IceBreak

Programmers Guide

Programmers Guide

Creating server applications

by Niels Liisberg

.

This book applies to IceBreak version: V4R04BLD0395 and above.

IceBreak

System & Method A/S

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Printed: januar 2013 in Copenhagen

Publisher

Comprendo.dk

Managing Editor

Kaj Vang Jensen

Technical Editors

Niels Liisberg
Bent Ronne

Cover Designer

Kenneth Riber Andersen

Team Coordinator

Martin Hect Olsen

Production

Comprendo.dk

Special thanks to all the dedicated IceBreak contributors:

Tom Grønkjær Nielsen
Jens Berg Churchill
Anders Thostrup-Clemmensen
Bjarke Aronso
John Foldager
Syd Nicholson
Dan Foldager
Peder Zacho
Jim Cooper
Tracy Eastwood
Edward Wright
Milan Zdimal
Frank Doherty
Per Rahr
Oran Paul
Lars Erik Brodersen
Jens Erik Mejlsbye
John Quarentello
Jørgen Wisbech
Erik Rex
Bent Ronne
Jeppe Klinge Bundgaard
Claus Rytter Larsen
Henning Orensoe

Table of Contents

Part X Administration menu 176

Part XI Appendix 193

```
if (Problem());
   exsr FixIt;
else;
   NOP();
endif;
```

Part I

1 Getting started

1.1 First Step

Getting Started:

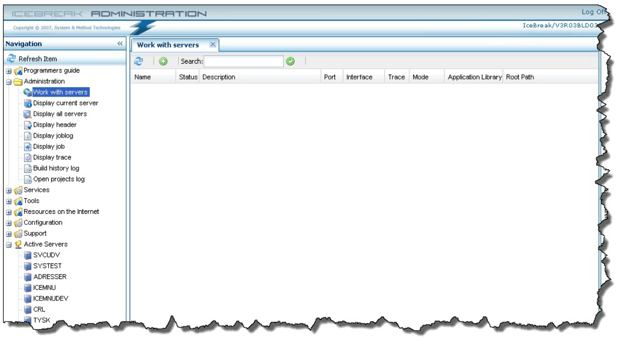
When the IceBreak server is installed an running you can used the administration menu. Here you need to create your own applications server instance where you can run your own applications. Also - you can visit the ApplicationStore on the web from where you can download application, frameworks, plugins and tools...

Now let's get started:

Create my own server instance.

The server instance is a combined webserver and applications server for ILE programs. You can serve plain html, text and image files directly from the IFS AND you can run ILE RPG, COBOL programs from IBMi libraries and JAVA classes. You need to define a root path on the IFS where streamfiles are placed. You also need to define a library name where you application is placed. Logon to the

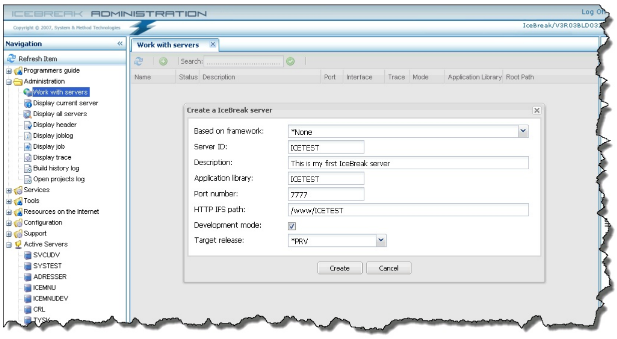
- Logon to the "Administration menu"
- Click "Administration"
- Click "Work with Servers"
- Click the green + or press F6 for "Create a new server":



Now fill in the form - let's call our first server ICETEST

- Enter "icetest" in the server id and give it a good description.

- The application library is automatically set to ICETEST
- The "HTTP IFS path" (or the root path) is automatically set to "/www/ICETEST"
- Now set the port number to 7777. (You can ensure it is available if it not used in the NETSTAT *CNN list)
- Ensure it is in "development mode" which will cause the compilers to be available.
- Finally press the "Create" button.

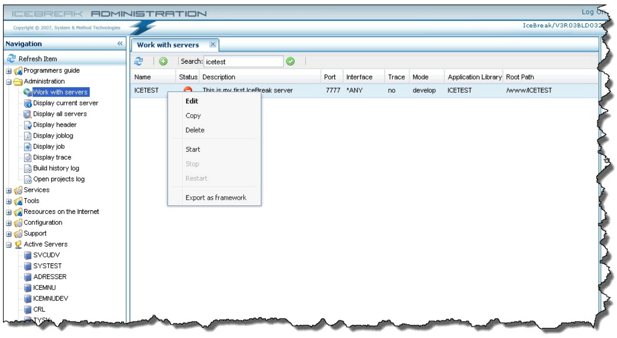


Create a IceBreak server

Based on framework: *None
Server ID: ICETEST
Description: This is my first IceBreak server
Application library: ICETEST
Port number: 7777
HTTP IFS path: /www/ICETEST
Development mode: ☑
Target release: *PRV

Now you have created you server. Now lets start it:

- Right click on the row "ICETEST" in the grid.
- On the drop down menu select "Start"
- Press the "Refresh button"
- The server will now switch from red to green.



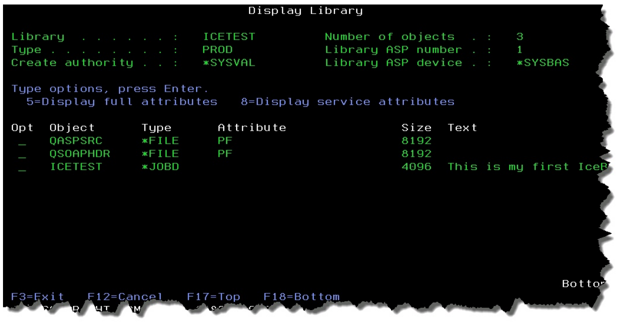
Search: icetest

Name	Status	Description	Port	Interface	Trace	Mode	Application Library	Root Path
ICETEST	🔴	This is my first IceBreak server	7777	*ANY	no	develop	ICETEST	/www/ICETEST

Congratulation !! Now you have an running server you can work with.

If you open a green-screen you can see it has created:

1. Directory /www/ICETEST
2. Library ICETEST
3. Job description ICETEST in ICETEST
4. Source file QASPSRC in ICETEST - This is where the IceBreak pre-compiler will place the program source
5. Source file QSOAPHDR in ICETEST - this is where the IceBreak pre-compiler will place prototypes for SOAP WebServices
6. IceBreak also creates a datashare named ICETEST so you can access the Root path directly. We will need that in a moment.



```
                              Display Library

Library  . . . . . . :   ICETEST         Number of objects  . :    3
Type . . . . . . . . :   PROD            Library ASP number . :    1
Create authority . . :   *SYSVAL         Library ASP device . :    *SYSBAS

Type options, press Enter.
  5=Display full attributes   8=Display service attributes

Opt  Object      Type      Attribute               Size  Text
 _   QASPSRC     *FILE     PF                      8192
 _   QSOAPHDR    *FILE     PF                      8192
 _   ICETEST     *JOBD                             4096  This is my first IceB

                                                                         Botto
F3=Exit   F12=Cancel   F17=Top   F18=Bottom
```

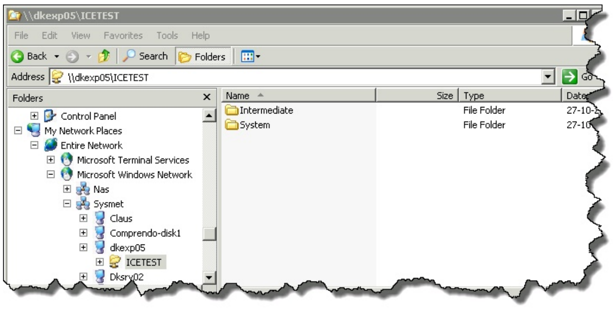
My First application

Any good programming tool can make a "hello world" in a couple of lines. Now lets do exactly that in IceBreak:

First you need to have access to the application Root path. If you open your file explorer you should be able to work with it just bu typing the name of your IBMi and the share name. In my case my IBMi is called DKEXP05 so I enter the address:

\\dkexp05\ICETEST

Which looks like:



Address: \\dkexp05\ICETEST

```
Control Panel
My Network Places
  Entire Network
    Microsoft Terminal Services
    Microsoft Windows Network
      Nas
      Sysmet
        Claus
        Comprendo-disk1
        dkexp05
          ICETEST
        Dksrv02
```

Name	Size	Type	Date
Intermediate		File Folder	27-10-2
System		File Folder	27-10

You will see that IceBreak has created two file folders - "Intermediate" which is a working sandbox and "System" where you will find a lot of common components like images, styles, scripts complete frameworks. Do not delete any of these folders or their contents.

Now lets create a new program file:

- Right click and select "New"
- Select "New Text document"
- Now right click and rename "New Text Document.txt" to hello.aspx

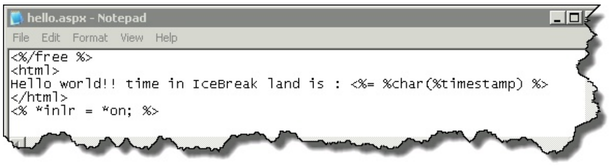
- Open "hello.aspx" with an editor of you choice. I'll use "notepad" in this case.
- Now enter the source code:

```
<%/free%>
<html>
Hello world!! time in IceBreak land is : <%= %
char(%timestamp) %>
</html>
<% *inlr = *on; %>
```

Let's look at the code before you save and run it.

The first thing you will notice is the code is surrounded by <% and %> Every thing that is not enclosed in this escape sequence will be sent to the browser (the client). Next - character strings can be set to the browser if you include the "=" equal sign. So in this case we are converting a timestamp to a character string and sending a dynamic value to the browser. There are several ways to do tat trick in IceBreak, but for now we will just use this monolithic design :)

Yeps - it time to save the source. Ensure your folder now contains hello.aspx (If you have hidden suffixes ensure it is not "hello.aspx.txt" it will not work)



```
<%/free %>
<html>
Hello world!! time in IceBreak land is : <%= %char(%timestamp) %>
</html>
<% *inlr = *on; %>
```

It is time to fire up the baby - and see if it works.

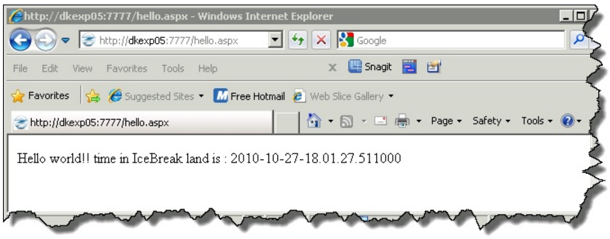
- Open your browser
- In the URL - enter http:// the name of your IBMi : the port / hello.aspx

Since my IBMi is called DKEXP05 the URL looks like:

http://dkexp05:7777/hello.aspx

Notice the name of the server is not shown here - it is only referred to by the port number.

If you have copied and pasted correctly your browser will show something like this:


http://dkexp05:7777/hello.aspx - Windows Internet Explorer
http://dkexp05:7777/hello.aspx
Google
File Edit View Favorites Tools Help
Snagit
Favorites
Suggested Sites
Free Hotmail
Web Slice Gallery
http://dkexp05:7777/hello.aspx
Page
Safety
Tools

Hello world!! time in IceBreak land is : 2010-10-27-18.01.27.511000

Congratulation !! Now you have an running application in your server.

What happens under the hood is what is called a JIT - Just In Time compilation. If IceBreak can see the hello. aspx source but the HELLO program object in ICETEST is missing or outdated, the JIT compiler kicks in and (re) compiles the source and creates the program object.

Even though this sample only contains a few lines of code it illustrates how easy you can bring your IBMi ILE code to the web. How advanced is up to you.

3

Using the component wizard

It is nice to be creative and write applications fast. However some times a little help from the toolbox is required to speed up the process. In IceBreak we have a couple of tools to assist you in this process.

The component wizard is by far the most easy generator to use. It comes in two flavours: vanilla and strawberry - or rather web 1.0 and web 2.0.

Web 1.0 is clean html and has the flavour of the web.

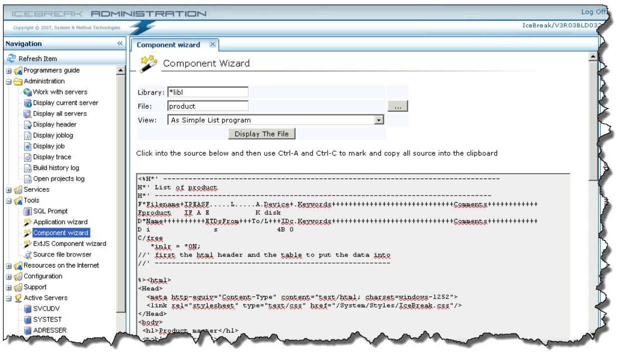
web 2.0 is more javascript and dynamic html and has the flavour of an windows/mac/linux desktop application which requires some kine of frame work to look nice and behave correctly. IceBreak ships with the ExtJS framework which also makes up the administration application.

But first things first - lets have a look at the web 1.0 component wizard and create a simple html based report.

- Open the administration console
- Open "Tools"
- Open "Component wizard"
- In the "library" field enter ***LIBL**
- In the "file" file enter **product**

- In the "view" select **As Simple List Program**
- Now click "display the file" button

Your browser now shows this:



Library: *libl

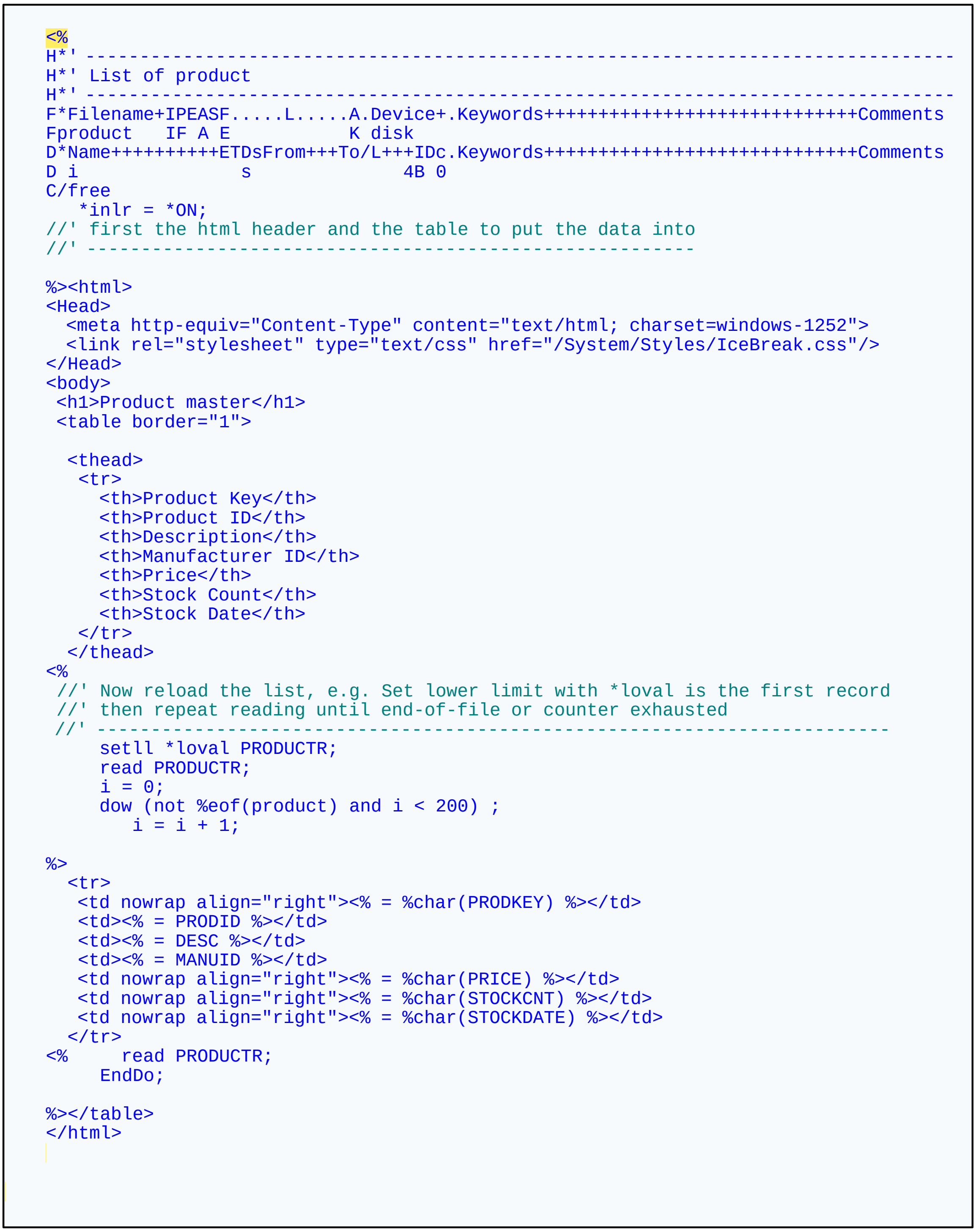
File: product

View: As Simple List program

Click into the source below and then use Ctrl-A and Ctrl-C to mark and copy all source into the clipboard

```
<%H*' --------------------------------------------------------------------------------
H*' List of product
H*' --------------------------------------------------------------------------------
F*Filename+IPEASF.....L.....A.Device+.Keywords+++++++++++++++++++++++++++Comments++++++++++++
Fproduct   IF A E             K disk
D*Name+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++Comments++++++++++++
D i                s               4B 0
C/free
   *inlr = *ON;
//' first the html header and the table to put the data into
//' ------------------------------------------------------------

%><html>
<Head>
  <meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
  <link rel="stylesheet" type="text/css" href="/System/Styles/IceBreak.css"/>
</Head>
<body>
 <h1>Product master</h1>
```

The complete text area will contain the final program:



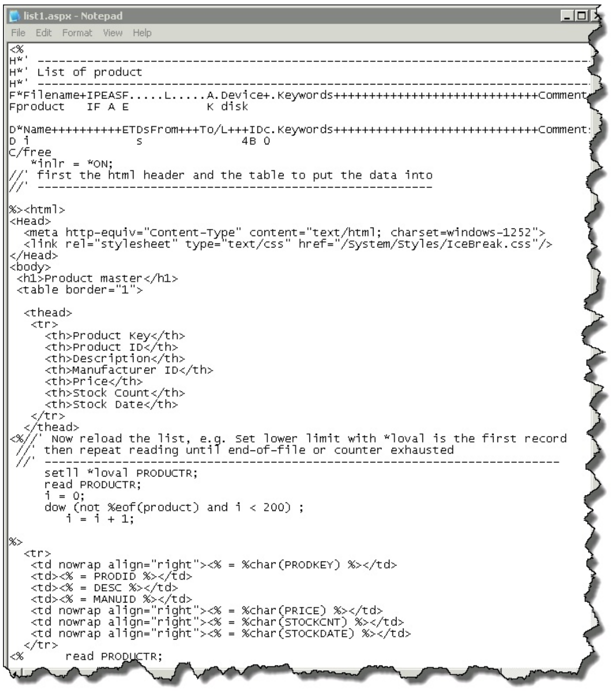
```
<%
H*' --------------------------------------------------------------------------------
H*' List of product
H*' --------------------------------------------------------------------------------
F*Filename+IPEASF.....L.....A.Device+.Keywords+++++++++++++++++++++++++++++Comments
Fproduct   IF A E           K disk
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments
D i               s              4B 0
C/free
   *inlr = *ON;
//' first the html header and the table to put the data into
//' ---------------------------------------------------------

%><html>
<Head>
  <meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
  <link rel="stylesheet" type="text/css" href="/System/Styles/IceBreak.css"/>
</Head>
<body>
 <h1>Product master</h1>
 <table border="1">

  <thead>
   <tr>
     <th>Product Key</th>
     <th>Product ID</th>
     <th>Description</th>
     <th>Manufacturer ID</th>
     <th>Price</th>
     <th>Stock Count</th>
     <th>Stock Date</th>
   </tr>
  </thead>
<%
 //' Now reload the list, e.g. Set lower limit with *loval is the first record
 //' then repeat reading until end-of-file or counter exhausted
 //' ------------------------------------------------------------------------
     setll *loval PRODUCTR;
     read PRODUCTR;
     i = 0;
     dow (not %eof(product) and i < 200) ;
        i = i + 1;

%>
  <tr>
   <td nowrap align="right"><% = %char(PRODKEY) %></td>
   <td><% = PRODID %></td>
   <td><% = DESC %></td>
   <td><% = MANUID %></td>
   <td nowrap align="right"><% = %char(PRICE) %></td>
   <td nowrap align="right"><% = %char(STOCKCNT) %></td>
   <td nowrap align="right"><% = %char(STOCKDATE) %></td>
  </tr>
<%     read PRODUCTR;
     EndDo;

%></table>
</html>
```

Now this looks more like a program - with SETLL and READ and ENDDO - I like that.

Lets create a new file in our server called **list1.aspx**
Just follow that same procedure as with the hello.aspx.

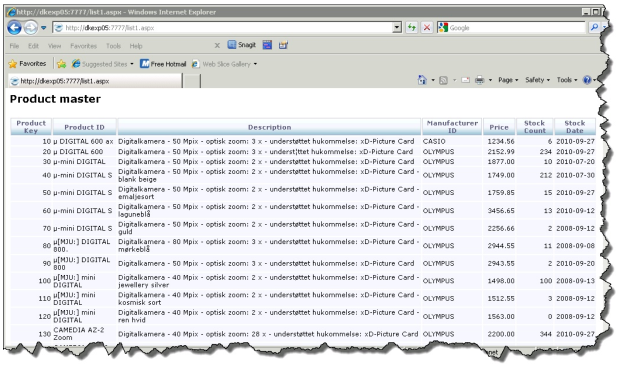
- Right click and select "New"
- Select "New Text document"
- Now right click and rename "New Text Document.txt" to list1.aspx
- Open "list1.aspx" with an editor of you choice. I'll use "notepad" in this case.
- Go back to the Component wizard.
- Click in the code-area and press Ctrl-A and Ctrl-C which is select all and copy to clipboard.
- Go back to notepad
- Now press Ctrl-V for paste
- ... your code has arrived

Your notepad will look like this:



```
<%
H*' -----------------------------------------------------------------------------
H*' List of product
H*' -----------------------------------------------------------------------------
F*Filename+IPEASF.....L.....A.Device+.Keywords+++++++++++++++++++++++++++++Comment
Fproduct   IF A E           K disk

D*Name+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comment
D i               s               4B 0
C/free
   *inlr = *ON;
//' first the html header and the table to put the data into
//' ------------------------------------------------------------

%><html>
<Head>
  <meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
  <link rel="stylesheet" type="text/css" href="/System/Styles/IceBreak.css"/>
</Head>
<body>
 <h1>Product master</h1>
 <table border="1">

  <thead>
   <tr>
     <th>Product Key</th>
     <th>Product ID</th>
     <th>Description</th>
     <th>Manufacturer ID</th>
     <th>Price</th>
     <th>Stock Count</th>
     <th>Stock Date</th>
   </tr>
  </thead>
<%//' Now reload the list, e.g. Set lower limit with *loval is the first record
 //' then repeat reading until end-of-file or counter exhausted
 //' ------------------------------------------------------------------------
     setll *loval PRODUCTR;
     read PRODUCTR;
     i = 0;
     dow (not %eof(product) and i < 200) ;
        i = i + 1;

%>
  <tr>
   <td nowrap align="right"><% = %char(PRODKEY) %></td>
   <td><% = PRODID %></td>
   <td><% = DESC %></td>
   <td><% = MANUID %></td>
   <td nowrap align="right"><% = %char(PRICE) %></td>
   <td nowrap align="right"><% = %char(STOCKCNT) %></td>
   <td nowrap align="right"><% = %char(STOCKDATE) %></td>
  </tr>
<%      read PRODUCTR;
```

When you save it and run it it produces a rather basic report:


http://dkexp05:7777/list1.aspx - Windows Internet Explorer
http://dkexp05:7777/list1.aspx
Google
File Edit View Favorites Tools Help
Snagit
Favorites
Suggested Sites
Free Hotmail
Web Slice Gallery
http://dkexp05:7777/list1.aspx
Page
Safety
Tools

Product master

Product Key	Product ID	Description	Manufacturer ID	Price	Stock Count	Stock Date
10	µ DIGITAL 600 ax	Digitalkamera - 50 Mpix - optisk zoom: 3 x - understøttet hukommelse: xD-Picture Card	CASIO	1234.56	6	2010-09-27
20	µ DIGITAL 600	Digitalkamera - 50 Mpix - optisk zoom: 3 x - underst¦ttet hukommelse: xD-Picture Card	OLYMPUS	2152.99	234	2010-09-27
30	µ-mini DIGITAL	Digitalkamera - 50 Mpix - optisk zoom: 2 x - understøttet hukommelse: xD-Picture Card	OLYMPUS	1877.00	10	2010-07-20
40	µ-mini DIGITAL S	Digitalkamera - 50 Mpix - optisk zoom: 2 x - understøttet hukommelse: xD-Picture Card - blank beige	OLYMPUS	1749.00	212	2010-07-30
50	µ-mini DIGITAL S	Digitalkamera - 50 Mpix - optisk zoom: 2 x - understøttet hukommelse: xD-Picture Card - emaljesort	OLYMPUS	1759.85	15	2010-09-27
60	µ-mini DIGITAL S	Digitalkamera - 50 Mpix - optisk zoom: 2 x - understøttet hukommelse: xD-Picture Card - laguneblå	OLYMPUS	3456.65	13	2010-09-12
70	µ-mini DIGITAL S	Digitalkamera - 50 Mpix - optisk zoom: 2 x - understøttet hukommelse: xD-Picture Card - guld	OLYMPUS	2256.66	2	2008-09-12
80	µ[MJU:] DIGITAL 800.	Digitalkamera - 80 Mpix - optisk zoom: 3 x - understøttet hukommelse: xD-Picture Card - mørkeblå	OLYMPUS	2944.55	11	2008-09-08
90	µ[MJU:] DIGITAL 800	Digitalkamera - 50 Mpix - optisk zoom: 3 x - understøttet hukommelse: xD-Picture Card	OLYMPUS	2943.55	2	2010-09-20
100	µ[MJU:] mini DIGITAL	Digitalkamera - 40 Mpix - optisk zoom: 2 x - understøttet hukommelse: xD-Picture Card - jewellery silver	OLYMPUS	1498.00	100	2008-09-13
110	µ[MJU:] mini DIGITAL	Digitalkamera - 40 Mpix - optisk zoom: 2 x - understøttet hukommelse: xD-Picture Card - kosmisk sort	OLYMPUS	1512.55	3	2008-09-12
120	µ[MJU:] mini DIGITAL	Digitalkamera - 40 Mpix - optisk zoom: 2 x - understøttet hukommelse: xD-Picture Card - ren hvid	OLYMPUS	1563.00	0	2008-09-12
130	CAMEDIA AZ-2 Zoom	Digitalkamera - 40 Mpix - optisk zoom: 28 x - understøttet hukommelse: xD-Picture Card	OLYMPUS	2200.00	344	2010-09-27

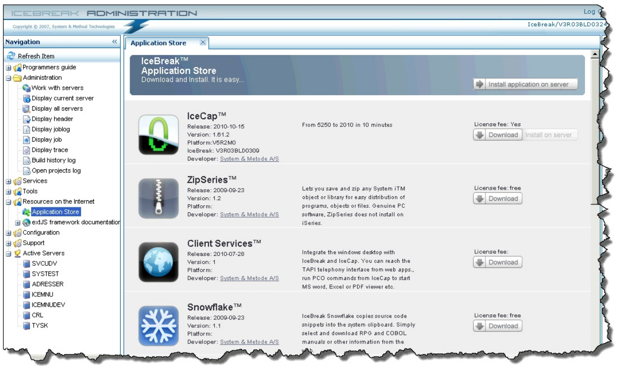
4

Explore the Application Store

Some tools a not installed in the same go as when you install the IceBreak core. A nice (free) tool is **Snow flake** which installs as a companion tool on your windows desktop and contains program snippets you can cut and past into your own work. RPG specs rulers for fixed format - just copy from snowflake and paste.

- Open the administration console
- Open "Resources on the internet"
- Click "Application store"
- Locate "Snowflake"
- Click "Download"

This will bring you through the windows installation process and soon you will have Snowflake installed on your desktop tray



IceBreak™
Application Store
Download and Install. It is easy...

IceCap™
Release: 2010-10-15
Version: 1.61.2
Platform:V5R2M0
IceBreak: V3R03BLD0309
Developer: System & Metode A/S
From 5250 to 2010 in 10 minutes
License fee: Yes

ZipSeries™
Release: 2009-09-23
Version: 1.2
Platform:
Developer: System & Metode A/S
Lets you save and zip any System iTM object or library for easy distribution of programs, objects or files. Genuine PC software, ZipSeries does not install on iSeries.
License fee: free

Client Services™
Release: 2010-07-28
Version: 1
Platform:
Developer: System & Metode A/S
Integrate the windows desktop with IceBreak and IceCap. You can reach the TAPI telephony interface from web apps., run PCO commands from IceCap to start MS word, Excel or PDF viewer etc.
License fee:

Snowflake™
Release: 2009-09-23
Version: 1.1
Platform:
Developer: System & Metode A/S
IceBreak Snowflake copies source code snippets into the system clipboard. Simply select and download RPG and COBOL manuals or other information from the web.
License fee: free

When installing applications that requires to run on the

IBM, you first click the download, which will place the installation files on your desktop - THEN Click **Install application on server.**

These two step allows you to not have access to the internet while accessing the IBMi which sometimes are required for security reasons.

1.2 Prerequisites for installation

Getting Started:

Before installing please check that you have the following information ready:

- First write down the TCP/IP address for your IBMi
- You will need a user profile and password with *SECADM class for installing IceBreak
- Also your FTP-server must be started on your IBMi

Now let's install the IceBreak server system:

- Open your internet browser and find http://icebreak.org
- Select the "download" option
- Fill in your name, company name and e-mail address.
- Click the "Start download" Now you will have a IceBreakSetupxxxx.exe (xxxx indecate the build) which will start the installation on your IBMi / iSeries/ i5 / AS/400.
- Enter the TCP/IP address for your IBMi
- Enter the user profile and password with *SECADM class.
- Now you can see the blue progress bar working while IceBreak is being unpacked and installed
- Finally you will be prompted using the "black/green" environment or the browser based environment .. Just click on the URL
- Congratulation!! - The IceBreak server is now up and running.

Change the administration server to prompt for a user profile and password:

- Open the "Work with server"
- Select the ADMIN server and click "Edit server settings"
- In the "General" tab, Find "logon required" and change it to YES
- Click "Save settings"
- Select the ADMIN server and click "Restart"

1.3 Introducing IceBreak

Getting Started:

IceBreak is a native HTTP-application server for IBMi, iSeries, i5 and AS/400. This server brings the power of the Internet right into the heart of ILE environment on the IBMi so you can create modern web applicatins with you RPG skils and bring legacy applications to the web. The IceBreak Server is an advanced, user-friendly server technology, enabling you to build web applications on the IBM i5/ IBMi/ AS/400 server platform. Yes - You can reuse your RPG coding skills and legacy code.

IceBreak saves source files on the IFS. This gives you the freedom to use any text editor to produce your applications. From low cost or free web designing tools like NotePad, pspPad, HTML-kit and NotePad ++ to professional tools like WDSc, Microsoft Visual Web Development, FrontPage or DreamWeaver - you make the choice depending on your needs and experience. You can even create IceBreak programs using tools on a Mac like iWeb.

You are able to create very efficient web applications running directly under the i5/OS OS/400 / IBM I operating system as native compiled program objects. Thanks to the IceBreak native server your applications are protected against subtle activity on the Internet.

The key to a simple web-development approach is ".aspx", Compiled Active Server Pages. This technology has been used in .NET on Microsoft web servers for a quite some time. Now we are bringing ASPX to the IBMi "Application Server Program eXtension". The syntax is similar and the performance is faster with i5/OS OS/400 / IBM I program objects executing directly from the IceBreak server.

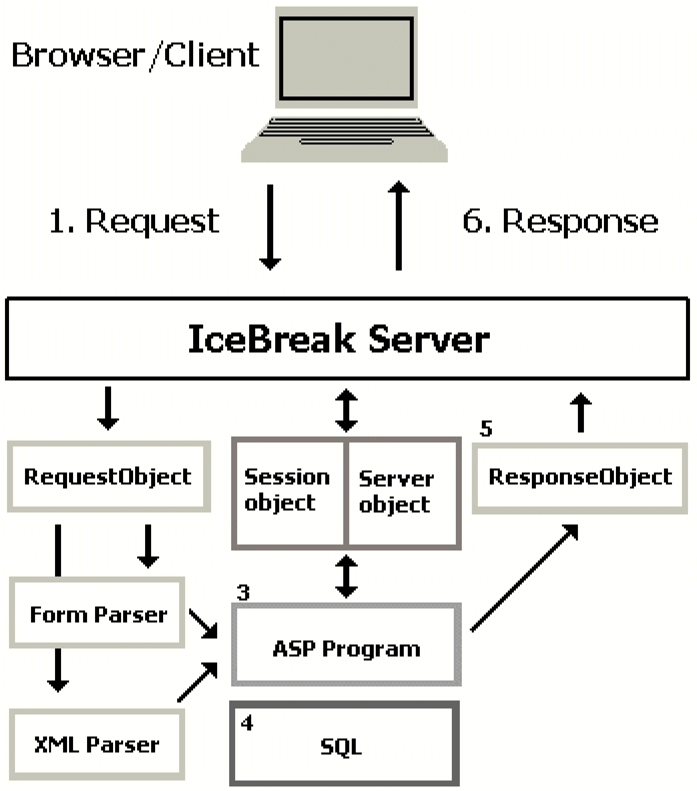
The Object model

The host languages in IceBreak are all the ILE languages: RPG, COBOL or CLP. However, these languages don't have an object model known in the Object Oriented programming world (OOP). IceBreak introduces an interface, so you are able to access the object model directly from RPG, COBOL and CLP.

The object model is hidden from your application-layer, however IceBreak allows the user via an API interface to communicate directly with all objects instantiated by the IceBreak server in RPG, COBOL or CLP.

Take a look at the communication flow between the IceBreak server and client web browsers:

1. The client places a request at the IceBreak server which creates a request object
2. The Request object is processed: Dependent on the content type either the form-parser for HTML or the XML-parser is executed for XML input.
3. ASPX Program initiates- You can retrieve data from the request object(2) and render the response object (5)
4. You can use imbedded SQL to make database I/O, call legacy code, use IBM API's or call IceBreak API's etc.
5. The Response object is prepared to the clients codepage and encoding scheme
6. IceBreak sends the response back to the client



The Objects used in the IceBreak server

When the client (the Browser) requests a HTML document from a web server, it will send a request to the server. The server then receives the response and render the result in the browser canvas.

Let's follow each of these objects involved in the process from the request to the response:

The Request Object:

The Request object contains all data sent from the client. In the HTTP protocol that is the ***header*** and ***content.*** The header describes the content which can be either a "Form Object" or an "XML-Object".

The Form Parser:

The Form Parser initiates automatically when it is a HTML Form-object. The following functions are available in your ASPX-program:

- Myvar=Form('FormVar');
- MyNumvar=FormNum('FormNumVar');
- MyVar=QryStr('URLParameter')
- MyVarNum=QryStr('NumericURLParameter')
- GetHeader(KeyWord)
- GetHeaderList(Name:Value:'*FIRST' | '*NEXT')

The XML parser:

The XML parser is invoked automatically when the content in the request is a XML-document. That is when the content-type is set to "text/xml".

Example:

- Myvar=XmlGetValue('/element1/element2/element[elementnumber]@anattribute' : 'Defaultvalue');

The Session Object:

Fact: The Internet is "stateless" - your program must be designed to process input and produce output - and then terminate. You cannot have open files or static memory between panels.

A Break-through: The IceBreak server is maintaining a session with the client by creating a "session cookie". Even if the client browser is blocking "cookies" a session cookie is normally allowed. Sometimes you have to change the "allow session cookies" flag in the browser for the applications to work correctly.

Sessions are maintained as long as you want, and you can configure the duration of a session using the WRKICESVR or WRKXSVR command. A session has a very small "footprint" and uses little system resources.

Session object functions:

- MyVar = SessGetVar('MyVariable');
- MyVarNum = SessGetVarNum('MyNumericVariable');
- SessSetVar(''MyVariable');
- SessSetVarNum(''MyNumericVariable');

The Server Object:

The server object contains static server information like the configuration parameters, but also dynamic information from the client.

Server object functions:

- MyVar = GetServerVar('AserverVariable');
- PointerToMyVar = GetServerVarPtr('AserverVariable');

The Response Object:

The Response object contains all data sent from the server to the client. In HTTP protocol that is the header and content. The header describes the content which can be of any type HTML, XML, GIF, TIF, PDF, etc. This is the place where you render your dynamic output data with the ASPX syntax.

When your program quits IceBreak will the send the response to the client.

Response object functions:

- ResponseWrite(Value)
- ResponseWriteNL(Value)
- ResponseNLWrite(Value)
- ResponseEncTrim(Value)
- SetContentType(MimeType)
- SetCharset(CharSet)
- AddHeader('Name' : 'Value')
- SetHeader('Name' : 'Value')
- Redirect('ToUrl')
- SetStatus(Code)
- SetCacheTimeout(Minutes)
- SetEncodingType(*HTML (default) | *XML | *NONE)

Follow the rest of the tutorial so you can master ASPX and IceBreak. Enjoy!

Part II

2 Basic Features

2.1 Response object

2.1.1 Placing runtime data in a HTML document

Basic Functions:

The Basic Syntax Rule

An IceBreak-ASPX file normally contains HTML tags, just like an HTML file. However, an ASPX file can also contain **server scripts/code**, surrounded by the delimiters **<%** and **%>**. Server scripts/code are **executed on the server,** and can contain any expressions, statements, procedures, or operators valid for the scripting language you prefer to use. In the following tutorial we will use Free-RPG.

ASPX uses an escape sequence between the HTML document and the code portion.

To start the code escape insert **<%**

To end the code insert **%>**

To place variables from the code insert **<%= pgmvar %>**

A small sample "Hello world" written in Free-RPG might look like:

```
<%/free%>
<html>
Hello world!! time in IceBreak land is : <%= %char(%time) %>
</html>
<% *inlr = *on; %>
```

Line1:
- First we insert **<%** to switch into "script code-mode".
- The RPG-compiler is then set to use "free format-mode".
- Switch back to HTML using **%>** .

Line 2,3,4:
- In HTML-mode you are able to write any valid HTML text.
- Place the value of a function call to retrieve the time value by inserting **<%=**.
- All response is in text format: Convert the result of the **%time** -function with the **%char**-function.
- Switch back to HTML with **%>** .

Line 5:
- Again insert **<%** to switch into "code-mode".
- Insert ***INLR** to terminate the program by setting this switch to ***ON**.
- Finally we switch back to HTML with **%>** .

You cannot view the ASPX source code by selecting "View source" in a browser. You will only see the output from the ASPX file which is plain HTML. This is because the scripts are executed on the server before the result is sent back to the browser.

In our ASPX tutorial, every example displays the hidden ASPX source code. This will make it easier for you to understand how it works.

2.1.2 Placing runtime data using "Markers"

Basic Functions:

Using "Markers"

"Markers" are used to place data in the response object just like using <% = variable %>. A marker value is inserted into the the response object just before data is sent back to the browser.

Markers data will be encoded according the current value set by **SetEncodingType('*html' (default) | '*xml' | '*none');** So it can contain any kind of data: HTML, Scripts etc. but has a limit of 32766 bytes in RPG

There are two steps in using a marker:

- Insert the the marker in the response object. The syntax is: **<%$ Mymarker %>**
- Then assign the marker to a value. This is done by calling the build in function: **SetMarker ('MyMarker' : 'this is the value for my marker');**

A small sample "hello world" written in Free-RPG might look like:

```
<%/free%>
<html>
   The marker value is:
   <%$ MyMarker %>
  ...
<%
   SetMarker('MyMarker': 'hello world!!');
   *inlr= *ON;
%>
```

Separating business logic and presentation layer

Markers also make it possible to separate .aspx code and HTML or XML documents. Here are the two steps:

1. Place all your HTML or XML code in a separate document which can be designed and validated by any HTML or XML editor.
2. Make a .aspx program which refers to a specific tag in the HTML / XML document at runtime.

Your connection with variables in the final response is through markers.

The syntax is the same as for static include in the presentation layer:
<!--#tag="*tagname*"-->

Next you referfollowing to that tag from the program by setting markers and finally render the result by:

ResponseWriteTag('*filename*' : '*tagname*');

Filename:
The file name and path can be absolute or relative for the **ResponseWriteTag(Filename : TagName)** according to the following

• /path/filename.ext	This absolute from the IFS root
• ./Filename.ext	This is relative to the Server instance root path
• Filename.ext	This is relative to the browser relative "refer location"

Tagname:

The tag name is the location in the file - until next tag or end of file (which comes first)

The special values *FIRST can be used as tag name to refer to the first protion of the file prior to any tags. This is useful when dealing with XML-response files since they can be validated. An XML file can not start with a comment, which the tag looks like from a XML file editor’s perspective.

Example:

The HTML document (let's call it ex01marker1.htm) will contain the following:

```
<html>
<body>
<h1>Dynamic placing data using markers</h1>
<table>
<!--#tag="detail"-->
<tr>
  <td><%$ MyCounter %></td>
  <td><%$ MyTime %></td>
</tr>
<!--#tag="end"-->
</table>
</body>
</html>
```

The ASPX program is HTML free and only use the **SetMarker(variable : value);** and the **ResponseWriteTag(Filename : TagName);** build-in function.

```
<%
d i               s             10i 0
  /free
  ResponseWriteTag('./tutorials/ex01marker1.htm' : '*FIRST');
  for i = 1 to 1000;
     SetMarker('MyCounter': %char(i));
     SetMarker('MyTime'   : %char(%timestamp));
     ResponseWriteTag('./tutorials/ex01marker1.htm' : 'detail');
  endfor;
  ResponseWriteTag('./tutorials/ex01marker1.htm' : 'end');
  return;
%>
```

2.1.3 Dynamically including stream files

Basic Functions:

The IceBreak ASPX-compiler has the powerful "include" function which dynamically enables you to include any stream file object in the response. You can import Microsoft Word documents, PDF files, Excel spread sheets, HTML, XML etc.

The Include-function loads files according to the http path entered in the IceBreak server configuration. The file path can be absolute or relative according to the folowing

• /path/filename.ext	This absolute from the IFS root
• ./Filename.ext	This is relative to the Server instance root path
• Filename.ext	This is relative to the browser relative "refer location"

How to let the Browser open the XLS-sheet, not asking for downloading the file.

Run an IceBreak ASPX-program creating the file contents. Then let the browser open the associated file type. For that purpose we use double suffix. IceBreak detects the first suffix, and the browser detects the following suffix.

The file name might be:

```
MyApp.aspx.XLS
```

The contents type however has to be set so the browser can open the resulting file. Otherwise a dialog box will appear asking you to download the file.

The following code include a simple sheet:

```
<%
D*'Name+++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments++++++
++++++
D errstr          s            512    varying
/free

// We will place Excel in the response object
// the "setContentType" must be executed before placing data in the response object
   SetContentType('application/x-msexcel');
   SetCacheTimeOut(240);

// Load the stream file into the response object
   errstr = Include('./tutorials/sheet1.xls');
   if ( errstr > '') ;
      SetContentType('text/html');
      SetCacheTimeOut(0);
      %><%= errstr %><%
   endif;

   return;
%>
```

Please note:

The use of **SetCacheTimeOut(240)** is required. Otherwise the message "404 document not found" is issued because the browser first places the file into the cache - and then opens it with the associated application, in this case Microsoft Excel. The dialog box is avoided by setting the contents type: **SetContentType('application/x-msexcel')**;

How to let the Browser open the DOC with Microsoft Word, not asking for downloading the file

Word documents can also be created in the same way. Now you have to change the contents type to Word document and the application suffix to DOC:

1) **SetContentType('application/msword')**;

2) **MyApp.aspx.DOC**

The following code include a simple Word-document

```
<%
D*'Name+++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments++++++
++++++
D errstr          s            512    varying
/free

// We will place a word-document in the response object
// the "setContentType" must be executed before placing data in the response object
   SetContentType('application/msword');
   SetCacheTimeOut(240);

// Load the stream file into the response object
   errstr = Include('./tutorials/testword.doc');
   if ( errstr > '') ;
      SetContentType('text/html');
      SetCacheTimeOut(0);
      %><%= errstr %><%
   endif;

   return;
%>
```

Can I do that for any file type?

Yes! As long you know the **SetContentType** and the **FileSuffix.** Maybe you need to search the Internet to find the right Content type, also called the MIME type but here is a popular list:

Application to start	File Suffix	SetContentType / MIME type
Microsoft Word	DOC	application/msword
Microsoft Excel	XLS	application/x-msexcel
Adobe Acrobat	PDF	application/pdf
Kodak imaging	TIF	image/tiff
Microsoft Power Point	PPT	application/vnd.ms-powerpoint
Rich Text	RTF	application/rtf
Application programs	EXE .. Others	application/octet-stream

Hint: The "ShowSample.aspx" used in this tutorial includes an example displaying the source by wrapping the APS/HTML content in **<XMP></XMP>** tags.

IceBreak includes a DB/2 table called MIM00 which controls the default MIME-type for different file extensions. You can modify this table but first backup this file. You can replace the contents with your own version. This table indicates which extensions ASPX-programs are using.

The default default MIME type for an filename or attribute can be retrieved by:

MyMimetype = GetMimeType('FilenameOrAttribute.ext')

The following code is including a PDF file and find the content type using the GetMimeType function:

```
<%
D*'Name+++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments++++++
++++++
D errstr          s            512    varying
/free

// We will place PDF file in the response object
// the "setContentType" must be executed before placing data in the response object
   SetContentType(GetMimeType('./tutorials/sample.pdf'));
   SetCacheTimeOut(240);

// Load the stream file into the response object
   errstr = Include('./tutorials/sample.pdf');
   if ( errstr > '') ;
      SetContentType('text/html');
      SetCacheTimeOut(0);
      %><%= errstr %><%
   endif;

   return;
%>
```

2.1.4 Sending the response back to the client

Basic Functions:

The response is by default build up in a response buffer, which will be transferred back to the client (the browser) in one go - when your applications programs returns.

