? test test
%>
```

Figure 14

When you run this code you will see the error information returned to the web browser with a detailed description of the error. Only the first error encountered will be returned, first compile errors, then link errors, then runtime errors.

## <CXP/> Class Reference

As of the writing of this document, much of the class reference documentation has not been completed however perusing the .CXP files in c:\inetpub\wwwroot\websamples will show how these classes are used.

### CxpAbstractPage()

This class implements the core features for any <CXP/> page created by the CxcBuilder.

All <CXP/> pages are by default derived from this class. Whenever a <CXP/> page needs to be rebuilt, the CxcBuilder first creates an intermediate PRG file for the page. This file contains the implementation of a class which by default is derived from CxpAbstractPage. You can override this default behaviour by using the inherits directive.

Unless specified otherwise, the code and HTML markup contained in a CXP file is emitted into the :Render() method of the page class. This means that for a simple <CXP/> page, the framework just executes :Render() to display the page at runtime. In this case, the HTML markup together with any output generated with the ??? commands will be used to create the response send back to the browser after a page request.

# Internet Technologies in XBase++ 2.0

Frankfurter Xbase++ Tracks 2015

Roger Donnay

For more complex scenarios, <CXP/> applications can use the code-locality directive to control the location where code is emitted in the intermediate PRG file created by the CxcBuilder. This way, application code can be made to execute not only during :Render(), but also when a page is loaded, for example. In addition, by using the HttpResponse Object and its dedicated output methods, the :Render() method of a <CXP/> page can return other content than just plain HTML.

## Configuration

<a href="#">:CachePolicy</a>	Determines the cache policy of the page.
<a href="#">:ConfigFile</a>	Fully-qualified filename of the config file of the page.
<a href="#">:Config</a>	Represents the root node of the configuration data for the page.

## Properties

<a href="#">:Application</a>	References the CxpApplication object the page belongs to.
<a href="#">:Session</a>	References the session object handling session state for the page.
<a href="#">:Host</a>	References an object for accessing functionality of the host machine or WWW server the <CXP/> application is running on.
<a href="#">:Data</a>	References a container object which handles the data associated with the page.
<a href="#">:Params</a>	Contains a container object for the parameters passed to the page.
<a href="#">:File</a>	Contains the fully-qualified URL of the <CXP/> file, such as localhost/samples/index.cxp.
<a href="#">:Path</a>	Contains the path to the page, such as localhost/samples/.
<a href="#">:PhysicalPath</a>	Contains the path of the physical file location of the page, eg. c:\inetpub\samples\.
<a href="#">:HttpRequest</a>	References a HttpRequest object representing the incoming request to the page.
<a href="#">:HttpResponse</a>	References a HttpResponse object representing the response created by the page.
<a href="#">:Trace</a>	Enables or disables tracing output.

## Layout Management

<a href="#">:HasLayout()</a>	Determine if the page has a layout associated.
<a href="#">:HasSection()</a>	Determine if a specific Section has been defined
<a href="#">:RenderSection()</a>	Renders the content of the given section by name
<a href="#">:SelectSection()</a>	Selects a section by name.

## Deferred Methods

<a href="#">:Render()</a>	Renders the page content which is returned to the client browser.
<a href="#">:Load()</a>	Gets executed when the page is loaded into memory.
<a href="#">:Unload()</a>	Gets executed when the page is unloaded from memory. Page unloading occurs when a <CXP/> worker process is recycled or when the www server shuts down.

## Error-handling and tracing-support

<a href="#">:Trace()</a>	Output trace information to the Windows Debug interface.
<a href="#">:ErrorHandler()</a>	The default handler for runtime errors occurring during page execution.

## CxpApplication()

The function CxpApplication() returns the class object of the CxpApplication class.

The documentation is not completed.

## HttpManager()

The class *HttpManager* controls the behaviour of the application on the HTTP protocol-level.

### Configuration

<a href="#">:UseUploadFolder</a>	Specifies whether uploaded files are stored in a local drive
<a href="#">:UploadFolder</a>	The folder where uploaded files are stored in

## HttpRequest()

Objects of this class encapsulate the incoming request from the client.

### Properties

<a href="#">:Form</a>	Provides access to the form variables of the request.