In some cases you might find it convenient to send the buffer back in smaller pieces. Some situations might be:

- Your application runs for a long time and produces output occasionally.
- You program produces more that 16MB which is the maximum size of the response buffer
- You are creating a "Web Push technology" applications.

In the above situations you might to exploit the "**chunked**" transfer encoding.

Chunked.

IceBreak automatically used the chunked transfer encoding after you call the **setChunked();** IceBreak procedure. The header will immediately be sent back to the client (the browser) so it is important that you setup all headers (if any) before calling **setChunked()**;Likewise you must set the chunked before you send any data to the response object.

The setChunked() procedure also takes minimum size of a chunk buffer you want to sent. Don't make it to small since it will have a performance impact - a good approximation might be around 4K. Each time this threshold is exceeded the output buffer will be flushed and sent back to the client.

If you want to control when the buffer is actually sent back to the client, you can call the **responseFlush();** which sending the buffer and afterwards clear it. It only works in Chunked mode. When used in "normal" mode the output buffer only gets cleared.

Syntax:

setChunked(BufferSize);

BufferSize:

- The buffer threshold value in bytes for automatically flushing the response to the client (browser)
- CHUNK_AUTO_FLUSH used it your application uses markers.

Syntax:
responseFlush();

Sample:

```
<%
d i                 s             10i 0
/free
  setChunked(4096);
%><html><%
  for i = 1 to 1000;
    %>Sending chunked data at :<% = %char(%TimeStamp()) %><br/><%
  endfor;
  *inlr= *ON;
%></html>
```

Also use this feature if you having a processes bar or for long running task as status information purposes:

```
<%
d i                 s             10i 0
/include qasphdr,posix
/free
   setChunked(1024);
%><html><%
   for i = 1 to 10;
     %>Running step <%= %char(i) %> of 10<br>
     <%                 // An extra new-line is required before the browser displays the
line
     responseFlush();   // Here the response is being sent back to the client
     sleep(1);          // This is a C-library function you can access if you include the
POSIX member from QASPHDR
   endfor;
   *inlr= *ON;
%>
</html>
```

Markers and Chunks

When you are using markers you have to bare in mind that your static response and the dynamic marker values are rendered when you send the response back to the client. You have to synchronize that behaviour with the chunks being sent.

The easiest way is to set the chunk minimum size to CHUNK_AUTO_FLUSH. That causes the response to be rendered and flushed in one go when you use the responseWriteTag(); procedure. Otherwise you can use the combination parseMarker() / responseFlush();

Sample:

```
<%
d i               s             10i 0
  /free
  setChunked(CHUNK_AUTO_FLUSH);
  responseWriteTag('./tutorials/ex01mark1.htm' : '*FIRST');
  for i = 1 to 1000;
     setMarker('MyCounter': %char(i));
     setMarker('MyTime'   : %char(%timestamp));
     responseWriteTag('./tutorials/ex01mark1.htm' : 'detail');
  endfor;
  responseWriteTag('./tutorials/ex01mark1.htm' : 'end');
  return;
%>
```

The following sample is if you want to control the flow 100% by you self. This has actually a better performance since the buffer is only sent to client for each 100 resulting rows.

```
<%
d i               s             10i 0
  /free
  setChunked(1000000);
  responseWriteTag('./tutorials/ex01mark1.htm' : '*FIRST');
  for i = 1 to 1000;
     setMarker('MyCounter': %char(i));
     setMarker('MyTime'   : %char(%timestamp));
     responseWriteTag('./tutorials/ex01mark1.htm' : 'detail');
     if (%rem(i : 100) = 0);
        responseFlush();
     endif;
  endfor;
  responseWriteTag('./tutorials/ex01mark1.htm' : 'end');
  return;
%>
```

2.1.5 Trimming the response object

Basic Functions:

Depending on the nature of your application you can set up how the response is trimmed before returning to the client.

The default trimming option is set on the server with **CHGICESVR** and the keyword **DFTTRIM**

The DFTTRIM can have 4 values:

- *NONE - The field value goes as it is to the client
- *LEFT - All leading blanks from left is removed before sending the result to the client
- *RIGHT - All trailing blanks from right is removed before sending the result to the client
- *BOTH - Both leading and trailing blanks are removed before sending the result to the client

You can override this value at runtime by calling the build-in **setTrim();**

setTrim() can process 4 values:

- **setTrim(TRIM_NONE)** The field value goes as it is to the client
- **setTrim(TRIM_LEFT)** All leading blanks from left is removed before sending the result to the client
- **setTrim(TRIM_RIGHT)** All trailing blanks from right is removed before sending the result to the client
- **setTrim(TRIM_BOTH)** Both leading and trailing blanks are removed before sending the result to the client

If you want to remove white space from the source stream you can use the compiler directive **trimoutput='YES'**

2.2 Request object

2.2.1 The "Form" function

Request object:

When you use input fields in an IceBreak-ASPX page - you make a "form" in HTML. Input field surrounded with the "form" can be "posted" back into the IceBreak Server:

When you want to use the content of the posted form use the "Form" Icebreak-ASPX function to retrieve the posted value from the request object.

The following example is IceBreak ASPX code in Free-RPG:

```
<%
d myName          s            256    varying
/free

// Take the "myName" attribute from the form object an place it
// into a program variable
// ----------------------------------------------------------
   myName = form('myName');

%>
<html>
<form action="ex01Form.aspx" method="post" name="form1">
  Enter your name, and press enter :
  <input type="text" name="myName">
</form>
When I come back to this page my name is: <%= myName %>
</html>
<% *inlr = *on; %>
```

Note the relation between "<form ... post .... input .. name="MyName" ...

"MyName" is an input text field on the posted form.

Listing all fields in a form

You can also list both values and field names placed on a form and returned to the IceBreak application like:

```
<%
 *' -------------------------------------------------------------------------- *
 *' Demo : Show all form fields received from the browser-client
 *' -------------------------------------------------------------------------- *
d  Field          s            256    varying
d  Value          s           4096    varying
c/free
%>
<html>
<head>
 <link rel="stylesheet" type="text/css" href="/System/Styles/IceBreak.css"/>
</head>
<body>
<form method="post" id=form2 name=form2>
<input type=text name=myinput />
<input type=text name=moreinput />
<input type=submit name=ok />
<table>
  <thead>
   <th>Field</th>
   <th>Value</th>
  </thead>
<%
    GetFormList (Field: value : '*FIRST');
    dow (Field > '') ; %>
       <tr>
         <td><%= Field %></td>
         <td><%= Value %></td>
       </tr>
<%     GetFormList (Field: value : '*NEXT');
    Enddo;

*inlr = *on;
%>
</table>
</form>
</body>
</html>
```

2.2.1.1 Using the "FormNum()" function

Request object:

Just like the "Form()" ASPX function you can use the "FormNum()" function to retrieve numeric values from a form.
The "FormNum()" function returns the numeric value from a "form" field Into a RPG variable or use the value in an expression.

The value returned is always a "packed decimal(30,15)" To allow a wide range of numeric data sizes. The conversion in respect to number of digits and decimal position is done by the RPG-runtime.

All formatting chars like -,;$ etc. are striped out. Therefore is this function is also useful to converting date field, since date formatting is striped out. The decimal sign , or . is used in respect to the underlying

IceBreak server job description.

ex:

```
eval Number = FormNum('MyFormField');
```

or:

```
if (FormNum('MyFormField') <= 0) then;
%> My Form Field has to be a valid number <%
endif;
```

```
<%
d myNumber        s              9  5
/free

 myNumber = FormNum('myNumber');


%>
<html>
<head>
 <link rel="stylesheet" type="text/css" href="/System/Styles/IceBreak.css"/>
</head>
<body>
<p>The value of "myNumber" is : " <%= %char(myNumber) %>
</p>
<form name="form1" method="post" action="ex01FormNum.aspx">
  Enter a number. Use decimal points and signs as you please:
    <input name="myNumber" value="<%= Form('myNumber') %>">
    <input type="submit" name="Submit" value="Submit">
</form>
</body>
</html>
<%
  *inlr = *on;
%>
```

2.2.2 Using the "QryStr" function to parse parameters along the URL

Request object:

The "QryStr" Function.

The "QryStr" takes the value from a parameter parsed along with the URL.

There are (at least) two ways to build the URL in a HTML document:

- Passing the variable name and value along the <a href ... tag
- use the method= "GET" on the form object.

The following example illustrates both ways in IceBreak ASPX code Free-RPG:

```
<%/free%>
<html>
```

```
<head>
 <link rel="stylesheet" type="text/css" href="/System/Styles/IceBreak.css"/>
</head>

<form method="get">
  <p>Enter your name, and press enter : </p>
  <input type="text" name="myname">
  <p>When I come back to this page my name is: <%= QryStr('myname') %></p>
  <a href="ex01QryStr.aspx?myname=ABC">This link takes the value "ABC" along the URL back
to the application</a>
</form>
</html>
<% *inlr = *on; %>
```

Listing all fields in the query string

You can also list both values and field names parsed on the query string returned to the IceBreak application like:

```
<%
 *' --------------------------------------------------------------------------- *
 *' Demo : Show all query string fields received from the browser-client
 *' --------------------------------------------------------------------------- *
d  Field          s            256    varying
d  Value          s           4096    varying
c/free
%>
<html>
<head>
 <link rel="stylesheet" type="text/css" href="/System/Styles/IceBreak.css"/>
</head>
<body>
<form method="get" id=form1 name=form1>
<input type=text name=myinput />
<input type=text name=moreinput />
<input type=submit name=ok />
<table>
  <thead>
   <th>Field</th>
   <th>Value</th>
  </thead>
<%
    GetQryStrList (Field: value : '*FIRST');
    dow (Field > '') ; %>
       <tr>
         <td><%= Field %></td>
         <td><%= Value %></td>
       </tr>
<%     GetQryStrList (Field: value : '*NEXT');
    Enddo;

*inlr = *on;
%>
</table>
</form>
</body>
</html>
```

2.2.2.1 The "QryStrNum" Function

Request object:

The "QryStrNum" takes the numeric value from a parameter parsed along with the URL

To retrieve input fields from an ASPX page you use the "form" tag in HTML. Input field surrounded with the "form" can be sent along the URL by the "GET" function back into the IceBreak Server:

The following example is IceBreak ASPX code in Free-RPG

```
<%
D*'Name+++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments++++++
++++++
D MyNumber        s              15  5
/free
  MyNumber = QryStrNum('MyNumber');
%>
<html>
<head>
 <link rel="stylesheet" type="text/css" href="/System/Styles/IceBreak.css"/>
</head>
<body>
<form action="ex01QryNum.asp" method="get" name="form1">
  <p>Enter a number or a date and press enter :
  <input type="text" name="MyNumber"></p>
  <p></p>When I come back to this page the number is: <%= %char(MyNumber) %></p>
  <a href="ex01QryNum.asp?MyNumber=1234,56">This link takes the value "1234,56" along the
URL back to the application</a>
</form>
</html>
<% *inlr = *on; %>
```

2.3 Server object

2.3.1 Controlling the HTTP Header

Server Behaviour:

The HTTP header is the way you control the Browser behavior. Here are the functions available:

Function name	Description	Sample
SetHeader	Set a header attribute in the response	SetHeader('Location':'http://www.agentdata.com');
Redirect	Force the browser to reload a page from another URL	Redirect('http://www.agentdata.com');
SetCharset	Set the character set used for the resulting document	SetCharset('windows-1252');
SetContentType	Set the MIME type on the resulting document	SetContentType('text/html');
SetContentType	Set the MIME type and character set used for the resulting document	SetContentType('text/html; charset=windows-1252');
SetStatus	Changes the HTTP status code for	SetStatus('401 Access denied');

	the response. See the complete list ***here***	
GetHeader	Returns the value of a given header keyword in this case the full URL	val= GetHeader('Referer');
GetHeaderList	Returns all headers one by one	GetHeaderList(Keyword , Value, '*FIRST' | '*NEXT');

Here is a sample

```
<%
 *' -------------------------------------------------------------------------- *
 *' Demo : How to make a redirect
 *' -------------------------------------------------------------------------- *
 /free

//' One way to make a "redirect"
    Redirect('http://www.google.com');

//' The same thing made "by hand"
//    SetStatus('302 Redirect');
//    SetHeader('Location' : 'http://www.google.com');

%> Never say Hello world - i'll be redirected to google before that ....<%
*inlr = *on;
%>
```

Run the sample   Show the sample

This sample shows all headers received from the browser-client

```
<%
 *' -------------------------------------------------------------------------- *
 *' Demo : Show all headers received from the browser-client
 *' -------------------------------------------------------------------------- *
D  Keyword        S            256    varying
D  Value          S           4096    varying
 /free
%>
<head>
 <meta http-equiv=Content-Type content="text/html; charset=windows-1252">
 <link rel="stylesheet" type="text/css" href="/System/Styles/IceBreak.css"/>
</head>
<body>

<br><br>
Get a header value direct: Host = <%= GetHeader('host') %>
<br><br>

 <table>
  <thead>
   <th>Keyword</th>
   <th>Value</th>
  </thead>
<%
    GetHeaderList (Keyword: value : '*FIRST');
    dow (Keyword > '') ; %>
       <tr>
         <td><%= Keyword %></td>
         <td><%= Value %></td>
       </tr>
<%     GetHeaderList (Keyword: value : '*NEXT');
    Enddo;

*inlr = *on;
%>
 </table>
</body>
</html>
```

Run the sample   Show the sample

2.3.2 Retrieving Server Information

Server Behaviour:

The server has many configuration parameters and runtime information which are available to you ASPX-program. Call the the **GetServerVar()** function to place values in your program variable.

Example:

```
MyServerId = GetServerVar('SERVER_ID');
```

The complete list with actual run-time values:

Server variable	Description
SERVER_SOFTWARE	The name and version of the running server
SERVER_ID	Name of current server instance
SERVER_TOKEN	The enumerated number of the server instance
SERVER_DESCRIPTION	Description of the server instance
SERVER_LOCAL_PORT	The TCP/IP portnumer (from 1 to 65535) where the server is polling for requests
SERVER_INTERFACE	The TCP/IP interface where the server is polling for requests. (Interface created by i5/OS command ADDTCPIFC)
SERVER_JOB_NAME	The underlaying i5/OS job name
SERVER_JOB_USER	The underlaying i5/OS job user name (not the active user)
SERVER_JOB_NUMBER	The underlaying i5/OS job internal job number
SERVER_JOB_MODE	The state of the server: "*PROD" or "*DEVELOP"
SERVER_LOGON_REQUIRED	An authenticaton prompt is automatically displayed when the user request the page and are "unknown" (Not available yet)
SERVER_STARTUP_TYPE	Whether the server should start when the subsystem is activated or must be started by **STRICESVR** or **STRXSVR**
SERVER_DEFAULT_USERPROFILE	The user profile used when starting a server instance process
SERVER_ROOT_PA	The IFS path used for web document. This can not be relative but is fixed to the IFS-

Server variable	Description
TH	root
SERVER_DFT_DOC	The web document displayed when no specific document is requsted. This is relative to the SERVER_ROOT_PATH
SERVER_CACHE_TIMEOUT	Number of seconds before a document expires. 0=immediately. Servers in *DEVELOP mode always overrides this value to 0 so the cache always is refreshed
SERVER_INPUT_BUFFER_SIZE	Number of bytes in the input buffer (the Request Object). When zero a default of 1Mbytes is used
SERVER_OUTPUT_BUFFER_SIZE	Number of bytes in the output buffer (the Response Object). When zero a default of 1Mbytes is used
SERVER_COOKIE_BUFFER_SIZE	Number of bytes in the output buffer (the Response Object). When zero a default of 64Kbytes is used
SERVER_INITIAL_PGM_NAME	Name of program to set extra libray list etc. It is called when the server instance is initiate - Not each time a new client connects or ***NONE**
SERVER_INITIAL_PGM_LIB	Name of library where the initial program exists
SERVER_JOBQ_NAME	The jobqueue from where the server process is started
SERVER_JOBQ_LIB	Name of library where the jobqueue exists"
SERVER_APPLICATION_LIB	This is where all your APS-programs are placed when they are compiled. This is where the QASPSRC file is created with your precompiled ASPX-program sources
SERVER_TGTRLS	The default target i5/OS release for programs created on this server instance
SERVER_TRACE_FILE	The name of the trace file created when **TRACE=*ON** When blank, the file name defaults to **Trace.txt** in the **SERVER_ROOT_PATH**
SERVER_SYSTEM_NAME	The system name from the network attribute
SERVER_OS_VER	The current version and relase of i5/OS
SERVER_CCSID	The current CCSID or *AUTO if the CCSID is dynamic seletected
SESSION_TIMEOUT	Number of seconds before the session automatically is terminated Default is 1440 seconds
SESSION_ID	The Unique Session Timestamp-id; also it is the time when the session was started
SESSION_NUMBER	The Unique Session number; Also it is the job number of the first lightwaight job that initiated the session
SESSION_USERID	The i5/OS userprofile logged on. When no logon was issued it returns ***DEFAULT**
REMOTE_ADDR	The remote TCP/IP address of the client web browser
REMOTE_PORT	The remote TCP/IP port number negotiated by the TCP/IP layer
REQUEST_HOST_NAME	The TCP/IP address or name for the requested server
REQUEST_METHOD	The method the document was requested: GET parameters is parsed along the URL POST Parameters are parsed in the form object
REQUEST_URL	The complete URL. E.g. the resource filename with path and extension and parameters
REQUEST_FULL_PATH	The resource filename with path and extension
REQUEST_REF_PATH	The refered path from the http header if given. Otherwise the reffence path where the resource were requested
REQUEST_PATH	The path portion only of the request
REQUEST_FILE	The filename and extension only
REQUEST_RESOURCE	The resource (filename only) portion only of the request in uppercase
REQUEST_FIRST_EXTENSION	The first extension for the resource of the request in uppercase. If only one extension exist this will be the same value
REQUEST_EXTENSION	The extension for the resource of the request in uppercase
REQUEST_HEADER	The complete header string
REQUEST_RAW	The complete request, excluding the content

Server variable	Description
REQUEST_CONTENT	The content string
QUERY_STRING	Parameters sent along the GET or POST request after the document. The URL in the browser
HIVE_NAME	If running i a hive this is the virtual path name (the hive name). Otherwise blank
HIVE_PATH	If running i a hive this is the physical path to the IFS. Otherwise blank
HIVE_LIB	If running i a hive this is the application library for the current . Otherwise blank

All i5/OS system values are also available by prefixing the system value with **"SYSVAL_".**

like:

```
   DayOfWeek = GetServerVar('SYSVAL_QDAYOFWEEK');
```

The complete list of system values can be found in the i5/OS command **WRKSYSVAL** Here are samples values

Server variable	Sysval	Description
SYSVAL_QDAYOFWEEK	QDAYOFWEEK	The day of week
SYSVAL_QTIME	QTIME	The current time

Run the sample   Show the sample

2.3.3 System Global Variables

Basic Functions:

System Global Variables are persistent between sessions and servers. I.e. if you create a system global variable in one session on one server it will be visible in all other servers and session. So be carful - especially when you clear system global variables - it has an impact on the entire system.

Functions:

globalSetVar (*variableName* **:** *value* **);**

myVar = globalGetVar(*variableName* **:** *[defaulrValue]* **);**

globalSetVarNum (*variableName* **:** *numeric value* **);**

myNumVar = globalGetVarNum(*variableName* **:** *[default Numeric Value]* **);**

globalClrVar (*qualified name* **);**

Variable Name:

You can use any name for the global variable, however we suggest that you qualify you names according to X-path naming, since you will be able to import and export using XML in releases to come. Also for the reason that the name you use is system wide. At least provide some kind of name space - for instance the current server name.

Value:

Any string expression - up to 32K bytes

Default value:

Any string expression - up to 32K bytes. Applied when the variable does not exists in the System Global domain. When the variables exists, the default value is ignored.

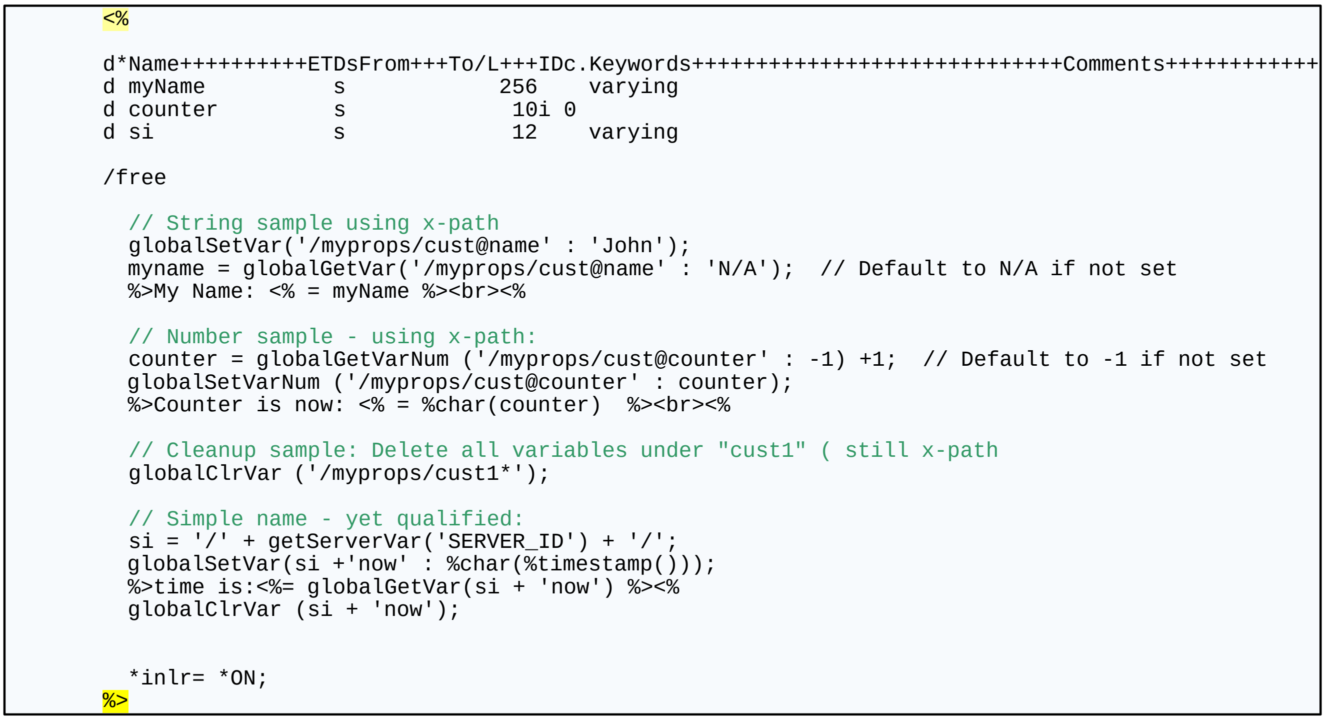
Numeric value:

Numeric expression that can be stored in a 31.15 decimal variable

Qualified name:

Any string expression. You can apply the * to give a generic value to clear - up to 32K bytes. Applied when the variable does not exists in the System Global domain. When the variables exists, the default value is ignored.

Example:



```
<%

d*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments++++++++++++
d myName          s            256    varying
d counter         s             10i 0
d si              s             12    varying

/free

  // String sample using x-path
  globalSetVar('/myprops/cust@name' : 'John');
  myname = globalGetVar('/myprops/cust@name' : 'N/A');  // Default to N/A if not set
  %>My Name: <% = myName %><br><%

  // Number sample - using x-path:
  counter = globalGetVarNum ('/myprops/cust@counter' : -1) +1;  // Default to -1 if not set
  globalSetVarNum ('/myprops/cust@counter' : counter);
  %>Counter is now: <% = %char(counter)  %><br><%

  // Cleanup sample: Delete all variables under "cust1" ( still x-path
  globalClrVar ('/myprops/cust1*');

  // Simple name - yet qualified:
  si = '/' + getServerVar('SERVER_ID') + '/';
  globalSetVar(si +'now' : %char(%timestamp()));
  %>time is:<%= globalGetVar(si + 'now') %><%
  globalClrVar (si + 'now');


  *inlr= *ON;
%>
```

Note:

The data is stored in the DB/2 table SGV00 - and you can manipulate it directly by i.e. SQL, but be aware that this might change from release to release.

Also, if you need to mirror IceBreak between and use System global variables - you need to mirror this

DB/2 table SGV00.

2.4 Session object

2.4.1 Session management

Basic Functions:

The internet has no session mechanism - the internet is stateless. However, state information is required to track data from one page to another in a modern web application. IceBreak support two different ways to keep track of sessions namely by session cookies and by URL Rewriting. Both methods have some pros and cons. What you choose to use depends on your application.

Session cookie

By default all session in iceBreak is maintained by a session cookie. The session ID is automatically placed in the http header and sendt back and forth between the browser and the IceBreak server. However, browsers may disable the use of cookies - even harmless session cookies. Also, some browsers have difficulties with serving the right session cookie to the right application in a multi-frame environment. I.e. when the outer frame is served by one server - let's say an IIS and the inner frame is served by IceBreak. In both cases you need to use the URL rewriting method.

URL Rewriting (session stability by URL redirection)

URL Rewriting is one of the popular session tracking methods used for Non-cookie browsers to save session information. URL Rewriting tracks the user's session by including the session ID in the URL (to carry information from one HTML page to another). The information is stored inside the URL, as an additional parameter where all the links on a page are re-written so that the server side program receives the old as well as new data. Thus a session (or a connection is maintained between multiple pages) for every user.

Since the session ID is exposed to the user it might be changed. In that case IceBreak just creates a new session.

Some pitfalls/considerations:

- Beware that you only use relative redirection.
- Ensure that Ajax calls refer to the same session URL.
- Sessions might be bookmarked and even transferred to other clients

Accessing the session object

You can put data into the session object and retrieve data back by some simple IceBreak functions:

- MyVar = SesGetVar('MyVariable');
- MyVarNum = SesGetVarNum('MyNumericVariable');
- SesSetVar('MyVariable' : MyValue);
- SesSetVarNum('MyNumericVariable': 1234567.89);

Also ILOB's are in maintained by the session object. More on that later in the ILOB chapter.

2.4.2 Log on to an icebreak session

Basic Functions:

In IceBreak you can use either the native i5/OS user profile login or you can use an LDAP server or Windows server running active directory to authticate a IceBreak session.

Log on using an I5/OS user profile

In many cases you will build applications that requires running authenticated under another user profile than the default profile that was used when the server instance was started.

For that purpose the IceBreak-ASPX has the following build-in functions:

LogOn

Syntax:

Message = LogOn (Userprofile : Password);

Field Name	Data type	Description
UserProfile	CHAR(10)	The i5/OS User profile
Password	CHAR(128)	The associated password for the above user profile
Returns		
Message	VARCHAR(512)	If blank the logon attempt was successful; otherwise it contains a descriptive message

When the "LogOn" executes successfully the underlying job for the IceBreak will change the session profile and all successive jobs associated with that session will use same profile handle.

Ex:

```
<%
d msg             s            512          varying
/free
msg = LogOn(Userid:Password);
if (msg <= '');
   Initpage();
   return;
else;
   exsr ShowLogon;
endif;
...
 %>
```

Run the sample   Show the sample

Log on using an LDAP server or Windows server running Active Directory

This is a simple way to authenticate the user profile and password against LDAP or Windows server. However, since there is no connection between the user profile on i5/OS and LDAP/Windows this will only be an authentication vailidation, but no credentials are given.

For that purpose the IceBreak-ASPX has the following building functions:

LDAP_Logon

Syntax:

flag = LDAP_Logon (Server : Userprofile : Password);

Field Name	Data type	Description
Server URL	VARCHAR(128)	Server name or TCP/IP adrress. Explicit host list.
UserProfile	VARCHAR(128)	The User profile on LDAP / Windows Active Directory
Password	CHAR(128)	The associated password for the above user profile
Returns		
flag	BOOL (indicator)	When *ON an error occured and you can examin that error by calling GetLastError();

Explicit Host List Specifies the name of the host on which the LDAP server is running. The *host* parameter may contain a blank-separated list of hosts to try to connect to, and each host may optionally be of the form *host:port* .

The following are typical examples:

```
error =  LDAP_Logon('server1' : MyUser : MyPwd);
error =  LDAP_Logon('server2:1200':MyUser : MyPwd);
error =  LDAP_Logon('server1:800server2:2000 server3' :MyUser : MyPwd);
```

You can also use a default host. i.e when the *host* parameter is set to ("ldap://") the LDAP library will attempt to locate one or more default LDAP servers, with non-SSL ports, using the SecureWay

error = LDAP_Logon ('ldap://' : MyUser : MyPwd);

If more than one default server is located, the list is processed in sequence, until an active server is found.

The "Server URL" can include a Distinguished Name (DN), used as a filter for selecting candidate LDAP servers based on the server's suffix (or suffixes). If the most significant portion of the DN is an exact match with a server's suffix (after normalizing for case), the server is added to the list of candidate servers. For example, the following will only return default LDAP servers that have a suffix that supports the specified DN:

```
error = LDAP_Logon ('ldap:///cn=niels, dc=cph, dc=agentdata, dc=com' :MyUser: MyPwd);
```

In this case, a server that has a suffix of "dc=austin, dc=ibm, dc=com" would match. If more than one default server is located, the list is processed in sequence, until an active server is found.

If the "Server URL" contains a host name and optional port, the host is used to create the connection. No attempt is made to locate the default server(s), and the DN, if present, is ignored.

For example, the following two are equivalent:

```
error =  LDAP_Logon('ldap://myserver' : MyUser : MyPwd);
error = LDAP_Logon('myserver':MyUser : MyPwd);
```

In general you will always test the result of the log on by retrieving the messages from the GetLastError() function.

Ex:

```
<%
....
error = LDAP_Logon(Server:Userid:Password);
if (error);
   <% = GetLastError() %>
   return;
else;
   exsr ReadyToGo;
endif;
...
 %>
```

Show the sample

LoggedOn

Syntax:

Flag = LoggedOn ();

Field Name	Data type	Description
Flag	Indicator/ Logical / boolean	Returns *ON if a session is logged on.

The "LoggedOn()" function can be used to redirect the user to the correct logon ASPX page where the user id/password can be entered.

Or it can be used to determine whether or not the session is run as Anonymous session.

Ex:

```
<%
....
if (not LoggedOn());
   exsr doLogon;
   return;
endif;
...
 %>
```

batchLogin

You can also create an IceBreak session from a batch or interactive job. This is useful when you run IceBreak in mixed environments with CGIDEV2, PHP or java from the same IBMi box.

From you within a (no-IceBreak) RPG program you issue a "bachLogin" which in turn gives you a session handle to an IceBreak session. Notice that you need to provide a userid and password - This is native IBMi profile information.

Also your RPG program has to bind to the ICEBREAK bin directory so you need ICEBREAK on your library list to compile and run it.

You can now concatenate the session id into an URL to your icebreak application and application will run with the credentials specified:

```
h bnddir('ICEBREAK')
 /include qasphdr,icebreak
d session         s             22    varying
 /free
  session = batchLogon('SYSTEST  ' : 'DEMO' : 'DEMO');
  url = 'http://myIBMI:1234/'
      + session
      + '/myProgram.aspx';

  // now provide the url to the client
  ....
  *inlr = *ON;
```

When you are done with the session you can explicitly terminate the session if needed with a batchLogoff:

```
h bnddir('ICEBREAK')
 /include qasphdr,icebreak
d session         s             22    varying
 /free
  ...
  batchLogoff ( 'SYSTEST  ' :   session);
  *inlr = *ON;
  ....
```

2.5 Server behaviour

2.5.1 Setting up HTTPS

Server Behaviour:

See the videocast on from the Internet

HTTPS is the secure version of the HTTP protocol. It is the same but uses the Secure Socket Layer - SSL as the transportations.

You need a certificate to run IceBreak with HTTPS. This certificate can be issued by Verisign (tm) among others - or you can build one your self. In this tutorial we will step through the configuration of the IceBreak server instance and build a certificate using the buld-in "Digital Certificate Manager" that ships with i5/OS - OS/400.

Step 1: Create a certificate.

- Open the "IBMi task": Click on the following link http://MySystemI:2001
- Logon as QSECOFR
- Click on "Digital Certificate Manager"
- Click on "Select a Certificate store"
- Select "*SYSTEM"
- Now it shows the path to the certificate - it might be "/QIBM/USERDATA/ICSS/CERT/SERVER/DEFAULT.KDB".
- Write the file path down - it must be entered into the IceBreak configuration later.
- Enter the password for the certificate - the password is case-sensitive. If you don't know what it is - the click on "reset password"
- Write the password down - it must be entered into the IceBreak configuration later.
- Create a new Certificate:
- 1: Select "Server / client certificate" [Continue]
- 2: Select "Local Certificate Authority" [Continue].
- 3: Fill in required values. [Continue]
- 4: Don't select anything. [Continue]
- Assign the Certificate to the certificate store:
- 1: Select "manage certificate Store".
- 2: Select "Set default certificate" [Continue].
- 3: Select the certificate you just made [Continue].

Step 2: Configure the IceBreak server instance.

- Go to the iceBreak administration menu.
- Click on work with servers
- Stop the server you want to run with HTTPS
- Select and edit the server you want to run with HTTPS
- Click on the advanced button
- Set the protocol type to "HTTPS"
- Set the certificate file path and password to what you wrote down in step 1.
- Restart the server.

You don't have to make any changes in your application to utilize HTTPS.

2.5.2 Use the pre-request exit program

Server Behaviour:

The pre-request program is an .aspx program that is executed before each and every "normal" request is processed in IceBreak.

It can be used for custom designed security, URL overriding, generic heading handling etc.

How to configure

The .aspx program you want to use must be compiled by referring to it from a normal browser URL. Due to performance reasons the normal JIT compilation is bypassed for the pre-request exit program.

Now - change the server instance to refer to the exit program. Use either CHGICESVR command or use the "ADMIN" page "work with server instances" and change the pre-request exit program parameter to the name of your program. Also you can set the library name if you want to let more server instances point to the same exit program.

Use the exit program for Internet / Intranet security

When you are making web applications you will face that some resources might be public available where the user is anonymous. Also you might want to build your own internet user account which has nothing to do with i5/OS security at all.

Now you can put your security logic into the pre-request exit program and secure all your resources.

You can control whether or not IceBreak will server a specified resource with the build in function SetBreak. If you use SetBreak(*ON) the normal http serving is bypassed and only the response object from your pre-request exit program is sent back to the client (the browser)

Syntax:

SetBreak (*ON | *OFF);

This example let IceBreak serve anything but .PDF files:

```
<%
...
extention = GetServerVar('REQUEST_FIRST_EXTENSION');
if (extention = 'PDF');
   SetContentType('text/html; charset=windows-1252');
   %>You can not view PDF files - sorry <%
   SetStatus('401 Access denied');
   SetBreak (*ON);
endif;
return;
%>
```

Performance considerations

Since this program is called each and every time a request is made to the icebreak server instance you have to design it to use very little resources and keep the program on the "stack" (e.g. don't seton *INLR). Also keep files open if any.

Advanced login

Then next sample show how to make a basic authentication using a pre-request exit program.

Thanks to Jens Berg Churchil - KTP, Denmark:

```
<%

  D*ame+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++
  D pos             s              5U 0
  D Flag            s              1S 0 INZ(0)
  D lo              c                   'abcdefghijklmnopqrstuvwxyzæøå'
  D up              c                   'ABCDEFGHIJKLMNOPQRSTUVWXYZÆØÅ'
  D Auth64          s           1024A   VARYING
  D Auth            s           1024A   VARYING
  D Username        s            128A   VARYING
  D Password        s            128A   VARYING
  D Status          s            128A   VARYING
  D Message         s            512A   VARYING

  /free

  // Only if the user is not logged on yet
  if not LoggedOn();
    Auth64 = GetHeader('Authorization');
    pos = %Scan(' ' : %Trim(Auth64));

    if (pos = 6); // Expect 'Basic BASE64_ENCODED_STR'
      Auth64 = %Subst(Auth64 : pos + 1);
      Auth64 = xlateStr(Auth64 : 0 : 1250);
      Auth = Base64DecodeStr(Auth64);
      Auth = xlateStr(Auth : 1250 : 0);

      pos = %Scan(':' : Auth);
      if (pos > 1);
        Username = %Subst(Auth : 1 : pos - 1);
        Password = %Subst(Auth : pos + 1);

        Username = %Xlate(lo : up : Username);

        Message = Logon(Username:Password);

        if LoggedOn();
          SesSetVar('USERNAME' : Username);
        endif;

        if (%Scan('deaktiveret' : Message) > 0);
          Flag = 2;
        endif;
      else;
        Flag = 1;
        Message = 'Invalid Authorization header';
      endif;
    endif;

    if not LoggedOn();
      if (Flag = 1);
        Status = '400 Bad Request';
      elseif (Flag = 2);
        Status = '403 Forbidden';
      else;
        Status = '401 Unauthorized';
        SetHeader('WWW-Authenticate':'Basic realm="My Webserver name"');
      endif;

      SetContentType('text/html; charset=windows-1252');
      SetStatus(Status);
      SetHeader('x-login-message' : Message);

      SetMarker('Message' : Message);
      SetMarker('Status' : Status);

%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/
TR/1999/REC-html401-19991224/loose.dtd">
<HTML>
  <HEAD>
    <TITLE><%$ Status %></TITLE>
    <META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=ISO-8859-1">
  </HEAD>
  <BODY><H1><%$ Status %>.</H1><P><%$ Message %></P></BODY>
</HTML>
<%

      SesEnd();
      SetBreak (*ON);
    endif;
  endif;

  return;

  /end-free
%>
```

Part III

3 Building applications

3.1 Debugging IceBreak applications

Programming:

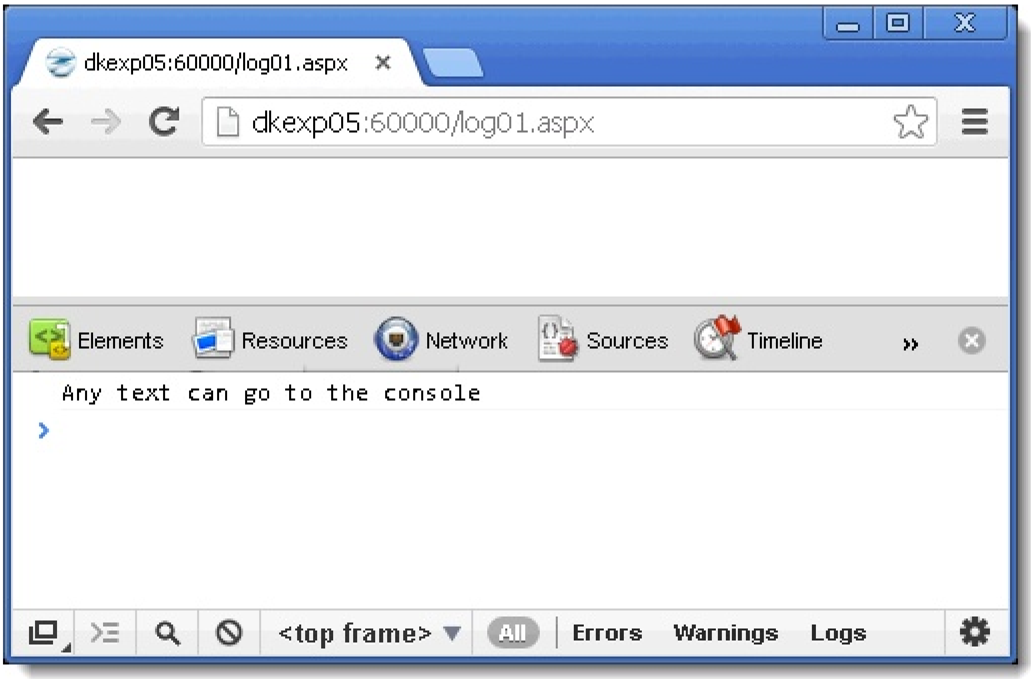
Debugging a web application can be very difficult, IceBreak has some build in features which make life a lot more easy. You can use the 5250 debugger, the GUI debugger or simply use the console to log data values. Use what ever fits your needs.

Using the "Console":

The simplest ways to debug it just to send values of variables directly to the console. This can me achieved by the adding the following code to your RPGLE program:

```
<%
 /free
    consoleLog('Any text can go to the console');
    *inlr  = *ON;
%>
```

This text will now occur in the browser debugger console along with other consol log notifications:



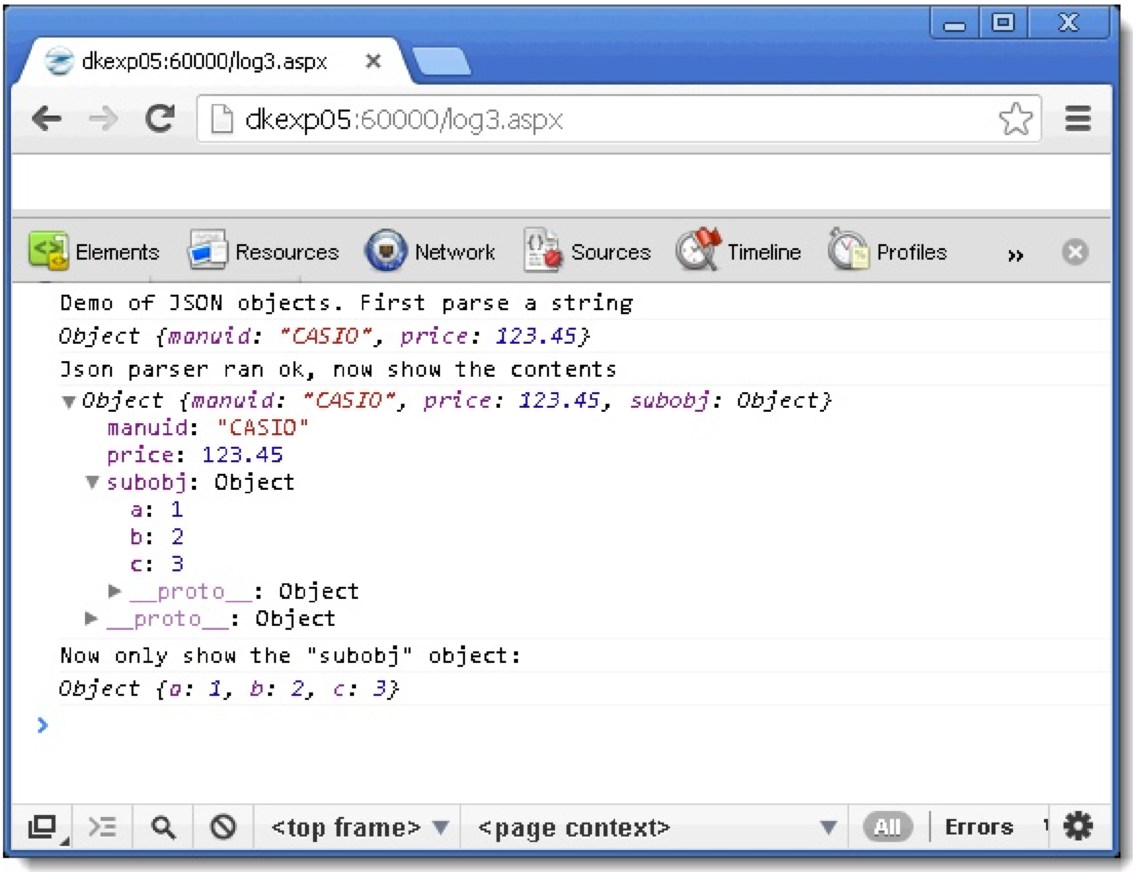
```
Any text can go to the console
```

Pleas notice, that the **consoleLog** uses a protocol called "WildFire" whitch requires a pluging like FirePHP to be installed in you Chrome of FireFox browser.

When you are building XML or JSON based applications you can log objets and arrays from the IceBreak XML and JSON parser directly. The following code is using both simple strings and complex JSON objects:

```
<%
// Example:
// Using the "consoleLog" on Json objects
// ------------------------------------------------------------------

d pJson           s               *
d subobj          s               *

 /include qasphdr,jsonparser
 /free

  *inlr  = *ON;
  consoleLog('Demo of JSON objects. First parse a string');

  consoleLog('{        +
    "manuid":"CASIO",  +
    "price" :123.45    +
  }');


  pJson = json_ParseString ('{  +
    "manuid":"CASIO", +
    "price" :123.45,  +
    "subobj": {       +
      "a": 1,         +
      "b": 2,         +
      "c": 3          +
    }                 +
  }');

  if json_Error(pJson) ;
    consoleLog('Json error:' + json_Message(pJson));
    json_Close(pJson);
    return;
  endif;

  consoleLog('Json parser ran ok, now show the contents');
  consoleLogJson(pJson);

  consoleLog('Now only show the "subobj" object:');
  subobj = json_locate(pJson : 'subobj');
  consoleLogJson(subobj);

  json_Close(pJson);
```

Which will result in the following output in firebug / chrome console:



```
Demo of JSON objects. First parse a string
Object {manuid: "CASIO", price: 123.45}
Json parser ran ok, now show the contents
▼Object {manuid: "CASIO", price: 123.45, subobj: Object}
    manuid: "CASIO"
    price: 123.45
  ▼subobj: Object
      a: 1
      b: 2
      c: 3
    ▶__proto__: Object
  ▶__proto__: Object
Now only show the "subobj" object:
Object {a: 1, b: 2, c: 3}
>
```

Using traditional server side debuggers:

IceBreak also provides support for both GUI and basic 5250 debugging. Please ensure to put the server into *MULTITHREAD mode before starting any of the debugging approaches. Otherwise it can be rather difficult to find which jog is actually service you application. But in *MULTITHREAD you have one IBMi job for each "WebJob" so You can use both System I debuggers: The GUI debugger and the 5250 service job debugger.

The GUI debugger

If you have IBMi Navigator installed, you have already installed the system debugger. Otherwise read the chapter installing the system debugger.

You need some settings before you can use the debugger:

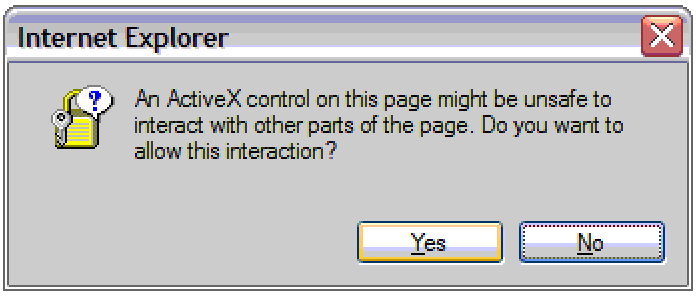
- You browser must allow Active-X components
- The debug server must run: **STRDBGSVR** and **STRTCPSVR SERVER(*DBG)** (or CALL QSYS/QTESSTRSVR for V5R1)
- The IceBreak server must be in development mode
- The IceBreak server must be running in *MULITHREAD (or *SINGLE) mode

From the URL where you application you want to debug type:
/system/strdbg.aspx

E.g if you have a server instance running on port 60001 it will look like:

http://mysystem:60001/system/strdbg.aspx

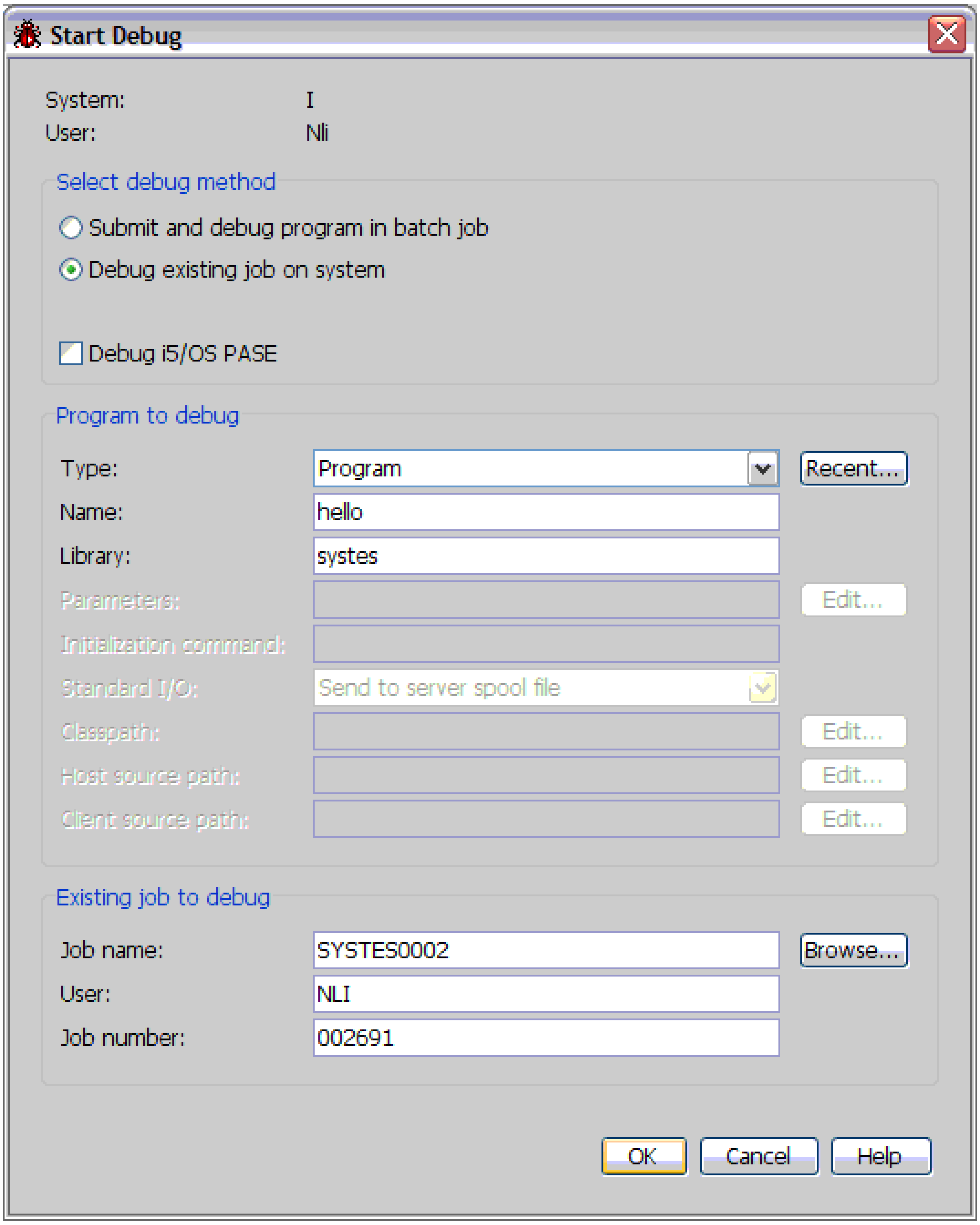
An Active-X warning might pup up now. please continue.



Internet Explorer

An ActiveX control on this page might be unsafe to interact with other parts of the page. Do you want to allow this interaction?

Short after the system debugger will appear. You will have to fill in the name and library of the program to debug:



Start Debug

System: I

User: Nli

Select debug method

○ Submit and debug program in batch job

● Debug existing job on system

□ Debug i5/OS PASE

Program to debug

Type: Program

Name: hello

Library: systes

Parameters:

Initialization command:

Standard I/O: Send to server spool file

Classpath:

Host source path:

Client source path:

Existing job to debug

Job name: SYSTES0002

User: NLI

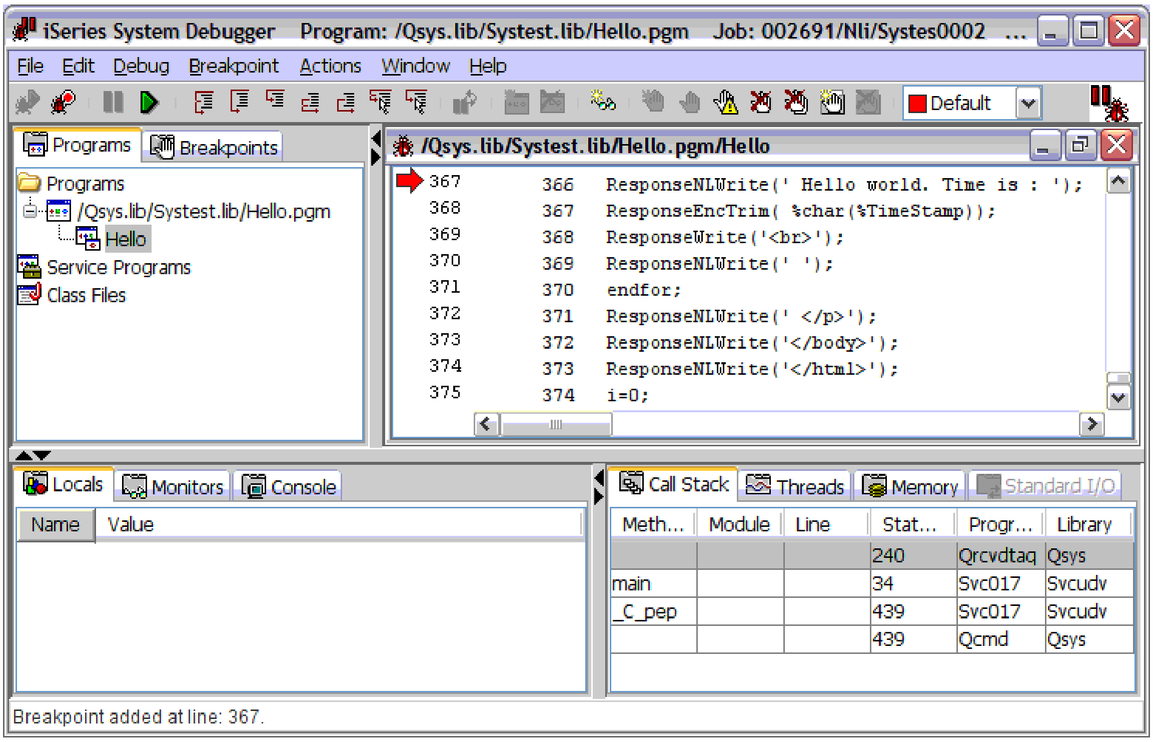
Job number: 002691

Now press OK and the intermediate source of your program will appear. Place a break-point and press the green "Run" button.

Go back to the browser and run the program by entering the name at the URL like:

http://mySystemi:60001/hello.aspx

Application now runs to the breakpoint line. You can not step through the source by F11. Use the build in help to investigate the debugger functionalities:


iSeries System Debugger Program: /Qsys.lib/Systest.lib/Hello.pgm Job: 002691/Nli/Systes0002 ...
File Edit Debug Breakpoint Actions Window Help
Default
Programs
Breakpoints
Programs
/Qsys.lib/Systest.lib/Hello.pgm
Hello
Service Programs
Class Files
/Qsys.lib/Systest.lib/Hello.pgm/Hello
Locals
Monitors
Console
Name
Value
Call Stack
Threads
Memory
Standard I/O

```
367     366   ResponseNLWrite(' Hello world. Time is : ');
368     367   ResponseEncTrim( %char(%TimeStamp));
369     368   ResponseWrite('<br>');
370     369   ResponseNLWrite(' ');
371     370   endfor;
372     371   ResponseNLWrite(' </p>');
373     372   ResponseNLWrite('</body>');
374     373   ResponseNLWrite('</html>');
375     374   i=0;
```

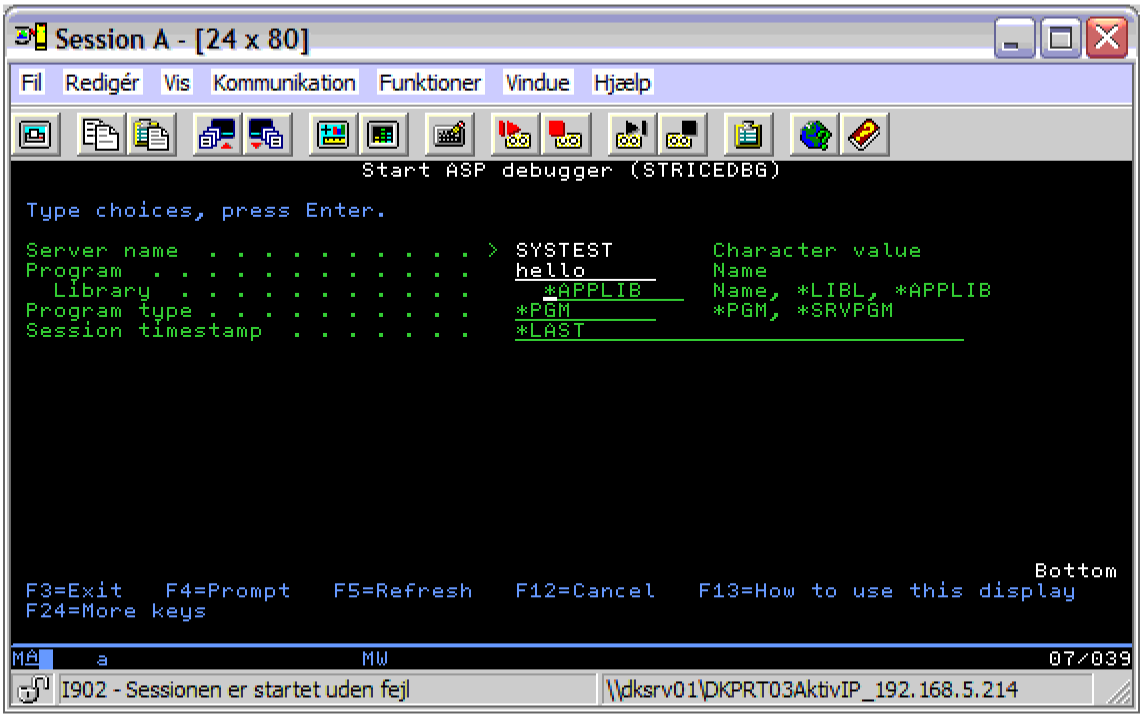
Meth...	Module	Line	Stat...	Progr...	Library
			240	Qrcvdtaq	Qsys
main			34	Svc017	Svcudv
_C_pep			439	Svc017	Svcudv
			439	Qcmd	Qsys

Breakpoint added at line: 367.

Using the 5250 debugger.

If You prefer the green-screen debugger, you can still use that.

- Navigate to the IceBreak menu with **GO ICEBREAK**
- Enter command **WRKICESVR**
- Enter option 17 at the server you want to debug which will prompt the **STRICEDBG** command


Session A - [24 x 80]
Fil Redigér Vis Kommunikation Funktioner Vindue Hjælp

```
                        Start ASP debugger (STRICEDBG)

 Type choices, press Enter.

 Server name  . . . . . . . . . . . > SYSTEST       Character value
 Program  . . . . . . . . . . . . .   hello         Name
   Library  . . . . . . . . . . . .     *APPLIB     Name, *LIBL, *APPLIB
 Program type . . . . . . . . . . .   *PGM          *PGM, *SRVPGM
 Session timestamp  . . . . . . . .   *LAST

                                                                       Bottom
 F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F13=How to use this display
 F24=More keys
MA     a                  MW                                           07/039
```

I902 - Sessionen er startet uden fejl
\\dksrv01\DKPRT03AktivIP_192.168.5.214

Enter the name of the program you want to debug. Now the service debugger will appear.

If you will be absolutely sure that you hit the right session then use /system/dspsvrinf.aspx from browser URL and find the session timestamp for the session you are debugging. The value ***LAST** refer to the most reason session that was started in the IceBreak subsystem

http://MySystemi.org:60000/system/dspsvrinf.aspx

Installing the GUI debugger.

You can install this and it work equaly well on Windows, Linux and Mac:

If you are running I5/OS V5R1 or V5R2 you need a few PTF's

V5R1 :

- Server PTF SI09825
- Client PTF SI06031

V5R2:

- Server PTF SI09834, SI08512
- Client PTF SI09844

Java 1.3 (or later) run-time environment must be installed on your IBMi. This is product 5722-JC1 option 5.

You will also need a recent JRE on your client PC.

You must copy 3 ".jar" (Java Archive) files to your PC. Locate or create a directory and copy from the IBMi:

- /QIBM/ProdData/HTTP/public/jt400/lib/jt400.jar
- /QIBM/ProdData/HTTP/public/jt400/lib/tes.jar
- jhall.jar from http://java.sun.com/products/javahelp

Place them in **C:\i5debug**

Change the CLASSPATH to refer to C:\i5debug.

If you're debugging from “outside” the office firewall, you will need port 4026 open which is the “single point of contact” for the user interface. Note that port 4026 is registered in the IBMi service table as “as-debug”.

3.2 Using compiler directives

Precompiler:

When the IceBreak compiles a program it uses a Just-In-Time technology. The JIT compiler is invoked only if the server instance is in *DEVELOP mode and if the source files has been modified since the last compilation. That means - if you change a program it will be re-compiled when you open it from the browser next time.

The IceBreak compiler has 4 stages:

- IceBreak pre-compiled
- IBM SQL pre-compiled [Optional]
- IBM ILE compiler [RPG/COBOL/C++/CLP]
- IBM Binding [CRTPGM or CRTSRVPGM]

All steps can be controlled by compiler directives within the source file starting the source file with a **<%@** and terminated by **%>** like:

```
  <%@ language="SQLRPGLE" sqlopt="COMMIT(*NONE)" modopt="text('demo')" pgmopt="BNDDIR
(UTIL)" %>
```

For each step that utilize IBM commands you can override the default values with *sqlopt, modopt* and *pgmopt*

Directive	Value	Default	Description
language	RPGLE	*	Source code is in ILE-RPG format
	CBLLE		Source code is in ILE-COBOL format
	CPP		Source code is in ILE-C++ format
	CLLE		Source code is in ILE-CL
	SQLRPGLE		Source code is in ILE-RPG format with SQL
	SQLCBLLE		Source code is in ILE-COBOL format with SQL
	SQLCPP		Source code is in ILE-C++ format with SQL
pgmtype	PGM	*	Bound program with access to the IceBreak Object model
	SRVPGM		Service program with access to the IceBreak Object model
	NOASPMOD		Program Module. No access to the IceBreak Object model
	NOASPSRVPGM		Service program. No access to the IceBreak Object model
	WEBSERVICE		Service program. With access to the IceBreak Object model and all procedures exposed as WebServices

Directive	Value	Default	Description
trimoutput	NO	*	The response object is rendered as is
	YES		White space are removed from the response object, reducing the final output
srcstyle	FREE	*	RPG columns are aligned automatically, Block comment /* */, and free SQL syntax are allowed
	WDSC		RPG syntax is strict according to columns, comments and SQL
sqlopt			Any parameters for CRTSQLxxx command
modopt			Any parameters for CRTRPGMOD/CRTCBLMOD/CRTCPPMOD/CRTCLMOD command
pgmopt			Any parameters for CRTPGM/CRTSRVPG command

3.3 How to include external resources

Programming:

Dynamic include

When you include data into your ASPX-program it can be done in to different ways - either static or dynamic. Dynamic include is done by the IceBreak server at run time by the **ResponseObject**, which are covered in :

Placing runtime data using "Markers"

Dynamically including stream files

Static include

Static include is done by the IceBreak pre compiler or by the host language compiler.

Host language include

If you use RPG as the host language you might be familiar with the **/COPY** or **/INCLUDE** or if you use COBOL it looks like **"copy MemberName in SourceFile"**.
With these copy function you can place only code into **source physical file**.

IceBreak, however also has a include feature in the pre compiler which allows you to both include code as well as XML and HTML into **source physical file** from where the IBM compiler is running.

The include can be placed anywhere in your source allowing you to initialize constants from included stream files, program code, and for the response object both HTML, XML, JSON etc.

This is also very convenient if you still like to use the PDM as you preferred source editor, because you can make one very small ASPX-program in the IFS that only includes the complete source from a source file in a library.

IceBreak pre-compiler include

In a IceBreak ASPX program we use the ASPX notation for including resources. the syntax is:

```
<!--#include file="FileToInclude.htm" -->
```

The "FileToInclude.htm" above is any stream file relative to the **current resource** where you issued the include statement. Of cause you can address any stream file in the IFS. But then you have to prefix the name with at **"/"** like:

```
<!--#include file="/root/www/FileToInclude.htm" -->
```

You can also include **relative** to the server root path by prefixing the file name with a **"."** like:

```
<!--#include file="./System/Includes/debug.inc.rpgle" -->
```

You can put this include anywhere in your source, so now you are able to include both HTML and ASPX-code, initialize constants with values from the include file etc.
Include can be done recursively with a max level to 200 files...

IceBreak extension to the ASPX-include

We have made an extension to the ASPX-include syntax. we have introduced **Tags**. Tags can be used to include a fragment from the include file. The syntax is:

```
<!--#include file="./System/Includes/debug.inc.rpgle" -->
```

The include file can now contain tags that will be included. Like:

```
<!--#include file="./System/Includes/debug.inc.rpgle" -->
```

The pre compiler will now include all data just after the tag until it finds a new tag or the end of file.
This technique can be used to externally describe HTML files - just like a normal DDS-displayfile. This is a way to separate code and HTML into two or more different files.

Consider the following ASPX program **ex01IncApp.aspx**:

```
<%/free

  %><!--#include file="ex01IncApp.htm" tag="header"--><%
  select;
    when Form('Screen') = 'Screen03';
      %><!--#include file="ex01IncApp.htm" tag="Screen03"--><%
    when Form('Screen') = 'Screen02';
      %><!--#include file="ex01IncApp.htm" tag="Screen02"--><%
    Other;
      %><!--#include file="ex01IncApp.htm" tag="Screen01"--><%
  endsl;
  %><!--#include file="ex01IncApp.htm" tag="footer"--><%
  *INLR = *ON;
%>
```

Now its counter part - the html file **ex01IncApp.htm** :

```
<!--#tag="header"-->
<html>
 <head>
  <title>This shows the use of externally described HTML files</title>
 </head>
 <body>
  <form method="POST">

<!--#tag="screen01"-->
<p>This is screen 01</p>

<!--#tag="screen02"-->
<p>This is screen 02</p>

<!--#tag="screen03"-->
<p>This is screen 03</p>

<!--#tag="footer"-->
   <input type="submit" name="Screen" value="Screen01">
   <input type="submit" name="Screen" value="Screen02">
   <input type="submit" name="Screen" value="Screen03">
  </form>
 </body>
</html>
```

This technique is very important when you migrate legacy programs into new ASPX programs - but also when you design new applications. With this approach you can let a web designer handle just the html file. Because the **tag** is only a HTML comment and is accepted by all web-design tools

Run the sample   Show the sample

Reference to the current file.

If you have separated your application into a .ASPX file and a .HTML file which shares the same name excluding the extension - then you can refer to the include file with a # The syntax is:

```
    <!--#include file="#.htm"  tag="AnyTagInTheIncludeFile"-->
```

Reference to XML / XHTML include files.

When you are dealing with XML and XHTML you might know that "doctype ..." must be the first in the file. There for you can not give this part of the file a tag name. For that purpose you can use the pseudo tag name *FIRST, which includes until the next tag in the include file:

```
    <!--#include file="#.htm"  tag="*FIRST"-->
```

3.4 A simple database application

Programming:

The most powerful feature in IceBreak is the ability to integrate the i5/OS or OS/400 DB-2 database using ILE-programs. This is why IceBreak has such outstandingly fast file access.

Let's take a look at a simple program that displays the first 200 records from a database file.

First, we need to declare the file we are using. In this case it is a product file with some digital cameras. We also need a counter 'i' for the counting record occurrences later.

We use the <% escape sequence to go into RPG-code mode. Declare and switch back to HTML using the %> sequence.

```
<%
f*Filename+IPEASF.....L.....A.Device+.Keywords+++++++++++++++++++++++++++++Comments++++++
++++++
fProduct   IF A E           K disk

d*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments++++++
++++++
d i               s             10U 0
 /free
%>
```

Build a valid HTML-document where the header and the body are written in plain HTML.

```
<html>
<Head>
  <link rel="stylesheet" type="text/css" href="/system/styles/icebreak.css">
</Head>
<body>
 <table border= "1">
  <thead>
   <th>Product ID</th>
   <th>Description</th>
   <th>Manufacturer ID</th>
   <th>Price</th>
   <th>StockCount</th>
   <th>StockDate</th>
  </thead>
```

After the table header we are ready to put in real data from the database. So we use <% escape to go into FREE-RPG, set the file pointer to the first record, and read that. If we have a valid record, then we enter a DO-loop that repeats reading until end-of-file or the number of records is reached.

```
<%
    setll *loval ProductR;
    read ProductR;
    i = 0;
    dow (not % eof(Product) and i < 200) ;
       i = i + 1;
%>
```

Place the record data into the HTML-table row. This is done in plain HTML, but with the **<%=** syntax. If we had some numeric values here we might want to convert them with %char or %edit functions:

```
      <tr>
         <td nowrap ><% = PRODID %></td>
         <td><% = DESC %></td>
         <td nowrap><% = MANUID %></td>
         <td nowrap align="right"><% = %char(PRICE) %></td>
         <td nowrap align="right"><% = %char(STOCKCNT) %></td>
         <td nowrap><% = %char(STOCKDATE) %></td>
        </tr>
<%
      read ProductR;
    EndDo;
%>
```

We finish up by terminating the program so the result can be shown:

```
 </table>
 </body>
</html>
<% return;%>
```

The final program looks like:

```
<html>
<head>
  <link rel="stylesheet" type="text/css" href="/System/Styles/IceBreak.css"/>
</head>
<body>
  <table border="1">
     <thead>
```

```
       <th>Product ID</th>
       <th>Description</th>
       <th>Manufacturer ID</th>
       <th>Price</th>
       <th>Stock Count</th>
       <th>Stock Date</th>
     </thead>
<%
 //' Now reload the list, e.g. Set lower limit with *loval is the first record
 //' then repeat reading until end-of-file or counter exhausted
 //' -------------------------------------------------------------------------
     setll *loval ProductR;
     read ProductR;
     i = 0;
     dow (not %eof(Product) and i < 200) ;
        i = i + 1;
%>
       <tr>
         <td nowrap ><% = PRODID %></td>
         <td><% = DESC %></td>
         <td nowrap><% = MANUID %></td>
         <td nowrap align="right"><% = %char(PRICE) %></td>
         <td nowrap align="right"><% = %char(STOCKCNT) %></td>
         <td nowrap><% = %char(STOCKDATE) %></td>
       </tr>
<%     read ProductR;
     EndDo;
%>
 </table>
</html>
<% return; %>
```

Run the sample   Show the sample

3.5 Database maintenance application

Programming:

Let's go one step further, and expand the simple "List" application so the database can update/delete and write.
This sample combines the ***Form()*** and ***QryStr()*** functions from the first tutorials .

Also we use the "File Field Descriptor" tool which can produce the components of the application we a building.

To simplify the program we have put the logic into separate subroutines:

1.Init:
The initial settings from the HTML document, references to style sheets and j-scripts.

2.Main logic
This part determines the functions based on previous incarnations of the ASPX-page state.

3.Load List
Similar to database application, tutorial 5, this reads the database and builds the html table

4.Edit
This Shows a form with the selected database occurrence, and buttons for update, delete and new

5.Exit

This cleans up, closes, and terminates the program

We also have traditional RPG for file and data declarations F-spec and D-spec etc. The "init" has nearly the same layout as the previous sample, but all initialization should be placed here in the future:

```
<%
//' --------------------------------------------------------------------------------
//' Init; setsup the HTML header and stylesheet / (evt. the script links)
//' --------------------------------------------------------------------------------
Begsr Init;
%>
<html>
 <Head>
  <link rel= "stylesheet"type  ="text/css" href= "/system/styles/icebreak.css">
 </Head >
 <h1>Work with Products</h1>

<%
Endsr;
%>
```

Now to the core applications - the "Main" logic

The ***Form('Option')*** used in the multi way if (select) is the result of any prior buttons "clicks" on the "edit" form

The state can be extracted from that.

The ***QryStr('Key')*** returns the unique key from the Query String in the request object i.e. the URL

```
//' --------------------------------------------------------------------------------
//' The Main logic controlling the state from previous incarnations
//' --------------------------------------------------------------------------------
Begsr Main;

//' Previous function parses update or add parameter
//' ---------------------------------------------
    select;


  //' The "New" button is clicked in the edit form
      when Form('Option') = 'New' ;
         reset ProductR;
         Exsr  Edit;

  //' The "Update" button is clicked in the edit form
      when Form('Option') = 'Update';
         prodKey = FormNum('product.prodKey');
         chain  prodKey ProductR;
         if (%found);
           exsr   Form2db;
           update ProductR;
         else;
           reset ProductR;
           setll *hival ProductR;
           readp ProductR;
           MaxKey = ProdKey + 1;
           exsr   Form2db;
           ProdKey = MaxKey;
           write  ProductR;
         endif;
         Exsr LoadList;
  //' The "Delete" button is clicked - and a prodKey exists; now delete that record
      when Form('Option') = 'Delete';
         prodKey = FormNum('product.prodKey');
         chain prodKey ProductR;
         if (%found);
            delete ProductR;
         endif;
         Exsr LoadList;

  //' The "Return" button is clicked
      when Form('Option') = 'Return';
         Exsr LoadList;

  //' When Clicking on a row in the table the "<a href .." returns the "prodId" along in
the "QryStr"
      when QryStrNum('prodKey') > 0;
         prodKey = QryStrNum('prodKey');
         chain  prodKey ProductR;
         unlock Product;
         Exsr   Edit;

      other;
         Exsr LoadList;
    endsl;

    Exsr Exit;
EndSr;
```

Load list reads all records from the database file and puts them into a HTML table. Real data from the database is ready to be placed under the table header. Use <% escape to go into FREE-RPG and the set the file pointer to the first record and read that. If a valid record is shown, enter a DO-loop that repeats reading until end-of-file or the number of records is reached.

Note: Here a Form is used in conjunction with the "POST" method and a "New" button. This causes the same ASPX page to be redisplayed, but with the "Option" Form-Field set to the value "New". The main logic will respond to that by clearing input fields and running the "Edit" routine for the new record.

The list table is created by the **"Component Wizard (File Field Description tool)"** found on the **administration** menu under "tools" . This produces the HTML structure for any database file:

```
<%
//' --------------------------------------------------------------------------------
//' loadList; is much like a "load subfile". All records are placed into a html table
//' --------------------------------------------------------------------------------
Begsr LoadList;
%>
<form method="POST" id="form1" name="form1">
  <input type="submit"  value="New" name="Option">
</form>

<table class="ListeHead">
 <thead>
   <th></th>
   <th>Product ID</th>
   <th>Description</th>
   <th>Manufacturer ID</th>
   <th>Price</th>
   <th>Stock Count</th>
   <th>Stock Date</th>
 </thead>
<%

 // Now reload the list
 // -------------------
    Count = 0;
    setll *loval ProductR;
    read(n) ProductR;

    dow (not %eof(Product) and Count < 2000) ;
       Count = Count + 1;
%>
       <tr>
          <td><a href="?prodKey=<%= %char(ProdKey) %>"><img src="image/plus2d.jpg"></a>
          <td nowrap><% = PRODID %></td>
          <td><% = DESC %></td>
          <td nowrap><% = MANUID %></td>
          <td nowrap align="right"><% = %editc(PRICE:'J') %></td>
          <td nowrap align="right"><% = %editc(STOCKCNT:'J') %></td>
          <td nowrap><% = %char(STOCKDATE) %></td>
       </tr>
<%
       read(n) ProductR;
    EndDo;
%>
</table>
<%
EndSr;
%>
```

The Edit routine features all the record fields into form fields in detail level.

Note: the Form in conjunction with the Post method causes the ASPX page to be redisplayed, but with the "Form" decorated with all data from this routine. Also, the "Option" contains either "Update" "Delete" or "Return" depending on which button is pressed.

The input form **"Component Wizard (File Field Description tool)"** found on the **administration** menu under "tools" . This produces the HTML structure for any database file:

```
<%
//' --------------------------------------------------------------------------------
//' Edit; Is just showing a form which is being posted back with all input fields
//' Filled
//' --------------------------------------------------------------------------------
Begsr Edit;
%>
<form method="POST" name="form1" id="form1">
 <input type="hidden" name="product.PRODKey"  value ="<% = %char(PRODkey) %>">
 <table>
  <tr>
   <td nowrap>Product ID</td>
   <td><input type="text" name="product.PRODID" size="<%= %char(%size(PRODID))%>"
maxlength="<%= %char(%size(PRODID))%>" value ="<% = PRODID %>"></td>
  </tr>
  <tr>
   <td nowrap>Description</td>
   <td><input type="text" name="product.DESC" size="<%= %char(%size(DESC))%>" maxlength="
<%= %char(%size(DESC))%>" value ="<% = DESC %>"></td>
  </tr>
  <tr>
   <td nowrap>Manufacturer ID</td>
   <td><input type="text" name="product.MANUID" size="<%= %char(%size(MANUID))%>"
maxlength="<%= %char(%size(MANUID))%>" value ="<% = MANUID %>"></td>
  </tr>
  <tr>
   <td nowrap>Price</td>
   <td><input type="text" name="product.PRICE" value ="<% = %trim(%editc(PRICE:'J')) %>
"></td>
  </tr>
  <tr>
   <td nowrap>Stock Count</td>
   <td><input type="text" name="product.STOCKCNT" value ="<% = %trim(%editc
(STOCKCNT:'J')) %>"></td>
  </tr>
  <tr>
   <td nowrap>Stock Date</td>
   <td><input type="text" name="product.STOCKDATE" size="<%= %char(%size(STOCKDATE))%>"
maxlength="<%= %char(%size(STOCKDATE))%>" value ="<% = %char(STOCKDATE) %>"></td>
  </tr>
  <tr>
   <td colspan="2">
      <input type="submit"  value="Update" name="Option">
      <input type="submit"  value="Delete" name="Option">
      <input type="submit"  value="Return" name="Option">
   </td>
  </tr>
 </table>
</form>
<%
EndSr;
%>
```

Finally we finish up by terminating the HTML (table, body and document) and terminate the program so the result can be shown:

```
<%
//' -------------------------------------------------------------------------------
//' Exit Finish up the the complete HTML and quits the program
//' -------------------------------------------------------------------------------
Begsr Exit;
%>
  </body>
</html>
<%
  return;
Endsr;
/end-free%>
```

Run the sample | Show the sample

3.6 Split your program logic and design into separated files

Programming:

In this sample we will split the logic and the design into to files (.ASPX and .HTML). We are using the "Pre-compile time Include with tags" technique in the sample. This will make the ASPX source code much easier to understand and the HTML include file will only hold the presentation.

List program

The following list program “Tutorials/ex19WrkPrd.aspx” is written in RPGLE and it loops through a file called Product. All records are putted into a HTML table described in the included file “Tutorials/ ex19WrkPrd.htm”.

Show the RPGLE source:

Show the sample RPGLE-fixed Show the sample RPGLE /free

So the user interface, the presentation layer is only in this HTML files:

Show HTML presentation layer

Record maintenance program

From the list program you can maintain products by use of maintain program “Tutorials/ex19EdtPrd.aspx” and the included “Tutorials/ex19EdtPrd.htm”.

Show the RPGLE source:

Show the sample RPGLE-fixed Show the sample RPGLE /free

Try the sample

3.7 Writing and reading stream files

Build in functions :

A handy feature in IceBreak is the ability to write and read stream files directly from strings.

What it does

It retrieves a string buffer and writes the contents to a stream file in the IFS according to the server directory. If you need to locate the file in the root of the IFS, then start the file name with an "/". e.g. "/MyFile.text". Optionally, you can convert the data from EBCDIC to ASCII by setting *ON in the conversion option.

Subdirectories must exists but files are created.

Write Stream files from strings

Use the built-in function "putStreamString(...)" to write a text file from a string.

Syntax:

error = putStreamString (FileName:FileOptions: StringtoWrite: xLate);

Field Name	Data type	Description	Default
FileName	VARCHAR(256)	path and name of the stream file to be created/appended to	

FileOptions	VARCHAR(256)	These are the options from the STDIO.H
		w=write, a=append, b=binary, t=text
		codepage=1252 is windows codepage	

StringToWrite	VARCHAR (32768)	Any data to put into the file	
xLate	BOOL	Convert from EBCDIC to ASCII	*OFF

Note: if you need to write new lines use the **x'0D25'** sequence, or <CR><LF> sequence in EBCDIC.

Reading Stream files into strings

Use the built-in function "getStreamString(...)" to read a text file from a string.

Syntax:

error = getStreamString (FileName: String : Offset : MaxLength : xLate);

Field Name	Data type	Description	Default
FileName	VARCHAR(256)	Path and name of the stream file to be read	
String	VARCHAR(1 to 32768)	This is where the file contents is placed. The Contents is truncated if the file contains more that 32768 or **MaxLength** number of bytes.	
Offset	LONGINT	Starting position in the file where 1=the first byte	
MaxLength	LONGINT	Maximum number of bytes to place into the result. no padding is done if the file is shorter. Use **%size(String)-2** since the result is a varying string (-2 to omit the length)	
xLate	BOOL	Convert from ASCII to EBCDIC	*OFF

Here is an example.

The file now contains:
Note: if you need to write new lines use the **x'0D25'** sequence, or <CR><LF> sequence in EBCDIC.

```
<%@ language="RPGLE" %>
<%
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments++++++
++++++
D Error           s               N
D CRLF            s              2A   inz(x'0d25')
D str             s          32760    varying


/free
// This appends a string to a ascii stream file - converting the input text string to
ascii

   Error = PutStreamString(
             'test.txt':                                   // The output file name
             'ab,codepage=1252':                           // w:write a:append, b:
binary codepage=1252 is windows
```

```
             'My log file at: ' + %char(%timestamp) + CRLF:   // The text string to write
             *ON);                                            // Conversion from EBCDIC
to ASCII

// Now read the complete test log file back into "str" - converting the input file to
EBCDIC
   Error = GetStreamString(
             'test.txt':                                      // The input file name
             str:                                             // where to place the
contents
             1:                                               // Starting position in the
file; 1=first byte
             %size(str)-2:                                    // Maximum number of bytes
to read
             *ON);                                            // Conversion from ASCII to
EBCDIC
%>The file now contains: <pre><%= str %><pre><%
   return;
%>
```

Run the sample Show the sample

3.8 AJAX Async JavaSript and XML

Programming:

When you want to create application based on components - AJAX is an excellent choice.

AJAX lets you handle "onClick" round-trips to the server, so you don't need to reload a complete page - but rather load fragments.

IceBreak has a small, yet powerful implementation of AJAX, which allows you to replace any DIV / SPAN / P tag with data of your choice - on the fly

Look at the following static HTML page, and see how we can make it alive with an AJAX call

```
<%@ language="RPGLE" %>
<html>
<head>
 <link rel="stylesheet" type="text/css" href="/System/Styles/IceBreak.css"/>
 <script language="JavaScript1.2" src="/System/Scripts/ajax.js"></script>
</head>
<body>
  <input type=button value='Click here to run theAjaxrequest' onclick="adAjaxCall
('myResult','ex22ajax.aspx?manuid=SONY');" />
  <div id=myResult></div>
</body>
</html>
```

- The first AJAX relatet line is 5: This refer to the script containing all the AJAX implementation. AJAX is always a client technology

- Next is line 8: "adAjaxCall" is the round trip to the server. The first parameter is WHERE you want to place data. The second parameter is HOW you want it. This is an URL to a resource - in this case an ASPX program that creates a table ... more on that later.

- Last - line 9: A <div> tag where I want to put my data when the call is complete. The "id" attribute is the key here.

The next step is to make the server side application. It only have to return the table element - not a complete html document. Otherwise it is pretty straight forward - making a table based on a result set:

```
<%
F*Filename+IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++++++++Comments++++++
++++++
Fproduct1  if   e           k disk

d pmanuid         s                   like(manuid)

/free
*inlr = *on;
pmanuid = qrystr('manuid');
%>
<table>
<%
  chain pmanuid productr;
  dow not %eof(product1) and manuid = pmanuid;
%><tr>
    <td><% = prodid %></td>
    <td><% = desc  %></td>
  </tr>
<%  read productr;
  enddo;
%>
</table>
```

As you can see it only create the "raw" table. It have no idea in what context it is used - it just create the list/table. The formatting is provided by the client document we just saw before.

It looks easy - and it is. The only problem with AJAX in the real world is:

- Debugging: It might sometimes be difficult to find the right component doing what.
- Caching: The browser might in some cases be confused about resources in the inner HTML and will therefore not cache it.
- JavaScript might not be available in the target browser.

Run the sample

Now lets change the first HTML to an ASPX with a drop-down list so you can feel the real power of AJAX:

```
<%
F*Filename+IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++++++++Comments++++++
++++++
FMANUFACT  IF   E           K DISK
/free
*inlr = *on;
%>
<html>
<head>
<%//' For using Ajax - just include the Ajax script in the system folder
%>
```

```
 <script language="JavaScript1.2" src="/System/Scripts/ajax.js"></script>
 <link rel="stylesheet" type="text/css" href="/System/Styles/IceBreak.css"/>
</head>

<body>
  <form>
   Select a manufacturer:

<%
//' The adAjaxcall takes two parameters:
//' 1) the id name for an div or paragraph to hold the result. In this case a table
//' 2) The url to the APS program to produce the result. in this case we build up the
QueryString with parameters - the key to our database lookup
%>

   <select name="manufacturer" onchange="adAjaxCall('productsDiv','ex22ajax.aspx?manuid='
+ this.value)">

<%
  read manufactr;
  dow not %eof(manufact);
%>  <option value="<% = manuid %>"><% = Desc %></option>
<%
    read manufactr;
  enddo;
%>
  </select>
  <div id="productsDiv"><b>Product info will be listed here.</b></div>
 </form>
</body>
</html>
```

Run the sample

3.9 Creating XML files Dynamically

Programming:

Until now we have only studied IceBreak ASPX creating HTML files dynamically. In other cases you might need to produce XML files dynamically, which can be done very easily with the IceBreak ASPX.

We learned in "Tutorial 4" how to include Word or Excel sheets dynamically. You will now expand that knowledge and let IceBreak ASPX create the XML contents. Again we use a double suffix. IceBreak detects the first suffix, and the browser detects only the last.

The resulting file name might be: a

MyApp.aspx.XML

The contents type, should be set for the browser to open the resulting file. Otherwise it will ask you to download the file (which may be the purpose in another case).

The following case creates a simple XML file for a Web shop.

Step 1 - The legacy code that reads the data

First lets take a look at an classic program that read all manufactures of digital cameras and produces a list of all available cameras for each manufacturer:

```
<%@ language="RPGLE" %>
<%
 *' -------------------------------------------------------------------------- *

 *' Program ex07XmlA.asp

 *'

 *' Reads all manufacturers of digital cameraes and coresponding products

 *'

 *' -------------------------------------------------------------------------- *

F*Filename+IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++++++++Comments++++++
++++++
FMANUFACT  IF   E           K DISK    prefix('MANU_')

FPRODUCT1  IF   E           K DISK    prefix('PROD_')

 *' Read manufacturer

C     *LOVAL        SETLL     MANUFACTR

C                   READ      MANUFACTR                              8080

C     *IN80         DOWEQ     *OFF

 *' Read all products for that manufacturer

C     MANU_MANUID   CHAIN     PRODUCTR                             80

C     *IN80         DOWEQ     *OFF

C     MANU_MANUID   READE     PRODUCTR                               8080

C                   ENDDO

C                   READ      MANUFACTR                              8080
C                   ENDDO
C                   SETON                                          LR
%>
```

Step 2 - Building the XML

Now we let the ASPX features extend the code to produce the XML data.

First we need to tell the browser that we are dealing with XML. That is done with the SetContentType which takes a MIME type as parameter ... this is done in FREE-RPG mode.

```
/free
 SetContentType('application/xml; charset=utf-8');
/end-free
```

Second we need a XML header and a root tag that wraps all the XML stuff:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<ProductList>
```

Then we need a Manufacture tag with the following data from the file:

```
  <Manufacture  ManufactureID="<%= Manu_ManuId %>"
                Description="<%= Manu_Desc %>"
                LogoURL="<%= Manu_LogoURL %> "/>
```

And the the Product tag with the following data from the file:

```
  <Product  ProductID="<%= Prod_ProdId %>"
              Description="<%= Prod_Desc %>"
              Price="<%= %char(Prod_Price) %>"
              Stock="<%= %char(Prod_StockCnt) %>" />
```

The program logic must insert the end tags of cause. Now The Program looks like:

```
<%
 *' --------------------------------------------------------------------------- *

 *' Program ex07XmlA.asp

 *'

 *' Reads all manufacturers of digital cameraes and coresponding products

 *'

 *' --------------------------------------------------------------------------- *

F*Filename+IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++++++++Comments++++++
++++++
FMANUFACT  IF   E           K DISK    prefix('MANU_')

FPRODUCT1  IF   E           K DISK    prefix('PROD_')


 *' This program deals with XML. UTF-8 is the prefered charset for that
/free
SetContentType('application/xml; charset=utf-8');
/end-free

 *' XML data is mostly in UTF-8 format

%><?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<ProductList>
<%
 *' Read manufacturer

C     *LOVAL        SETLL     MANUFACTR

C                   READ      MANUFACTR                            8080

C     *IN80         DOWEQ     *OFF

 *' Now put the manufacturer informations into the response object
%>
  <Manufacture  ManufactureID="<%= Manu_ManuId %>"
                Description="<%= Manu_Desc %>"
                LogoURL="<%= Manu_LogoURL %> ">
<%
 *' Read all products for that manufacturer

C     MANU_MANUID   CHAIN     PRODUCTR                           80
```

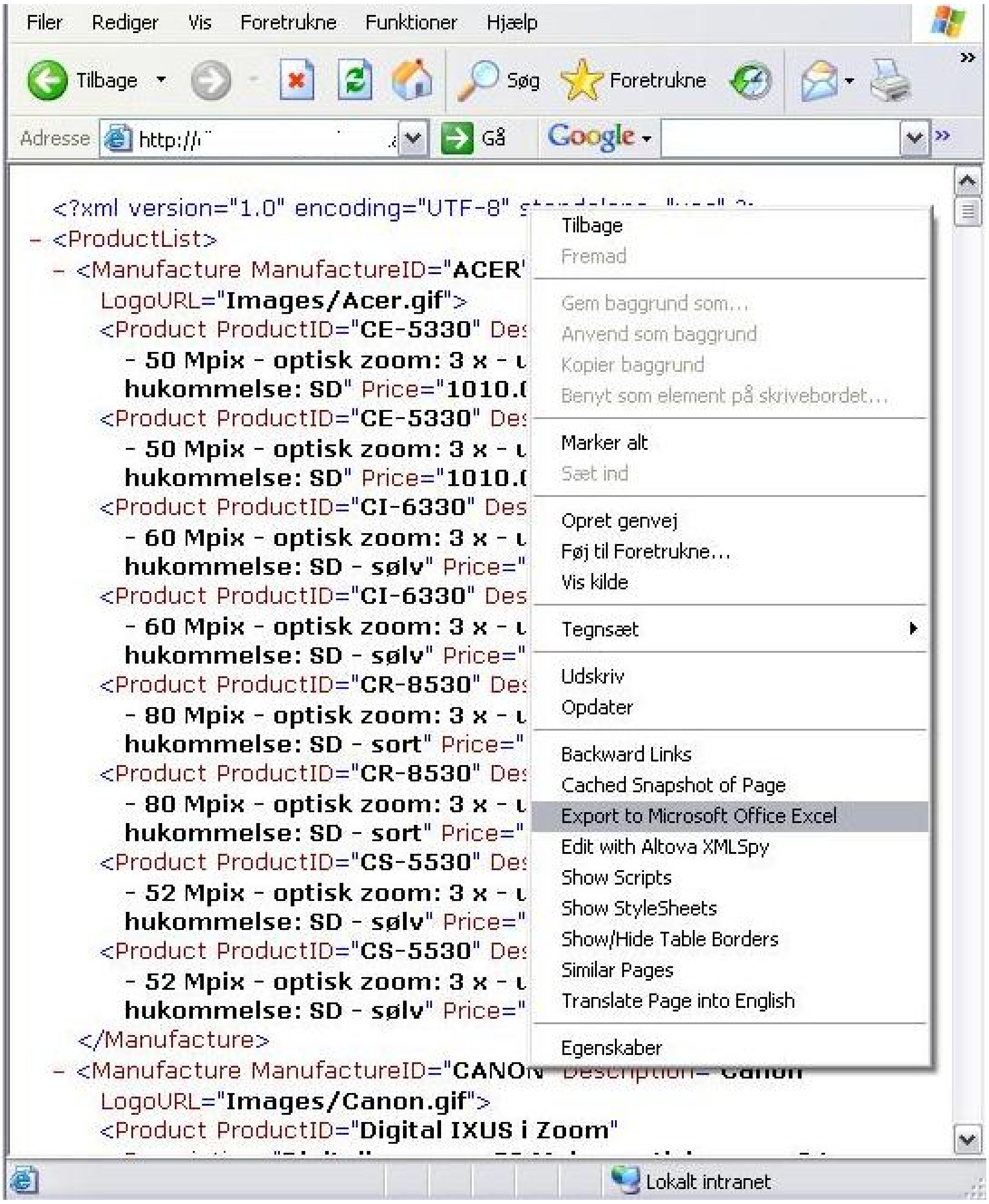
```
C     *IN80         DOWEQ     *OFF

 *' Now put the product informations into the response object
%>
    <Product  ProductID="<%= Prod_ProdId %>"
              Description="<%= Prod_Desc %>"
              Price="<%= %char(Prod_Price) %>"
              Stock="<%= %char(Prod_StockCnt) %>" />
<%
C     MANU_MANUID   READE     PRODUCTR                             8080
C                   ENDDO

%>
  </Manufacture>
<%
C                   READ      MANUFACTR                            8080
C                   ENDDO
C                   SETON                                        LR
%>
</ProductList>
```

Thats all you need. If you run this sample you browser will format the data as a XML document. You can collapse and expand each node in the XML-tree

Run the sample   Show the sample

Step 3 - Using the XML data

If you have installed Microsoft Office 2003 or greater - then you can use the XML data right away.
In the browser window right-click on the mouse and select "Export to Microsoft Office Excel". You will be prompted for where to put the data into sheet. And last - you go a Pivot table.