<a href="#">:Cookies</a>	Provides access to the cookies of the request.
<a href="#">:Files</a>	Provides access to the files uploaded from the browser.
<a href="#">:Headers</a>	Provides access to the headers of the request.
<a href="#">:DocumentRoot</a>	Contains the physical path to the document root of the site.
<a href="#">:QueryString</a>	Contains the query string as posted by the browser.
<a href="#">:AcceptLanguages</a>	Provides access to the accept languages order as posted by the browser.
<a href="#">:ContentEncoding</a>	Contains the content encoding of the request.

### Manipulation

<a href="#">:GetVariable()</a>	Retrieve the value of a CGI/FastCGI variable.
--------------------------------	---



## HttpResponse()

Objects of this class encapsulate the outgoing HTTP response.

### Properties

<a href="#">:ClientCachePolicy</a>	Controls the browser's client cache policy for the response.
<a href="#">:ContentType</a>	Contains the content type requested by the client.
<a href="#">:ServerCachePolicy</a>	Determines the cache policy of this request.
<a href="#">:Response</a>	The response as send to the server.

### HTTP Header Management

<a href="#">:AddHeader()</a>	Add a header token and value to the response.
<a href="#">:DelHeader()</a>	Remove a header token from the response.
<a href="#">:DelHeaders()</a>	Remove all headers, or just the headers with a given token from the response.
<a href="#">:GetHeader()</a>	Get a header value given its token.

### Cookie Management

<a href="#">:AddCookie()</a>	Add a cookie.
<a href="#">:DelCookie()</a>	Delete a cookie given its name.
<a href="#">:DelCookies()</a>	Delete all cookies.

### Manipulation

<a href="#">:WriteRaw()</a>	Write an arbitrary value to the response.
<a href="#">:Write()</a>	Write an arbitrary value to the response.
<a href="#">:WriteStr()</a>	Write one or more values to the response with CRLF appended.
<a href="#">:WriteBinary()</a>	Write binary data directly to the response.
<a href="#">:WriteFile()</a>	Sends back a file to the client.
<a href="#">:Transfer()</a>	Transfers the execution path to another page or controller.
<a href="#">:Redirect()</a>	Redirects the browser to another URL.
<a href="#">:Reset()</a>	Discards any pending content.
<a href="#">:Send()</a>	Sends the response to the client.

# Internet Technologies in XBase++ 2.0

Frankfurter Xbase++ Tracks 2015

Roger Donnay

## HttpUploadedFile()

Instances of the class *HttpUploadedFile* represent files which are uploaded from within a <CXP/> page. The *HttpUploadedFile* objects are created automatically by the <CXP/> infrastructure and can be accessed via the [:Files](#) instance variable of the *HttpRequest* object.

### Properties

<a href="#">:Data</a>	Contains the file content.
<a href="#">:IsOnDisk</a>	Specifies whether the file is stored on disk.
<a href="#">:LastError</a>	Contains the error code of the last file operation.
<a href="#">:NameOnDisk</a>	Contains the full pathname of the file on the web server's hard drive.
<a href="#">:Name</a>	Contains the filename.
<a href="#">:Size</a>	Specifies the size of the file.

## How to create your own Web Server with HttpEndPoint()

An object of the class *HttpEndpoint* serves as a connection point for incoming HTTP or WebSocket connections. The connection point is active after calling the method [:start\(\)](#). [:start\(\)](#) automatically creates a new thread for listening on the port associated with the *HttpEndPoint*. The port number must be passed as a parameter to the method [:new\(\)](#).

For HTTP requests with an URL referring to a local file, the *HttpEndpoint* loads the file from the hard drive and returns it to the connected web client. The following table lists the extensions and the corresponding MIME types of the resources which are supported by default:

Default extensions and MIME types

Extension MIME type

.html	text/html
.htm	text/html
.gif	image/gif
.png	image/png
.jpg	image/jpeg jpeg jpg jpe
.js	application/x-javascript
.css	text/css
.woff	font/woff
.eot	application/vnd.ms-fontobject
.otf	application/octet-stream
.ttf	application/octet-stream
.svg	image/svg+xml
.json	application/json

# Internet Technologies in XBase++ 2.0

Frankfurter Xbase++ Tracks 2015

Roger Donnay

A web request to a resource without an extension must be handled in application code. For this, a user-defined class must be written containing the logic for handling the request. The class must have the same name as the resource which is requested, and it must be derived from a base class which corresponds to the connection type.

If the client tries to establish a websocket connection, an object of a user-defined class that must be derived from *WebSocketHandler* is created. The following URL instantiates an object from the class *MyWebSocketHandler*:

```
ws://<servername>:<port>/MyWebSocketHandler
```

In case no websocket connection is to be established, an object is instantiated from a class that must be derived from the class *WebHandler*. The following URL instantiates an object from the class *MyWebHandler*, and calls the method *:echo()*. The parameters specified in the URL are forwarded to the handler method:

```
http://<servername>:<port>/MyWebHandler/echo?paramname1=value1
```

The following code fragment (Figure 15) demonstrates the implementation of the handler that corresponds to the URL above:

```
CLASS MyWebHandler FROM WebHandler
  EXPORTED:
    METHOD echo
  ENDClass

<...>

METHOD MyWebHandler:echo( cParamName1 )
  LOCAL cHtml
  cHtml := ""
  cHtml += "<html><title>MyWebHandler:echo() was called</title><body>"
  cHtml += "Parameter cParamName1 is " + cParamName1
  cHtml += "</body></html>"
  RETURN cHtml
```

Figure 15

## Class methods

[:new\(\)](#) Creates an instance of the class *HttpEndpoint*.

## Connection

[:start\(\)](#) Start the listener thread

[:stop\(\)](#) Terminates the listener thread.

## Implementing a web endpoint

```
// The example creates a html file in the current directory.
// The HttpEndpoint loads the file and returns it to the connected client on request.
#include "inkey.ch"
#include "web.ch"

#define PORT 81
PROCEDURE Main()
    LOCAL cPort
    LOCAL oHttpEndpoint
    LOCAL nKey
    LOCAL cHtml

    //
    // Creates a sample html file on the local drive.
    //
    IF .NOT. File( "default.html" )
        cHtml := ""
        cHtml += "<h1>Hello World!</h1>"
        cHtml += "<h2>Greetings from: "+AppName()+"</h2>"
        MemoWrit( "default.html", cHtml )
    ENDIF

    cPort := AllTrim(Str(PORT))
    ? "HttpEndpoint sample."
    ?
    ? "With a web browser navigate to:"
    ?
    ? "    http://localhost:"+cPort+"/default.html"
    ?
    ? "Press ESC to quit"

    //
    // Create an end point for handling incoming
    // connections on the specified port
    //
    // Note: Requests to .html files are handled
    //       automatically by the Http end point!
    //       No code must be written for returning
    //       the sample .html file.
    //
    oHttpEndpoint := HttpEndpoint():new( PORT )
    oHttpEndpoint:start()

    // Continue after the ESC key was pressed
    nKey := 0
    DO WHILE nKey <> K_ESC
        nKey := Inkey(1)
    ENDDO

    //
    // Stop the endpoint before the application
    // terminates
    //
    oHttpEndpoint:stop()

RETURN
```

Figure 16

## WebHandler()

On web connections the class `HttpEndpoint` instantiates objects from user defined classes that must be derived from the class *WebHandler*. Those sub classes implement methods being called to serve the request.

### Incoming

[:execute\(\)](#) Handle the request and provide the content to be returned to the client.

### Properties

[:HttpRequest](#) References a `HttpRequest` object representing the incoming request to the web handler.

[:HttpResponse](#) References a `HttpResponse` object representing the response created by the web handler.

## Implement a web endpoint