```
  <?xml version="1.0" encoding="UTF-8"
- <ProductList>
  - <Manufacture ManufactureID="ACER"
      LogoURL="Images/Acer.gif">
      <Product ProductID="CE-5330" De
        - 50 Mpix - optisk zoom: 3 x - u
        hukommelse: SD" Price="1010.(
      <Product ProductID="CE-5330" De
        - 50 Mpix - optisk zoom: 3 x - u
        hukommelse: SD" Price="1010.(
      <Product ProductID="CI-6330" Des
        - 60 Mpix - optisk zoom: 3 x - u
        hukommelse: SD - sølv" Price="
      <Product ProductID="CI-6330" Des
        - 60 Mpix - optisk zoom: 3 x - u
        hukommelse: SD - sølv" Price="
      <Product ProductID="CR-8530" De
        - 80 Mpix - optisk zoom: 3 x - u
        hukommelse: SD - sort" Price="
      <Product ProductID="CR-8530" De
        - 80 Mpix - optisk zoom: 3 x - u
        hukommelse: SD - sort" Price="
      <Product ProductID="CS-5530" De
        - 52 Mpix - optisk zoom: 3 x - u
        hukommelse: SD - sølv" Price="
      <Product ProductID="CS-5530" De
        - 52 Mpix - optisk zoom: 3 x - u
        hukommelse: SD - sølv" Price="
    </Manufacture>
  - <Manufacture ManufactureID="CANON" Description="Canon"
      LogoURL="Images/Canon.gif">
      <Product ProductID="Digital IXUS i Zoom"
```

It's really useful!!!

Step 4 - Formating the XML data

Browsers are able to reformat the XML data to be more readable to humans. It uses a transformation style sheet, a XSL file.
We have made a small XSL file called **Product.XSL** which reformats the XML you just created.

Now you have to refer to the transformation style sheet from within the XML file before the transformation occurs. It is done in the XML header with

```
   <?xml-stylesheet type="text/xsl" href="Product.xsl"?>
```

The final ASPX program now looks like:

```
<%
 *' --------------------------------------------------------------------------- *

 *' Program ex07XmlB.asp

 *'

 *' Reads all manufacturers of digital cameraes and coresponding products

 *'

 *' --------------------------------------------------------------------------- *

F*Filename+IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++++++++Comments++++++
++++++
FMANUFACT  IF   E           K DISK    prefix('MANU_')

FPRODUCT1  IF   E           K DISK    prefix('PROD_')


 *' This program deals with XML. UTF-8 is the prefered charset for that
/free
SetContentType('application/xml; charset=utf-8');
/end-free
 *' Here we use the transformations stylesheet "Product.xsl"

%><?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<?xml-stylesheet type="text/xsl" href="Product.xsl"?>
<ProductList>
<%
 *' Read manufacturer

C     *LOVAL        SETLL     MANUFACTR

C                   READ      MANUFACTR                            8080

C     *IN80         DOWEQ     *OFF

 *' Now put the manufacturer informations into the response object
%>
  <Manufacture  ManufactureID="<%= Manu_ManuId %>"
                Description="<%= Manu_Desc %>"
                LogoURL="<%= Manu_LogoURL %> ">
<%
 *' Read all products for that manufacturer

C     MANU_MANUID   CHAIN     PRODUCTR                           80

C     *IN80         DOWEQ     *OFF

 *' Now put the product informations into the response object
%>
    <Product  ProductID="<%= Prod_ProdId %>"
```

```
              Description="<%= Prod_Desc %>"
              Price="<%= %char(Prod_Price) %>"
              Stock="<%= %char(Prod_StockCnt) %>" />
<%
C     MANU_MANUID   READE     PRODUCTR                              8080
C                   ENDDO

%>
  </Manufacture>
<%
C                   READ      MANUFACTR                             8080
C                   ENDDO
C                   SETON                                         LR
%>
</ProductList>
```

When you try to run the program your browser will reformat the XML document into a nice HTML document. The point here is - that is it the same data provided for both the human and for the application.

Run the final sample

3.10 Inter process communication

Programming:

If you want to send messages across the internet i.e. if you are building a chat-room you will need a method to communicate between processes - hence the name inter process communication. If you are familiar with data queues on the IBMi you will be happy to know that you can use data queues right out of the box in IceBreak. Data queues will not interrupt the normal HTTP request/response flow but rather run in a separated thread in the IceBreak core.

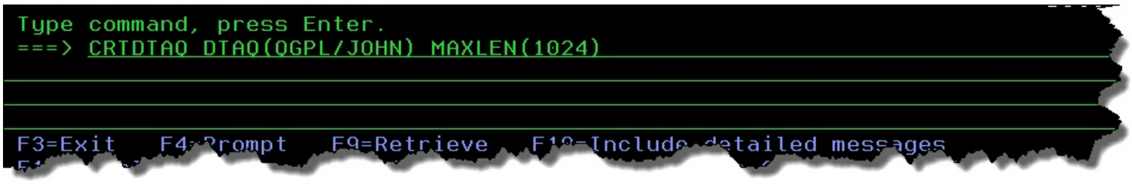
Sending and receiving data to a data queue from the browser:

In the "system" hive there is two IceBreak build in functions that sends data to the data queue and receives data from the data queue. Normally you will wrap these functions in AJAX calls from a web framework like ExtJs, DOJO or jQuery. However, you can try the functionality directly from the browser URL

First let us create a queue to work with. On the command line - enter:

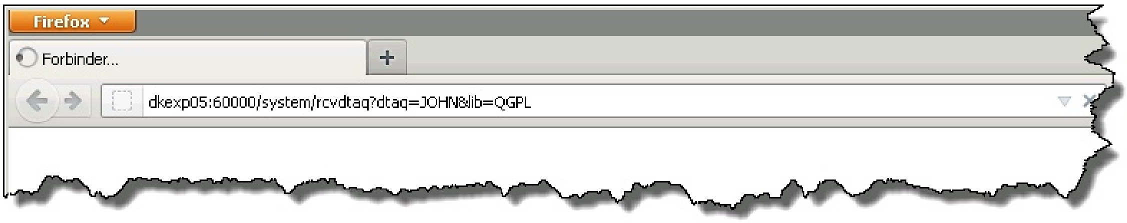
CRTDTAQ DTAQ(QGPL/JOHN) MAXLEN(1024)

Like this:



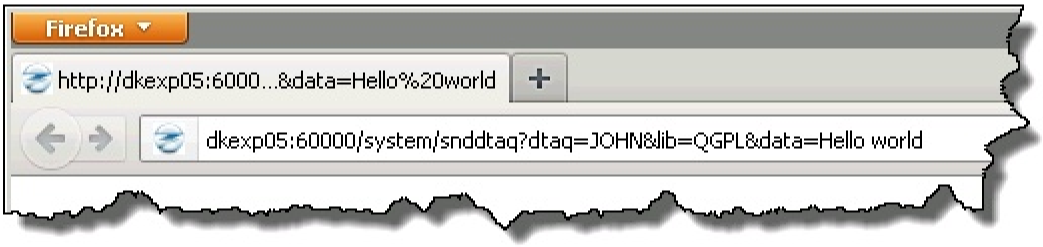
```
Type command, press Enter.
===> CRTDTAQ DTAQ(QGPL/JOHN) MAXLEN(1024)



F3=Exit   F4=Prompt   F9=Retrieve   F10=Include detailed messages
```

Now start a browser and let it wait for an event on that queue. This is done by the **/system/rcvdtaq** command. Notice it will just "spin" the waiting symbol until data arrives in the data queue, and notice you'll need to write the data queue name and library in uppercase:



dkexp05:60000/system/rcvdtaq?dtaq=JOHN&lib=QGPL

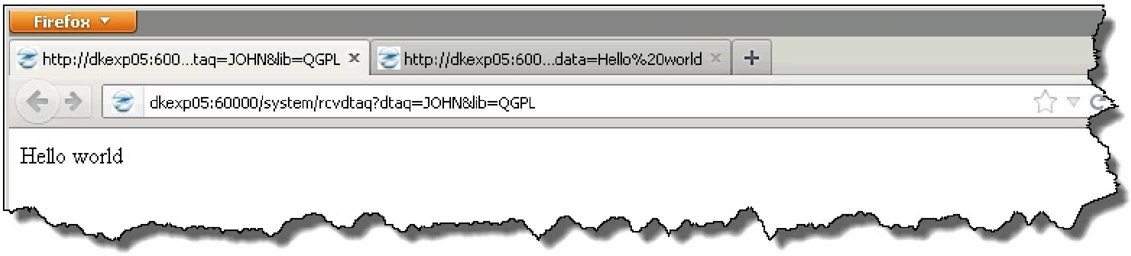
Next: Let's send the text "Hello world" into the messages queue **JOHN** in **QGPL**. This is done by the **/system/snddtaq** command



http://dkexp05:6000...&data=Hello%20world

dkexp05:60000/system/snddtaq?dtaq=JOHN&lib=QGPL&data=Hello world

Now: As soon you press the Enter and send the request, immediately after the waiting process will wake up and show the text "Hello world"



http://dkexp05:600...taq=JOHN&lib=QGPL

http://dkexp05:600...data=Hello%20world

dkexp05:60000/system/rcvdtaq?dtaq=JOHN&lib=QGPL

Hello world

You can control the following parameters:

Parameter	Description
dtaq	Name of the data queue. This values has to be in uppercase
lib	The name of the library containing the dataqueue. This values must be in uppercase. The special values *LIBL and *CURLIB can be used
data	String data representation of the data **snddtaq** will place in the data queue. This will be converted between UTF-8 and EBCDIC according to the current CCSID of the server job
timeout	Number of seconds **rcvdtaq** will wait before returning an empty response. Don't set this value longer than the AJAX call timeout value, it might fill up you communication stack
key	If a keyed dataqueue, this is the char presentation of the key
keylen	number of bytes in the total keylen (the key will be padded with blanks to fit this length). Defaults to the string length of the key

keytype	The type of entry to wait for: EQ, GE, LE, GT, GT. Defaults to EQ - has to be in uppercase

Normally you will place the **rcvdtaq** and **snddtaq** in some AJAX call. You will notice that no application server job is started. The dataqueues are handled by the multithreaded core, and will not interrupt your normal program call flow.

Also notice. It does not matter which server port you are sending or receiving at - all queues are available on all server instances despite the server port number. Just notice that if the server requires you to logon - then you can't send or receive on data queues before the logon process is completed normally.

3.11 Uploading files to a IceBreak server

Programming:

In this tutorial we will create a simple IceBreak program which uploads files - either to the IFS or to an internal large object (ILOB). Basically this program is just a simple HTML form. The magic however is in the combination of the form encoding type set to multipart/form-data and the special input fields of type "file".

Configure the allowed destination:

When you upload files they will be placed with a temporary name in the /tmp folder on the IFS - from here it is your responsibility to move them into your designated folder. You can, however restrict/allow the client to upload into one or more dedicated folders. You simply place a configuration file in Your server root path called webConfig.xml

The configuration file has a upload element which again can contain a map of valid directories an the corresponding alias as the client refer to it. Finally You can give a generic folder path for any invalid upload filenames. This is indicated by an * as the alias name. By default that would be the /tmp folder on the IFS.

The webConfig.xml file is hidden from the client. It is only available in the physical path, not in the virtual counterpart.

The webConfig.xml contains (among other stuff) the following.

```xml
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <upload>
    <map alias="frameworks" path="frameworks" />
    <map alias="upload" path="/www/anyplace/docs" />
    <map alias="*" path="/tmp" />
  </upload>
</configuration>
```

The configuration above says:

- Any uploaded files designated to virtual folder "frameworks" will be placed in the sub-folder "framework" in the server root path.

- Any uploaded files designated to virtual folder "upload" will be placed in the absolute path "/www/anyplace/docs"
- The final "*" alias is a "catch all" - so all other virtual folder references will be placed into the "/tmp" folder

The code:

Next we will take a look at the file type, how it is used to upload files to your IceBreak server and how you can control the uploaded file. The combination of the form encoding type set to multipart/form-data and the special input fields of type "file" makes it work.

Two components are needed, <Form> and <Input> tags:

<form>

```
   <form METHOD="post" ENCTYPE="multipart/form-data" ACTION="Upload.aspx" id=form1 name
="form1" accept-charset="utf-8">
```

The <form> tag type has a special feature called "ENCTYPE". This is the encoding type for file-uploading to "multipart/form-data". It informs the browser to include all form elements like attachments in an email instead of passing the data in the form object.

Also notice that IceBreak only supports the UTF-8 charset when you have inputs fields on you upload form. You always need to specify accept-charset="utf-8" if you also submits input fields. Input fields can be retrieved by the **form()** api

<input> for IFS stream files

```
   <input NAME="upload/MyUpload1.txt" TYPE= "file">
```

The NAME paramter must contain a valid virtual path and file. the path must be a name found in the webConfig.xml file in the upload element. The name must conform to any valid file name for the IFS.

The name can be set to the special value "*" if you want IceBreak to create a unique name and place the file into the /tmp IFS folder.

<input> for ILOB's (Internal Large Objects)

```
   <input NAME="*ILOB:MyUpload" TYPE= "file">
```

The <input> tag type is set to "file". This informs the browser to attach a file in form attachment. The browser automatically places a Browse-button next to the file name.

The "NAME" is the destination name of an ILOB, but you have to prefix it by *ILOB:

Now you have some objects in the resulting ASPX request object which you can retrieve by the Form('') or FormNum('') functions.

Field Name	Description
FormNum('file.count')	Returns the number of attachments in the file upload
Form('file.n.uploadname')	Returns the intended name from the form "name" - field
Form('file.n.localname')	Returns the name of the uploaded filename on the IFS
Form('file.n.remotename')	Returns the name of the selected filename on the client
form('file.n.size')	Returns the size of the file in bytes

Where 'n' is the file number in the attachment between 1 and "FormNum('file.count')"

The final sample:

```
    <%
       //' I use the "var" icebreak macro to declare my variables- they will expand to
D-specs
       var i            like(int)
       var FilesOnForm like(int)
       var oldName      like(String)
       var newName      like(String)

    /include qasphdr,ifs
    /free
    %>
    <html>
     <head>
      <link rel="stylesheet" type="text/css" href="/System/Styles/IceBreak.css"/>
     </head>
    <body>
    <h1>Files uploads on the form was: <%= form('file.count') %></h1>.
    <br>
    <%
       //' the first uses the default file name and will be placed temporary in
/tmp/xxxxx.upload
       //' the second uses the "UPLOAD" which is mapped to location: /www/anyplace/docs
on the IFS via the webConfig.xml file placed in the server root path
       //' the last is trying to hack into a invalid path, so it will fallback to the
default /tmp path
    %>
    <form method="post" enctype="multipart/form-data" action="Upload.aspx" accept-charset
="utf-8">
       <br/>File to upload :<INPUT NAME="*" TYPE="file" size=40>
       <br/>File to upload :<INPUT NAME="UPLOADa/MyUpload2.dat" TYPE="file" size=40>
       <br/>File to upload :<INPUT NAME="/this is an invalid path/MyUpload3.dat" TYPE
="file" size=40>
       <br/>Aditional text :<INPUT NAME="text1" TYPE="text" size=40>
       <br/><input TYPE="submit" VALUE="Upload">
    </form>
    <h2><% = Form('text1') %></h2>
    <table>
    <%
    //'The table header
    %>
    <tr>
      <td>Filenumber</td>
      <td>Name on the upload form</td>
      <td>Uploaded to</td>
      <td>Uploaded from</td>
      <td>Size in Bytes</td>  </tr>
    <%
       //' Now build a simple table with info about the uploaded files
       //' we iterate through each file-field on the request form
       FilesOnForm = FormNum('file.count');
       for i = 1 to FilesOnForm;
          %><tr>
              <td><%= %char(i)%> </td>
              <td><%= form('file.' + %char(i) + '.uploadname') %></td>
              <td><%= form('file.' + %char(i) + '.localname') %></td>
              <td><%= form('file.' + %char(i) + '.remotename') %></td>
              <td><%= form('file.' + %char(i) + '.size') %></td>
            </tr><%
       endfor;
    %></table>
     </BODY>
    </html>

    <%
       //' Finally - if we use the method with temporary filename
       //' we rename and move the temporary file to the decided finally location
       //' Note we include the prototype IFS from qasphdr to utilize the IFS api "rename"
       if form('file.1.remotename') > '';
          oldName = form('file.1.localname');                         // This is the
teporary name
          newName = '/www/anyPlace/docs/upload1.dat';  // this is the resulting name
          if rename( oldName :newName) = 0;
            %>File was moved from: <% = oldName %> to: <% = newName %><%
          else;
            %>Not able to moved from: <% = oldName %> to: <% = newName %><%
          endIf;
       endIf;
       return;
    %>
```

3.12 Controlling the state of a "CheckBox"

Programming:

Check boxes are used for logical states ON/OFF .. TRUE/FALSE. This is done by the "checked" attribute in the HTML.

The following code builds a HTML-form containing a check box

```
<form name="form1">
  <input type="checkbox" value="checked" name="IsItOk" checked>
</form>
```

By using the value "checked" it is very easy to transfer the logical state from the ASPX:

```
<form name="form1">
  <input type="checkbox" value="checked" name="IsItOk" <%= form('IsItOk') %>>
</form>
```

RPG has logical variables and indicators which can be used for direct control of these check boxes.

```
<%
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments++++++
++++++
D IsItOk          s               N

/free
 IsItOk = form('IsItOk') > *blanks;

 if (IsItOk);
 %>This Checkbox is Checked<%
 else;
 %>This Checkbox is not Checked<%
 endif;
%>
```

When the value is retrieved from the checkbox, it will contain "checked" or blank. Now take the check state into logical variables and control the check box.

Run the sample   Show the sample

3.13 Mixing Jave-Script and RPG

Programing:

With HTML you are able to use scripts in your ASPX code in IceBreak. This can be data from a DB2 database which contain the options in a menu structure. Here is a small sample.

How to put the system time into an "alert" popup message:

This is done by the following code:

```
<%/free
  *inlr = *ON;
%>
<html>
 <h1>Script sample</h1>
 <script>
   alert(' <%= %char(%time) %> ');
 </script>
</html>
```

When the value is retrieved from the checkbox, it will contain "checked" or blank. Now take the check state into logical variables and control the check box.

Script sample

It does not look like much, but these few lines do quite a lot of work:

- When the ASPX program is executed the %time() value is placed into the script
- The total HTML file is the sent to the browser
- The browser displays the HTML "Script sample"
- Then it interprets the script showing the alert prompt with the time

Run the sample   Show the sample

Tip: When you want to include JavaScript into your project you can find free snippets and complete components on the Internet.

The following example shows how to build a menu. The first part only declares some of the RPG variables.

```
<%
 D*Name+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments+++++
++++++
 D url              s            256    varying
/free
%>
```

Now we make reference to style sheets and JavaScripts.

```
<html>
 <head>
  <link rel="stylesheet" type="text/css" href="/System/Scripts/iNavigate/style/iNavigate.
css">
  <script language="JavaScript1.2" src="/System/Scripts/iNavigate/jscript/iNavigate.js">
  </script>
 </head>
```

The following code makes up the menu items in Java Script

```
<script>
function menu() {
  MenuBegin("Icebreak Introduction");
  AddSubMenu("What is it?" , "navigator.asp?func=Default.htm");
  AddSubMenu("Creating programs" , "navigator.asp?func=CreatePgm.htm");
  AddSubMenu("Into details" , "navigator.asp?func=intro3.htm");
  AddSubMenu("Tutorial" , "navigator.asp?func=tutorial.htm");
  AddSubMenu("System &amp; Metode" , "navigator.asp?func=http://www.System-metode.dk");
  AddSubMenu("Agent Data " , "navigator.asp?func=http://www.agentdata.com");
  MenuEnd();

  MenuBegin("Administration");
  AddSubMenu("Working with servers" , "navigator.asp?func=SvcMng01.asp");
  AddSubMenu("System settings" , "navigator.asp?func=Default.htm");
  AddSubMenu("User profile" , "navigator.asp?func=Default.htm");
  MenuEnd();

  MenuBegin("Back office");
  AddSubMenu("Work with spool Files" , "navigator.asp?func=Default.htm");
  MenuEnd();

  MenuBegin("Email &amp; Faxing");
  AddSubMenu("Send Fax/Mail" , "navigator.asp?func=Default.htm");
  AddSubMenu("Work with jobs" , "navigator.asp?func=Default.htm");
  AddSubMenu("Work with receipients" , "navigator.asp?func=Default.htm");
  MenuEnd();

  iNavigate_SetParentPage('navigator.asp' , '?func=<%=%trim(url)%>');
  iNavigate_BeforeLoad();
}
</script>
```

The complete page is built by a table with the Java script menu on the left side and the contents on the right made by an IFRAME (Inline frame).

```
<html>
<body topmargin="0" leftmargin="0" marginwidth="0" marginheight="0" onload
="iNavigate_AfterLoad();">
<%
  url = QryStr('func');
  if url = *blanks;
     url = 'default.htm';
  endif;
%>

<table border="0" cellpadding="0" cellspacing="0" height="100%">
  <tr>
    <td width="10"><img src="/iNavigate/graphics/blank.gif" width="7" height="1"></td>
    <td valign="top" width="200">
      <script language="JavaScript1.2">
        menu();
      </script>
    </td>

    <!--- VERTICAL LINE SEPERATOR  -->
    <td width="10"><img src="/iNavigate/graphics/blank.gif" width="7" height="1"></td>
    <td width="1" bgcolor="#808080"><img src="/iNavigate/graphics/blank.gif" width="1"
height="1"></td>
    <td width="10"><img src="/iNavigate/graphics/blank.gif" width="7" height="1"></td>


    <!--- CONTENT START --->
    <td valign="top" width="100%">
      <iframe src="<%=url%>" width="100%" frameborder="0" marginwidth="0" height="100%"
marginheight="0" align="top" scrolling="auto" hspace="0" vspace="0"></iframe>
    </td>
    <!--- CONTENT END --->

  </tr>
</table>

</body>
</html>

<%return;%>
```

You can take this navigator menu and load the item from a DB2 database to make a user dependent menu.

3.14 VB-Scripting. Merge i5/OS data into a MS-Word document

Programming:

The power of building script using an IceBreak ASPX-program is relay amazing. Here we will start MS-word as an Active-X component and then build a document based on data from an DB2-table.

The first step is to bring up MS-word and just fill in some data:

```
<html>
<head>
```

```
</head>
  <script language="vbscript">
    sub OpnDoc()

    ' Create Word object
      Dim oWd, doc
      Dim Crlf
      crlf = chr(13) + chr(10)
      On Error Resume Next
      Set oWd = GetObject(, "Word.Application")
      If TypeName(oWd) <> "Application" Then
        Set oWd = CreateObject("Word.Application")
      End If

      oWd.Visible = false
      oWd.Documents.Add ,,0
      oWd.Activate
      oWd.Selection.TypeParagraph
      oWd.Selection.TypeText "Customer name" + CrLf
      oWd.Selection.TypeText "Address way 1" + CrLf
      oWd.Selection.TypeText "US 888 555 Anyplace" + CrLf
      oWd.Selection.TypeText "Thecountryname" + CrLf
      oWd.Selection.TypeParagraph
      oWd.Selection.TypeText "This is the text we want to write to the customer"
      oWd.Visible = True

   end sub
 </script>
<body>
<input type="button" onclick="OpnDoc" value="Word">
</body>
</html>
```

When You click on the "Word" button the VB-Script is started. The construction getObject followed by CreateObject is used to reuse word is it is already started. The hole magic is to use the "TypeText" method on the word object:

Run the sample Show the sample

Next step is to bring live data into the document with an IceBreak ASPX Program. Now we can iterate through a database table and build the script dynamically.

```
<%
 *' --------------------------------------------------------------------------- *

 *' Program ex17Word2.asp

 *'

 *' Reads all manufacturers of digital cameraes and coresponding products

 *'
 *' This demonstrates the interaction with Microsoft Word scriptiong capabilities
 *' You need to allow "active-X" to run from your icebreak server - otherwise it will not
work
 *'
 *' --------------------------------------------------------------------------- *

F*Filename+IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++++++++Comments++++++
++++++
FMANUFACT  IF   E           K DISK    prefix('MANU_')

FPRODUCT1  IF   E           K DISK    prefix('PROD_')

//'
------------------------------------------------------------------------------------------
-----
%>
<html>
<head>
 <script language="vbscript">
   sub OpnDoc()
      Dim oWd, doc
      Dim Crlf
      Dim lf
      crlf = chr(13) + chr(10)
      lf = chr(10)

       ' Create Word object - if it exists the reuse it
      On Error Resume Next
      Set oWd = GetObject(, "Word.Application")
      If TypeName(oWd) <> "Application" Then
        Set oWd = CreateObject("Word.Application")
      End If

    ' Use a preformated document with logos etc as template - we load it from the same
server as we get this request
      oWd.Visible = false
      oWd.Documents.Add "http://<%= getHeader('host')%>/tutorials/template.doc", False, 0
      oWd.Activate

<%
 *' Read manufacturer

C     *LOVAL        SETLL     MANUFACTR

C                   READ      MANUFACTR                            8080

C     *IN80         DOWEQ     *OFF

%>
    ' Now list the manufacturer
      oWd.Selection.TypeParagraph
      oWd.Selection.TypeText "<%
      /free
      ResponseWrite(%trim(Manu_Desc));
      /end-free
      %>" + Lf
      oWd.Selection.TypeText
"------------------------------------------------------------" + CrLf
<%
 *' Read all products for that manufacturer

C     MANU_MANUID   CHAIN     PRODUCTR                           80

C     *IN80         DOWEQ     *OFF

C     MANU_MANUID   READE     PRODUCTR                             8080

%>
      oWd.Selection.TypeText "<%
      /free
      ResponseWrite(%trim(Prod_Desc));
      /end-free
      %>" + CrLf
<%
C                   ENDDO
%>
```

```
      oWd.Selection.InsertBreak wdPageBreak
<%
C                   READ      MANUFACTR                            8080
C                   ENDDO
C                   SETON                                          LR

%>
      oWd.Visible = True

   end sub
</script>
</head>
<body>
<p>The report is ready press OK to open it in word</p>
<input type="button" value="ok" onclick="OpnDoc()">
</body>
</html>
```

As you can see - the structure is the same, but the RPG-logic i reading the entire file and inserting data in the document with "TypeText" method.

Hint: Use word macro recording tool to show you all you need to do to produce, print and save (etc.) a document.

Hint2: You can use the same technique to produce Excel spreadsheets.

Run the sample Show the sample

.

3.15 Using keyCodes (SPAM Prevention)

Programming:

Use "KeyCodes" to prevent a spammer from creating unwanted accounts from your web-application.

What it does

It prompts the user for a sequence of random numbers presented in a graphic format which is not easy for OCR-software to recognize. Also the information behind "KeyCodes" can not be exposed neither the "query string" or the "form" object.

How it is done

The key-code is generated as a random number using a timestamp as the base. Here we use four digits.

```
//' Create a new key code:
//' Get a pseudo random number .. that is the current timestamp sec cat milli sec part
//' Time is in format: 2005-05-05-12.34.56.123456
    rand =  Num(%subst(%char(%timestamp()):21:2)
         +      %subst(%char(%timestamp()):18:2));
    rand = %rem(rand : 10000);                    //' Limit it to between 0 and 9999
    KeyCode = %subst(%char(rand + 10000) : 2: 4);  //' convert it to 4-char string with
prefix zeros
    SesSetVar('KeyCode' : KeyCode);                //' Store it into session
```

A .GIF file for each digit is created with an almost unreadable font i.e. the "Curlz" font. Each digit from the keyCode is returned to the response object not exposing the original name, as .GIF image data.

Here we use an ASPX that includes the .GIF data, and sets the contents type to "image/gif". Remember to set the cache time out to zero, otherwise the browser will just get the first image and then reuse that from the cache. You are creating the image contents dynamically!

```
<%
//' Return the "Curlz" image for the KeyCode position in the query string
    KeyPos = QryStrNum('KeyPos'); // The Key Digit position .. First 1 then 2  then 3
and last 4..
    if (KeyPos >= 1);
       KeyCode = SesGetVar('KeyCode');
       KeyPicture = %subst(KeyCode : KeyPos : 1);
       errstr = include('image/curlz' + KeyPicture + '.gif');
       if (errstr = '');
          SetContentType('image/gif');
          SetCacheTimeout(0);
       else;
          %><%= errstr %><%
       endif;
       return;
    endif;
%>
```

Run the sample   Show the sample

3.16 Using COBOL as ASPX programming language

Programming:

By migrating a COBOL program to IceBreak ASPX, you have a good opportunity to move legacy code towards the Internet.

This release supports the basic features of the "form" object and the "response" object. These features are sufficient for writing COBOL-ASPX for the IceBreak server.

Let's look into the souce code:

First, set the precompiler to COBOL (CBLLE).

```
<%@ language="CBLLE" %>
```

In the "Environment division" you will need some special linkage features for the IceBreak server. This is done by including the member "httpxlink" in file "qasphdr".

```
 Environment division.
 Configuration section.
 Special-names.
     copy httpxlink in qasphdr.
```

In the "Data division" you need to include IceBreak-internal work variables using the member "httpxdata" in file "qasphdr".

```
Data Division.
Working-Storage Section.
copy httpxdata in qasphdr.
```

In the "procedure division" you are now able to utilize the "request" and the "response" object within the IceBreak server. When placing data in the response object you call on one of the following procedures and macros.

Syntax	Description
<%= MyVariable %>	URL-Encodes and blank-trims a variable into the response object
"Response_Write"	Writes the parameter/literal to the response object as it is
"Response_Write_NL"	Writes the parameter to the response object. Then appends a new line characters
"Response_NL_Write"	Writes a new line of characters into the response and appends the parameter/literal

The final sample might look like this using basic COBOL paragraphing.

```
<%@ language="CBLLE"%><%
*' ------------------------------------------------------------- '*
 Environment division.
 Configuration section.
 Special-names.
     copy httpxlink in qasphdr.
*' ------------------------------------------------------------- '*
 Data Division.
 Working-Storage Section.
     copy httpxdata in qasphdr.

 Procedure Division.
*' ------------------------------------------------------------- '*
 Main section.

*' Insert the html heading text
%>
<html>
  <h1>This is a test from a COBOL.ASP program  </h1>
</html>
```

Run the sample   Show the sample

Use the following macros when you want to retrieve data from form input fields.

Syntax	Description
MyVariable = Request.Form("MyFormField");	Retrieve a char field from the input form
MyVariable = Request.FormNum("MyFormField");	Retrieve a number field from the input form
MyVariable = Request.QueryString("FormField");	Retrieve a char field from the URL
MyVariable = Request.QueryStringNum ("FormField");	Retrieve a number from the URL

Please Note:

1) Macros in IceBreak-COBOL are terminated by ";" and they look likefunction calls in "Visual-Basic".

2) The first parameter (the left side of the =) is the name of the variable in your COBOL program to receive the content.

3) The second parameter is the form field name. The name of any HTML <input> tag.

Show the complete list of macros

Let's extend the sample above placing a form field and showing what was typed into that field

This sample might look like this:

```
<%@ language="CBLLE"%><%
*' ------------------------------------------------------------- '*
 Environment division.
 Configuration section.
 Special-names.
     copy httpxlink in qasphdr.
 Input-output section.
 File-Control.

*' ------------------------------------------------------------- '*
 Data Division.
 Working-Storage Section.
     copy httpxdata in qasphdr.

 01  Globals.
     05  MyField                     pic x(30).

 Procedure Division.
*' ------------------------------------------------------------- '*
 Main section.

 Main01.

*' Insert the html heading text
%>
<html>
  <h1>This is a test from a COBOL.ASP program</h1>
  <form action='ex10cobol2.asp' method='post' name='form1'>
     Enter a string:<input type='text' name='MyField'>
<%   MyField = Request.Form("MyField"); %>
     <br>The string was: <%= MyField %>
   </form>
</html>
<%
 MainEnd.
     exit.
%>
```

Run the sample

Finally let's look at a COBOL-ASPX application producing a list of data using the DB2-database

Show the sample Run the sample

When using the procedure interface, it might look like this:

call procedure "Response_Write" using content "

```
call procedure "Response_Write" using content
     "<html><h1>This is a test from a COBOL.ASP program</h1>"
end-call
```

Use the "using content" calling convention when placing text literals, or "using reference" as the calling convention. Remember that text literals have a maximum length of 255 characters.

3.17 Using Legacy RPG fixed form code

Programming:

You can even use old style "input primary files" and the old-style "cycle", however, you have to migrate old RPG-III programs to IceBreak which can be done with the following guide-lines:

- Convert the RPG-III to RPG-ILE using the CVTRPGSRC command
- Move the source to the IFS using "Source File Browser" tool found in the Adminitration menu
- Isolate the business logic from display files into routines
- Rewrite the user interface into HTML
- Use the "File Field Descriptor" tool to create HTML tables and HTML forms

The fixed form business logic might be used with little or no change at all:

```
<%@ language="RPGLE" %>
<%
FPRODUCT   IP   E              DISK
%><% = prodid %><%= Desc %><br/>
```

Run the sample

Part IV

4 Internal Large Objects - ILOB's

4.1 ILOB's - Internal Large Objects

ILOB's:

Host languages like RPG and COBOL have a limit in program variable size. IceBreak breaks that limit by utilizing userspaces and associates them to a session. The transfer of data back and forth between Request object or Response object can be done with help from ILOB's. Also SQL CLOB's and BLOB's can be mapped to ILOB’s. See ILOB's and SQL

Consider a file upload application. When the user hits the “upload” button on a form this data can be placed in an ILOB up to 2Gbytes. This ILOB can now save itself as a stream file. The XML parser can parse it or it can be placed into DB/2 CLOB field.

ILOB's is by default session maintained, but can also be persistent so it is ideal to share data between server instances and session instances with ILOB's.

ILOB's contains a body and a header which maps directly to the HTTP protocol. Therefore ILOB is used for IceBreak webservices.

i5/OS userspaces are used to implement ILOB's. They are wrapped in an easy-to-use IceBreak-API's. However, you can use IBM supplied API's for manipulating userspaces if you create the basic type ILOB. see the API sample.

The access to IOB functions is available by including:

```
/include qasphdr,ilob
```

ILOB Functions:

- `IlobPtr = ILOB_OpenPersistant(Library : Name);`
- `Ok = ILOB_DeletePersistant(IlobPtr);`
- `Ok = ILOB_Read(IlobPtr: String: Offset : Length);`
- `Ok = ILOB_ReadNext(IlobPtr: String :Length);`
- `Ok = ILOB_Write(IlobPtr : String :Offset);`
- `Ok = ILOB_LoadFromBinaryStream (IlobPtr: FileName);`
- `Ok = ILOB_LoadFromTextStream (IlobPtr: FileName);`
- `Ok = ILOB_SaveToBinaryStream (IlobPtr: FileName);`
- `Ok = ILOB_SaveToTextStream(IlobPtr: FileName);`
- `OK = ILOB_Xlate(IlobPtr: fromCCSID : toCCSID);`
- `Ok = ILOB_Append(IlobPtr : String);`
- `Ok = ILOB_Clear (IlobPtr);`
- `DataPtr= ILOB_GetDataPtr(IlobPtr); // Obsolete - now the IlobPointer is the same as the data p`
- `Len = ILOB_GetLength(IlobPtr);`
- `ILOB_SetData (IlobPtr : Length : Data | *NULL);`
- `ILOB_SetWriteBinary(IlobPtr; *ON|*OFF);`
- `ILOB_Close(IlobPtr);`
- `ILOB_Ilob2Clob( ClobLocator : IlobPtr : Offset : Length );`
- `ILOB_Clob2Ilob( IlobPtr : ClobLocator : Offset : Length );`
- `ILOB_SubIlob (OutIlobPtr: InIlobPtr : FromPos : Length | ILOB_ALL : ToPos | ILOB_END); // Pos`

Also functions directly accessible from the response, request, session object and the XML parser:

- `Form2ILOB(IlobPtr: FormFieldName);`
- `ResponseWriteILOB(IlobPtr);`
- `SetResponseObject(IlobPtr | *NULL);`

- `IlobPtr = SesGetILOB(IlobName : [Defultsize=8192] : [HeaderSize=4096] );`
- `IlobPtr = SesGetUsrSpc(usrspcname: IlobName : [Defultsize = 8192] );`
- `XmlPtr = Xml_ParseILOB ( IlobPtr : Options: InputCcsid : OutputCcsid);`
- `error = Xml_GetIlobValue (IlobPtr: XmlPtr : xPathNode);`

ILOB’s are also used for response and request objects in some application server session dispatcher modes.

First let's play with a ILOB as a way to preserve session variables for an application. A good practice might be to prefix the ILOB name with the name of the program for program global session variables, and "globals." for cross program variables. Simply base all variables you want to preserve in a session on a pointer served by an ILOB. If the name does not exists - it will automatically be created:

```
<%@ language="RPGLE"%>
<%

/include qasphdr,ilob

D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments++++++
++++++
d MyIlob          s               *
d MySession       ds                  based(MyIlob)
d  Counter                      10I 0
d  first                         1A

C/free
   *inrt        = *on;                    // break the cycle
   MyIlob = SesGetILOB('Thispgm.MyIlob'); // All ILOB's in the following is based on
"MyIlob" session ILOB
   if ( first  <> *ON);                   // Initialize the ilob fields
      Counter = 0;
      First = *ON;
   endif;
   Counter = Counter + 1;                 // Increment the counter in the ilob
   %><% = %char(Counter) %><%
%>
```

Show the sample   Run the sample

You can manipulate ILOB's with ILOB_SubIlob, where you can take a fraction or all data from an ILOB and insert it or append it to another ILOB

```
<%@ language="RPGLE" modopt="DBGVIEW(*LIST)" %>
<%

D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments++++++
++++++
d firsttime       s               n   INZ(*ON)
d ilob1           s               *
```

```
d ilob2           s               *

/include qasphdr,ilob


C/free


// Get two pointers to some session ILOB's
   ilob1  = SesGetILOB('ilob1');
   ilob2  = SesGetILOB('ilob2');

// Clear Ilobs. The second only the first time
   Ilob_Clear(ilob1);
   if (firsttime);
     Ilob_Clear(ilob2);
     firsttime = *OFF;
   endif;

// Put data in the first ilob, so we redirect the response to the first ilob
   SetResponseObject(ilob1);

// Use the normal ASPX syntax to produce data, this time however it ends up in "ilob1"
%>
<tr><td><%= %char(%timestamp()) %></td></tr>
<%
// Switch back to default reposnse object
   SetResponseObject(*NULL);

// Now append that data to the second ilob
   Ilob_SubIlob(ilob2 : ilob1 : 1 : ILOB_ALL : ILOB_END);

// Build the result
%>
<html>
 <Head>
  <link rel="stylesheet" type="text/css" href="/System/Styles/IceBreak.css"/>
 </Head>
 <body>
 <h1>ILOB Demo - Press refresh to append to the list</h1>
 <table>
<%
 // take the final ilob which contains all table rows and put it into the table
    ILOB_SetWriteBinary(ilob2 : *ON); // This is already in ASCII
    ResponseWriteIlob(ilob2);
%></table>
 </body>
</html>
<% return ; %>
```

Show the sample   Run the sample

4.2 ILOB's and SQL

ILOB's can map CLOB and BLOB coloumns in a SQL table. It requires, however, that the SQL tables a journal'ed and commit are allowed.

In this sample we are using SQL and maps fields to SQL CLOB fields, it requires a "locator", but with that you are able to save and restore SQL data in a session.

```
<%@ language="SQLRPGLE" options="COMMIT(*ALL)" modopt="DBGVIEW(*LIST)" %>
<%
 *' --------------------------------------------------------------------------------
 *' Use CLOB Fields directly from input form and the response object
 *' --------------------------------------------------------------------------------

/include qasphdr,ilob

D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments++++++
++++++
d  MyIlob         s               *
d  MyClob         s                  sqltype(CLOB_LOCATOR)
d  lobind         s              5i 0
d  ALL            s             10I 0 inz(-1)

 *' --------------------------------------------------------------------
 *' Main logic
 *' --------------------------------------------------------------------
/Exec sql
   Set Option Optlob=*YES, Commit=*ALL, Closqlcsr=*ENDACTGRP
/End-exec

C/free
   *inrt  = *on;                     // break the cycle
   MyIlob = SesGetILOB('MyIlob'); // All ILOB's in the following is based on "MyIlob"
session ILOB

%>
<html>
 <Head>
  <meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
  <link rel="stylesheet" type="text/css" href="/System/Styles/IceBreak.css"/>
 </Head>
 <body>
 <h1>CLOB/ILOB Demo</h1>
 <form method=POST id=form1 name=form1>
  <textarea name=lob></textarea>
  <br/>
  <input type=submit name=loadfromstream value="Load ILOB From Stream">
  <input type=submit name=savetostream value="Save ILOB to Stream">
  <input type=submit name=Showilob value="Show ILOB">
  <input type=submit name=form2clob value="Form to Clob">
  <input type=submit name=ilob2clob value="ILOB to Clob">
  <input type=submit name=Delete value=Delete>
<%
   if Form('form2clob') > '';
      Form2Clob(myClob : 'lob');
      Exsr Sql_Insert;
      %><pre><% responseWriteCLOB(MyClob); %></pre><%

      ILOB_Clob2Ilob(MyIlob:MyClob:1:ALL);
   endif;

   if Form('Ilob2clob') > '';
      ILOB_Ilob2Clob(MyClob:MyIlob:1:ALL);
      Exsr Sql_Insert;
      %><pre><% responseWriteCLOB(MyClob); %></pre><%

   endif;

   if Form('showilob') > '';
      %><pre><% responseWriteILOB(MyILOB); %></pre>
      <%
   endif;
   if Form('loadfromstream') > '';
      ILOB_LoadFromStream(MyIlob :'/www/systest/space.txt':'r');
      %><pre><% responseWriteILOB(MyILOB); %></pre>
      <%
   endif;

   if Form('savetostream') > '';
      Form2ILOB(myIlob : 'lob');
      ILOB_SaveToStream(MyIlob :'/www/systest/space.txt':'w,ccsid=277');
   endif;

   if Form('Delete') > '';
      Exsr Sql_Delete;
   endif;
%>

<table border="1">
 <thead>
  <TR>
    <th>Clob</th>
  </TR>
 </thead>

<%
// Now reload the list
// -------------------
   Exsr Sql_Open;                     // Declare and Open the SQL cursor
   Exsr Sql_Fetch;                    // Get the first row from the result set
   Dow (SqlCod = 0) ;                 // repeat until max number of rows found or EOF
%>
    <tr>
      <td><pre><% responseWriteCLOB(MyClob); %></pre></td>
    </tr>
<%
     Exsr Sql_Fetch;                 // Get the next row from the result set
   EndDo;
   Exsr Sql_Close;                   // Always remember to close cursors after use
%>
  </table>
 </form>
 </body>
</html>
<%
   return;

//' --------------------------------------------------------------------------------
//' Open is declaring And opening the SQL Cursor
//' Actually the declare ends up as a comment in the final object
//' --------------------------------------------------------------------------------
Begsr Sql_Open;

/Exec SQL
   Declare List cursor for
   Select  yclob
   from    y
   for update of yclob
/End-exec
```

```
/Exec SQL
    Open List
/End-Exec

   Exsr Sql_Monitor;
Endsr;
//' --------------------------------------------------------------------------------
//' Fetch - is retrieving the newt row from the result set
//' --------------------------------------------------------------------------------
Begsr Sql_Fetch;

/Exec SQL
     Fetch list into :MyClob :lobind
/End-Exec

   Exsr Sql_Monitor;
Endsr;
//' --------------------------------------------------------------------------------
//' Close - Just close the cursor
//' --------------------------------------------------------------------------------
Begsr Sql_Close;


/Exec SQL
     Commit hold
/End-Exec

   Exsr Sql_Monitor;

/Exec SQL
     Close list
/End-Exec

   Exsr Sql_Monitor;

Endsr;
//' --------------------------------------------------------------------------------
//' Insert a new row
//' --------------------------------------------------------------------------------
Begsr Sql_Insert;

/Exec SQL
     Insert into y values :MyClob
/End-Exec

   Exsr Sql_Monitor;
Endsr;
//' --------------------------------------------------------------------------------
//' delete all rows
//' --------------------------------------------------------------------------------
Begsr Sql_Delete;

/Exec SQL
     Delete from y
/End-Exec

   Exsr Sql_Monitor;
Endsr;
//' --------------------------------------------------------------------------------
//' Sql_Monitor is the global monitor for sql-errors
//' The SQL_SetError sends the message back to the IceBreak server.
//' Then use 'GetLastError' to display the formatted error message
//' --------------------------------------------------------------------------------
Begsr Sql_Monitor;
   select;
      when SqlCod = 0;
      when SqlCod = 100;
      other;
         SQL_SetError(SqlCod:SqlErm);
%>
```

```
         <script>
         alert ("<%= GetLastError('*MSGTXT') %>");
         </script>
<%
   Endsl;
EndSr;
%>
```

4.3 ILOB's with IBM supplied API's

If you want to use traditional userspaces with in a session and let IceBreak handle the memory management, you can use a special userspace ILOB which has no ILOB's features except for the plain userspace and the fact it will be connected to your session and reclaimed when the session ends.

Syntax:

IlobPtr = SesGetUsrSpc(UsrSpcName: IlobName : [Defultsize=8192]);

Field Name	Usage	Data type	Description
UsrSpcName	OUTPUT	CHAR(20)	Automatically generated name of the resulting qualified output name and library for the user space
Name	INPUT	VARCHAR(256)	Your session global wide alias name for the ILOB userspace managed for your session. (Not case sensitive)
Name	INPUT - optional	INT4	The optional initial size for the userspace in bytes. Default i 8K if omitted
Return value			
IlobPtr	OUTPUT	POINTER	Pointer to userspace

Sample:

```

<%@ language="RPGLE"  %>
<%
 *' --------------------------------------------------------------------------------
 *' Demo: userspace managed by ilobs
 *' --------------------------------------------------------------------------------

/include qasphdr,ilob

d MyIlob          s                *
d pObj            s                *
d ReadCount       s               5u 0
d lstobj          s              20

 * ------------------------------------------------------------- *
D gh              ds                   Based(MyIlob)
D                                      LikeDs(QUSH0100)
```

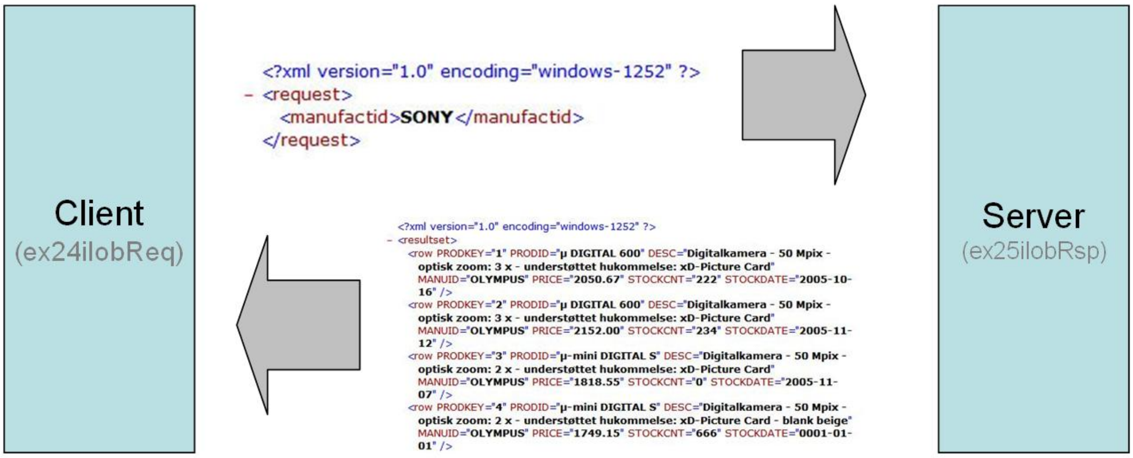
```
 * -------------------------------------------------------------- *
D ol               ds                  Based(pObj)
d                                      likeds(QUSL010003)
 * -------------------------------------------------------------- *
D quslobj          pr                  extpgm('QUSLOBJ')
D  SpaceName                     20
D  Format                         8    const
D  ObjQual                       20    const
D  ObjType                       10    const


 /Include QSysInc/qRpgleSrc,QUSGEN
 /Include qasphdr,QUSLOBJ

 *' --------------------------------------------------------------------
 *' Main logic
 *' --------------------------------------------------------------------

C/free

%><html>
<head>
 <link rel="stylesheet" type="text/css" href="/System/Styles/IceBreak.css"/>
</head>
<body>
 <h4>Tutorials</h4>
 <h1>Using Userspace like an ilob.</h1>
 <table>
 <%

   *inlr = *on;
   MyIlob = SesGetUsrSpc(lstobj
                        : 'ListOfmyObjects'
                        : 4096 );

   quslobj ( lstobj
           : 'OBJL0100'
           : '*ALL      QGPL      '
           : '*ALL      ');

   pObj    = myilob + gh.QUSOLD;      // Position to Record format
   for ReadCount = 1 to gh.QUSNBRLE;  // ReadCount < Number List Entries
     %><tr><%
       %><td><% = ol.QUSOLNU  %></td><%
       %><td><% = ol.QUSOBJNU %></td><%
       %><td><% = ol.QUSOBJTU %></td><%
     %></tr><%
     pObj     = pObj + gh.QUSSEE;     // Size Each Entry
   endfor;

%>
</table>
</body>
</html>
```

4.4 ILOB's, httpRequest and XML

ILOB's:

In a Service Oriented Architecture world (SOA) the ability to intercommunicate huge data streams is essential. As we saw earlier ILOB's is a way deal with large data from within an ILE program.

This tutorial will show how to use ILOB's and make HTTP request with XML based data stream.



```
<?xml version="1.0" encoding="windows-1252" ?>
- <request>
    <manufactid>SONY</manufactid>
  </request>
```

```
  <?xml version="1.0" encoding="windows-1252" ?>
- <resultset>
    <row PRODKEY="1" PRODID="μ DIGITAL 600" DESC="Digitalkamera - 50 Mpix -
      optisk zoom: 3 x - understøttet hukommelse: xD-Picture Card"
      MANUID="OLYMPUS" PRICE="2050.67" STOCKCNT="222" STOCKDATE="2005-10-
      16" />
    <row PRODKEY="2" PRODID="μ DIGITAL 600" DESC="Digitalkamera - 50 Mpix -
      optisk zoom: 3 x - understøttet hukommelse: xD-Picture Card"
      MANUID="OLYMPUS" PRICE="2152.00" STOCKCNT="234" STOCKDATE="2005-11-
      12" />
    <row PRODKEY="3" PRODID="μ-mini DIGITAL S" DESC="Digitalkamera - 50 Mpix -
      optisk zoom: 2 x - understøttet hukommelse: xD-Picture Card"
      MANUID="OLYMPUS" PRICE="1818.55" STOCKCNT="0" STOCKDATE="2005-11-
      07" />
    <row PRODKEY="4" PRODID="μ-mini DIGITAL S" DESC="Digitalkamera - 50 Mpix -
      optisk zoom: 2 x - understøttet hukommelse: xD-Picture Card - blank beige"
      MANUID="OLYMPUS" PRICE="1749.15" STOCKCNT="666" STOCKDATE="0001-01-
      01" />
```

Basically we want to send a XML request to a server program and in return receives the XML response. By using ILOB's we are able to break the 32K limit that RPG normally has.

The Server program

```
<%@  language="RPGLE" %>
<%
 *'
-------------------------------------------------------------------------------------------
---
 *' Runs an SQL query for the XML request received
 *'
-------------------------------------------------------------------------------------------
---

D*Name+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments++++++
++++++
d manufactid      s             16    varying
D Error           s               N
D SqlCmd          s           1024    varying
D MaxRows         s             10i 0

/free
//' The first thing is always to set charset and content type
    SetContentType ('application/xml; charset=windows-1252');

//' Take the parameter from the XML request
    manufactid = reqXmlGetValue('/request/manufactid' : '');

//' Build the SQL command
    sqlcmd = 'select * from product where manuid = ''' +manufactid+ '''';
    MaxRows = 10000;

//' Now run the SQL query
    %><?xml encoding="windows-1252" version="1.0" ?><%
    Error   = SQL_Execute_XML(sqlcmd : maxrows);
    if (Error);
      %><error help="<% = getLastError('*HELP') %>" msg="<% = getLastError('*MSGTXT') %>
"></error><%
    endif;
    return;
%>
```

1. The *SetContentType()* to "application/xml" is essential. We both sends ad receives data in XML format. if IceBreak "see" the /XML in the content type it will automatically run the XML-parser.
2. The xml request is already parsed so we have access to an XML object that IceBreak maintains. Now we can use the X-path syntax to refer to the value of the element "*manufactid*" in the "*request*" elemet.
3. The SQL statement is constructed. Just adding the "*where*" clause and we are ready to run the SQL.
4. Finally the surroundings for the XML is set up. Note that the encoding tag in the XML header has to be the same as we used in "*SetContentType()*"
5. SQL_Execute_XML() simply returns the SQL result set as an XML document, which is placed directly in the response object.

You can see the number of variables in a program like this i kept to a minimum. All the data manipulation is done behind the scenes in the request and response object.

The Client program

The client program is a little more sophisticated the the client. It is dealing with the access to the ILOB's and parser directly:

```
<%@ language="RPGLE" %>
<%

d*' Include the ILOB and xml parser prototypes
/include qasphdr,ilob
/include qasphdr,xmlparser

d*Name+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments++++++
++++++
d manufactid      s             16    varying
d path            s             64    varying
d Error           s               N
d reqILOB         s                *
d respILOB        s                *
d xmlPtr          s                *
d i               s             10i 0
d rows            s             10i 0

/free

//' Bacically this program just toggle between the manufactures
    manufactid = form('manufactid');

//' The first thing is always to set charset and content type
    SetContentType ('text/html; charset=windows-1252');

//' First I crerate my work ILOB's
    reqILOB  = SesGetILOB('request');   // get a pointer to a session ILOB used as my
request
    respILOB = SesGetILOB('response');  // get a pointer to a session ILOB used for the
response from the server

//' Reset my ilob
    ILOB_Clear(reqILOB);
    ILOB_Clear(respILOB);

//' Now i want to populate a simple XML request. - my ILOB header need a content type XML
//' All httpRequest are routet to the same default job hench the cookie
    ILOB_SetHeaderBuf(
      reqILOB :
      'Content-Type: application/XML; charset=windows-1252'
    );

//' I reroute my response object to the request ILOB, so I can use the ASP sysnax for
```

```
creating the XML
    SetResponseObject(reqILOB);

%><?xml version="1.0" encoding="windows-1252" ?>
<request>
  <manufactid><% = manufactid %></manufactid>
</request><%

//' Now i redirect my response back to the default responce objet ( what my browser
receives >
    SetResponseObject(*NULL);

//' My request is now ready to to to the server, so i call the httpRequest for ILOB's
    Error = ILOB_httpRequest(
       '/tutorials/ex24ilobsvr.asp': // The URL in form:"http://server:port/resource"
       30:                            // Number of seconds before timing out
       'POST':                        // The "POST" method
       reqILOB:                       // pointer to my Request ILOB
       RespILOB                       // pointer to my Response ILOB
    );

//' Check for errors
    if error;
       %><% =  GetlastError('*MSGTXT') %><%
       return;
    endif;

//' My data has arrived - I'll fire up the XML parser
    xmlPtr = XML_ParseILOB ( //' Returns XML-object tree from an ILOB
       RespILOB:             //' Pointer to an ILOB object
       'syntax=loose':       //' Parsing options
       1252:                 //' The ccsid of the input ilob  (0=current job)
       0                     //' The ccsid of the XML tree (0=current job)
    );

//' Check for errors
    if XML_Error(xmlPtr);
       %><%= XML_Message(xmlPtr) %><%
       XML_Close(xmlPtr);
       return;
    endif;

//' The Result will be a table
%><html>
<head>
 <link rel="stylesheet" type="text/css" href="/System/Styles/IceBreak.css"/>
</head>
<h1>Products by: <% = manufactid %></h1>

<form action="" method="post" id=form1 name=form1>
 <select name="manufactid">
    <option value="ACER">ACER</option>
    <option value="CANON">CANON</option>
    <option value="CASIO">CASIO</option>
    <option value="FUJIFILM">FUJIFILM</option>
    <option value="HP">HP</option>
    <option value="KODAK">KODAK</option>
    <option value="KONICA">KONICA</option>
    <option value="NIKON">NIKON</option>
    <option value="OLYMPUS">OLYMPUS</option>
    <option value="PANASONIC">PANASONIC</option>
    <option value="SAMSUNG">SAMSUNG</option>
    <option value="SONY">SONY</option>
 </select>
 <input type="submit" value="Go"/ id=submit1 name=submit1>
</form>

<table>
  <thead>
    <th>Product id</th>
    <th>Product Desciption</th>
    <th>Product Price</th>
  </thead><%
```

```
//' now print the report
    rows =  num(XML_GetValue(xmlPtr: '/resultset/row[ubound]' :'0'));
    for i = 0 to rows -1;
      path = '/resultset/row[' + %char(i) + ']@';
      %><tr>
        <td><% = XML_GetValue(xmlPtr: path + 'prodid' :'N/A') %></td>
        <td><% = XML_GetValue(xmlPtr: path + 'desc'   :'N/A') %></td>
        <td><% = XML_GetValue(xmlPtr: path + 'price'  :'N/A') %></td>
       </tr><%
    endfor;
%>
 </table>
</html><%

//' Always release the memory used
    XML_Close(xmlPtr);
    return;
%>
```

Show the sample Run the sample

1. Like the server program we need to set up the contents type. It need to map to the same values as the server.
2. The we create two ILOB's used for request and response
3. The clear of the the Request ilob is essential. Otherwise data will just append to the ILOB.
4. The ILOB has a header which is used for HTTP communication. You have access to that header by the *"ILOB_SetHeaderBuf()".* Note that you replaces the complete header by using the method.
5. The "SetResponseObject()" is used to reroute the response data to an ILOB so the data goes to the ILOB and not the normal response object. When you want to switch back, the just call "SetResponseObject()" with a *NULL parameter and you have your normal response object back which goes to your browser. This method can also be applied to creating XML files on the fly by building up the ILOB and the save the ILOB to stream file.
6. Now with the request XML in the "request" ILOB we just runs the "*ILOB_httpRequest()*" method and receives the "response" ILOB.
7. All the data in the ILOB is in ASCII, so when we fire up the XML parser we have to tell it which ccsid the ILOB is sored in. In this case "windows-1252" maps to "ccsid 1252".
8. Now are finished with the ILOB's the rest of the program just formats the XML tree to the response object that goes to the end-users browser. Here again - like in the server - we are extracting data using the X-Path syntax.
9. Finally always remember to use XML_Close() to free up the memory used by the XML parser.

4.5 ILOB's secures your streamfile upload to your IceBreak server

Next Lets upload data to an ILOB and save it to a stream file. Note the ***ILOB:** in the input field. The name just after is the ILOB field.

This is a good way to protect files in an upload until you decide where to save the uploaded file (see tutorial 3 on uploads)

On the form you set the "`enctype`" to "`multipart/form-data`". This ensure that the browser uploads the attached file:

```
<form method="post" enctype="multipart/form-data" name="form1">
```

Set the input type to file and the name prefixed by *ILOB: to ensure the up loaded stream is placed in the ILOB. From there you can manipulatet it in any way you like with the ILOB build in functions.

```
<input type="file" name="*ILOB:MyUploadIlob" size="40"/>
```

Now the final program:

```
<%@ language="RPGLE" %>
<%

/include qasphdr,ilob

D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments++++++
++++++
d  MyUploadIlob   s               *

C/free
   *inrt        = *on; // break the cycle
   MyUploadIlob = SesGetILOB('MyUploadIlob');

%>
<html>
 <Head>
  <link rel="stylesheet" type="text/css" href="/System/Styles/IceBreak.css"/>
 </Head>

 <body>
 <h1>ILOB upload Demo</h1>

 <form method="post" enctype="multipart/form-data" name="form1">
  <br/>File to upload :<br/>
  <input type="file"   name="*ILOB:MyUploadIlob"  size="40"/>
  <input type="submit" name=submit1 VALUE="Upload"/>
 </form>
<%
  if Form('file.1.RemoteName') > '';

  %><p>The Remote file name is: <%= form('file.1.Remotename') %></p>
    <p>It was uploaded to an ILOB : <%= form('file.1.Localname') %></p>
    <p>The size of the received file was: <%= form('file.1.size') %></p>
  <%
  // Now save it to disk
     ILOB_SaveToBinaryStream(MyUploadIlob : 'MyUploadIlob.txt');

  endif;
%>
 </body>
</html>
```

Show the sample

Run the sample

4.6 ILOB's in WebServices

ILOB's

ILOB's or Internal Large Objects is heavily used in IceBreak to store large chunks of data. In this example I'll show you how to run a SQL query based on a request in a WebService. The XML result is placed in an ILOB and returned to the WebService client as a string.

```
<%@ language="RPGLE" pgmtype="webservice" %><%
h NOMAIN

/include qasphdr,ilob

p*name++++++++++..b...................keywords++++++++++++++++++++++++++comments+++++++++
+++
p GetProductList...
p                 B                   export

d                 PI
d SqlCmd                      1024    input varying
d xmlout                          *   output

d*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments++++++
++++++
d Error           s               N
d i               s             10i 0

/free

  xmlout = SesGetILOB('sqllist');     //' The SQL result set will go directly to the ILOB
in the response
  setResponseObject ( xmlout );       //' Redirect all output to the response object to
the ILOB
  ILOB_SetWriteBinary(xmlout : *ON);  //' This is in ASCII so don't x-late it later.


  %><?xml version="1.0" encoding="windows-1252" ?><%


  //' Now run the SQL query and put all data into the response object
  MaxRows = 10000;
  Error   = SQL_Execute_XML(sqlcmd : maxrows);

  if (Error);
     %><error msg="<% = getLastError('*MSGTXT') %>"
              help="<% = getLastError('*HELP') %>"/><%
  endif;

  setResponseObject (*NULL);         //' Go back to use the normal response object from
here

/end-free
P GetProductList...
P                 E
%>
```

As you can see - it is pretty much a normal service program - except for the input/output keywords. It is a service program with no mainline. It simply exports a procedure. However, You are not restricted to only one.

The trick here is that **xmlout** is defined as as pointer. When IceBreak recognize a pointer in th WebService declaration it will assume it is a ILOB'pointer. In this example I have used the

SQL_Execute_XML() to produce the SQL result set XML by redirecting the **ResponseObject** to the **xmlout** ILOB.

Now IceBreak is tranfering the resultset in the WebService interface.

Part V

5 SQL

5.1 Using embedded SQL

SQL:

SQL programs can have another subtype besides RPGLE, the other is called SQLRPGLE. All you have to do is tell the precompiler that another programming language is being used, and override how the commitment control is being used. The first line of ASPX code does that, it is called Compiler-directives:

```
<%@ language="SQLRPGLE" sqlopt="COMMIT(*NONE)" %>
```

The "sqlopt" parameter can override any additional parameters on the CRTBNDRPG and CRTSQLRPGI.

Host variables are needed to hold the result of the SQL-query. This is a standard RPG d-spec definition.

```
<%
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments++++++
++++++
D MyCounter       s               10U 0
D Now             s                Z
/free
%>
```

The last thing is the actual SQL-statement. This is:

```
<%
Exec SQL Select count(*)  , Current Timestamp
         into   :MyCounter, :Now
         from   Product;

%>
```

Or if you prefer the classic style syntax - enclosed by **/exec** and **/end-exec**:

```
<%
/Exec SQL Select count(*)  , Current Timestamp
          into   :MyCounter, :Now
          from   Product
/End-Exec
%>
```

Both syntaxes above are supportet both within fixed or free format from i5/OS ver. 5R1 with help from the IceBreak pre-compiler.

Note, that colon in **:MyCounter** and **:now** is the method to describe an RPG host variable when you are writing SQL-statements.

Finally, you are able to present the result in HTML like this.

```
<p>
 The number of rows in table "Product" is <%= %char(MyCounter) %> and time is: <% = %char
(now) %>
</p>
```

The complete example looks like this.

```

<%@ language="SQLRPGLE" sqlopt="COMMIT(*NONE)" %>
<%
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments++++++
++++++
D MyCounter       s             10U 0
D Now             s               Z
/free
%>
<html>
<h1>Using SQL</h1>
<%
   Exec SQL Select count(*)  , Current Timestamp
            into  :MyCounter , :Now
            from product;
%>
  <p>
   The number of rows in table "product" is <%= %char(MyCounter) %> and time is: <% = %
char(now) %>
  </p>
</html>
<%
  return;
%>
```

Show the sample   Run the sample

5.2 SQL - Producing a list using a cursor

SQL:

If you want to use SQL to create result sets and read rows into a HTML table, then you need a "cursor".

Take a look at the following example which is a embedded-SQL version of a List application.

Here we place all SQL-statements into subroutines, so it doesn't interfere with the main logic. Also notice the use of **SQLMonitor** which is a method to determine if SQL-statements have completed normally.

```

<%@ language="SQLRPGLE" options="COMMIT(*NONE)" %>
<%
H*' --------------------------------------------------------------------------------
H*' A List using embedded SQL
H*' --------------------------------------------------------------------------------

D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments++++++
++++++
D Rows            s             10i 0
D MsgId           s              7
D MsgFile         s             10
D Sql_Cmd         s           1024    Varying

 *' --------------------------------------------------------------------
 *' The SQL host variables are included as an external data-structure
 *' --------------------------------------------------------------------
D Product       E ds

 *' --------------------------------------------------------------------
 *' Main logic
 *' --------------------------------------------------------------------
 /free

   Exsr Init;
   Exsr LoadList;
   Exsr Exit;

//' --------------------------------------------------------------------------------
//' Init; sets up the HTML header and stylesheet / (evt. the script links)
//' --------------------------------------------------------------------------------
Begsr Init;
%>
<html>
 <Head>
  <link rel="stylesheet" type="text/css" href="/System/Styles/IceBreak.css"/>
 </Head>
 <Body>
<%
Endsr;
//' --------------------------------------------------------------------------------
//' loadList; is much like a "load subfile". All records are placed into a html table
//' --------------------------------------------------------------------------------
Begsr LoadList;

// First build the table header
// ----------------------------
%>

<table border="1">
  <thead>
   <th>Product ID</th>
   <th>Description</th>
   <th>Manufacturer ID</th>
   <th>Price</th>
   <th>Stock Count</th>
   <th>Stock Date</th>
  </thead>

<%
// Now reload the list
// -------------------
   Rows = 0;                          // reset the row counter

   Exsr Sql_Open;                     // Declare and Open the SQL cursor
   Exsr Sql_Fetch;                    // Get the first row from the result set
   Dow (SqlCod = 0 and Rows < 2000) ; // Repeat until max number of rows found or EOF
     Rows = Rows + 1;                 // Count number of rows so fare
%>
     <tr>
        <td nowrap><% = PRODID %></td>
        <td><% = DESC %></td>
        <td nowrap><% = MANUID %></td>
        <td nowrap align=right><% = %editc(PRICE:'J') %></td>
        <td nowrap align=right><% = %editc(STOCKCNT:'1') %></td>
        <td nowrap><% = %char(STOCKDATE) %></td>
     </tr>
<%
     Exsr Sql_Fetch;                 // Get the next row from the result set
   EndDo;
   Exsr Sql_Close;                   // Always remember to close cursors after use
%>
</table>
<%
EndSr;
//' --------------------------------------------------------------------------------
//' Exit Finish up the the Final html and quits the program
//' --------------------------------------------------------------------------------
Begsr Exit;
%>
  </body>
</html>
<%
   return;
Endsr;
//' --------------------------------------------------------------------------------
//' Open is declaring And opening the SQL Cursor
//' Actually the declare ends up as a comment in the final object
//' --------------------------------------------------------------------------------
Begsr Sql_Open;

  Sql_Cmd  = 'Select * from Product '
           + 'Where prodId  > '' '' '
           + 'Order by ManuId';

  Exec SQL Prepare pList from :Sql_Cmd;
  Exsr Sql_Monitor;

  Exec SQL Declare List cursor for pList;
  Exsr Sql_Monitor;

  Exec SQL Open List;
  Exsr Sql_Monitor;

Endsr;
//' --------------------------------------------------------------------------------
//' Fetch - is retrieving the newt row from the result set
//' --------------------------------------------------------------------------------
Begsr Sql_Fetch;

  Exec SQL Fetch list into :product;
  Exsr Sql_Monitor;

Endsr;
//' --------------------------------------------------------------------------------
//' Close - Just close the cursor
//' --------------------------------------------------------------------------------
Begsr Sql_Close;

  Exec SQL Close list;
  Exsr Sql_Monitor;

Endsr;
//' --------------------------------------------------------------------------------
//' Sql_Monitor is the global monitor for sql-errors
//' The SQL_SetError sends the message back to the IceBreak server.
//' Then use 'GetLastError' to display the formatted error message
//' --------------------------------------------------------------------------------
Begsr Sql_Monitor;
   select;
      when SqlCode = 0;
      when SqlCode = 100;
      other;
         SQL_SetError(SqlCode:SqlErm);
%>
         <script>
         alert ("<%= GetLastError('*MSGTXT') %>");
         </script>
<%
   Endsl;
EndSr;
```

Show the sample   Run the sample

5.3 SQL to the ResponseObject

SQL:

You can use the Structured Query Language - SQL in several ways in IceBreak.

The classic way is to incorporate embedded-SQL into your code with Embedded SQL.

But IceBreak also lets you use SQL-result sets directly in your application. IceBreak has two build-in functions that return either a HTML-table or a complete XML document directly into your response object.

SQL to a HTML-table

If you just want to list the content of a SQL select statement (the result set) , then the **SQL_Execute_HTML** build-in function is the easiest way to incorporate database information into your application. This function formats the data values in respect to the data types in the result set.

The result is placed directly in the response-object just where the function is executed:

```
<%
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments++++++
++++++
D Error           s               N
D SqlCmd          s           1024    varying
D MaxRows         s             10i 0
/free
%>
<html>
<head>
 <link rel="stylesheet" type="text/css" href="/System/Styles/IceBreak.css"/>
</head>
<body><%

  *inlr   = *on;
  SqlCmd  = 'Select * from product';
  MaxRows = 1000;
  Error   = SQL_Execute_HTML(sqlcmd : maxrows);

  if (Error); %>
     <% = getLastError('*MSGTXT') %><br>
     <% = getLastError('*HELP')   %><%
  endif;
%>
</body>
</html>
```

Show the sample   Run the sample

The **SQL_Execute_HTML** build-in function returns logical *ON if an error occurs, which sets the "LastError" property. You can then retrieve the last error with **GetLastError('*MSGTXT | *HELP | *MSGID')** which returns the last error in a string.

The **SqlCmd** parameter can be any SQL select statement up to 32760 bytes long.

The **MaxRows** parameter determines the maximum number of rows allowed in the result set.

SQL to a XML-document

The **SQL_Execute_XML** build-in function returns the root element of an XML-document called **<resultset>.** Each row of the result set is called **<row>** and has an attribute named after the SQL column name.

However, you have to fill in the XML header with the version 1.0 and the encoding type of your choice - i.e. combine this function with **SetContentType()** . And ofcause you can append a formatting style sheet that reformats the XML into a XHTML document or what ever.

The result is placed directly in the response-object just where the function is executed:

```
<%
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++Comments++++++
++++++
D Error           s               N
D SqlCmd          s           1024    varying
D MaxRows         s             10i 0
/free
  SetContentType ('application/xml; charset=windows-1252');
  %><?xml version="1.0" encoding="windows-1252" ?>
  <%
  *inlr   = *on;
  SqlCmd  = 'Select * from product';
  MaxRows = 10000;
  Error   = SQL_Execute_XML(sqlcmd : maxrows);

  if (Error);
     %><error msg="<% = getLastError('*MSGTXT') %>"
              help="<% = getLastError('*HELP') %>"/><%
  endif;


%>
```

Show the sample   Run the sample

The **SQL_Execute_XML** build-in function returns logical *ON if an error occurs, which sets the "LastError" property. You can then retrieve the last error with **GetLastError('*MSGTXT | *HELP | *MSGID')** which returns the last error in a string.

The **SqlCmd** parameter can be any SQL select statement up to 32760 bytes long.

The **MaxRows** parameter determines the maximum number of rows allowed in the result set.

SQL to any custom format

The **SQL_Execute_Callback** build-in function calls your own custom function which makes you able to build any kind of output; from JSON to CSV etc. Basically it works as the above but you parse it your own function which is called for each cell in the result set -including the header.

The following sample creates a simple HTML list:

```
<%
/include qasphdr,sqlvar

d*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments++++++
++++++
d Error           s               N
d SqlCmd          s           1024    varying
d FromRow         s             10i 0
d MaxRows         s             10i 0

d RenderCell      pr            10i 0
d  CellValue                  4096    varying
d  Row                          10i 0 value
d  OfRows                       10i 0 value
d  Col                          10i 0 value
d  OfCols                       10i 0 value
d  SqlVar                             likeds(I_sqlvar)

/free
%>
<html>
 <head>
  <link rel="stylesheet" type="text/css" href="/System/Styles/IceBreak.css"/>
 </head>
<body>
<%

  SqlCmd = 'Select * '
         + 'from product '
         + 'order by MANUID';

  FromRow = 1;
  maxRows = 10000;

  SQL_Execute_CallBack(
      SqlCmd:                 //' The Select * from ...
      FromRow:                //' From row number in the result set where 1 is the first
      MaxRows:                //' Maximum number of rows returned in the resultset
      %paddr(RenderCell)      //' Custom Cell rendering call-back function
  );

  *inrt   = *on;
%>
</body>
</html><%
/end-free

//'
-----------------------------------------------------------------------------------------
--
//' The "RenderCell" is called for each cell in the result set
//'
-----------------------------------------------------------------------------------------
--
p*name++++++++++..b...................keywords+++++++++++++++++++++++++++++comments++++++
++++++
p RenderCell      B
d                 PI            10i 0
d  CellValue                  4096    varying
d  Row                          10i 0 value
d  Rows                         10i 0 value
d  Col                          10i 0 value
d  Cols                         10i 0 value
d  SqlVar                             likeds(I_sqlvar)
 /free

    Select;
```

```
       //' Row zero i the header only
       When ( row = 0);

          if (col = 1);
             %><H4>Number of rows in the result set: <%= %char(Rows) %></h4>
               <table><thead><%
          endif;

          %><th><% = sqlVar.SqlName  %></th><%

          if (col = cols);
             %></thead><%
          endif;


       //' Row one and abowe is cell data from the result set
       When ( row >= 1);
          if (col = 1); //' First coloumn
             %><tr><%
          endif;

          %><td><% = CellValue %></td><%

          if (col = cols); //' Last coloumn
             %></tr><%
          endif;

       //' Row -1 indicates that is an EOF indications - no data is returned. This event
always comes once as the last event
       When ( row = -1);
         %></table><%
   Endsl;

//' Return I_CONTINUE as long a you want to iterate. If you want to break the loop then
return I_BREAK
   return I_CONTINUE;

 /end-free
p RenderCell      e
%>
```

Show the sample   Run the sample

The **SQL_Execute_Callback** build-in function returns logical *ON if an error occurs, which sets the "LastError" property. You can then retrieve the last error with **GetLastError('*MSGTXT | *HELP | *MSGID')** which returns the last error in a string.

The **SqlCmd** parameter can be any SQL select statement up to 32760 bytes long.

The **FromRow** parameter is the starting row you want to retrieve from the result set. This makes it perfect to scroll through a dataset with a paging logic.

The **MaxRows** parameter determines the maximum number of rows you will have returned on each subsequent call.

The **Callback** is the procedure address of your cell function.

Initially it is call with just the header with the row value of zero.

Each subsequent call returns the relative row number in the sub-result-set.

Finally you receive a value of -1 which is the EOF indication that allows you to mach up the final response.

The SQLVAR template structure is defined in the QASPHDR file and is received with appropriate value for the actual cell. The cell value is formatted to a default string, but you have access to the binary data from with in the SQLVAR structure along with the database data type.

Advanced use of the SQL_Execute_Callback()

Suppose you want to produce an Excel spread sheet on the fly base on XML and all data is loaded using SQL. In that case **SQL_Execute_Callback()** build-in function is perfect. This solution has two components:

- A XML template file
- The ASPX source using **SQL_Execute_Callback()**

First let's take a look at the XML template. This template is produced by saving a spreadsheet from Excel in XML format. Then we have modified all dynamic data to be referred to by static include and tag names.

```
<?xml version="1.0"?>
<?mso-application progid="Excel.Sheet"?>
<Workbook xmlns="urn:schemas-microsoft-com:office:spreadsheet"
 xmlns:o="urn:schemas-microsoft-com:office:office"
 xmlns:x="urn:schemas-microsoft-com:office:excel"
 xmlns:ss="urn:schemas-microsoft-com:office:spreadsheet"
 xmlns:html="http://www.w3.org/TR/REC-html40"
 xmlns:x2="http://schemas.microsoft.com/office/excel/2003/xml"
 xmlns:udc="http://schemas.microsoft.com/data/udc"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:udcxf="http://schemas.microsoft.com/data/udc/xmlfile">
 <DocumentProperties xmlns="urn:schemas-microsoft-com:office:office">
  <Version>11.8132</Version>
 </DocumentProperties>
 <ExcelWorkbook xmlns="urn:schemas-microsoft-com:office:excel">
  <WindowHeight>10995</WindowHeight>
  <WindowWidth>21360</WindowWidth>
  <WindowTopX>0</WindowTopX>
  <WindowTopY>645</WindowTopY>
  <ProtectStructure>False</ProtectStructure>
  <ProtectWindows>False</ProtectWindows>
  <FutureVer>11</FutureVer>
 </ExcelWorkbook>

 <Styles>
  <Style ss:ID="Default" ss:Name="Normal">
   <Alignment ss:Vertical="Bottom"/>
   <Borders/>
   <Font/>
   <Interior/>
   <NumberFormat/>
   <Protection/>
  </Style>
  <Style ss:ID="s21">
   <Font ss:Bold="1"/>
  </Style>
  <Style ss:ID="s22">
   <NumberFormat/>
  </Style>
  <Style ss:ID="s23">
   <NumberFormat ss:Format="@"/>
  </Style>
  <Style ss:ID="s24">
   <NumberFormat ss:Format="Short Date"/>
  </Style>
 </Styles>

 <Worksheet ss:Name="<%= SheetName  %>">
  <Names>
   <NamedRange ss:Name="_FilterDatabase" ss:RefersTo="=<%= SheetName  %>!R1C1:R<%= %char
(Rows+1) %>C<%= %char(Cols) %>" ss:Hidden="1"/>
  </Names>

  <Table ss:ExpandedColumnCount="<%= %char(Cols) %>" ss:ExpandedRowCount="<%= %char(Rows
+1) %>" x:FullColumns="1" x:FullRows="1">

<!--#tag="col"-->
    <Column ss:Width="<% = %char(len * 2) %>"/>

<!--#tag="style"-->
   <Row>

<!--#tag="stylecell"-->
     <Cell ss:StyleID="s21"><Data ss:Type="String"><% = sqlHdr(i).sqlColName %></Data><
NamedCell ss:Name="_FilterDatabase"/></Cell>

<!--#tag="stylecellend"-->
   </Row>

<!--#tag="type"-->
   <Row>

<!--#tag="typecell"-->
     <Cell ss:StyleID="<% = style %>"><Data ss:Type="<% = type %>"><% = CellValue %></
Data><NamedCell ss:Name="_FilterDatabase"/></Cell>

<!--#tag="typeend"-->
   </Row>

<!--#tag="WorksheetOptions"-->
  </Table>
  <WorksheetOptions xmlns="urn:schemas-microsoft-com:office:excel">
   <Selected/>
   <Panes>
    <Pane>
     <Number>1</Number>
     <RangeSelection>R1C1:R<%= %char(Rows+1) %>C<%= %char(Cols) %></RangeSelection>
    </Pane>
   </Panes>
   <ProtectObjects>False</ProtectObjects>
   <ProtectScenarios>False</ProtectScenarios>
  </WorksheetOptions>
 </Worksheet>
 <x2:MapInfo x2:HideInactiveListBorder="false">
  <x2:Schema x2:ID="Schema1" x2:Namespace="">
   <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element nillable="true" name="resultset">
     <xsd:complexType>
      <xsd:sequence minOccurs="0">
       <xsd:element minOccurs="0" maxOccurs="unbounded" nillable="true" name="row" form
="unqualified">
        <xsd:complexType>

<!--#tag="attribute"-->
          <xsd:attribute name="<% = sqlHdr(i).SqlColName %>" form="unqualified" type
="xsd:<% = type %>"></xsd:attribute>

<!--#tag="attributeend"-->
        </xsd:complexType>
       </xsd:element>
      </xsd:sequence>
     </xsd:complexType>
    </xsd:element>
   </xsd:schema>
  </x2:Schema>
  <x2:Map x2:ID="resultset_Map" x2:SchemaID="Schema1" x2:RootElement="resultset">
   <x2:Entry x2:Type="table" x2:ID="1" x2:ShowTotals="true">
    <x2:Range><%= SheetName  %>!R2C1:R<%= %char(Rows+1) %>C<%= %char(Cols) %></x2:Range>
    <x2:HeaderRange>R1C1</x2:HeaderRange>
    <x:FilterOn>True</x:FilterOn>
    <x2:XPath>/resultset/row</x2:XPath>

<!--#tag="field"-->
    <x2:Field x2:ID="<% = sqlHdr(i).SqlColName %>">
     <x2:Range>RC[<%= %char(i-1) %>]</x2:Range>
     <x2:XPath>@<% = sqlHdr(i).SqlColName %></x2:XPath>
     <x2:XSDType><% = type %></x2:XSDType>
     <ss:Cell/>
     <x2:Aggregate>None</x2:Aggregate>
    </x2:Field>

<!--#tag="fieldend"-->
   </x2:Entry>
  </x2:Map>
 </x2:MapInfo>

<!--#tag="workbookend"-->
</Workbook>
```

This is a bit long, however that is required to make a XML list in Excel. Please note that the template is prepared for static include so RPG variables and buldings are used in the template

Now let's have a look at the ASPX code.

```
<%
/include qasphdr,sqlvar

d*Name+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments++++++
++++++
d Sql_Cmd         s           1024    varying

D start           s             10i 0
D limit           s             10i 0
d sheetname       s             32    varying
d style           s             16    varying
d type            s             32    varying
d q               s              1    inz('''')
D i               s             10i 0
D len             s             10i 0
d SqlHdr          ds                  dim(200) likeds(I_sqlHdr)

d RenderCell      pr              n
d  CellValue                  4096    varying
d  Row                          10i 0 value
d  Rows                         10i 0 value
d  Col                          10i 0 value
d  Cols                         10i 0 value
d  SqlVar                             likeds(I_sqlvar)

d CallbackHead    pr
d  Cols                         10i 0 value
d  SqlHdr                             dim(200) likeds(I_sqlHdr)

/free
//'
-----------------------------------------------------------------------------------------
--
//' Mainline
//'
-----------------------------------------------------------------------------------------
--
```

```
//' These 3 lines causes:
//' 1) Excel to open it
//' 2) prompt for a download
//' 3) Keep the temporary file to stay in the cache until excell has it open
   SetContentType('application/vnd.ms-excel; charset=utf-8');
   SetHeader ('content-disposition'
             : 'attachment; filename=AnXMLexcelList.xls');
   SetCacheTimeout(10);

   sheetname = 'Sheet1';
   start = 1;
   limit = 99999; //' allways all !!!

   Sql_Cmd = 'Select * '
           + 'from product ';

   SQL_Execute_Header(
         sql_cmd:
         %paddr(CallbackHead)
   );

   SQL_Execute_CallBack(
         sql_cmd:              //' The Select * from ...
         Start:                //' From row number in the result set where 1 is the first
         Limit:                //' Maximum number of rows returned in the resultset
         %paddr(RenderCell)    //' Custom Cell rendering call-back function
   );
   *inrt   = *on;

 /end-free
//'
-----------------------------------------------------------------------------------------
--
//' This is called once when the coloumn names are ready
//' we only copy the header array to a global array for later use
//'
-----------------------------------------------------------------------------------------
--
p*name++++++++++..b...................keywords+++++++++++++++++++++++++++++comments++++++
++++++
p CallbackHead    B
d                 PI
d  Cols                         10i 0 value
d  Sql_Hdr                             dim(200) likeds(I_sqlHdr)

d i               s             10i 0
d comma           s              1    varying
 /free
   SqlHdr = Sql_Hdr;
 /end-free
p CallbackHead    E
//'
-----------------------------------------------------------------------------------------
--
//' The renderCell is called for each cell in the result set. here we produce the
complete XML
//'
-----------------------------------------------------------------------------------------
--
p*name++++++++++..b...................keywords+++++++++++++++++++++++++++++comments++++++
++++++
p RenderCell      B
d                 PI               N
d  CellValue                  4096    varying
d  Row                          10i 0 value
d  Rows                         10i 0 value
d  Col                          10i 0 value
d  Cols                         10i 0 value
d  SqlVar                             likeds(I_sqlvar)
```

```
 /free

//' Row zero is the header only. from row = 1 etc it also contains data
//' So we build the initial XML stuff when the first column is detected
    if (row = 0 and col = 1);
    %><!--#include file="ex09sqlExcel.xml"  tag="*FIRST"--><%

       for i = 1 to cols;
          len = %len(%trim(sqlHdr(i).sqlColName)) * 4;
          if ( sqlHdr(i).SqlLen >  len) ;
             len = sqlHdr(i).SqlLen;
          endif;
          %><!--#include file="ex09sqlExcel.xml"  tag="col"--><%

       endfor;
       %><!--#include file="ex09sqlExcel.xml"  tag="style"--><%
       for i = 1 to cols;
          %><!--#include file="ex09sqlExcel.xml"  tag="stylecell"--><%
       endfor;
        %><!--#include file="ex09sqlExcel.xml"  tag="stylecellend"--><%
//' from row = 1 and forward it also contains data from the resultset
//' we build each data cell here
    elseif (row > 0);
      if (col = 1);
        %><!--#include file="ex09sqlExcel.xml"  tag="type"--><%
      endif;
      select;
      when sqlHdr(col).SqlType >= 480
      and  sqlHdr(col).SqlType <= 490;
         type  = 'Number';
         style = 's22';
      when sqlHdr(col).SqlType >= 491
      and  sqlHdr(col).SqlType <= 501;
         type = 'Number';
         style = 's22';
      other;
         type = 'String';
         style = 's23';
      endsl;

      %><!--#include file="ex09sqlExcel.xml"  tag="typecell"--><%

      if (col = cols);
         %><!--#include file="ex09sqlExcel.xml"  tag="typeend"--><%
      endif;

//' row -1 is the EOF indication. This event always comes once at last
//' here we finalize the XML with all the xPath stuff
    elseif (row = -1 );
       %><!--#include file="ex09sqlExcel.xml"  tag="WorksheetOptions"--><%

       for i = 1 to cols;
          select;
          when sqlHdr(i).SqlType >= 480
          and  sqlHdr(i).SqlType <= 490;
             type = 'double';
          when sqlHdr(i).SqlType >= 491
          and  sqlHdr(i).SqlType <= 501;
             type = 'integer';
          other;
             type = 'string';
          endsl;
          %><!--#include file="ex09sqlExcel.xml"  tag="attribute"--><%
       endfor;
       %><!--#include file="ex09sqlExcel.xml"  tag="attributeend"--><%
       for i = 1 to cols;
          select;
          when sqlHdr(i).SqlType >= 480
          and  sqlHdr(i).SqlType <= 490;
             type = 'double';
          when sqlHdr(i).SqlType >= 491
          and  sqlHdr(i).SqlType <= 501;
             type = 'integer';
          other;
             type = 'string';
          endsl;
          %><!--#include file="ex09sqlExcel.xml"  tag="field"--><%
       endfor;
       %><!--#include file="ex09sqlExcel.xml"  tag="fieldend"--><%

       %><!--#include file="ex09sqlExcel.xml"  tag="workbookend"--><%
    endif;

//' Return *ON as long a you want to itterate. If you want to break the loop then return
*OFF
    return *ON;

/end-free
p RenderCell
```

Note that the CallbackHead() is using array data structures as parameters. This data type was introduced in i5/OS R5V2M0. This Admin pages can run back to R5V1M0 so you have to copy the source to your own server instance with a target release ar R5V2M0 or above - to try it out.

Part VI

6 XML, Webservices and proxies

6.1 WebServices - the easy way

WebServices:

WebServices is the backbone in Service Oriented Architecture - SOA.

IceBreak lets you create SOA complaint WebServices from any RPG service program in few easy steps.

- Set the compiler directive **type="webservice".**
- Set the **"export"** keyword on the function you want to populate.
- Use the IceBreak keyword extentions **Input, Output or InOut** on the parameter list.
- Suffix your source file with the **.asmx** extension.

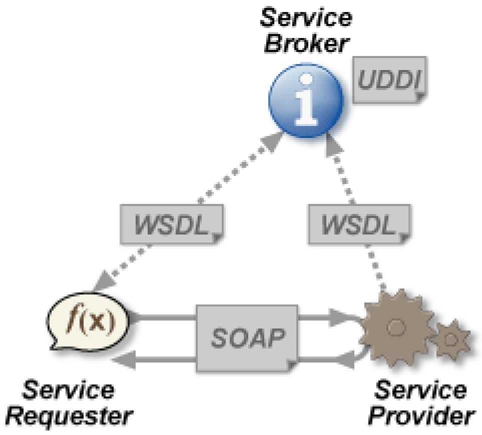
When you refers to the WebService from the Browser URL IceBreak will:

- Compile the RPG program into a service program
- Create a source file QSOAPHDR with prototypes for your functions, so you can reuse them in a non SOAP environment.
- Build the SOAP wrapper functions for your exported functions (They have the same name suffixed by an "_")
- Build a WSDL file that describes how to invoke the webservice

What is SOAP and WSDL

Soap is an XML document that is sent back and forth between the WebService consumer and the WebService provider with the parameters and which function to invoke.

The WSDL describes the invocation, functions names and their parameters and types.



Now lets se how to create the WebService - step by step.a

1

Create a webservice source

Open your favorite editor and paste the following in

```
<%@ language="RPGLE" pgmtype="webservice" %>
<%
h nomain

p*name++++++++++..b...................keywords++++++++++++++++++++++++++comments+++++++++
+++
p Calculator      B                   export

d Calculator      pi
d    x                          10i 0 Input
d    y                          10i 0 Input
d    z                          10i 0 Output

/free

  z = x + y;