```
// The example shows a server that implements the Endpoint
// CurrentTime/get. Requests to other endpoints on CurrentTime
// encounter a default handling

#include "inkey.ch"
#include "web.ch"

#define PORT 81
PROCEDURE Main()
    LOCAL cPort
    LOCAL oHttpEndpoint
    LOCAL nKey

    cPort := AllTrim(Str(PORT))
    ? "Time Server sample."
    ?
    ? "With a web browser navigate to:"
    ?
    ? "    http://localhost:"+cPort+"/CurrentTime/get"
    ? "    http://localhost:"+cPort+"/CurrentTime/unknown"
    ?
    ? "Press ESC to quit"

    //
    // The listener thread is established
    //
    oHttpEndpoint := HttpEndpoint():new( PORT )
    oHttpEndpoint:start()

    // This sample is not doing something special
    nKey := 0
    DO WHILE nKey <> K_ESC
        nKey := Inkey(1)
    ENDDO

    //
    // Stop the endpoint before the application
    // is terminated
    //
    oHttpEndpoint:stop()

RETURN

//
// Handler for the current time
//
CLASS CurrentTime FROM WebHandler
    EXPORTED:
        METHOD get
        METHOD execute
ENDCLASS
```

Figure 17

```
//
// The method :get() handles the RESTful path CurrentTime/get
//
METHOD CurrentTime:get()
  LOCAL cThreadId
  LOCAL cResponse := ""

  cThreadId := AllTrim(Str(ThreadId()))

  cResponse += "<!DOCTYPE html>"
  cResponse += "<html>"
  cResponse += "  <head><title>"+ProcName()+"</title></head>"
  cResponse += "  <body>"
  cResponse += "    <h1>Current Time:</h1>"
  cResponse += "    <p>"+Time()+" (HH:MM:SS)</p>"
  cResponse += "    <small>By thread: "+cThreadId+"</small>"
  cResponse += "  </body>"
  cResponse += "</html>"

RETURN cResponse

//
// Dispatch the request to existing methods. Return a generic
// web page for all other requests.
//
METHOD CurrentTime:execute( cMethodName )
  LOCAL cResponse := ""

  IF IsMethod(SELF, cMethodName)
    RETURN SUPER:execute( cMethodName )
  ENDIF

  cResponse += "<!DOCTYPE html>"
  cResponse += "<html>"
  cResponse += "  <head><title>Unhandled request</title></head>"
  cResponse += "  <body>"
  cResponse += "    <h1>No handler for: "+cMethodName+"</h1>"
  cResponse += "    <p>Please try with: /CurrentTime/get</p>"
  cResponse += "    <small>Sorry</small>"
  cResponse += "  </body>"
  cResponse += "<html>"

RETURN cResponse
```

Figure 17a

## How to create a CXP Web Server with HttpEndPoint()

The below code will be supported in the near future by Alaska Software.

It is understood that the full capabilities of <CXP/> will be supported without the need for IIS or Apache. This is an example of how easy it will be to add <CXP/> support to your custom web server (Figure 18).

```
/// <summary>
/// We override DefaultProcessor so we can add our
/// CXP and HTMLFile processors which we need for
/// web debugging.
/// </summary>
CLASS CxpHttpEndPoint FROM HttpEndPoint
    EXPORTED:
        METHOD DefaultProcessors()
        METHOD DefaultConfig()
ENDCLASS

METHOD CxpHttpEndPoint:DefaultProcessors()
    LOCAL aRet := SUPER:DefaultProcessors()
    AAdd( aRet , "CxpProcessor" )
    AAdd( aRet , "HtmlFileProcessor" )
RETURN(aRet)

METHOD CxpHttpEndPoint:DefaultConfig(cToken)
    LOCAL oDO := DataObject():New()

    IF( !("CDXDBE" $ dbelist()))
        DbeLoad("CDXDBE")
    ENDIF
    IF( !("FOXDBE" $ dbelist()))
        DbeLoad("FOXDBE")
    ENDIF
    IF( !("FOXCDX" $ dbelist()))
        DbeBuild("FOXCDX","FOXDBE","CDXDBE")
    ENDIF

    cToken := Lower(AllTrim(cToken))