/end-free
p Calculator      E
%>
```

This sample takes **x** and **y** as input parameters and calculates the sum **z.**

As you can see - it is pretty much a normal service program except for the input/output keywords in the D-Spec.

basically it is a normal service program with no mainline. It simply exports a number of procedures. You are not restricted to only one.

The following data types are supported:

i5/OS type	WebService type
char	s:string
packed	s:decimal
zoned	s:decimal
integers	s:decimal
float	s:decimal
time	s:time
date	s:date
timestamp	s:datetime
pointer to ILOB	s:string

You can pass "atomic" parameters and arrays. Also, you can pass external described data structures as complex types. Even arrays of externally described structures which becomes arrays of complextypes

If you want to pass arrays or data structures, then click here to see howto use arrays in webservices

You can use ILOB's to pass large data chunks (12MB) from and to the WebService. Click here to see how to use ILOB's in WebServices

2

Save the WebService

Now save the webservice as **WebService.asmx** in your development server directory**.** The suffix ASMX is very important, since it causes IceBreak to load the service program and locate the requested WebService functions within.

3

Compile the WebService

Simply Refer to the **WebService.asmx?WSDL** from a browser ... and that's it.

The **?WSDL** causes the WebService to return its own definition to the rest of the world.

If you had made any typos you will see the compilation post list as usual.

You are done !! The WebService is ready to use.

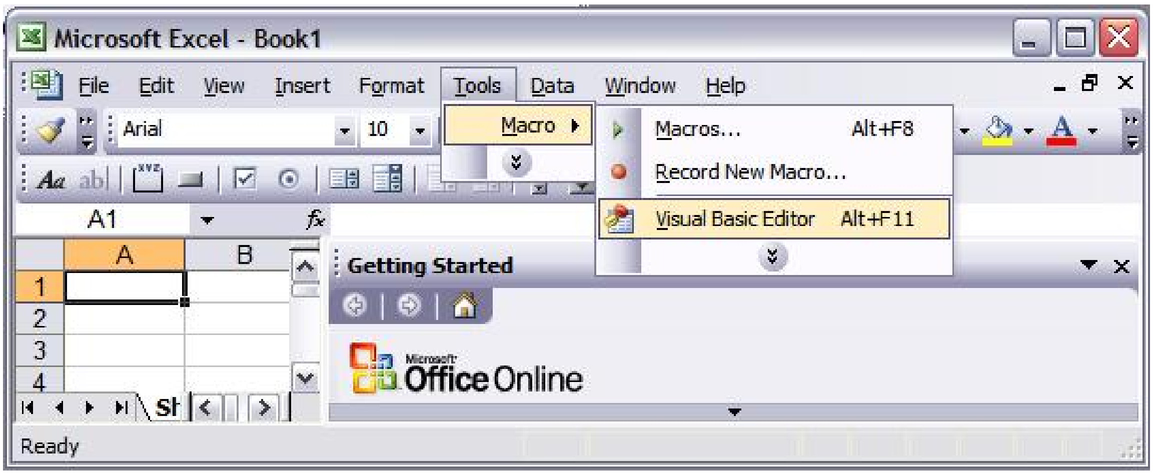
4

Use the WebService

You can use the WebService from WebSphere or .NET or any other SOA complaint architecture. One easy place is to integrate it into Microsoft office Word or Excel.

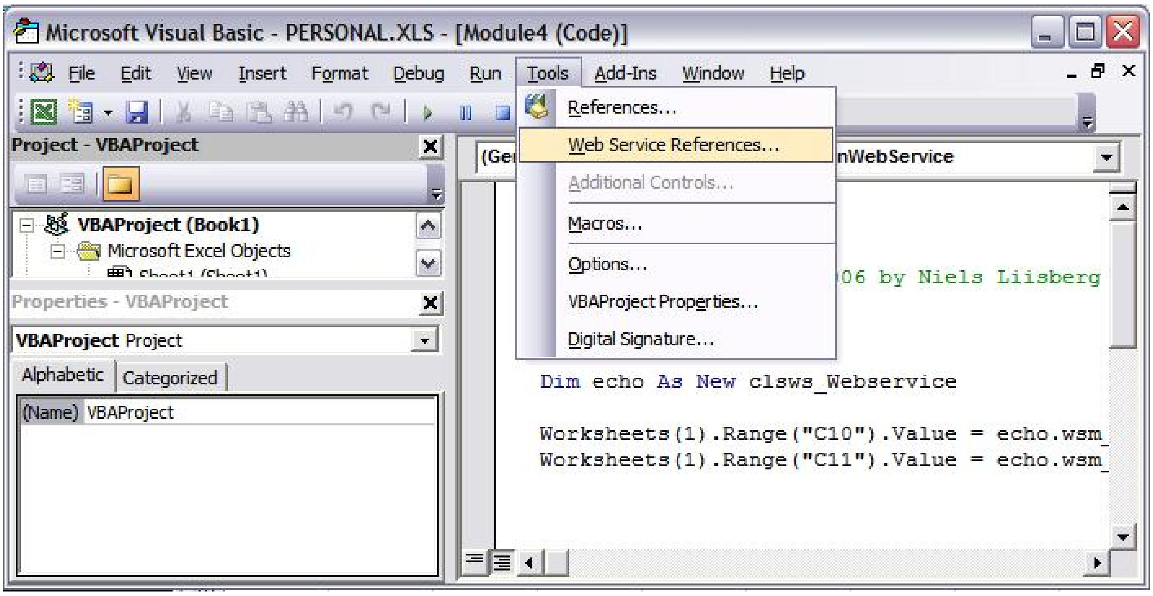
Here is an Excel example:

- Open Excel
- Click Tools
- Click Macro
- Click "Visual Basic Editor"



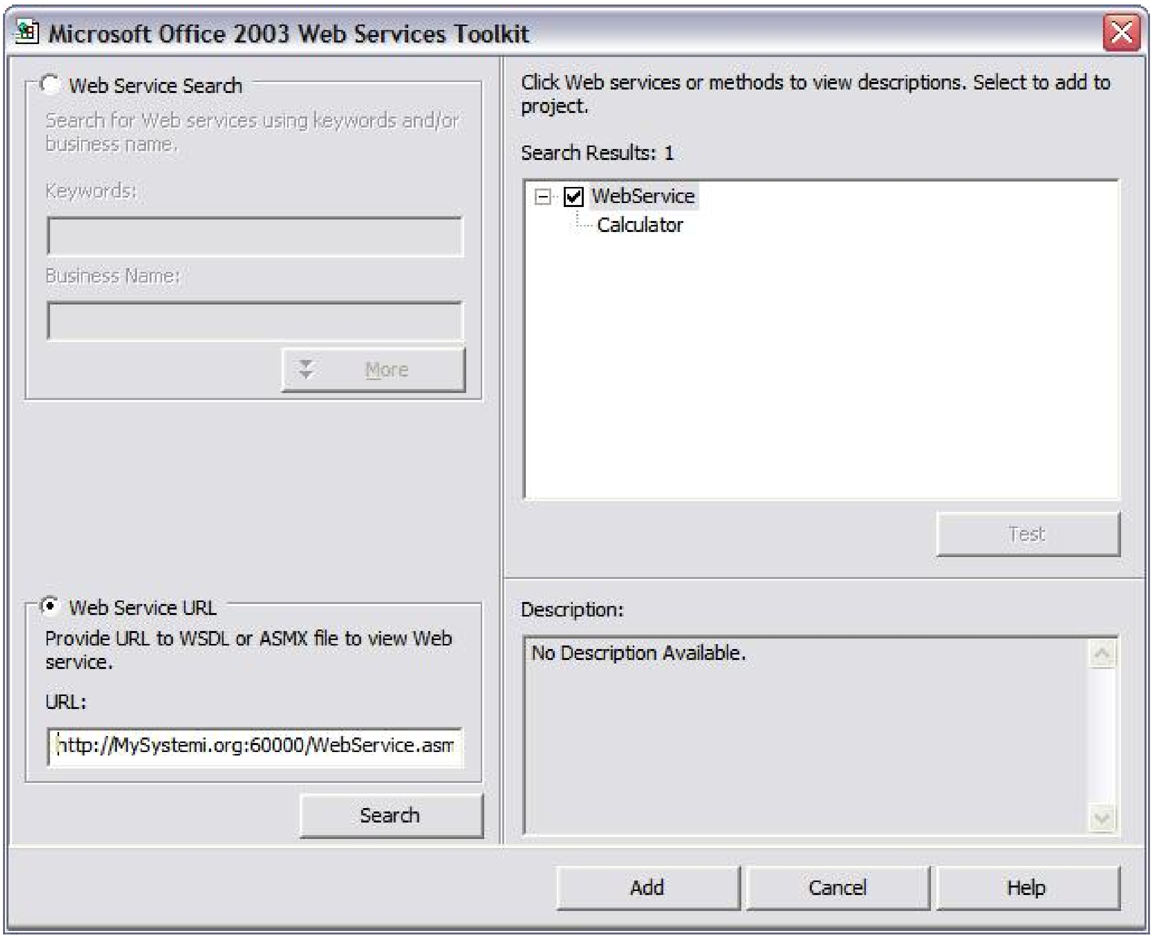
- Click "Tools"
- Click "Web Service References" (If it don't show - then download webservices for office tools from

Microsoft, it is free)



```
06 by Niels Liisberg

Dim echo As New clsws_Webservice

Worksheets(1).Range("C10").Value = echo.wsm_
Worksheets(1).Range("C11").Value = echo.wsm_
```

- Now click in "WebService URL"
- Enter the URL to your IceBreak server and the name of the WebService
- Click "Search"
- In the right panel your webservice is shown
- "Check" the "WebService"
- Click "add"



Microsoft Office 2003 Web Services Toolkit

○ Web Service Search

Search for Web services using keywords and/or business name.

Keywords:

Business Name:

● Web Service URL

Provide URL to WSDL or ASMX file to view Web service.

URL: http://MySystemi.org:60000/WebService.asm

Click Web services or methods to view descriptions. Select to add to project.

Search Results: 1

☑ WebService

Calculator

Description:

No Description Available.

Now all the code needed is created for you now just enter the following macro and you are ready:

```
Sub Calc() Dim Calculator As New clsws_WebService
   Worksheets(1).Range("C1").Value = _
      Calculator.wsm_Calculator( _
         Worksheets(1).Range("A1").Value, _
         Worksheets(1).Range("B1").Value _
      )
End Sub
```

The macro above takes cell A1 and B1 and call the WebService which in turn calculates the sum and place it in cell C1

6.1.1 Using arrays in webservices

On the service procedure you can specify the RPG keyword dim() to give an array a dimension. However it is not necessary all elements that is transmitted. To determine the number of elements that is received or sent by the SOAP service, you can call the IceBreak build in function **soap_ArrayElements()**. Beware that you need to pass the address of the array.

if you want to detect the number of elements received:

```
NumberOfReceivedElements  = soap_ArrayElements(%addr(InputArray));
```

if you want to set the number of array element that is returned to the soap client:

```
soap_ArrayElements(%addr(InputArray) : NumberOfElementeToSend);
```

This sample code receives an array of keys. For each key it returns the value fetched from the database in a corresponding output element to the client:

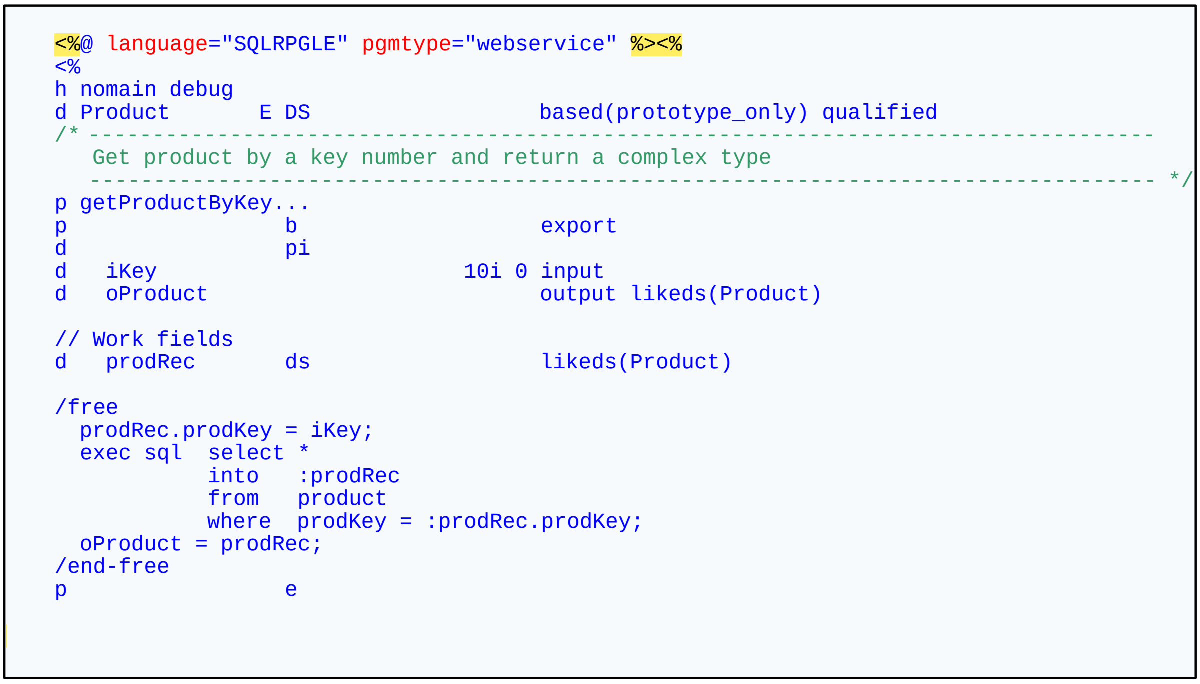
```
<%@ language="RPGLE" pgmtype="webservice" %>
<%
h nomain

fproduct   if   e           k disk

p*name+++++++++++..b...................keywords+++++++++++++++++++++++++comments++++++++++++
p Products        B                    export

d Products        pi
d    iKeys                       10i 0 Input  dim(1000)
d    oDesc                      256    Output dim(1000) varying
d    oPrice                      11  2 Output dim(1000)
d    oStockDate                    D   Output dim(1000)

d i               s              10i 0
d activeElements  s              10i 0
/free

  // Store number of elements received
  activeElements = soap_ArrayElements(%addr(ikeys));

  // read each product from the input array of active elements
  for i = 1 to activeElements;
    ProdKey = ikeys(i);
    chain ProdKey ProductR;
    if %found;
      oDesc(i) = Desc;
      oPrice(i) = Price;
      oStockDate(i) = stockDate;
    else;
      oDesc(i) = 'N/A';
      oPrice(i) = 0;
      oStockDate(i) = %date();
    endif;
  endFor;

  // Set the size of theh output array to return to the client
  soap_ArrayElements(%addr(oDesc)      : activeElements);
  soap_ArrayElements(%addr(oPrice)     : activeElements);
  soap_ArrayElements(%addr(oStockDate) : activeElements);

/end-free
p Products        e
%>
```

6.1.2 Using external described datastructures in webservices

SOAP services supports "complext types" which is a collection of atomic fields. This concept maps quite well the the data-structure concept in RPG. However, datastructures in RPG support overlaps, pointers, sub-definitions and other tweaks which are not good pratice in a service oriented architecture.

So IceBreak uses the best form the two worlds. By introducing external described data-structures as complex types you will have two fishes in the same catch: Dynamic defined data based on DDS or SQL together with the possibility to expose these structures in WebServices. And you can even make arrays of comples types.

Lets have an example:



```
<%@ language="SQLRPGLE" pgmtype="webservice" %><%
<%
h nomain debug
d Product       E DS                  based(prototype_only) qualified
/* -----------------------------------------------------------------------------------
   Get product by a key number and return a complex type
   ----------------------------------------------------------------------------------- */
p getProductByKey...
p                 b                   export
d                 pi
d   iKey                        10i 0 input
d   oProduct                          output likeds(Product)

// Work fields
d   prodRec       ds                  likeds(Product)

/free
  prodRec.prodKey = iKey;
  exec sql  select *
            into   :prodRec
            from   product
            where  prodKey = :prodRec.prodKey;
  oProduct = prodRec;
/end-free
p                 e
```

The magic begins at :

```
d Product       E DS                  based(prototype_only) qualified
```

This line of code will go to the library and import the definition of "Product" . The "based(prototype_only) " instructs the compiler to not assign any memory to that definition, but rather use the definition as a template.

The next magical line is the:

```
d   oProduct                          output likeds(Product)
```

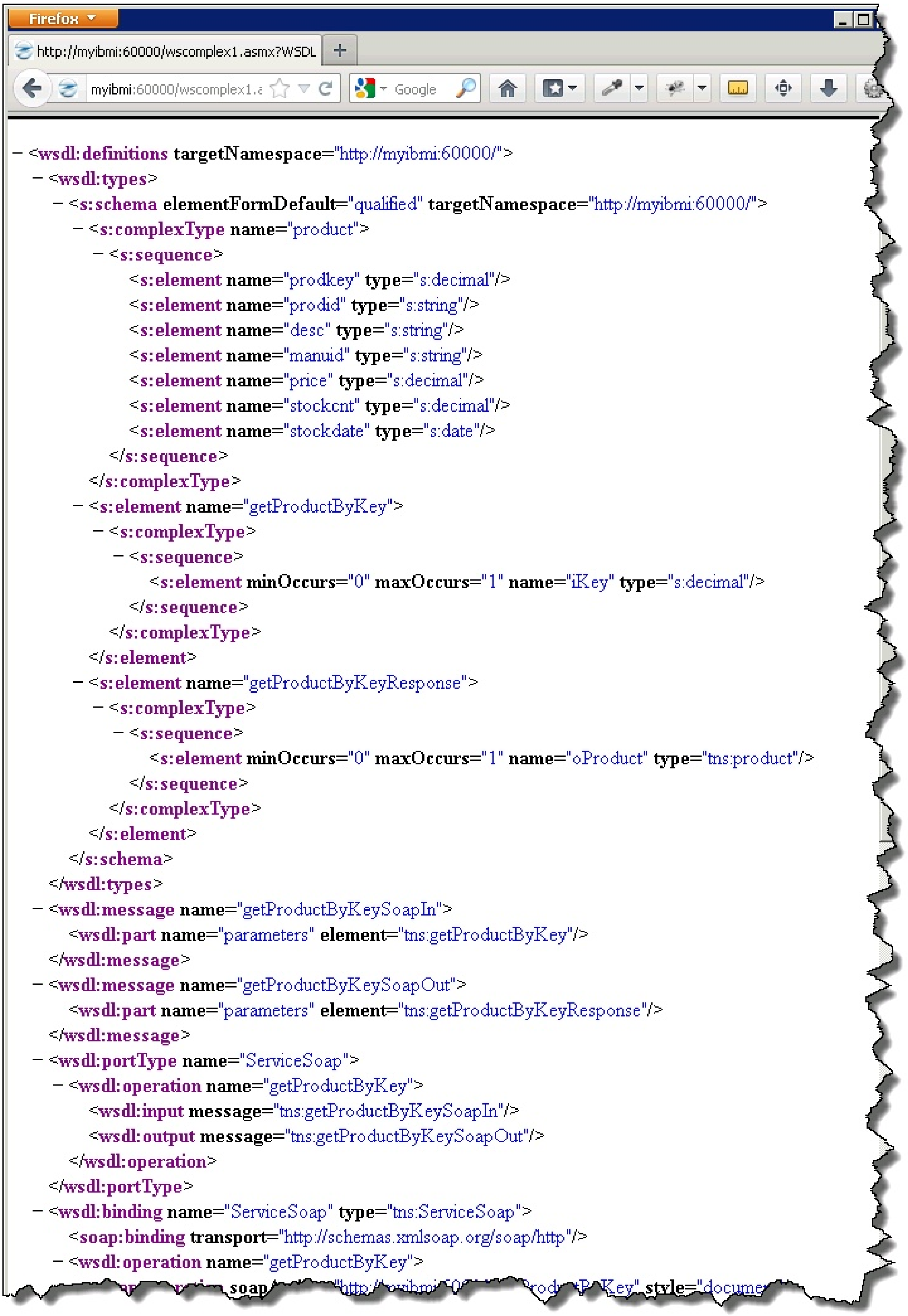
The oProduct output parameter with will have the same fields as the "product E DS" template form above. The "output" instructs the IceBreak pre-compiler to create a SOAP wrapper and build a WSDL definition for all the fields in that data-structure and, as well, include the structure as a "complext type" in the WSDL

Now we can use a simple SQL to fetch all the columns in one go. Take a look of the beauty - actually we have no real internal hard-coded definition of data in this little sample.

Now lets try this WebService:

1. Save the above code as wscomplex1.asmx on the IFS for any test IceBreak server.
2. Open your browser and refer to the WSDL definition of this service so it compiles - like: http://myIBMi:60000/wscomplex1.asmx?WSDL

Now you can see that the externally described structure is expanded in to a complex type in the WSDL. The complex type "product" are used as a return parameter on the SOAP call "getProductByKeyResponse "

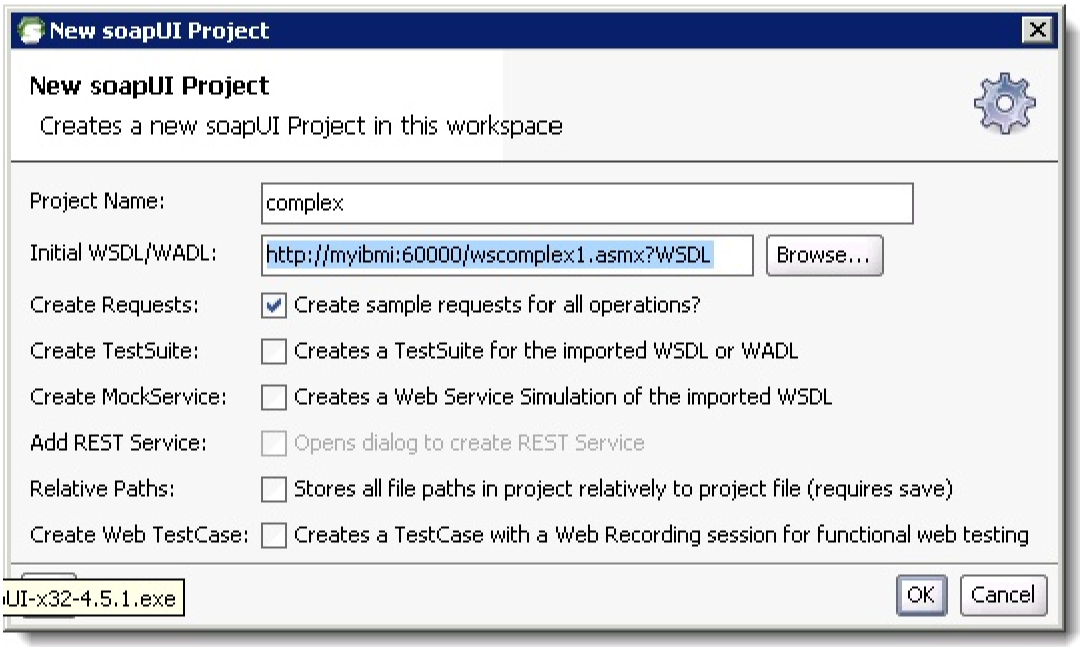

Firefox
http://myibmi:60000/wscomplex1.asmx?WSDL
myibmi:60000/wscomplex1.a
Google

```
- <wsdl:definitions targetNamespace="http://myibmi:60000/">
  - <wsdl:types>
    - <s:schema elementFormDefault="qualified" targetNamespace="http://myibmi:60000/">
      - <s:complexType name="product">
        - <s:sequence>
            <s:element name="prodkey" type="s:decimal"/>
            <s:element name="prodid" type="s:string"/>
            <s:element name="desc" type="s:string"/>
            <s:element name="manuid" type="s:string"/>
            <s:element name="price" type="s:decimal"/>
            <s:element name="stockcnt" type="s:decimal"/>
            <s:element name="stockdate" type="s:date"/>
          </s:sequence>
        </s:complexType>
      - <s:element name="getProductByKey">
        - <s:complexType>
          - <s:sequence>
              <s:element minOccurs="0" maxOccurs="1" name="iKey" type="s:decimal"/>
            </s:sequence>
          </s:complexType>
        </s:element>
      - <s:element name="getProductByKeyResponse">
        - <s:complexType>
          - <s:sequence>
              <s:element minOccurs="0" maxOccurs="1" name="oProduct" type="tns:product"/>
            </s:sequence>
          </s:complexType>
        </s:element>
      </s:schema>
    </wsdl:types>
  - <wsdl:message name="getProductByKeySoapIn">
      <wsdl:part name="parameters" element="tns:getProductByKey"/>
    </wsdl:message>
  - <wsdl:message name="getProductByKeySoapOut">
      <wsdl:part name="parameters" element="tns:getProductByKeyResponse"/>
    </wsdl:message>
  - <wsdl:portType name="ServiceSoap">
    - <wsdl:operation name="getProductByKey">
        <wsdl:input message="tns:getProductByKeySoapIn"/>
        <wsdl:output message="tns:getProductByKeySoapOut"/>
      </wsdl:operation>
    </wsdl:portType>
  - <wsdl:binding name="ServiceSoap" type="tns:ServiceSoap">
      <soap:binding transport="http://schemas.xmlsoap.org/soap/http"/>
    - <wsdl:operation name="getProductByKey">
```

Let's test the SOAP service:

Open your SoapUI - if you don't have a copy, then you can download and install it from here: http://sourceforge.net/projects/soapui/ or purchase it from http://www.soapui.org/

Now create an new project: Lets call it "complex" and enter the location of you webservice's WSDL, which is the name of youe service with the amsx extension and WSDL as the URL parameter (query string):



New soapUI Project

New soapUI Project
Creates a new soapUI Project in this workspace

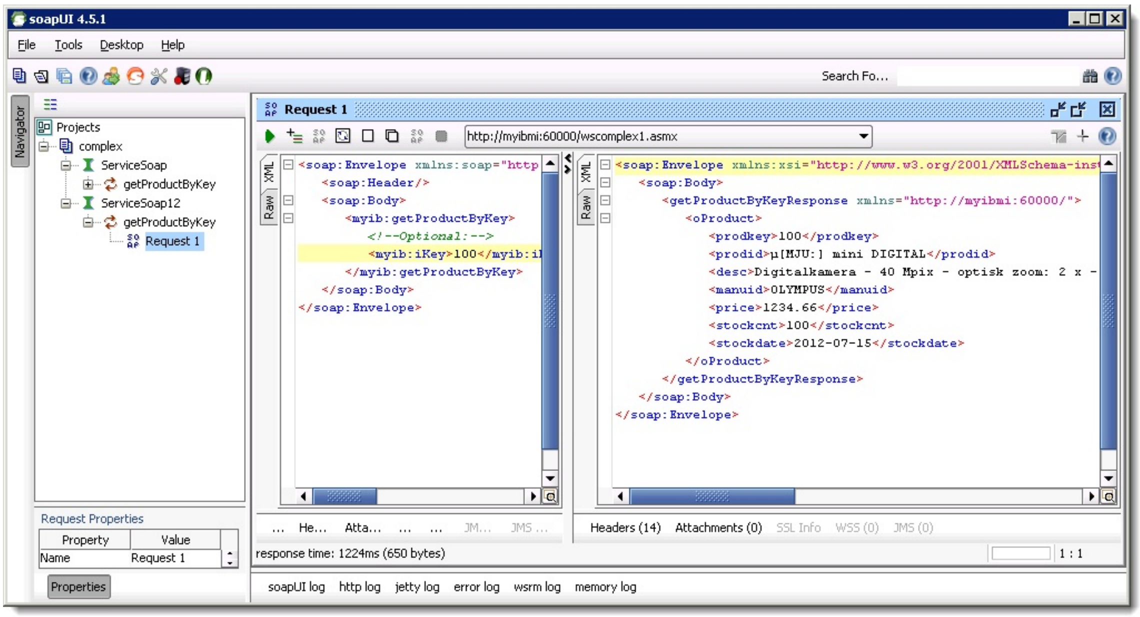
Project Name: complex
Initial WSDL/WADL: http://myibmi:60000/wscomplex1.asmx?WSDL
Create Requests: ☑ Create sample requests for all operations?
Create TestSuite: □ Creates a TestSuite for the imported WSDL or WADL
Create MockService: □ Creates a Web Service Simulation of the imported WSDL
Add REST Service: □ Opens dialog to create REST Service
Relative Paths: □ Stores all file paths in project relatively to project file (requires save)
Create Web TestCase: □ Creates a TestCase with a Web Recording session for functional web testing

Press the "OK" and let Soap-UI build the SOAP interfaces for you.

In the "projects"

1. Expand the "getProductByKey".
2. Fill in "100" in the **iKey** parameter.
3. Now press the green > run button.

Now you should see something similar to this:



```
<soap:Envelope xmlns:soap="http
   <soap:Header/>
   <soap:Body>
      <myib:getProductByKey>
         <!--Optional:-->
         <myib:iKey>100</myib:i
      </myib:getProductByKey>
   </soap:Body>
</soap:Envelope>
```

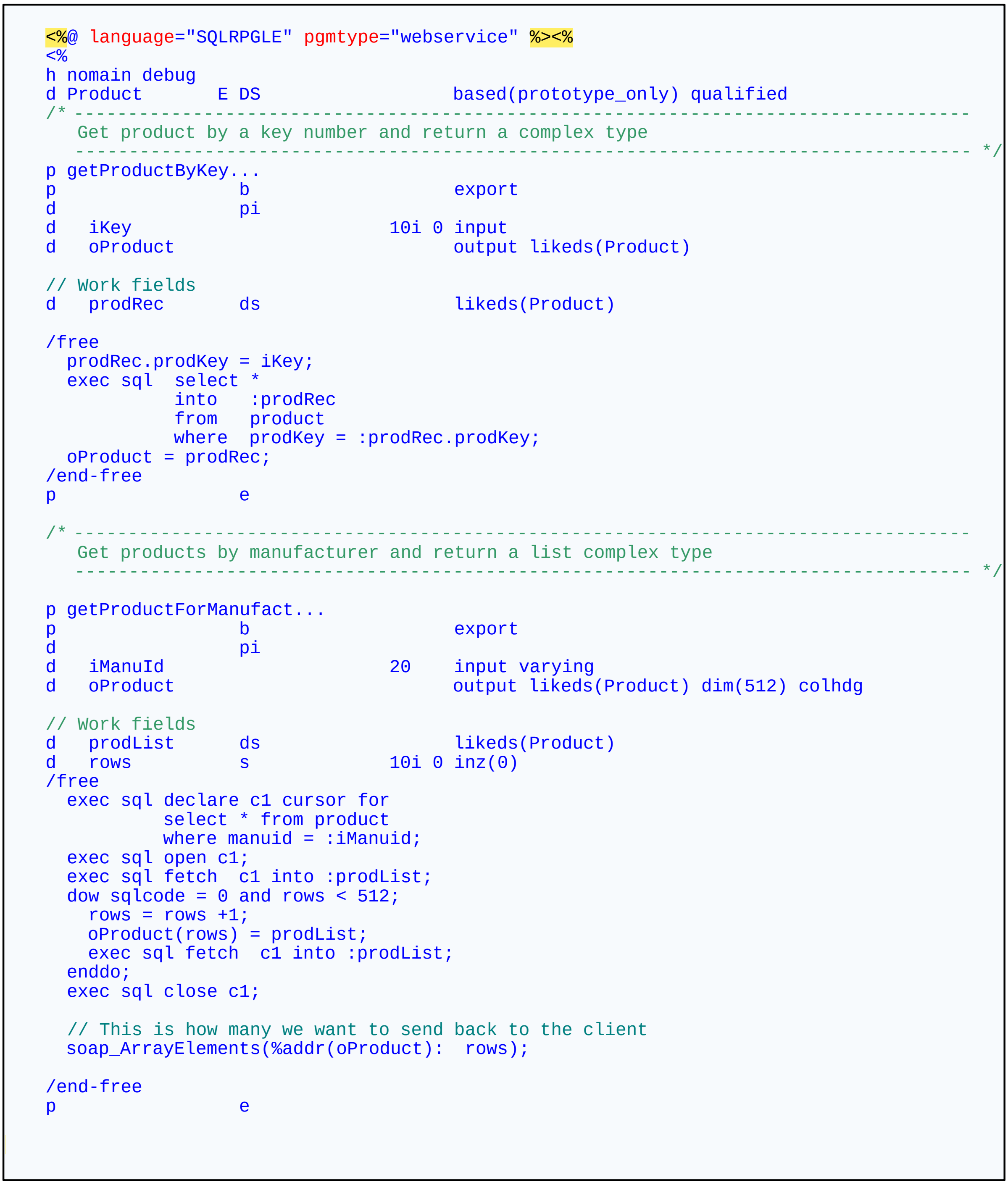
```
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-inst
   <soap:Body>
      <getProductByKeyResponse xmlns="http://myibmi:60000/">
         <oProduct>
            <prodkey>100</prodkey>
            <prodid>µ[MJU:] mini DIGITAL</prodid>
            <desc>Digitalkamera - 40 Mpix - optisk zoom: 2 x -
            <manuid>OLYMPUS</manuid>
            <price>1234.66</price>
            <stockcnt>100</stockcnt>
            <stockdate>2012-07-15</stockdate>
         </oProduct>
      </getProductByKeyResponse>
   </soap:Body>
</soap:Envelope>
```

response time: 1224ms (650 bytes)

List of complex types

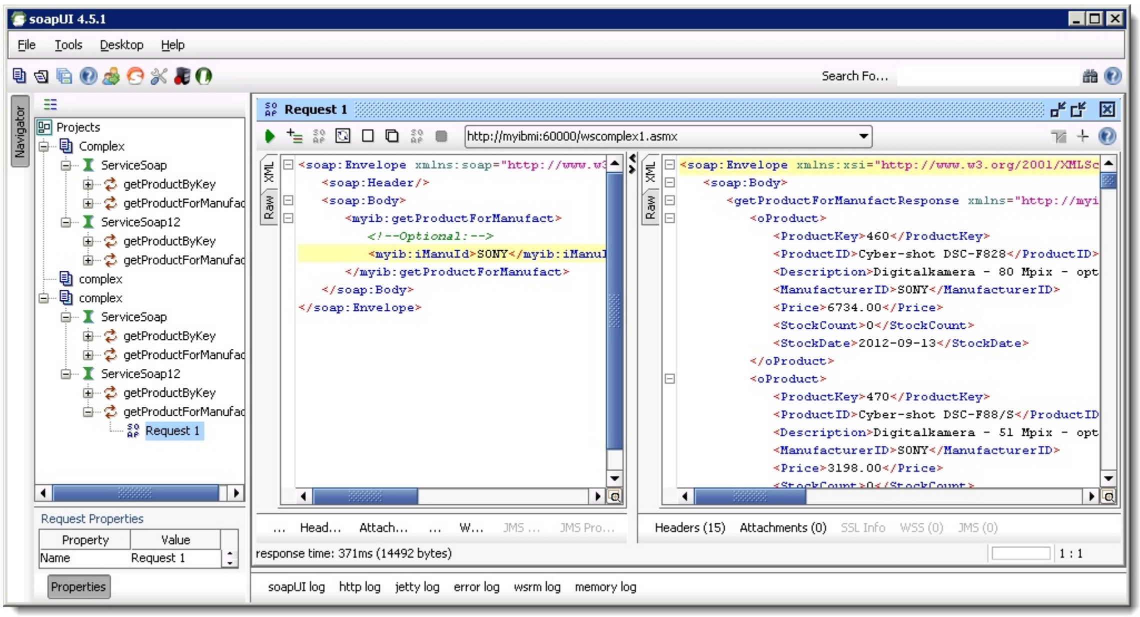
Now lets add a new webservice method that returns a list product from a manufacturer. Basically we want to return an array of complex types. And all we need to do is to create a loop on the SQL select statement, so we need a SQL cusrsor.

From the webservice perspective, we now need to define the array and this is done by a "**dim**" option on out output. Add the following `getProductForManufact` to you code:



```
<%@ language="SQLRPGLE" pgmtype="webservice" %><%
<%
h nomain debug
d Product       E DS                  based(prototype_only) qualified
/* ------------------------------------------------------------------------------------
   Get product by a key number and return a complex type
   ------------------------------------------------------------------------------------ */
p getProductByKey...
p                 b                   export
d                 pi
d   iKey                        10i 0 input
d   oProduct                          output likeds(Product)

// Work fields
d   prodRec       ds                  likeds(Product)

/free
  prodRec.prodKey = iKey;
  exec sql  select *
            into   :prodRec
            from   product
            where  prodKey = :prodRec.prodKey;
  oProduct = prodRec;
/end-free
p                 e

/* ------------------------------------------------------------------------------------
   Get products by manufacturer and return a list complex type
   ------------------------------------------------------------------------------------ */

p getProductForManufact...
p                 b                   export
d                 pi
d   iManuId                     20    input varying
d   oProduct                          output likeds(Product) dim(512) colhdg

// Work fields
d   prodList      ds                  likeds(Product)
d   rows          s             10i 0 inz(0)
/free
  exec sql declare c1 cursor for
           select * from product
           where manuid = :iManuid;
  exec sql open c1;
  exec sql fetch  c1 into :prodList;
  dow sqlcode = 0 and rows < 512;
    rows = rows +1;
    oProduct(rows) = prodList;
    exec sql fetch  c1 into :prodList;
  enddo;
  exec sql close c1;

  // This is how many we want to send back to the client
  soap_ArrayElements(%addr(oProduct):  rows);

/end-free
p                 e
```

Let's re-create the Soap-UI project and run it:



```
<soap:Envelope xmlns:soap="http://www.w3
   <soap:Header/>
   <soap:Body>
      <myib:getProductForManufact>
         <!--Optional:-->
         <myib:iManuId>SONY</myib:iManuI
      </myib:getProductForManufact>
   </soap:Body>
</soap:Envelope>
```

```
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSc
   <soap:Body>
      <getProductForManufactResponse xmlns="http://myi
         <oProduct>
            <ProductKey>460</ProductKey>
            <ProductID>Cyber-shot DSC-F828</ProductID>
            <Description>Digitalkamera - 80 Mpix - opt
            <ManufacturerID>SONY</ManufacturerID>
            <Price>6734.00</Price>
            <StockCount>0</StockCount>
            <StockDate>2012-09-13</StockDate>
         </oProduct>
         <oProduct>
            <ProductKey>470</ProductKey>
            <ProductID>Cyber-shot DSC-F88/S</ProductID
            <Description>Digitalkamera - 51 Mpix - opt
            <ManufacturerID>SONY</ManufacturerID>
            <Price>3198.00</Price>
            <StockCount>0</StockCount>
```

response time: 371ms (14492 bytes)

Now we have an array of 512 elements we can return to the client; eact row is produced by a SQL fetch the **oProduct** is the list we are returning so we finally use the soap_ArrayElements to set the number of active elements in the array - this is managed by the "**row**" counter.

Also notice we want some more advanced names in our SOAP service. So in the "**likeds"** you provide the "**colhdg**" IceBreak keyword on your **output** or **input** structures. IceBreak will convert the column heading from the externally described file/data-structure to the complex type field names. Internally in you RPG source you just work with the field names of externally described data structure, but from the SOAP service / the clients point of view we are use the coloumnheadings as names. Therefor: Beware the column heading can confirm to valid SOAP-field names, otherwise you client will not be able to consume the WSDL and therefore the SOAP service you are providing.

Final note:

This sample illustrate how easy you can provide a SOAP web-service using IceBreak and RPGLE. Your WSDL is automatically created, and in this sample you basically have no internal described data. Everything is provided by IBMi data management. In this sample we are only dealing with list of complex types send back to the client, you can however, as easy consume arrays of complex types by replacing the "**output**" with the "**input**" or "**inout**" keyword.

6.2 Using Proxy. Making HTTP request to web servers

httpRequest:

A proxy is a technique to execute a web-request on another web server.

What it does

IceBreak has a built-in "virtual browser" so you are able to perform "GET" and "POST" functions in your ASPX program. This allows you to send parameters along the URL and build dynamic form data in the request. It even allows you to include XML data in a SOAP envelope creating web-service applications.

How it is done

Use the built-in function "httpRequest(..)" to exchange data between your IceBreak ASPX-program and another web server. httpRequest supports both HTTP and HTTPS. When you use HTTPS you will net to set up the certificate store as described in:

Setting up HTTPS

httpRequest

Syntax:

error = httpRequest (ReqUrl: ReqTimeOut: ReqMethod: reqData: reqContentsType: reqXlate: respXlate: respHeader: respBody);

Field Name	Data type	Description	Default
reqUrl	VARCHAR(32768)	The URLinform:"http://server:port/Resource?parm1=value1&parm2=value2" You can use both http and https. If you skip "http://server:port" and use/Resource?parm1=value1&parm2=value2 the request with be executed on the current server instance. I.e. "loopback" port.	
reqTimeOut	INT(5)	Number of seconds before timing out	15
reqMethod	VARCHAR(10)	"POST" or "GET"	GET
reqData	VARCHAR(32768)	Data in the "form" when using "POST"	none
reqContentsType	VARCHAR(256)	The MIME type of the request	text/html
reqHeader	VARCHAR(32768)	Additional headers	none
reqXlate	CHAR(15)	The Request charset:utf-8 | iso-8859-1 | windows-1252	windows-1252
respXlate	CHAR(15)	The Response charset: utf-8 | iso-8859-1 | windows-1252	windows-1252
respHeader	VARCHAR(32768)	Output: the http header from the remote web server response	none
respBody	VARCHAR(32768)	Output: the http body from the remote web server response	Response object
Returns			
Error	boolean	If an error occurs, this	

Field Name	Data type	Description	Default
		will be set to "TRUE" and you can retrieve the reason with msg = GetLastError()	

Example 1. Include a entire web page directly into your response object:

```
<%@ language="RPGLE" %>
<%
D Error           s               N
/free
// This simply places the result in the response object
// using all default parameters
   Error = httpRequest(
             'www.ibm.com/us/'
             : 30
             : 'GET'
             :*OMIT:*OMIT:*OMIT:*OMIT:*OMIT:*OMIT:*OMIT);

   if error;
   %>
      <% = GetLastError('*MSGID') %>
      <% = GetLastError('*MSGTXT') %>
      <% = GetLastError('*HELP') %>
   <%
   endif;
%>
```

Example 2. This is an equivalent example but sets up all parameters.

```
<%@ language="RPGLE" %>
<%
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments++++++
++++++
D ReqUrl          s          32767    varying                              The URL in
form:"http://server:port/file"
D ReqTimeOut      s              5I 0                                      Number of
seconds before timing out
D ReqType         s             10    varying                              "POST" or
"GET"
D ReqData         s          32767    varying                              The form data
when using "POST"
D ReqContentType  s            256    varying                              The MIME type
of the request
D ReqHeader       s          32767    varying                              Aditional
Headers
D ReqXlate        s             15                                         The target
charset (valid: utf-8 | iso-8859-1 | windows-1252 )
D RespXlate       s             15                                         The charset
for the response (valid:utf-8 | iso-8859-1 | windows-1252)
D RespHeader      s          32767    varying                              Out: Response
Header
D RespData        s          32767    varying                              Out: Response
Body

D Error           s               N

/free

// This uses all parameters and returns both the header and body in separate variables
```

```
  Error = httpRequest(
             'www.ibm.com/us/':           // The URL in form:"http://server:port/
resource"
             30:                          // Number of seconds before timing out
             'GET':                       // "POST" or "GET"
             '':                          // The formdata when using "POST"
             'text/html; charset=utf-8':  // The MIME type of the request
             '':                          // Extra headers
             'UTF-8':                     // UTF-8 is the target charset (valid: utf-8 |
iso-8859-1 | windows-1252 )
             'UTF-8':                     // UTF-8 is the charset for the response
(valid:utf-8 | iso-8859-1 | windows-1252)
             RespHeader:                  // Out: Response Header
             RespData);                   // Out: Response Body

   if error;
   %>
      <% = GetLastError('*MSGID') %>
      <% = GetLastError('*MSGTXT') %>
      <% = GetLastError('*HELP') %>
   <%
   else;
      responseWrite(RespData);            // Just put the response from "www.ibm.com/us/"
into my response object
   endif;
   return;
%>
```

Show the sample   Run the sample

6.3 Using the built-in XML Parser on the request object

Sample 15:

XML contents in a HTTP request is the core of Web-Services used in SOAP. XML is the perfect transportation method of almost any information on the internet.

What it does

When a client is creating a XML-request object it will always set the content-type header attribute to manage XML, i.e. "text/xml" or "application/xml" or "ms-office/XML". IceBreak will automatically parse the input.

How it is done

You simply use the built-in function **reqXmlGetValue(...)** to retrieve data. This function is using the XPath syntax to navigate into the XML-object tree.

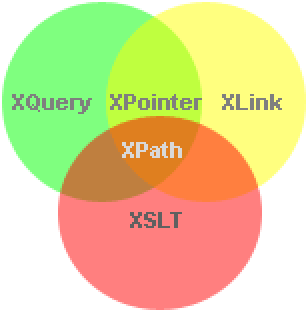
MyVar = reqXmlGetValue(Path : Defaultvalue);

Field Name	Data type	Description	Default
MyVar	VARCHAR(32760)	The return value of the XML-object	
Path	CHAR(*)	This is the path to the XML element or attribute in XPath format	
Default	CHAR(*)	Any default value if the "path" was not found in	

Field Name	Data type	Description	Default
		the XML-object tree	

XPath

XPath is a language for finding information in an XML document. XPath is used to navigate through elements and attributes in an XML document. XPath is a major element in the W3C's XSLT standard. XQuery and XPointer are both built on XPath expressions. Understanding XPath is crucial for working with advanced XML.



The XPath is a syntax to navigate into the XML object tree:

Path Expression	Description
/	Location from the root in the XML object tree
@	An attribute value
[ubound]	Upper boundary for elemet
[n]	Subscripting the index of **'n' 0=First, 1=next ...**

Consider the following XML request:

```
<Order>
  <Header Orderno='436533' Custname='John Doe'>
  <Detail>
    <Line lineno='1' itemno='4711' description='Nails 7"' qty='100'/>
    <Line lineno='2' itemno='3214' description='Bolts 2"' qty='50'/>
  </Detail>
<Order>
```

will give the following result:

Path Expression example	Description	Result
/Order/Header@Orderno	Returns the attribute "Orderno" in the element "Header" in element "Order"	436533
/Order/Header@CustName	Returns the attribute "Custname" in the element "Header" in element "Order"	John Doe
/Order/Detail/Line[ubound]	Returns the number of "line" Â´s in "detail" in "Order"	2
/Order/Detail/Line[1]@qty	Returns the attribute "qty" in the second element "line" in "Detail" in "Order"	50

Now let us create a small server:

1. It receives a XML-request (the layout as above) in charset UTF-8 format

2. Parse the XML request
3. Build a HTML response using the values found in the XML request
4. Finaly it sends the response back to the client

Let's take a closer look:

```
<%@ language="RPGLE" %>
<%

 *'
-----------------------------------------------------------------------------------------
---
 *' Web Service server: Parse the XML and return a HTML document
 *'
-----------------------------------------------------------------------------------------
---


D*Name+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments++++++
++++++
D path            s           1024    varying
D NumberOfLines   s              5  0
D i               s              5  0
/free
// Always set the charset before putting data into the response object
   SetCharset('utf-8');

%>
<html>
<header>
<link rel="stylesheet" type="text/css" href="/system/Styles/iceBreak.css">
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</header>
<body>
<h1>The XML request was:</h1>
<pre><% = GetServerVar('REQUEST_CONTENT') %></pre>
<h1>List the Order</h1>
Order Number : <% = reqXmlGetValue('/Order/Header@Orderno'
                                     : 'No number was given') %><br>
Customer name: <% = reqXmlGetValue('/Order/Header@CustName'
                                     : 'Customer name was not given') %><br>
Order date   : <% = reqXmlGetValue('/Order/Header@Date'
                                     : 'No date was not given') %><br>
<br>
<table>
 <thead>
  <th>Line No</th>
  <th>Item No</th>
  <th>Description</th>
  <th>Qty</th>
 </thead>
<%
 NumberOfLines  = num(reqXmlGetValue('/Order/Detail/Line[ubound]' : '0'));
 for i = 0 to NumberOfLines - 1;
   Path = '/Order/Detail/line[' + %char(i) + ']';
   %>
   <tr>
     <td><% = reqXmlGetValue(Path + '@LineNo'      : '0')   %></td>
     <td><% = reqXmlGetValue(Path + '@ItemNo'      : 'N/A') %></td>
     <td><% = reqXmlGetValue(Path + '@Description' : 'N/A') %></td>
     <td><% = reqXmlGetValue(Path + '@Qty'         : '0')   %></td>
   </tr>
<%
 endfor;
%>
</table>
</body>
</html>
<%
```

```
 return;
%>
```

But how do you produce a request with a XML content for that server? You might reuse the HTTP-proxy we made in Tutorial 13. It is able to make a HTTP-request with any kind of content, including XML.

What the client must do:

1. Set up charset UTF-8 - the default when dealing with XML and webservices.
2. Build a request XML structure.
3. Put it into the request content.
4. Ensure that the "content type" containing xml and charset e.g. "text/xml; charset=utf-8".
5. Execute the HTTP-request.
6. Present the result in the browser.

```
<%@ language="RPGLE" %>
<%

 *
-----------------------------------------------------------------------------------------
---
 * Send a XML request to the server and then just display the response data
 *
-----------------------------------------------------------------------------------------
---

D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments++++++
++++++
D xml             s         32767    varying                              *ON=Convert to
response to ebcdic
D RespHeader      s         32767    varying                              Out: Response
Header
D RespData        s         32767    varying                              Out: Response
Body
D ServerURL       s           255    varying                              The URL to the
web service

D Error           s               N
D crlf            c                  x'0d25'

/free

// When dealing with XML and webservices; the preferred charset is UTF-8
// Also - the charset must be selected before anything is placed in the responseobject
   SetCharset('utf-8');

// Then we build the request XML
xml =
'<Order>' + CRLF +
'  <Header Orderno="436533" -
           Custname="John Doe" -
           Date="' + %char(%timestamp()) + '"/>' + CRLF +
'  <Detail>' + CRLF +
'    <Line Lineno="1" -
           itemno="4711" -
           description="Nails 7 x 0.5-Ø" -
           qty="100"/>' + CRLF +
'    <Line Lineno="2" -
           itemno="3214" -
           description="Bolts 2 x 0.2-Ø" -
           qty="50"/>' + CRLF +
'  </Detail>' + CRLF +
'</Order>';
```

```
// I'll run the service on the same IceBreak server instance so we skip the "http://
server:port"
// and just go with the resource
   ServerURL = '/tutorials/ex15Server.asp';

// post the HTTP-request to the server ASP and return both the header and body in
separate variables
   Error = httpRequest(
             ServerURL:                  // The URL in form:"http://server:port/
resource"
             30:                         // Number of seconds before timing out
             'POST':                     // "POST" or "GET"
             xml:                        // The formdata or xml when using "POST"

             'text/xml, charset=utf-8':  // The MIME type of the request
             '':                         // Extra headers
             'utf-8':                    // UTF-8 is the target charset (valid: UTF-8 |
ISO-8859-1 | windows-1252 )
             'utf-8':                    // UTF-8 is the charset for the response
(valid: UTF-8 | ISO-8859-1 | windows-1252 )
             RespHeader:                 // Out: Response Header
             RespData);                  // Out: Response Body

// Take all the response and send it back to the browser
   responseWrite(RespData);

return;%>
```

Show the sample   Run the sample

6.4 Using the built-in XML Parser on a string or stream file

XML:

Using the built-in XML Parser on stream files or strings

The XML parser is also convenient for stream files and strings containing XML-structures.

What it does

You can parse a stream file or a string containing XML structures by adding an extra prototype to your ASP program, module or service program. The XML parser is a small and fast parser that use the XPath syntax to retrieve XML elements or attributes.

How it is done

- First you have to Include the XML-parser portotype into your ASP-source.
- Then declare a pointer to he XML-tree.
- Then use the function xml_ParseFile() or xml_ParseString() to process the stream filen or string.
- Now you can retrieve data in the XML-tree by xml_Getvalue(...).
- Finally you must reclaim the storage used by the parser with xml_Close().

Let us have a look at an example:

```
<%@ language="RPGLE" %><%
```

```
/include qasphdr,xmlparser

D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments++++++
++++++
D xmlPtr          s               *                                        Pointer to the
XML tree
D xmlString       s           1024    varying                              XML string to
parse
D val             s           1024    varying                              temp. value
D node            s            256    varying                              temp. value
D crlf            c                   x'0d25'                              the <CR><LF>
sequence

/free
%>
<html>
<head>
  <link rel="stylesheet" type="text/css" href="/system/Styles/iceBreak.css">
</head>
<body>
<%

// First build the request XML
xmlString =
'<Order>' + CRLF +
'  <Header Orderno="436533" +
          Custname="John Doe" +
          Date="' + %char(%timestamp()) + '"/>' + CRLF +
'  <Detail>' + CRLF +
'    <Line Lineno="1" +
          itemno="4711" +
          description="Nails 7 x 0.5" +
          qty="100"/>' + CRLF +
'    <Line Lineno="2" +
          itemno="3214" +
          description="Bolts 2 x 0.2" +
          qty="50"/>' + CRLF +
'  </Detail>' + CRLF +
'</Order>';

// Just show the XML
%>
<h1>The XML structrure</h1>
<pre><%=xmlString%></pre>
<br>
<h1>The Result when parsing the XML and finding the values with X-Path</h1>

<%



// Then Parser the string
   xmlPtr = XML_ParseString(xmlString:'syntax=LOOSE');

// If the parser is not able to parse the string; then display the error
   if XML_Error(xmlPtr);
   %><%= XML_Message(xmlPtr) %><%

// Otherwise - the parser ran ok, so we pick the second line description atribute
// remember that x-path use [0] as the first index  [1] as the next etc...
// the 'N/A' is the default value if i can not be found in the XML tree
   else;
      node = '/Order/Detail/Line[1]@description';
      val = XML_GetValue(xmlPtr:node:'N/A');
      %>Node:<b><%=Node%></b> has the value of: <b><%= val %></b><%
   endif;
   XML_Close(xmlPtr);
%>
</body>
</html>
<%
   return;
```

```
%>
```

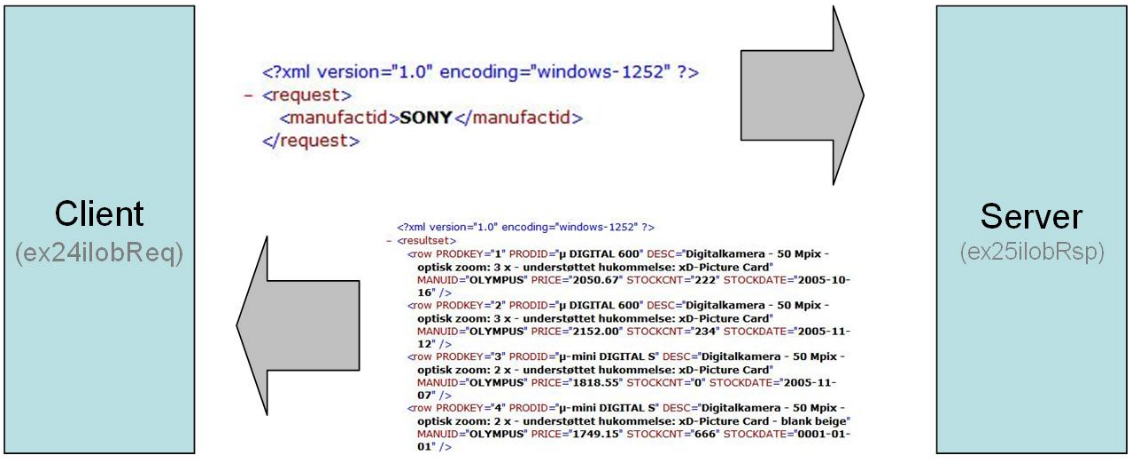
Show the sample   Run the sample

6.5 ILOB's, httpRequest and XML (SOA components)

SOA::

In a Service Oriented Architecture world (SOA) the ability to intercommunicate huge data streams is essential.

ILOB's as we saw earlier is a way deal with large data from within a ILE program.
This tutorial will show how to use ILOB's and make HTTP request with XML based data stream.



```
  <?xml version="1.0" encoding="windows-1252" ?>
- <request>
    <manufactid>SONY</manufactid>
  </request>
```

```
  <?xml version="1.0" encoding="windows-1252" ?>
- <resultset>
    <row PRODKEY="1" PRODID="µ DIGITAL 600" DESC="Digitalkamera - 50 Mpix -
      optisk zoom: 3 x - understøttet hukommelse: xD-Picture Card"
      MANUID="OLYMPUS" PRICE="2050.67" STOCKCNT="222" STOCKDATE="2005-10-
      16" />
    <row PRODKEY="2" PRODID="µ DIGITAL 600" DESC="Digitalkamera - 50 Mpix -
      optisk zoom: 3 x - understøttet hukommelse: xD-Picture Card"
      MANUID="OLYMPUS" PRICE="2152.00" STOCKCNT="234" STOCKDATE="2005-11-
      12" />
    <row PRODKEY="3" PRODID="µ-mini DIGITAL S" DESC="Digitalkamera - 50 Mpix -
      optisk zoom: 2 x - understøttet hukommelse: xD-Picture Card"
      MANUID="OLYMPUS" PRICE="1818.55" STOCKCNT="0" STOCKDATE="2005-11-
      07" />
    <row PRODKEY="4" PRODID="µ-mini DIGITAL S" DESC="Digitalkamera - 50 Mpix -
      optisk zoom: 2 x - understøttet hukommelse: xD-Picture Card - blank beige"
      MANUID="OLYMPUS" PRICE="1749.15" STOCKCNT="666" STOCKDATE="0001-01-
      01" />
```

Basically we want to send a XML request to a server program and in return receives the XML response.
By using ILOB's we are able to break the 32K limit that RPG normally has.

The Server program

```
<%@  language="RPGLE" %>
<%
 *'
------------------------------------------------------------------------------------------
---
 *' Runs an SQL query for the XML request received
 *'
------------------------------------------------------------------------------------------
---

D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments++++++
++++++
d manufactid      s             16    varying
D Error           s               N
D SqlCmd          s           1024    varying
D MaxRows         s             10i 0

/free
//' The first thing is always to set charset and content type
```

```
    SetContentType ('application/xml; charset=windows-1252');

//' Take the parameter from the XML request
    manufactid = reqXmlGetValue('/request/manufactid' : '');

//' Build the SQL command
    sqlcmd = 'select * from product where manuid = ''' +manufactid+ '''';
    MaxRows = 10000;

//' Now run the SQL query
    %><?xml encoding="windows-1252" version="1.0" ?><%
    Error   = SQL_Execute_XML(sqlcmd : maxrows);
    if (Error);
      %><error help="<% = getLastError('*HELP') %>" msg="<% = getLastError('*MSGTXT') %>
"></error><%
    endif;
    return;
%>
```

1. The SetContentType() to "application/xml" is essential. We both sends ad receives data in XML format. if IceBreak "see" the /XML in the content type it will automatically run the XML-parser.
2. The xml request is already parsed so we have access to an XML object that IceBreak maintains. Now we can use the X-path syntax to refer to the value of the element "manufactid" in the "request" element.
3. The SQL statement is constructed. Just adding the "where" clause and we are ready to run the SQL.
4. Finally the surroundings for the XML is set up. Note that the encoding tag in the XML header has to be the same as we used in "SetContentType()"
5. SQL_Execute_XML() simply returns the SQL result set as an XML document, which is placed directly in the response object.

You can see the number of variables in a program like this i kept to a minimum. All the data manipulation is done behind the scenes in the request and response object.

The Client program

The client program is a little more sophisticated than the client. It is dealing with the access to the ILOB's and parser directly:

```

<%@ language="RPGLE" %>
<%

d*' Include the ILOB and xml parser prototypes
/include qasphdr,ilob
/include qasphdr,xmlparser

d*Name+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments++++++
++++++
d manufactid      s             16    varying
d path            s             64    varying
d Error           s               N
d reqILOB         s               *
d respILOB        s               *
d xmlPtr          s               *
d i               s             10i 0
d rows            s             10i 0

/free

//' Basically this program just toggle between the manufactures
    manufactid = form('manufactid');

```

```
//' The first thing is always to set charset and content type
    SetContentType ('text/html; charset=windows-1252');

//' First I create my work ILOB's
    reqILOB  = SesGetILOB('request');   // get a pointer to a session ILOB used as my
request
    respILOB = SesGetILOB('response');  // get a pointer to a session ILOB used for the
response from the server

//' Reset my ilob
    ILOB_Clear(reqILOB);
    ILOB_Clear(respILOB);

//' Now i want to populate a simple XML request. - my ILOB header need a content type XML
//' All httpRequest are routed to the same default job hence the cookie
    ILOB_SetHeaderBuf(
      reqILOB :
      'Content-Type: application/XML; charset=windows-1252'
    );

//' I reroute my response object to the request ILOB, so I can use the ASP syntax for
creating the XML
    SetResponseObject(reqILOB);

%><?xml version="1.0" encoding="windows-1252" ?>
<request>
  <manufactid><% = manufactid %></manufactid>
</request><%

//' Now i redirect my response back to the default response objet ( what my browser
receives >
    SetResponseObject(*NULL);

//' My request is now ready to to to the server, so i call the httpRequest for ILOB's
    Error = ILOB_httpRequest(
       '/tutorials/ex24ilobsvr.asp': // The URL in form:"http://server:port/resource"
       30:                            // Number of seconds before timing out
       'POST':                        // The "POST" method
       reqILOB:                       // pointer to my Request ILOB
       RespILOB                       // pointer to my Response ILOB
    );

//' Check for errors
    if error;
       %><% =  GetlastError('*MSGTXT') %><%
       return;
    endif;

//' My data has arrived - I'll fire up the XML parser
    xmlPtr = XML_ParseILOB ( //' Returns XML-object tree from an ILOB
       RespILOB:             //' Pointer to an ILOB object
       'syntax=loose':       //' Parsing options
       1252:                 //' The ccsid of the input ilob  (0=current job)
       0                     //' The ccsid of the XML tree (0=current job)
    );

//' Check for errors
    if XML_Error(xmlPtr);
       %><%= XML_Message(xmlPtr) %><%
       XML_Close(xmlPtr);
       return;
    endif;

//' The Result will be a table
%><html>
<head>
 <link rel="stylesheet" type="text/css" href="/System/Styles/IceBreak.css"/>
</head>
<h1>Products by: <% = manufactid %></h1>

<form action="" method="post" id=form1 name=form1>
```

```
 <select name="manufactid">
    <option value="ACER">ACER</option>
    <option value="CANON">CANON</option>
    <option value="CASIO">CASIO</option>
    <option value="FUJIFILM">FUJIFILM</option>
    <option value="HP">HP</option>
    <option value="KODAK">KODAK</option>
    <option value="KONICA">KONICA</option>
    <option value="NIKON">NIKON</option>
    <option value="OLYMPUS">OLYMPUS</option>
    <option value="PANASONIC">PANASONIC</option>
    <option value="SAMSUNG">SAMSUNG</option>
    <option value="SONY">SONY</option>
 </select>
 <input type="submit" value="Go"/ id=submit1 name=submit1>
</form>

<table>
  <thead>
    <th>Product id</th>
    <th>Product Description</th>
    <th>Product Price</th>
  </thead><%

//' now print the report
    rows =  num(XML_GetValue(xmlPtr: '/resultset/row[ubound]' :'0'));
    for i = 0 to rows -1;
      path = '/resultset/row[' + %char(i) + ']@';
      %><tr>
        <td><% = XML_GetValue(xmlPtr: path + 'prodid' :'N/A') %></td>
        <td><% = XML_GetValue(xmlPtr: path + 'desc'   :'N/A') %></td>
        <td><% = XML_GetValue(xmlPtr: path + 'price'  :'N/A') %></td>
       </tr><%
    endfor;
%>
 </table>
</html><%

//' Always release the memory used
    XML_Close(xmlPtr);
    return;
%>
```

1. Like the server program we need to set up the contents type. It need to map to the same values as the server.
2. The we create two ILOB's used for request and response
3. The clear of the the Request ILOB is essential. Otherwise data will just append to the ILOB.
4. The ILOB has a header which is used for HTTP communication. You have access to that header by the "ILOB_SetHeaderBuf()". Note that you replaces the complete header by using the method.
5. The "SetResponseObject()" is used to reroute the response data to an ILOB so the data goes to the ILOB and not the normal response object. When you want to switch back, the just call "SetResponseObject()" with a *NULL parameter and you have your normal response object back which goes to your browser. This method can also be applied to creating XML files on the fly by building up the ILOB and the save the ILOB to stream file.
6. Now with the request XML in the "request" ILOB we just runs the "ILOB_httpRequest()" method and receives the "response" ILOB.
7. All the data in the ILOB is in ASCII, so when we fire up the XML parser we have to tell it which ccsid the ILOB is stored in. In this case "windows-1252" maps to "ccsid 1252".
8. Now are finished with the ILOB's the rest of the program just formats the XML tree to the response object that goes to the end-users browser. Here again - like in the server - we are extracting data using the X-Path syntax.
9. Finally always remember to use XML_Close() to free up the memory used by the XML parser.

Run the sample

Part VII

7 OEM, Bundle IceBreak

7.1 Ship IceBreak allong with your product (COPY)

OEM - (Original equipment manufacturer) means the original manufacturer of a component for a product, which may be resold by another company. In the case of IceBreak it means you can put all the IceBreak features into you own software package and ship your package and IceBreak in one go.

As an alternative, you can let your customers install IceBreak from the web, and just configure you OEM server (or servers) once and let your application have control over license authorization.

In both cases you have the option to run IceBreak servers in you own subsystem as a part of you own product.

1

Obtain a OEM licence key

Before you can make an OEM version of IceBreak you need to contact the IceBreak support team at mailto:support@system-method.com to get a OEM licence key. An OEM licence key let you create a server instance, and run that server from your own subsystem

2

OEM Licence Exit Program

You need a OEM licence exit program in you application library that returns your OEM key (or keys) to the IceBreak core. The name of this program is by default **SVCOEMEXIT**. This can be a simple RPG program or even a CLP as in the following example

```
    Pgm   Parm(&licXml)

       /* Simply return the Licence XML string for your product -
          ----------------------------------------------------- */
       Dcl  &licXml    *Char (2048)

       chgvar  &licXml ('-
          <license>-
            <product id="OEM-ICEADM" service="ADMIN" expires="*PERM" liccode="517-391-473" text="Ic
          </license>-
       ')
       return
    EndPgm
```

Basically this program returns a string containing a XML document. You can have one or more products of you own in this XML string. The OEM-exit program need to be placed in the application library for you server instance

- **id** - This is your global wide unique product identification. You receive this from the IceBreak support team.

- **service** - The service name you provide; this also the name of you application server in IceBreak. For OEM servers the name can be up to 8 char long.

- **expires -** The date when this licence code will expire in format CCYY-MM-DD or *PERM if this OEM licence continues infinitely.

- **text -** Any descriptive text of you choice to describe your product.

Now you have two options: You can imbed IceBreak as a part of your own application library or you can ship you application run on to of any type of IceBreak server installed. Let me explain both options:

3

Imbed IceBreak OEM in you own application library.

If you want to imbed IceBreak as a part of your application i.e. install IceBreak along with you product then you can extract all the necessary programs, files and other objects required into a save-file which you again restore into you product library either when you install on the customer site or when you save your product before shipment.

A good or bad thing (depends on how you look at it) is, that you lock your IceBreak version to version you extracts. Of cause you can redo the imbed process to upgrade to a new version of IceBreak. This, however, is not done automatically.

This is the process:

1. Extract the IceBreak core from you installation:
2. Install the extracted IceBreak core in you application library
3. Prepare your installation
4. Deploy and run

Log on as QSECOFR or a user with *SECADM and open an IBMi command prompt and enter:

```
===> GO ICEBREAK
===> EXPICEOEM OEMCODE('OEM-ICEADM') SAVF(QGPL/MYICEBREAK) WORKLIB(QTEMP)
===> RSTOBJ OBJ(*ALL) SAVLIB(QTEMP) DEV(*SAVF) SAVF(QGPL/MYICEBREAK) RSTLIB(MYICEOEM)
```

- **OEMCODE** - This is your global wide unique product identification. You receive this from the IceBreak support team.

- **SAVF** - Is the location where you store the core
- **RSTLIB -** Your application library to contain the icebreak core. **MYICEOEM** is just used for this example

Now you can prepare the server in your library

Relogon on as QSECOFR or a user with *SECADM and open an IBMi command prompt and enter:

```
        ===> ADDLIBLE MYICEOEM
        ===> WRKICESVR
```

Now add a server with the name of the service provided for you in the OEM licence agreement.

When you add a server then ensure that that you have a job queue to a active subsystem. If you don't have a subsystem provided by your application then you can use the system supplied job queue:

```
        QSYS/QSYSNOMAX
```

So your "add IceBreak server" command will look similar to:

```
        ===>  ADDICESVR SVRID(MYICEOEM) TEXT('My icebreak oem server') SVRPORT(60006) JOBQ
(QSYS/QSYSNOMAX)
```

Now you need to start the servers with **STRICEJOBS**, this must be done from at batch job either in you own subsystem by a call from a CL-program in i.e. the auto-start-job or you can submit it from the command line - again you can use QSYS/QSYSNOMAX job queue.

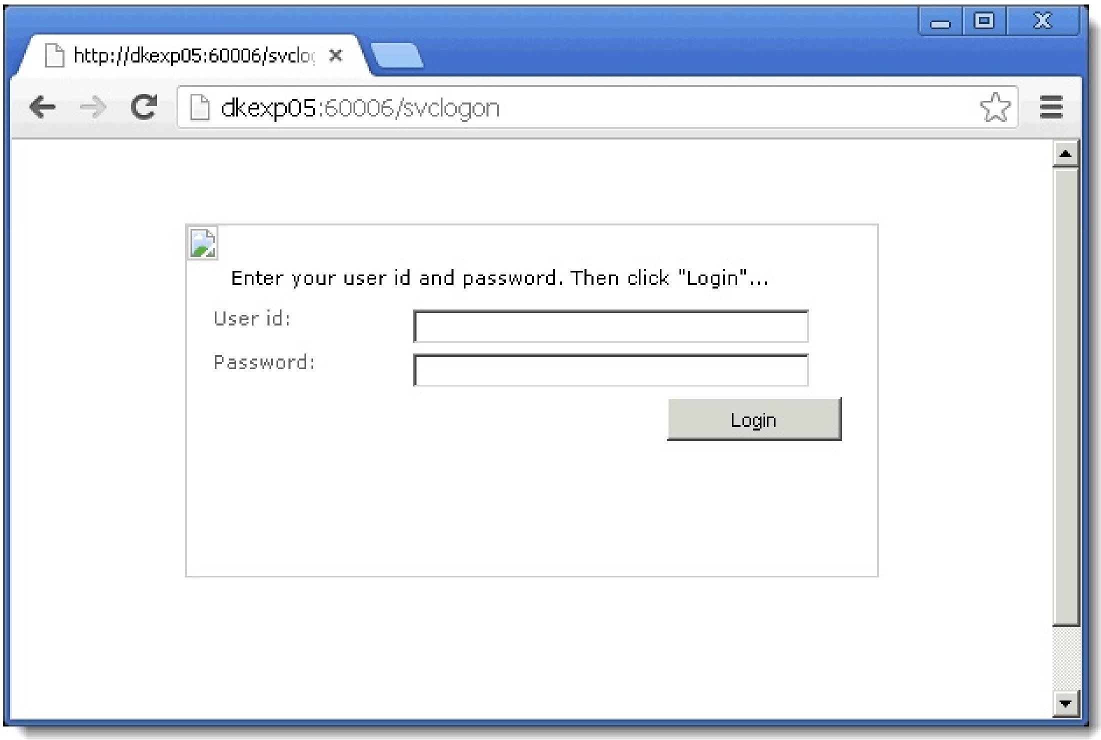
```
        ===>  SBMJOB CMD(STRICEJOBS) JOB(STRMYJOB) JOBQ(QSYS/QSYSNOMAX)
```

Note: Do not try to start the server from with in the WRKICESVR option 12 or the STRICESVR command. Your OEM servers are detached from the normal IceBreak subsystem, so the normal start features does not work as in a plain IceBreak installation. You can however use option 13=End server or ENDICESVR command. Always use the STRICEJOBS for OEM servers. (This, however, might change in future releases).

Now the server is up a running - you can test it by one of the build in servies like SVCLOGON:

1. Open your browser:
2. Enter the URL http://mysystemI:60006/svclogon.aspx

Now the default logon icebreak prompt is shown - note: system styiling is not in the default OEM build:



Enter your user id and password. Then click "Login"...

User id:

Password:

Finally you can now deploy you application library with IceBreak build in. That is all there is to it.

4

Use Your product with a plain IceBreak installation

Your application can coexist with any "nomal" plain IceBreak installation. Even demo and community edition, depending on which OEM licence code for your product you obtain. Normally; if your application just require the IceBreak Runtime, it will be sufficient to just install the newest version from the icebreak download site and configure your server (which can be done automatically by a CL program in you application package, and the finally start the IceBreak subsystem. You will not need to provide any licence codes for the IceBreak server it self

Part VIII

8 Using JAVA

8.1 Java as the host language

If you want to build java web applications then you can choose from many server technologies: Tomcat, WebSphere Glassfish - just to name a few.

IceBreak also supports java. But where Tomcat and WebSphere only support java, IceBreak on the other hand runs Java and ILE in the same session.

So when do You need to use Java in IceBreak?

- If you have mixed environments of ILE and Java.
- if you want to use some features that does not exists in ILE / RPG like JDBC to acces SQL databases on remote systems.
- If you need high JAVA performance
- If you are running on a small system where i.e. WebSphere can not run.
- If you want to use open-source java libraries like iText to produce PDF-files, xmlSec to sign XML files etc.

How does java work in IceBreak

Java for IceBreak is not a "standard" servlet container like Tomcat or websphere, but rather a way to let java use the IceBreak server object model. The layer between your java code and the IceBreak is extremely thin, so you will not have the same performance penalty in IceBreak that you have in a "standard" servlet container on the IBMi.

When you establish an IceBreak session it becomes equally available for ILE and java so you can refer to a RPG resource in one **httprequest** and a java resource in the next with no performance overhead. Your java and RPG session shares the following:

- The request object
- The response object
- The session with session variables and ILOB's
- Globals and server-vars.

Now lets get stated:

1

Create a java "hello World" in IceBreak

1) Fire up you Java environment of choice: eclipse, NetBeans, WDSc, RDi, IntelliJ - any will do.

2) Now create a new java project. Let's call it "IceBreakTest". Don't create a "Java Web project", since it will be for servlet containers which IceBreak is not.

3) Set environment: The version of java you are using are important.The version you compile to, need to be the same on the target IBMi system. In this sample I will used JDK 1.5. So ensure the you right click on the project the select project properties. Select i.e. JDK 1.5 or JDK 1.6 - just ensure you have the same JDK installed on the IBMi that you are compiling against. But for now select JDK 1.5

4) Copy and paste the version of IceBreak core integrations classes that matches you environment into your project. You find it on the IFS in the IceBreak core path, where x-x is the java JDK version you are using.. Typically copy it from here:

```
\\myIBMi\IceBreak\java\IceBreak-x-x.jar
```

In eclipse you simply:
a) go the the project properties
b) click on the "java build path"
c) click on "add external JAR"
d) Enter the share name to the icebreak-x-x.jar as described above.

This **jar** file contains all the access to the IceBreak core you need. Also you will find the documentation for IceBreak.jar in the IceBreak admin at the "Programmers guide" -> java documentation. Or you can open it from here:

```
\\myIBMi\IceBreak\java\javadoc\index.html
```

2

Write you java code

Now having the IceBreak jar in place you can start using it:

1) Add a package to you project, just callit IceBreakTest like the project. 3) Add a package to you project. Packages means you can have several classes with several methods in one package. IceBreak supports packages. It is a nice way to sperate logic into dedicated source files.

2) Add a class to the package - let's call it "simple". this will produce a source file called simple.java

3) In the source "**simple.java"** you can add the following code:

a) You need to give the name of the package it belongs to: IceBreakTest - was the name

b) Now import all the core classes from the IceBreak core: This is done by **import icebreak.core.***;

c) Add the class body for the simple class like **public class simple** - it need to be public so the IceBreak core can access is at runtime

d) Add a method to the simple class - let call it "**sayHey**" which will use the request object to get the name and the response object to reply back:

```
package IceBreakTest;
import icebreak.core.*;

public class simple {
  public static void sayHey () {
    String s = qrystr.get("name");
    response.write("Hey " + s);
```

```
        }
      }
```

... Not much code required to make a Web applicaitons with Java in IceBreak... :)

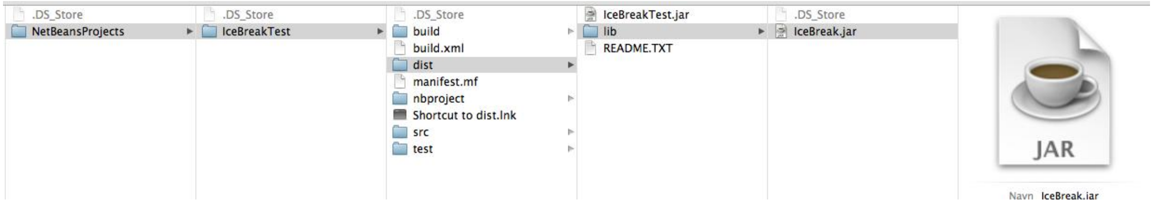
Compile and deploy "icebreaktest".

Click on "make project" and if you had success we are ready to deploy it to the IceBreak server:

On the IFS, in the application server "Http root path" you can make a directory called "**java"**, where you can put your applications. For my "**systest**" server it will be:

```
//myIbmi/www/systest/java
```

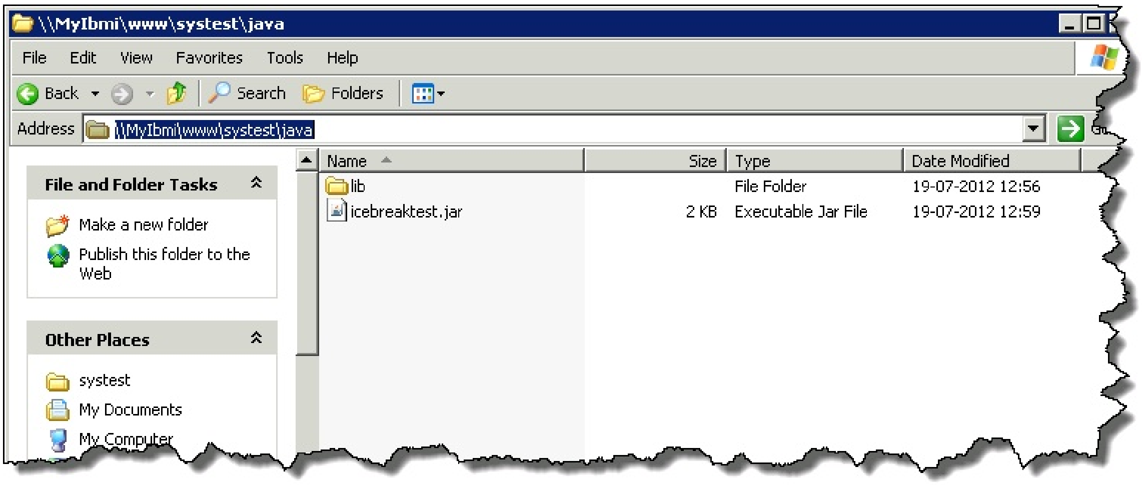
In Netbeans you simply copy the complete contents of the "**dist**" folder from the "NetBeansProject" folder and paste it to the "java" directory described above.



In eclipse/WSDc /RDi you export the jar file:

1) Click on properties on the project.
2) click on "export"
3) Select "java"->"JAR file"
4) Click "Next"
5) Select the "export destination" to be the IFS application server path selected in IceBreak
6) Click "Finish"

My IFS folder now looks like this:


\\MyIbmi\www\systest\java
File Edit View Favorites Tools Help
Back Search Folders
Address \\MyIbmi\www\systest\java
File and Folder Tasks
Make a new folder
Publish this folder to the Web
Other Places
systest
My Documents
My Computer

Name	Size	Type	Date Modified
lib		File Folder	19-07-2012 12:56
icebreaktest.jar	2 KB	Executable Jar File	19-07-2012 12:59

In the application server you need to setup the callspath for your JAVA. The classpath is where IceBreak java searches for your application .JAR file(s):

You will have to open the **webConfig.xml** in your application server "Http root path" - if it does not exists, simply create it, and add the java element:

```
<?xml version="1.0" encoding="utf-8" ?>
<Configuration>

  <java
    classpath="/www/systest/java/icebreaktest.jar"
    version="1.5">
  </java>

</Configuration>
```

Set the classpath to the complete location of where you did place you application jar file. This is from the IBMi perspective so if you afterwards use the "WRKLNK" command to verify the existence of the file.

Note: the classpath can be a list of files. Simply separate the jar files by a **:** as you will see in the next tutorial.

Version: In this sample we use JDK version 1.5 - again this have to match the compiled version from your IDE and you must have this version installed on your IBMi

Now save your webConfig.xml.

For each time you deploy/recompile you application or change the webConfig.xml you have to reload the IceBreak JVM. Simply restart the IceBreak server from a command line to accomplish this task.

```
ENDICESVR SVRID(SYSTEST) RESTART(*YES)
```

4

Run your "Hello world" application.

Now You should be good to go. Open you browser. And enter the URL of your application:

First: the name or IP-address and the port-number of you IBMi: In my case it is : http://myIBMi:60000/

Next: The package name **slash** the class name: In my case that is IceBreakTest/simple

Now you can access all "public void .. (void)" methods in that class with **colon** and the method name. Here it is :sayHey

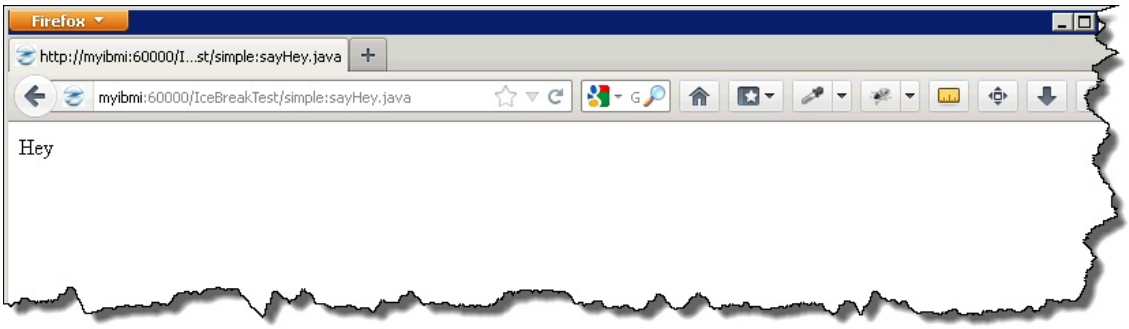
Finally - all java need to be suffixed by .java to route the request to the IceBreak JVM

So finally my URL looks like:

```
http://myibmi:60000/IceBreakTest/simple:sayHey.java
```

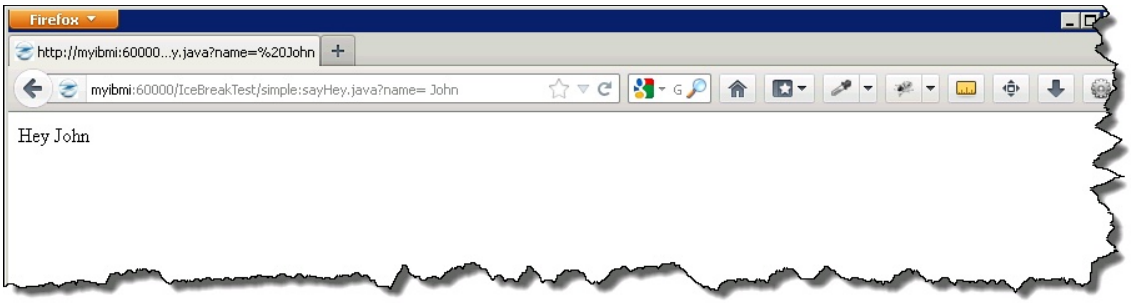
NOTE !! NOTE !!: Beware - Java is case sensitive so now the URL are case sensitive. Everything from the package name and forward need to be in the right upper or lower-case.

Let's try the it:


Firefox
http://myibmi:60000/I...st/simple:sayHey.java
myibmi:60000/IceBreakTest/simple:sayHey.java

Hey

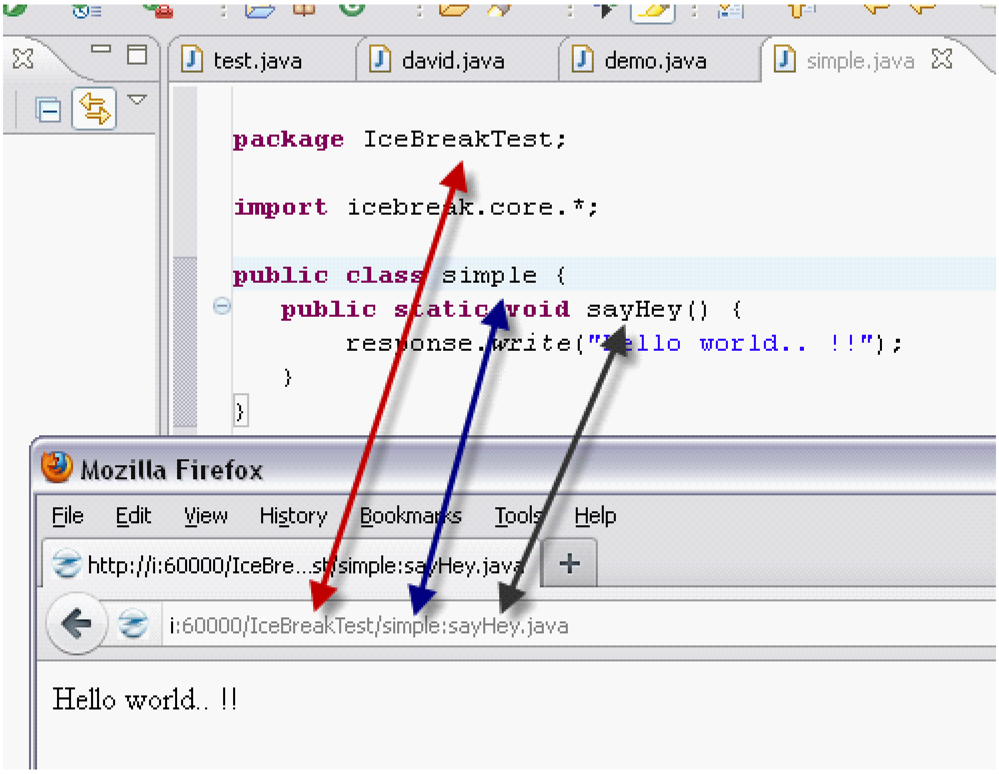
Take a look at the code again: We actually supplied the query string as well: so if we give the parameter **name=John** it will be echoed back. Lets take it for a spin:


Firefox
http://myibmi:60000...y.java?name=%20John
myibmi:60000/IceBreakTest/simple:sayHey.java?name= John

Hey John

Nice!! Now we receive a parameter from the URL, parse it into our Java-code and place that in our response. We are golden!!

It is important that you always keep the structure of the "Package" / "Class" : "Method" and ensure that you use the same case in all levels. If you fail to do so, you will receive a class not found message:



```
package IceBreakTest;

import icebreak.core.*;

public class simple {
    public static void sayHey() {
        response.write("Hello world.. !!");
    }
}
```

Hello world.. !!

8.2 Accesing remote data bases with JDBC

The main reason for you to use Java in IceBreak is to benefit from all the java libraries you can find out there on the internet. In this tutorial I'll show you how to access a MS SQL server from an IceBreak program. This sample also illustrates the use of properties in the webConfig.xml, classpath lists and more.

Creating a JDBC application

A Java Data Base Connection can be run against almost any SQL server. From MySql , Oracle, MS SQL to DB/2 on any platform. From tiny devices to mainframes.

So lets extend the "**IceBreakTest**" package from previous tutorial to exploit the benefit of JDBC.

1

Open your IDE of choice, open the previous tutorial and add the following **listHtml** method to you code:

```
import icebreak.core.*;
import java.sql.* ;

public class simple {
  public static void sayHey () {
    String s = qrystr.get("name");
    response.write("Hey " + s);
  }

  public static void listHtml() {
     try {
        // Load the database driver
        String driver = System.getProperty("driver");
        String connect = "jdbc:sqlserver://DKSRV01;databaseName=Northwind;user=xx;";
        String sqlcmd = "Select * from customers";

        // Load the database driver and get connection to the database
        Class.forName(driver) ;
        Connection conn = DriverManager.getConnection(connect);

        // Print all warnings
        for( SQLWarning warn = conn.getWarnings(); warn != null; warn = warn.getNextWarning() ) {
          response.writeln( "SQL Warning:" ) ;
          response.writeln( "State  : " + warn.getSQLState()  ) ;
          response.writeln( "Message: " + warn.getMessage()   ) ;
          response.writeln( "Error  : " + warn.getErrorCode() ) ;
        }

        // Get a statement from the connection
        Statement stmt = conn.createStatement() ;

        // Execute the query
        ResultSet rs = stmt.executeQuery( sqlcmd) ;

        // Loop through the result set
        response.writeln("<html>");
        response.writeln("<table>");
        while( rs.next() ) {
          response.writeln("<tr>");
          for (int i = 1 ; i <5; i++) {
            response.write("<td>");
```

```
                response.write(rs.getString(i));
                response.write("</td>");
              }
              response.writeln("</tr>");
            }
            response.writeln("</table>");
            response.writeln("</html>");

            // Close the result set, statement and the connection
            rs.close() ;
            stmt.close() ;
            conn.close() ;
         }
         catch( SQLException se ) {

            // Loop through the SQL Exceptions
            response.writeln( "SQL Exception:" ) ;
            while( se != null ) {
              response.writeln( "State  : " + se.getSQLState()  ) ;
              response.writeln( "Message: " + se.getMessage()   ) ;
              response.writeln( "Error  : " + se.getErrorCode() ) ;
              se = se.getNextException() ;
            }
         }
         catch( Exception e ) {
            response.writeln( e.toString() ) ;
         }
      }
    }
```

All the new code is in the listHtml method. Let me go through it step by step:

a) The first initialization of the string driver, connect and sqlcmd are simply string constants;

```
// Load the database driver
String driver = System.getProperty("driver");
String connect = "jdbc:sqlserver://DKSRV01;databaseName=Northwind;user=xx;";
String sqlcmd = "Select * from customers";
```

- The driver is the name that implements the JDBC in out case a driver supplied by Micosoft which you can find in the IceBreak core IFS path under /java/lib. The JAR files are discussed later when we talk about the classpath. The name is supplied by the driver attribute from "webConfig.xml", where you also have the classpath. so **System.getProperty()** returns values from your **webConfig.xml**, feel free to use that feature.

- Connect is the "connection string" where you have the URL to the SQL database server, the name of the database to connect to, and a user name/password.

- sqlcmd is just a simple select which resurens alle the rows in the customers database table.

b) class.forName loads the implementation of the JDBC, in this case it is the Microsoft driver, which must be found in the class-path, getConnection returns a handle to a logged on database connection (this might take a little while the first time)

```
// Load the database driver and get connection to the database
Class.forName(driver) ;
Connection conn = DriverManager.getConnection( connect);
```

c) If an error occurs, we can trap that and print it to the browser screen; Note this is a list and we use the IceBreak response.write to send the response back to the client

```
// Print all warnings
for( SQLWarning warn = conn.getWarnings(); warn != null; warn = warn.getNextWarning() ) {
 response.writeln( "SQL Warning:" ) ;
 response.writeln( "State  : " + warn.getSQLState()  ) ;
 response.writeln( "Message: " + warn.getMessage()   ) ;
```

```
      response.writeln( "Error  : " + warn.getErrorCode() ) ;
    }
```

d) Now we can run the SQL query; and return a resultset into "**rs**"

```
    // Execute the query
    ResultSet rs = stmt.executeQuery( sqlcmd) ;
```

e) For each row in the resultset we produce at HTML-table row using the icebreak **response.write** this tutorial just list the five first columns.

```
    // Loop through the result set
    response.writeln("<html>");
    response.writeln("<table>");
    while( rs.next() ) {
      response.writeln("<tr>");
      for (int i = 1 ; i <5; i++) {
        response.write("<td>");
        response.write(rs.getString(i));
        response.write("</td>");
      }
      response.writeln("</tr>");
    }
    response.writeln("</table>");
    response.writeln("</html>");
```

f) After the last row, we do a little cleanup; Close the result set, close the statement and finally close the connection.

```
    // Close the result set, statement and the connection
    rs.close() ;
    stmt.close() ;
    conn.close() ;
```

g) Exceptions - First we listen to SQL exceptions; if any occurs we simply list them to the browser screen.

```
   } catch( SQLException se ) {

    // Loop through the SQL Exceptions
    response.writeln( "SQL Exception:" ) ;
    while( se != null ) {
      response.writeln( "State  : " + se.getSQLState()  ) ;
      response.writeln( "Message: " + se.getMessage()   ) ;
      response.writeln( "Error  : " + se.getErrorCode() ) ;
      se = se.getNextException() ;
    }
```

h) More Exceptions - Finally we listen to generic exceptions, if any occurs we simply list them to the browser screen

```
  }catch( Exception e ) {
    response.writeln( e.toString() ) ;
  }
}
```

2

Now compile the IceBreakTest package again, and re-deploy it to the IBMi - However, we just need to tweak the classpath a bit to let the JVM load the database driver:

reopen the webConfig.xml:

- Add the MS SQL driver JAR til the classpath:

- Add the property driver with the values of "com.microsoft.sqlserver.jdbc.SQLServerDriver", which is used by the Microsoft implementation

```
<?xml version="1.0" encoding="utf-8" ?>
<Configuration>

  <java
    classpath="/www/systest/java/icebreaktest.jar:/icebreak/java/lib/sqljdbc.jar"
    version="1.5">
    <property name="driver" value="com.microsoft.sqlserver.jdbc.SQLServerDriver"/>
  </java>

</Configuration>
```

And remember: For each time you deploy/recompile you application or change the webConfig.xml you have to reload the IceBreak JVM. Simply restart the IceBreak server from a command line to accomplish this task.