    IF(cToken=="session")
        oDO:Provider := "CxpCookieSessionManager"
        oDO:Timeout := 15
    ELSEIF(cToken=="storage")
        oDO:Provider := "AppStorageManager"
        oDO:Store := "cxp-data\application-storage.dbf"
    ENDIF
RETURN(oDO)
```

Figure 18

## Sending email to a secure SMTP server

Many web apps require sending out an email response. This can be done in a <CXP/> page by using a GMAIL account as the secure SMTP server. If you are creating your own web server with HttpEndPoint() then just add the below code (Figure 19) to your project. If you wish to send email from a CXP page loaded via IIS or Apache, then you must add the below code () to the page-global section of the CXP page.



# Internet Technologies in XBase++ 2.0

Frankfurter Xbase++ Tracks 2015

Roger Donnay

```
//Sendmail.Prg

#Pragma Library("asinet10.lib")
#Pragma Library("dclipx.lib")

FUNCTION SendMail( cSender, cRecipient, cSubject, ;
                  cReplyTo, cAttachment, cText, cErrorText )

LOCAL oMail, oSender, oRecipient, oSmtP, lError := .t., cPrefix, ;
      cUserID, cPassword, cIniFile, nSMTPPort, aRef, aData, ;
      cSMTPServer

DEFAULT cSender := 'rogerdonnay@donnay-software.com', ;
       cRecipient := 'rogerdonnay@donnay-software.com'

aRef := { ;
  'SMTPServer', ;
  'SMTPPort', ;
  'UserID', ;
  'Password' }

aData := { ;
  Space(40), ;
  0, ;
  Space(40), ;
  Space(20) }

DC_IniLoad('Sendmail.Ini','EMAIL',aData,aRef)

cSMTPServer := Alltrim(aData[1])
nSMTPPort := aData[2]
cUserID := Alltrim(aData[3])
cPassword := Alltrim(aData[4])

IF Empty(cSMTPServer)
  cErrorText := 'No SMTP Server Address'
ELSEIF Empty(cSender)
  cErrorText := 'No Sender Address'
ELSEIF Empty(cRecipient)
  cErrorText := 'No Recipient Address'
ELSEIF Empty(cSubject)
  cErrorText := 'No Subject'
ELSE
  lError := .f.
ENDIF
IF lError
  RETURN .f.
ENDIF
```

Figure 19

# Internet Technologies in XBase++ 2.0

Frankfurter Xbase++ Tracks 2015

Roger Donnay

```
oMail      := MIMEMessage():new()
oSender    := MailAddress():new( Alltrim(cSender) )
oRecipient := MailAddress():new( Alltrim(cRecipient) )

// Assemble the e-mail
oMail:setFrom( oSender      )
oMail:setSubject( Alltrim(cSubject) )

oMail:setMessage( cText )
oMail:addRecipient( oRecipient )
IF !Empty(cReplyTo)
  oMail:addHeader( "Reply-To", Alltrim(cReplyTo) )
ENDIF
oMail:setContentType('text/html')

IF !Empty(cAttachment)
  oMail:attachFile( cAttachment )
ENDIF

oSmtP := SMTPClient():new( Alltrim(cSMTPServer), nSMTPPort, 'donnay-software.com' )

IF oSmtP:connect(cUserId,cPassword)
  IF !oSmtP:send( oMail )
    cErrorText := 'Unable to send message - Cause unknown'
  ENDIF
  oSmtP:disconnect()
  lError := .f.
ELSE
  cErrorText := "Unable to connect to mail server"
ENDIF

IF !Empty(cErrorText)
  WTL cErrorText
ENDIF

RETURN !lError

//Sendmail.Ini
[EMAIL]
SMTPServer=smtp.gmail.com
SMTPPort=465
UserID=<your user id>
Password=<your password>
```

Figure 19a

## Using WebSocketClient() and WebSocketHandler()

WebSocketHandler() and WebSocketClient() are used to create client/server applications in Xbase++.

### WebSocketHandler()

WebsocketHandler is an abstract class which must be used to implement the handlers for websocket endpoints. To do this, a user-defined handler class derived from the class *WebSocketHandler* must be implemented that overloads the abstract methods *:onText()*, *:onBinary()*, *:onConnect()* and *:onDisconnect()*.

On incoming websocket connections via the class *HttpEndpoint* an object of the user-defined handler class is instantiated to service the requests from the client. For this the methods *:sendText()* and *:sendBinary()* can be used to send messages to the connected client.

The class *WebSocketHandler* implements the websocket protocol according RFC 6455. If a HTML browser shall be used to establish a connection with the websocket endpoint then the browser must support the websocket protocol according this standard. The following table provides an overview about HTML browsers supporting websockets according RFC 6455:

Web browser	Minimum version
Internet Explorer	10
Firefox (PC)	11
Firefox (Android)	11
Chrome (PC, Mobile)	16
Safari (Mac, iOS)	6
Opera (PC, Mobile)	12.10
Android Browser	-

# Internet Technologies in XBase++ 2.0

Frankfurter Xbase++ Tracks 2015

Roger Donnay

An Xbase++ application can utilize the class `WebSocketClient()` to implement the client site suitable to connect to a websocket server. Below is the source for a sample server that echos data received from a client program (Figure 20).

```
#include "web.ch"

#define PORT 81

PROCEDURE Main()

LOCAL oHttpEndpoint

oHttpEndpoint := HttpEndpoint():new( PORT ):start()

? "HttpEndpoint is running."
wait

oHttpEndpoint:stop()

RETURN

// A websocket handler must implement the methods :onConnect(), onDisconnect(),
onText() // and :onBinary().
//
// The methods are triggered from the HttpEndpoint.

CLASS EchoServer FROM WebSocketHandler
EXPORTED:

INLINE METHOD EchoServer:onConnect()
RETURN self

INLINE METHOD EchoServer:onDisconnect()
RETURN self

INLINE METHOD EchoServer:onText( cText )
::sendText( cText )
RETURN self

INLINE METHOD EchoServer:onBinary( cBinary )
Unused( cBinary )
RETURN self

ENDCLASS
```

Figure 20

## WebSocketClient()

Use the `WebSocketClient` class to implement the client of a websocket connection. A user-defined websocket client is derived from the class `WebSocketClient` and implements the methods `:onConnect()`, `:onDisconnect()`, `:onText()` and `:onBinary()`.

Below is the source (Figure 21) for a sample program that sends text messages to an Echo Server created using `HttpEndPoint()` and `WebSocketHandler()`.

```
#define SERVERNAME  GetEnv("Computername" )
#define PORT        81

PROCEDURE Main

LOCAL oEchoClient, lHandled

oEchoClient := EchoClient():new( SERVERNAME,"EchoServer",PORT )

IF .NOT. oEchoClient:connect()
    ? "Connection failed"
    RETURN
ENDIF

DO WHILE oEchoClient:isConnected()
    lHandled := oEchoClient:handleMessage( .F. )
    IF .NOT. lHandled
        Sleep( 100 )
        oEchoClient:sendText( "Hello. The time is now " + Time() )
    ENDIF
ENDDO

RETURN

CLASS EchoClient FROM WebSocketClient

EXPORTED:
INLINE METHOD onText
? ProcName(), cText
RETURN self

INLINE METHOD EchoClient:onConnect()
? ProcName()
RETURN self

INLINE METHOD EchoClient:onDisconnect()
? ProcName()
RETURN self

ENDCLASS
```

Figure 21

# Internet Technologies in XBase++ 2.0

Frankfurter Xbase++ Tracks 2015

Roger Donnay

## A Picture Server

Look at **C:\users+\<user>\documents\Xbase++\source\samples\multitier\pictureserver**.

This Xbase++ sample demonstrates how an Xbase++ program can serve up pictures (images) to either a client program written in Xbase++, using `WebSocketClient()`, or a Web Browser.