```
ENDICESVR SVRID(SYSTEST) RESTART(*YES)
```

3

Run the report application:

Now we need to setup the URL in the browser - First: the name or IP-address and the port-number of you IBMi: In my case it is : http://myIBMi:60000

Next: The package name **slash** the class name: In my case that is IceBreakTest/simple

Now the method prefixed by a **colon** in this case :listHtml

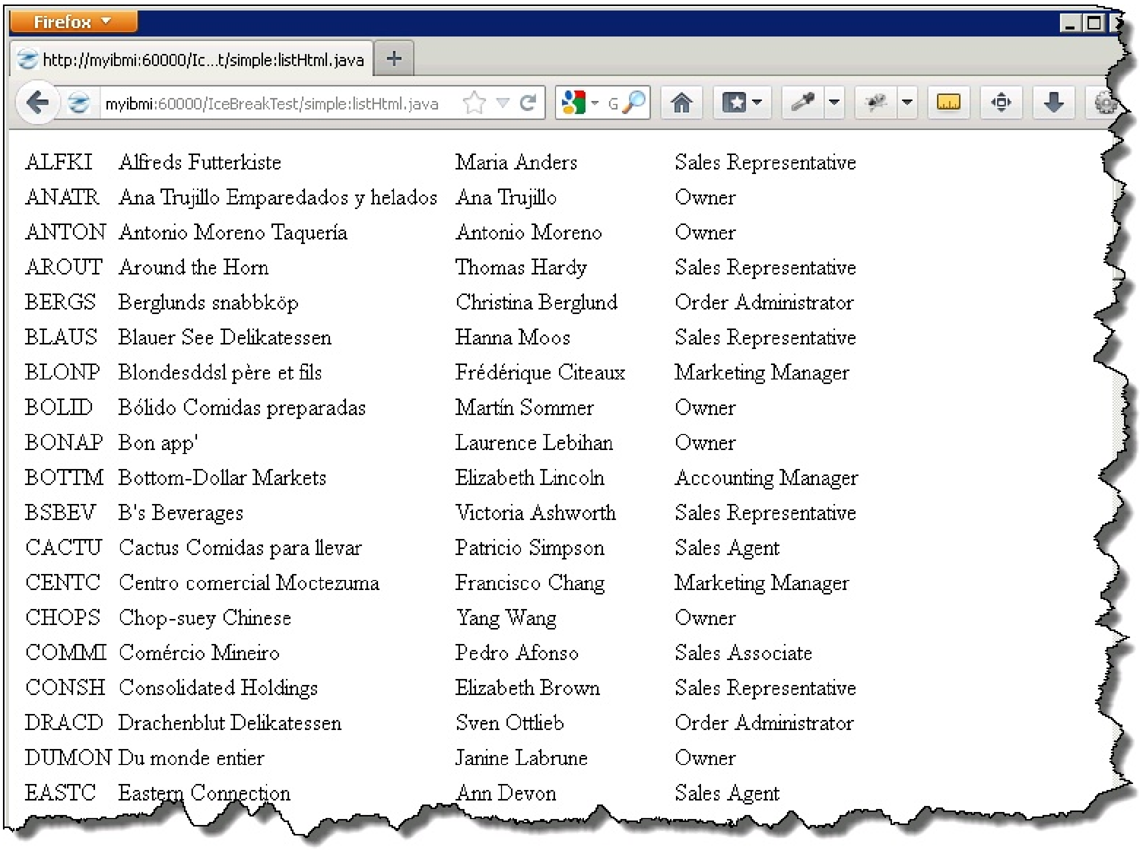
Finally - all java need to be suffixed by .java to route the request to the IceBreak JVM

So the URL looks like:

```
http://myibmi:60000/IceBreakTest/simple:listHtml.java
```

Again !! NOTE !! NOTE !!: Beware - Java is case sensitive so part of the URL are case sensitive. Everything from the package name and forward need to be in the right upper or lower-case.

If you have done it right, and configured a server in your network to allow JDBC and change the code above accordingly - it will produce the following report:


Firefox
http://myibmi:60000/Ic...t/simple:listHtml.java
myibmi:60000/IceBreakTest/simple:listHtml.java

ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner
AROUT	Around the Horn	Thomas Hardy	Sales Representative
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative
BLONP	Blondesddsl père et fils	Frédérique Citeaux	Marketing Manager
BOLID	Bólido Comidas preparadas	Martín Sommer	Owner
BONAP	Bon app'	Laurence Lebihan	Owner
BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager
BSBEV	B's Beverages	Victoria Ashworth	Sales Representative
CACTU	Cactus Comidas para llevar	Patricio Simpson	Sales Agent
CENTC	Centro comercial Moctezuma	Francisco Chang	Marketing Manager
CHOPS	Chop-suey Chinese	Yang Wang	Owner
COMMI	Comércio Mineiro	Pedro Afonso	Sales Associate
CONSH	Consolidated Holdings	Elizabeth Brown	Sales Representative
DRACD	Drachenblut Delikatessen	Sven Ottlieb	Order Administrator
DUMON	Du monde entier	Janine Labrune	Owner
EASTC	Eastern Connection	Ann Devon	Sales Agent

Amazing ... :)

Part IX

9 Utilities

9.1 Introduction

Utilities:

Included in the standard IceBreak installation you will find IceBreak Utility (Services program). The IceBreak Utility (IceUtility) is constantly increased with function useful for IceBreak programmers. This document describe how to use the IceUtility functions and how to bind from you own IceBreak program to the IceUtility service program.

All samples in this document are written in IceBreak RPGLE.

Why are these utilities not part of the IceBreak core?

All functions in the IceUtility is "stand alone" and does not requires the IceBreak core. hence, you can use it to other the web based applications. All functions in the iceBreak core relies on a server instance and the object model around that.

9.1.1 Bind to IceUtility

Every time you need to use one or more functions from the IceUtility you must refer to a Bind directory call ICEUTILITY like:

```
H BndDir('ICEUTILITY')
```

9.1.2 Prototyping

You must include member “ICEUTILITY” placed in file “QASPHDR” placed in the IceBreak library for the right prototyping of the functions use by the IceUtility like:

```
/Include qAspHdr,IceUTILITY
```

9.2 Functions

In the following I describe function by function how to use them – normally with a little sample.

Legend:

- bool => Indicator variable
- varying => Variable-Length Character field
- * => pointer

9.2.1 Administrator();

Returns *ON if the user has Administrator rights.

Sample:

```
If Administrator();
   // Do some Administrator stuff
Else;
```

```
   //  The user is Not an Administrator
endif;
```

9.2.2 Disabled(disable);

Returns “disabled=disabled” if disable is *ON

Use this function to make it easier to disable input for users depending on RPG fields directly.

Sample:

If field “MyBool” contains *ON the input field name “MyName” will be disabled and the user will NOT be able to enter data into it.

```
<Input <%=disabled(MyBool)%> Type="text" name="MyName" value ="<%=MyName %>"  >
```

9.2.3 DropDown(TagName: SqlCmd {:SelectedValue} {: HtmlExtend});

Writes a formatted HTML “select” back to the response object formatted with output from a SQL select command.

Parameters:

FormName:

Form name that can be read with the standard IceBreak Form function. e.g.:

```
Manuid = Form('PRODUCT.MANUID');
```

Sql command:
Enter a Sql command. The Sql output will be placed in the select dropdown list. If you select two or more fields in the select, the first one will be returned as the "key" field eg. <option value="CANON"... the last one will be used as the text. All in between the first and the last field will not be used.

Selected value (optional):
Use this parameter to set a certain record to be selected. Enter the "Key" value that the first Sql Select field should be compared with.

HtmlExtend (optional):
If you need to add some extra keywords to the genereted HTML this one can be used. E.g. 'onblur="MyScript();"' will execute the MyScript() when the user leave the field.

Sample:

```
 DropDown('PRODUCT.MANUID':
         'Select ManuId, desc from Manufact order by 1':
          ‘OLYMPUS’);
```

Will write the following to the response object:

```
<select name="PRODUCT.MANUID" id="PRODUCT.MANUID" />
  <option value="ACER">Acer</option>
  <option value="CANON">Canon</option>
  <option value="CASIO">Casio</option>
  <option value="FUJIFILM">Fujifilm</option>
  <option value="HP">HP</option>
  <option value="KODAK">Kodak</option>
  <option value="KONICA">Konica</option>
  <option value="NIKON">Nikon</option>
  <option selected=selected value="OLYMPUS">Olympus</option>
  <option value="PANASONIC">Panasonic</option>
  <option value="SAMSUNG">Samsung</option>
  <option value="SONY">Sony</option>
</select>
```

Check out “/icebreak/Tutorials/ex19.htm” for a demo of the function.

9.2.4 exists(varying);

Check IFS and return *ON if found

Sample:

```
if (exists('/path/file.ext'));
  // do some stuff for the STMF
endif;
```

9.2.5 FieldListOpen(Library: File);

Open and prepare a list of fields for a database file. Together with FieldListRead() you can generate a list of fields within the first record format for a file.

The first parameter must be the library name where the file name from the second parameter is located.

Sample:

FieldListOpen('MYLIB' : 'MYFILE');

For more samples check out FieldListRead();

9.2.6 FieldListRead();

Reads a row of data prepared with FieldListOpen(); The function returns its data into a data structure defined as a pointer to it.

Sample:

This sample opens a file in a library and loops through the “File Field Description” for the file and set the field “myField” to the name of the field name (ffd.fieldName) just read.

```
D pFFD            s               *
D FFD             ds                  likeds(FLDL0100) based(pFFD)

  if (FieldListOpen(UpperCase(Lib) : UpperCase(File)));
    pFFD = FieldListRead();
    dow (pFFD  <> *NULL);
      myField = ffd.fieldName;
      pFFD = FieldListRead();
    enddo;
  endif;
```

9.2.7 LowerCase(varying);

Convert and return a string in lowercase.

Sample:

```
UserID = LowerCase(UserID);
```

9.2.8 MemberListOpen(Library: File: Member {:Format} {:OverrideProcessing});

Together with function MemberListRead() it lets you generate a list of member names and descriptive information based on specified selection parameters.

The first parameter must be the library name where the file name from the second parameter is located.

“Format” is an optional parameter with default set to QUSL0200

“OverrideProcessing” is an optional parameter with default set to *OFF

Sample:

MemberListOpen('MYLIB': 'MYFILE': 'MYMEMBER);

For more samples check out MemberListRead();

For more information see OS/400 API QUSLMBR (http://publib.boulder.ibm.com/IBMi/v5r2/ic2924/index.htm?info/apis/quslmbr.htm)

9.2.9 MemberListRead();

Reads a row of data prepared with MemberListOpen() The function returns its data into a data structure defined as a pointer to it. You must select a structure that fits the chosen format in the MemberListOpen – that is:

MBRL0100 Member name use data structure QUSL010000
MBRL0200 Member name and source information use data structure QUSL0200 (Default)

For MBRL0310, MBRL0320, MBRL0330 check the description @: http://publib.boulder.ibm.com/IBMi/v5r2/ic2924/index.htm?info/apis/quslmbr.htm

Sample:

This sample open and prepare a list of members to be read and sets the field “MyMember” to the member name just read.

```
H BNDDIR('ICEUTILITY')
D pML             s               *
D ML              ds                  likeds(QUSL0200) based(pOL)

 /Include qAspHdr,IceUtility

  if (MemberListOpen(UpperCase(Lib): UpperCase(File): UpperCase(Member)));
    pML = MemberListRead();
    dow (pML  <> *NULL);
      MyMember = ml.QUSMN01;
      pML = MemberListRead();
    enddo;
  endif;
```

9.2.10 ObjectExists(Library: Object: ObjectType);

This function checks object existence and verifies the user's authority to the object before trying to access it.

Sample:

```
if (NOT ObjectExists('QSYS': 'MyLib': '*LIB'));
   // Library MyLib was not found
endif;
```

9.2.11 ObjectListOpen(Library: Object: Type {: Format});

Together with function ObjectListRead() it lets you generate a list of object names and descriptive

information based on specified selection parameters.

The third optional parameter "Format" may either be a specific object type, or a special value of *ALL. For a complete list of the available object types, see the OS/400 API QUSLOBJ manual.

Sample:

```
ObjectListOpen('MYLIB': 'MYFILE': '*FILE'));
```

This sample check for the existing of file MYFILE placed in library MYLIB

For more information see OS/400 API QUSLOBJ (http://publib.boulder.ibm.com/IBMi/v5r2/ic2924/index.htm?info/apis/quslobj.htm)

“Format” is an optional parameter with default set to OBJL0200

9.2.12 ObjectListRead();

Reads a row of data prepared with ObjectListOpen(); The function returns its data into a data structure defined as a pointer to it. You must select a structure that fits the chosen format in the ObjectListOpen – that is:

OBJL0100 Object names (fastest) use data structure QUSL010003
OBJL0200 Text description and extended attribute use data structure QUSL020002 (Default)
OBJL0300 Basic object information use data structure QUSL030000
OBJL0400 Creation information use data structure QUSL0400
OBJL0500 Save and restore information; journal information use data structure QUSL0500
OBJL0600 Usage information use data structure QUSL0600
OBJL0700 All object information (slowest) use data structure QUSL0700

Sample:

This sample open and prepare a list of objects to be read and sets the field “MyObject” to the object name just read.

```
H BNDDIR(' ICEUTILITY')
D pOL             s               *
D OL              ds                  likeds(QUSL020002) based(pOL)

 /Include qAspHdr,IceUTILITY

  if (ObjectListOpen(UpperCase(Lib): UpperCase(Obj): UpperCase(Type)));
    pOL = ObjectListRead();
    dow (pOL  <> *NULL);
      MyObject = ol.QUSOBJNU00;
      pOL = ObjectListRead();
    enddo;
  endif;
```

9.2.13 ReverseDNSlookup(IP address);

Reverse DNS lookup.

This function returns the host name for an IP address.

Sample:

```
HostName = ReverseDNSlookup ('129.42.16.103');
```

This HostName will hold 'www.ibm.com'

9.2.14 UpperCase(varying);

Convert and return a string in UPPERCASE.

Sample:

```
UserID = UpperCase(UserID);
```

9.2.15 UserExists(UserProfile);

Check OS/400 User profile and return *ON if found.

Sample 1:

```
 bool =  UserExists('JOHN');
```

Sample 2:

```
if UserExists(User);
   // The User exists
else;
  // The User does not exists
endif;
```

9.3 Samples

9.3.1 DropDown sample

This sample shows how to use the DropDown function in an IceBreak RPGLE program.

The code is separated in a RPGLE part and a HTML part. The IceBreak pre-compiler collects the needed code into one source code and compile the program (for more information see tutorials/ex01Include.htm).

9.3.1.1 HTML code

```
<!--#tag="Init"-->
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
    <link rel="stylesheet" type="text/css" href="/System/Styles/IceBreak.css"/>
    <script language="JavaScript"  type="text/javascript" src="/System/Scripts/util.js"></
script>
  </head>

  <!--#tag="Page01"-->
  <body onload="adSetFormStatus();">
    <form method="POST" name="form1" id="form1" action="#">
      <fieldset><legend>Maintain product information (<%=ModeText%>)</legend>
        <table>
          <tr>
            <td style="width: 119px">Product key</td>
                  <td><input class="<%=Class_Key%> NUM" type="text" name="PRODUCT.PRODKEY"
value ="<% = %char(PRODKEY) %>" ></td>
                  <td class="ErrorText" ><%=SesGetVar('PRODUCT.PRODKEY.error')%> </td>
               </tr>
               <tr>
                  <td style="width: 119px">Product ID</td>
                  <td><input type="text" name="PRODUCT.PRODID" size="<%=
%char(%size(PRODID))%>" maxlength="<%= %char(%size(PRODID))%>" value ="<% = PRODID %>"
></td>
                  <td class="ErrorText" ><%=SesGetVar('PRODUCT.PRODID.error')%> </td>
               </tr>
               <tr>
                <td>Description</td>
                <td><input type="text" name="PRODUCT.DESC" size="50" maxlength="<%=
%char(%size(DESC))%>" value ="<% = DESC %>" style="float: left"></td>
                  <td class="ErrorText" ><%=SesGetVar('PRODUCT.DESC.error')%> </td>
               </tr>
               <tr>
                <td>Manufacturer ID</td>
                <td>
                <%DropDown('PRODUCT.MANUID':
                  'Select ManuId, desc from Manufact order by 1':
                   Manuid);%>
                </td>
               </tr>
               <tr>
                <td>Price</td>
                <td><input type="text" class="NUM"  name="PRODUCT.PRICE" value ="<% =
%char(PRICE) %>"></td>
                  <td class="ErrorText" ><%=SesGetVar('PRODUCT.PRICE.error')%> </td>
               </tr>
               <tr>
                <td>Stock Count</td>
                <td><input type="text" onblur="adCheckRange(this , -99, 99);" name
="PRODUCT.STOCKCNT" value ="<% = %char(STOCKCNT) %>"></td>
                  <td class="ErrorText" ><%=SesGetVar('PRODUCT.STOCKCNT.error')%> </td>
               </tr>
               <tr>
                <td>Stock Date</td>
                <td><input type="text" name="PRODUCT.STOCKDATE" size="<%=
%char(%size(STOCKDATE))%>" maxlength="<%= %char(%size(STOCKDATE))%>" value ="<% =
%char(STOCKDATE: *ISO) %>"></td>
                  <td class="ErrorText" ><%=SesGetVar('PRODUCT.STOCKDATE.error')%> </td>
               </tr>
            </table>
         </fieldset>
         <input type="hidden" name="Mode" id="Mode" value="<%=Mode%>"/>
         <input type="hidden" name="showCount" id="showCount" value="<%=%char(showCount)%>
"/>
         <input type="hidden" name="inputOk" id="inputOk" value=true/>
         <input type="submit" value="OK" name="Func">
         <input type="button" onclick="history.go(<%=%char(showCount * -1)%>);" value
="Cancel" name="Cancel">
      </form>
      <!--#tag="GoBack"-->
      <script type="text/javascript">history.go(<%=%char(showCount * -1)%>);</script>
      <!--#tag="Alert"-->
      <script language=javascript type="text/javascript">
         alert("<%=AlertText%>");
      </script>

   <!--#tag="Exit"-->
   </body>
</html>
```

9.3.1.2 RPGLE code

```
<%
H DecEdit(*JOBRUN)                                                         Use correct decimal p
H BNDDIR('ICEUTILITY')

F*Filename+IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++++++++Comments++++++++++++
FPRODUCT   uf A E           K disk                                         The file beeing maint

/Include qAspHdr,IceUtils

D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments++++++++++++
D Func            S             10                                         Will contian 'OK' whe
D Mode            S             10                                         ChangeMode or AddMode
D Error           S               n                                        Error indicator
D AlertText       S             64                                         The Alert text to be
D showCount       S              5  0                                      Number of times this

D ModeText        S             64                                         The displayed text fo
D ChangeMode      c                   const('UPDATE')                      Define constant for C
D AddMode         c                   const('ADD')                         Define constant for A

D Class_Key       S             64                                         Special behaviour for

D System         SDS
D  Program          *PROC                                                  The program name

/free
 exsr init;

 // Select type of logic, when no function then assum get record
 select;

 when func = 'OK';
   exsr valInput;     // Validate input, if error then return
   if Error;
     exsr InputForm;  // Show the input form again
   else;
     exsr updateDB;   // update the record
   endif;

 other;
   exsr getRecord;    // Get the record from the QryStr
   exsr inputForm;    // Put the input form

 endsl;

 exsr exit;           // return to the web server and apply the response object

 // ------------------------------------------------------------------------
 // get the record. If not exists Jump into create mode...
 // ------------------------------------------------------------------------
 //LN01Factor1+++++++Opcode&ExtFactor2+++++++Result++++++++Len++D+HiLoEq....Comments++++++++++++

 Begsr getRecord;

   ProdKey = QryStrNum('PRODKEY'); // Read the key from the URL (e.g. http://myServer/MyPgm?Key=

   chain(n) ProdKey PRODUCTR;
   *IN80 = NOT %FOUND;
   if *in80 = *on;
     Mode = AddMode;               // Mark for ADD mode
     clear PRODUCTR;               // Clear the record format when creating a new record
```

```
  else;
    Mode = ChangeMode;            // Mark for UPDATE mode

  endif;

Endsr;
// -------------------------------------------------------------------------
// Validate input from the form
// -------------------------------------------------------------------------

Begsr valInput;

  Error = *off;                                  // No errors yet

  exsr form2DB;                                  // Move from form to internal fields

  // Validate input

  if Mode = AddMode AND ProdKey = 0;
    SesSetVar('PRODUCT.PRODKEY.error':           // Set error text for the form field in error
        'Must be > 0 when creating a record');
    Error = *on;                                 // Mark for at least one error
  endif;

  if ProdID = *blank;
    SesSetVar('PRODUCT.PRODID.error':
        'Required field!');
    Error = *on;
  endif;

  if Desc = *blank;
    SesSetVar('PRODUCT.DESC.error':
        'Required field!');
    Error = *on;
  endif;

endsr;

// -------------------------------------------------------------------------
// Now add or updates the record
// -------------------------------------------------------------------------

BegSr updateDB;

  select;

  when Mode = ChangeMode;
    chain ProdKey PRODUCTR;  // Chain to the record for update
    *IN80 = NOT %FOUND;
    if *in80 = *off;
      exsr Form2DB;          // Update fields from form fields into the records fields
      update PRODUCTR;       // Update the record
    endif;

  when Mode = AddMode;       // When add mode
    write PRODUCTR;          // Just write the record,
  endsl;

  // Include the HTML code for the "GoBack" tag here at compile time */
  // %><!--#include file="#.htm"  tag="GoBack"--><%

  showCount = 0;             // Next show will be the first
EndSr;

// -------------------------------------------------------------------------
// Form to database for the record
// -------------------------------------------------------------------------
BegSr Form2db;

  monitor;

    PRODKEY   = FormNum('PRODUCT.PRODKEY');        // Copy numeric data from the form into the
    PRODID    = Form('PRODUCT.PRODID');            // Copy character data from the form into th
    DESC      = Form('PRODUCT.DESC');
    MANUID    = Form('PRODUCT.MANUID');
    PRICE     = FormNum('PRODUCT.PRICE');
    STOCKCNT  = FormNum('PRODUCT.STOCKCNT');
    STOCKDATE = %date(Form('PRODUCT.STOCKDATE'));

  on-error *all;
    Error = *on;                                   // Error in the form, prepare for re-display
    exsr InputForm;
    AlertText = 'Error in the form, check ' +
        'numeric or date values entered!';
    // Include the Alert tag
    %><!--#include file="#"  tag="Alert"--><%

    exsr Exit;
  endmon;
EndSr;
// ------------------------------------------------------------------------
// Init set up the HTML header incl. stylesheets  and scripts
// ------------------------------------------------------------------------
Begsr Init;
  SesClrVar('PRODUCT.*');                // Clear all session vars used as error texts
  Func = Form('Func');                   // The last function used by the user (e.g. pressed OK
  Mode = Form('Mode');                   // AddMode og ChangeMode
  showCount = FormNum('showCount') + 1;  // Count number of times this panel was showed (e.g. w

  // Include the init tag
  %><!--#include file="ex19edtPrd.htm"  tag="init"--><%

endsr;

// ------------------------------------------------------------------------
// Write the input form
// ------------------------------------------------------------------------
BegSr InputForm;

  Class_Key = *blank;               // Key can be changed when blank
  if Mode = AddMode;
    ModeText = 'Add a new record';  // Put some "Create" info text on the panel
  else;
    Class_Key = 'PROTECT';          // Protect the key field when we are in change mode
    ModeText = 'Change a record';   // Put some "Change" info text on the panel
  endif;

  %><!--#include file="#"  tag="page01"--><%

EndSr;
// ------------------------------------------------------------------------
// Done
// ------------------------------------------------------------------------
Begsr exit;

  // Include the exit part of the HTML code here at compile time */
  %><!--#include file="#"  tag="exit"--><%

  return;      // Return to the browser with the response object build in this program

EndSr;
// ------------------------------------------------------------------------
```

Part X

10 Administration menu

10.1 Work with servers

This is where you maintain and configure the list of servers instances in your IceBreak sub system.

Creating a new server

At least you will need to give it a name. By default this name is used as description, rootpath and application library for your new server - more on that later.

The server name also is an eye catcher when you look in the iceBreak subsystem. Here you can see two jobs for each server when it is active. Once active job and one spare job (the backup job) which in status lock wait until it is needed - when the application pool is full or if anything happens to the the primary job.

Port

IceBreak server are listening on one specific port number in the TCP/IP protocol. When you assign port numbers it is recommended that you check the port number and see if it is available.

Use the `NETSTAT *CNN` command and see if your desired port is free for use. One port can only serve one server at the time. You can reuse the port number; however, you have to stop the server on that particular port before you start another.

Interface address

By default the value `*ANY` which of cause means that this IceBreak server is listening on all available interfaces. You can see the complete list of interfaces with the command `NETSTAT *IFC`. If you want to share the same port number between two or more IceBreak server - the just add a dedicated interface - giving it a new IP address

Mode

Where the server is in development mode it will check if the source for each particular application is changed compared to the running compiled version. If so the JIT (Just-in-time) compiler will kick in; compile the application and run it. If any kind of types or syntax error was found, the you will be presented the post list of the application.

The development mode is only available for Enterprise versions of IceBreak.

In "production" mode the JIT process is bypassed and the application will always be executed. The IceBreak PRO server version only allows the production mode, which also ensures the no application hacking can occur - according to SOX.

Target release

When compile programs the target release of the compiler can be selected with with option.

Root path

This is the IFS location where all your resources for your applications and web sites "lives". By default the root path will be a subdirectory to the "/www" path also used by the Apache server. This allows you to share resources like .css . gif .html files between IceBreak applications and Apache Web server.

Referring to the root path with in applications:

The file name and path can be absolute or relative when you build applications i.e. using the `ResponseWriteTag(Filename : TagName);`
This reference is according to the following rule:

- /path/filename.ext This absolute from the IFS root
- ./Filename.ext This is relative to the Server instance root path

- Filename.ext This is relative to the browser relative "refer location"

The above applies to application only. From the url to top level root is always the this root path for security reasons. If you want to make applications that serves other directories which is above the root, then make an application that includes this resource. then you have programmatic control over the security.

Application library

When the JIT compiler kicks in the output will be a genuine I5/OS program object. This program is placed in the application library. Since that is the case and the max length of a object in I5/OS i 10 char. you must ensure that the final object name is unique despite your IFS source is longer or is placed in subdirectories.

Default document

This is the document or application that is show if the URL to the server is blank. This document or application is also available even if the logon required parameter is set to *YES. Which allows you to build your own log on screen, and secure all resources with in the IceBreak server instance.

Default profile

Initially when a server instance is running it will run under this particular user profile - this can be overridden at runtime by the ***logon()*** build in function.

Protocol

The protocols available is HTTP and the secure version counterpart HTTPS. By default the more lightweight HTTP is selected, however if you want to make secure solutions like banking software HTTPS is the way to go. Click here to see how to configure HTTPS over SSL.

Library list

All ILE applications runs with a library list. This tab lets you set up the default library list for your job and the server instance which eventually is the same as you applications.

Trace

The trace tab gives you the possibility to enter a file name where Incoming requests and out going responses are logged.

*NO - no trace is made.
*YES will trace the headers.
*ALL will trace headers and contents. This generates a huge amount of data so be careful.

The TLOG00 database file also contains Trace for header but works as a round robin

Certificates

When you use the HTTPS protocol as described above , you will need a certificate. Click here to see how to configure HTTPS over SSL.

Advanced tab

You should normally only change the parameter on the advanced tab after you has consulted an IceBreak service representative.

Initial program

When the server instance is starting up (not the job carnation), this optional program is being called. This is convenient to set additional library list here.

Job queue

By default bot server instance and server job carnation is submitted accordingly to a job description placed with the server name in the application library. However, you can override the value from the job description with this value.

Cache timeout

This value is placed in all HTTP headers for static contents. When you are in development mode you might want to change this value to zero to avoid any client side caching.

This value can be overridden programmatically with the `setCacheTimeout(sec);` in your application.

Value is in seconds.

Session timeout

The number of minutes a session is maintained after it was detected inactive. (That is - no server round trips). When this timer expires the ICECORE will remove session related state information, variables, ILOB's.

For session stable servers also the session job is terminated causing files being closed, QTEMP library deleted, SQL cursors closed etc.

Value is in minutes.

Connection timeout

The number of seconds IceBreak is maintaining a TCP/IP connection after it was detected inactive. (That is - no server round trips). When this timer expires the server instance thread that is directly communicating with the client is terminated and resources is released back to the i5/OS.

The connection timeout has no impact on the session which will remain "alive" despite the connection is terminated. If the client reconnects, a new server instance thread will be created and reconnected to the session.

Value is in seconds.

Buffer size

The input, output and cookie buffer are only used when the server runs in multi processed mode. Buffers in that particular mode are allocated using teraspaces . The larger buffer the larger foot print.

The buffers in other server modes are allocated by ILOB's which again are based on userspaces.

Value in in bytes

Coded character set ID

The CCSID is used when converting back and forth between client codepage and server side codepage which can be ascii, unicode and utf-8 for the client and any EBCDIC for the server side. Also the result from the IceBreak pre-compiler places the code in QASPSRC in the application library. This source file will be created with this CCSID.

Default response trimming

When IceBreak places dynamic data in the response it will be blank trimmed accordingly to this parameter. See the tutorial.

Shared application library

Default NO. The application library might also contain additional programs which may not be accessible from the URL. in that case you can change this value to YES, which in turn causes an extra validation of the object - it must reside as a row in the DB/2 table DIR00 within IceBreak core library usually called ICEBREAK. When you use this feature you must provide a deployment feature to update the DIR00 table on the target IceBreak server.

This feature is basically used for OEM version of IceBreak.

We suggest you keep this value to NO and place your additionally programs in an sperate library which is accessible using the library list. The URL reference will always be calling the application in the application library.

Pre-request exit program

You can define your own security algorithms for any resource within IceBreak. This optional ASPX-program is called prior to any content is returned to the client.

See the tutorial.

Add File Server Share

When you select yes, a share to the server root path on the IFS is automatically created. Now you can map this share as a network drive in you Windows (or using SAMBA on Mac and Linux) to edit the source stream file files that eventually makes the website or application.

WebService URI

When you build webservices, it need to define a URI for you reaource. This is the name that will be statically bound to the web-service.

See the tutorial.

Days to keep transmission log

Each request / response is logged in the DB/2 table TLOG00 in the icebreak core library. Compared to the trace the TLOG00 is in tabular form, which makes it suitable for any network statistics, such as turn-around time and debugging because you can query it with e.g. SQL.

Rows from the TLOG00 table will be removed when they are older that this parameter.

Session type

A session can be maintained either by a cookie - a small token the browser will return to the server for each subsequent request. Or it can be maintained by redirection to a session unique virtual sub-path.

pros/cons:

- Cookies can be disabled - but session cookies are normally not;
- URL session can be bookmarked - but this might not cause a problem for you application.
- Cookie-2 is only supported by new (Microsoft) browsers, but can not be disabled.

Dispatcher method

IceBreak has several modes to dispatch the serving job. Each which has their own benefits - See technical documentation.

The short version:

- Use MULTITHREAD with pool size set to zero for intranet solutions.
- Use MULTITHREAD with pool size arround 5 jobs for internet solution.

Multi thread server options

The following option are only available for server running with the multi threaded IceBreak core.

Heartbeats

If you will assure that the IceBreak server core is running you can enter the TCP/IP address of the interface and the frequency you want to ping with a heartbeat.

If the heartbeat is not responded - IceBreak will launch a new carnation of the server job, and kill any server job that were suppose to serve.

Leave that blank if you have a monitoring system or don't want to take up bandwidth with network pings.

Max Sessions per Server job

This is the part of IceBreak load balancing. After around 250 concurrent network connections TCP/IP jobs start degrade the performance since the job will be overloaded.

If you exceed this parameter value a new instance of the server job will be started, which again is ready for up to 250 concurrent network connections.

You can set til value to a lower value on smaller system to gain overall performance for the IceBreak applications on the cost of the total system performance.

The math is simple - ex. 2500 concurrent users with the limit set to 250:

2500 / 250 = 10 IceBreak server jobs

Max transactions per Server job

You can renew the server instance job after a number of transactions to clean up heap/stack - to release memory.

0=No max; the server instance job in never renewed.

Max Sockets per Server job

Related to max session per server job, this is the actual number of physical connections to clients. if you keep these two values the same then any of the values that is exceed first will cause the new instance to start.

Server job run priority

The i5/OS job priority for the server instance - the job that is connected to the client.

Session job run priority

The i5/OS job priority for the session job - the job that is runs the application.

Job Pool Size

The pool size has only effect for multithreaded job dispatching.

This value set equal to zero disables the pool which again runs the server in session stable mode / job persistent mode. This is recommended for intranet applications, since you have a physical job for each client; just like 5250 application.

When you change this value to a value above zero this again reflect the number of physical jobs that serve application requests.

You can never be assure that any subsequent request from a client will hit the same server instance except when you set the value to on - since there will only one job to serve all request. This can be valuable knowledge when making data-queue like application / web-service like applications.

see technical documentation for dispatcher methods.

10.2 SQL prompt

Administration Menu:

You can can enter any select SQL select statement from this prompt. Other SQL statements like "update" or "delete" are however not allowed for security reasons.

Try the prompt

In the Administration Menu under the Tool menu you will find the SQL prompt. Try the following SQL statement produces a report over some digital cameras from a demo table:

```
Select * from product order by 1
```

You can select the output format to be in HTML or XML from the drop down box.

Using the output XML in Excel

If you right on the XML output, you will find that you can open the result set with Microsoft Excel:

Right click and select import.

External references to the SQL query

You can copy the final URL from the statement to you clip board so you can run the query from out side the the administration menu.

10.3 Application wizard

The application wizard builds a skeleton program that you can use as a foundation for your own programs

You can build your own templates for this application generator if you like. Just save your templates in the IFS path where IceBreak is in stalled. Locate the sub-folder for your programming language:

RPG: \IceBreak\Wizards\Templates\RPG
COBOL: \IceBreak\Wizards\Templates\COBOL

Remember to save your templates before you upgrade IceBreak to a new version.

10.4 Component wizard

Like the application wizard, the component wizard generates code. However, it ranges from code snippets to complete programs you can use as
foundation for your own application.

The component wizard uses a database file or a SQL table / view and generate the code corresponding to you selection it in the dropdown box.

The code or code snippet are shown in the text box where from you can select it by clicking the text box and press the key combination <CTRL><A> <CTRL><C> Then you can paste it into you source for your application.

10.5 Sencha Touch Component Wizard

The IceBreak Sencha Thouch code generator / component wizard is quite straight forward. After you fire it up, you simply fill in generation parameters and it will produce a directory with j a number of sub directories containing the iPhone / iPad / android application.

Basically the application is a HTML5 based web application, however i utilize all the features in HTML5 and / or the web-kit environment allowing a web application to have the same look and feel as a native application, but can be installed and run from the internet directly from your IceBreak Server. The directory also it contains compilation parameter required if you want to create a native handheld application that can be downloaded and installed from Apple AppStore or android market and run totally offline.

Prerequsits

Before you start you need to download a version of the Senche Touch frame work . The application generator is designed to work with Sencha touch version 2. This however will change over time. For copyrights reason the IceBreak server is not distributed with the Sencha touch frame work, the installation on the other hand i quite easy:

1. Download Sencha touch from http://www.sencha.com/products/touch/download/
2. Unpack the zip file
3. Place the Sencha Touch directory under the /System/Plugins directory of your IceBreak application server.
4. The "/System/Plugins" directory:

Please note that the **/System/Plugins** directory is a symbolic link and is therefore available in all server instances via the system hive. Also **/System/Plugins** are left unchanged when you update IceBreak to the next version. The **/System/Plugins** directory are not changed, deleted or updated by IceBreak, you, however, must have in mind that any changes you apply to **/System/Plugins** will be propagated to all server instances of IceBreak server on your system. Therefore: Do not delete it!!

Also the **/System/Plugins** directory is perfect for shared resources between IceBreak servers – just to name a few that might interest you:

- jQuery
- ExtJs
- Sencha Touch
- jQuery Touch
- DoJo
- Capuchino

Create My First App

Now lets fire up the code generator and build out first application:

- Open the IceBreak Adminstration
- Open the "tool" menu
- Click on the "Sencha Touch generor"
- Now you will se the generator wizard:
- Now lets fill in the wizard form:

1. First the fille in the library and file: Lets use the simple PRODUCT database table which is default available in *LIBL of you IceBreak installation.
2. Select the component type: Now just select the List and edit for application.
3. Select the server where it will be installed and run from: The server need to be in development mode before the wizard successfully can compile the RPGLE-REST service it will create.
4. Enter the name of the "app" this will also be the name of the directory created for you app, so don't enter a name so overwriting existing stuff will occur. Also avoid spaces and hyphens.
5. Enter a a description of the app: This will be the name you see on the iPhone/iPad/Androide desktop menu when you attaché the app tou you home screen – So give it at proper description but not to long – the phone desktop has limited space.

Now you are good to go!! Press the "generate" and let the magic begin - it might run for a while.

Run my first App

Open your iPhone/iPad/androide and open the browser: Enter the url of the application you just made:

```
http://myIBMI:1234/myapp/index.html
```

myIBMI is the ip address or name of you IBM
1234 denotes the port number you already have configured for you IceBreakserver
Myappis the exact name of you app you entered in the above opt 4
Index.html is The default document – just use that name for now

Now you will see an application similar to this:

It has the following features:

- It only loads 25 rows from the sever at the time to reduce net workload – this can of cause be changed. But when you page through it loads the 25 etc.
- It makes a generic search on all columns in the basing table – this can be modified later
- When you click on a row, it brings up a form where you can change the row contents, and update the contents.
- You can create new rows in the table
- When you swipe down on the first page it will reload the list contents.

The application will in all cases be based on the access to a DB/2 database file. So the application generator will build the following components for you:

- The Ajax based client code
- The REST service – which btw also contains a full-blown web 2.0 CRUD web-desktop application
- And IFS folder as a subfolder in you application server root that contains the source to the above.

10.6 Source file browser

Since IceBreak applications source is based on IFS stream files you need at too to migrate your legacy code from source physical files.

The "source file browser" lets cut and paste from a source physical file member and trim the first six left characters off if you like - not recommended if you are using WDSc, but is useful if you are using i.e. Microsoft Visual Web Developer tool.

10.7 Services

Services is a collection of features that enhance the IceBreak server core. Services can be installed as add-on products via framework deployment.

10.7.1 Web TAPI

Web TAPI is a service that brings the "telephony API" on the client available for IceBreak applications on system i.

This is excellent when you want to build web based call center applications or extend a CRM solution with a "dial customer button".

You need to download the "web TAPI client service" to each PC that will be using the feature.

You will find the "web TAPI client service" for download at http://icebreak.org/tools.htm

10.7.2 Web printers

Web printer is a service that brings clients printers available as output queues on the system i.

On the client you have to install the "Web printer client service" which will expose the client connected printers over HTTP. The transmission of data is 128 bit encrypted over a plain HTTP protocol.

You will find the "Web printer client service" for download at http://icebreak.org/tools.htm

10.8 Frameworks

In IceBreak you can deploy applications as "Frameworks". A framework is a generic for Applications , services, plug-ins and complete solutions. IceBreak frameworks were introduced to help developers deploying their applications and plug-ins to other IceBreak installations. You can use IceBreak frameworks to deploy you own applications or installing IceBreak applications and Plug-ins from the IceBreak community and software vendors. To find new software from the IceBreak community you can visit http://icebreak.org or select “Work with IceBreak Frameworks, Applications and Plug-ins.” from the main IceBreak menu.

When you have a server instance with all your own applications, then you can export all its contents both program objects from the application library - along with IFS objects .html, .gif's etc. directly to a stream file on the IFS.

Framework files has the extension of .IFW (IceBreak FrameWork).

Export a framework:

1. Logon to your IceBreak admin page
2. Click on work with servers
3. Select the server that contains what you want to export
4. Click "Edit server settings"
5. Click "Export as framework"

Fill in to export form and press "Export"

The Export Server panel are use to prepare and the export of a server as a framework.

Product information

Indicate the product information's of the server on the target system.

Description
The default description shown when the end user installs the framework.

Vendor
The name of the vendor of this framework.

Vendor URL
Enter a URL to the vendors homepage. Later when the end user are going to install the framework on the target system this link will be active.

URL
Enter a URL to the products homepage. Later when the end user are going to install the framework on the target system this link will be active.

Version
The version of the framework can be entered here. You will see that the "file name" will altered automatically including the version entered.

IceBreak Server ID
This Server ID will be the server ID that the end user will be shown as the default server ID.

Port number
Use this parameter to specify the default TCP/IP port used for this IceBreak server exported.

Application library
The default application library name for the target system.

Http Root Path
The IFS default root path for this framework on the target system.

User exit programs on the target installation
You can define user exit programs to be called on the target system before and after the installation has occurred.

The "before" exit program will be executed as the first process on the target system. The program will be restored into QTEMP and executed from their.

The post exit program will be called as the finale step in the installation process. The library QTEMP holds a data area called NEWSERVER with a logical value = '1' if the server was new and was created during the installation process.

Sample post exit CL program:

```
/* Post Framework install exit program */
pgm
  Dcl &NewServer  *lgl

  RtvDtaara Qtemp/NewServer &NewServer
  if (&NewServer) then(Do)
     clrpfm GROUP00
     monmsg cpf0000
     clrpfm QUERY00
     monmsg cpf0000
     RUNSQL     SQLCMD('Insert into GROUP00 ("GROUP", TEXT, +
                DESC) VALUES(''MYFIRST'', ''My first +
                group in Inspire'', ''This is My first +
                group in IceBreak Inspire'')')
     monmsg cpf0000
  enddo
  else do
   /* Upgrade */
     RunSQL SqlCmd('update query00 set qrytk = rrn(query00) where qrytk = 0')
     monmsg (sql0000 cpf0000)
  enddo

endpgm
```

Hint!

You can retrieve information of the framework just installed by use of the command RTVICESVRA *RECENT.

Sample:

If you need to know the name of the application library entered by the user under the installation process you can use the following line:

```
RtvIceSvrA SvrId(*Recent) AppLib(&AppLib)
```

The field &Applib will hold the name of the application library just installed into.

You can retrieve the following information:

```
Returned SVRID            (10)
Tcp/Ip Port              (5 0)
Tcp/Ip Interface          (15)
Description               (50)
Server type               (10)
Startup type               (7)  *AUTO, *MANUAL
Http Root Path           (128)
www Default Document      (32)
Initial Program           (10)
Initial Program Library   (10)
Job queue                 (10)
Job queue Library         (10)
Application Library       (10)
Shared Application Lib     (1)
Target release            (10)
Create Trace file          (1)
Trace Path               (128)
Days to keep trans.log   (9 0)
Mode                       (1)
Mode as text              (10)  *PROD, *DEVELOP
Log on required            (1)
Default User profile      (10)
Cache Timeout (sec.)     (9 0)
Session timeout          (9 0)
Connection timeout       (9 0)
Input buffer sizes       (9 0)
Output buffer sizes      (9 0)
```

```
Cookie buffer sizes      (9 0)
Session type              (10)  *COOKIE, *URLREDIR, *COOKIE2
TcpIp Heartbeat address   (15)
Sec.between Heartbeats   (9 0)
Max ses. pr Server job   (9 0)
Max trans.pr Server job  (9 0)
Max Sockets pr Server job(9 0)
Server job priority      (2 0)  0=*CLASS
Session job priority     (2 0)  0=*CLASS
Job Pool Size            (9 0)  0=*JOBSTABLE
Coded character set ID   (5 0)
Protocol                   (6)  *HTTP, *HTTPS
Certificate Path & File  (128)
Certificate Password      (32)
Prerequest exit program   (10)
Prerequest exit PGM LIB   (10)
WebService URI           (128)
```

What will be saved?

- All objects in the application library except the job description named after the IceBreak server ID.
- All objects in the IFS root path defined by the ADDICESVR's keyword HTTPPATH. Including almost all subdirectories. Directory /System and /Exclude will NOT be saved.
- The IceBreak server attributes needed for the framework installation program to create the server on the target system.
- All the save operations above are collected in one IFS object named by the user. Normally with file extension .ifw (e.g. /IceBreak/FrameWorks/MyFramework.ifw)

Note!

- The target release will be the same as the one from the IceBreak server being exported.
- Object owner of all objects in the http path are changed to QPGMR

Universal Framework ID

The first time you export a IceBreak server the export feature generates a Universal Framework ID. The Universal Framework ID are used to identify the framework when it is restored on the target system. When using this technique the installation process will be able to update the server next time the vendor releases a new version.

Import a framework:

1. Logon to your IceBreak admin page on the target system;
2. Click on Configuration:
3. Click Frameworks

The framework installations has three steps:

1. Downloading the IFW file from the IceBreak framework community website to your PC.
2. Uploading the IFW file from your PC to the IFS and register it to IceBreak.
3. Install the IFW file from IFS to a library and server root path, create all stuff needed and start the server.

Downloading the IFW file from a website to your PC.

- Click on the WEB->PC tab.
- Select the framework you want to download
- click green download arrow

You can skip this first step if you have a local copy of a framework and are locally connected to the system i where IceBreak runs.

Uploading the IFW file from your PC to the IFS and register it to IceBreak.

Now you have the IFW on your PC:

- Click on the PC->IFS tab.
- Find the file with “search”
- Click “Upload”

The IFW deployment file is then registered in the iceBreak server.

Install the IFW file from IFS to a library and server root path

- Click on the IFS->LIB tab.
- Select the framework on the list by clicking the “+” sign
- Follow the installation guide

The IceBreak Framework Installer are shown with the following information's.

Framework ID

Enter the ID for the framework ID

Port number

IceBreak server are listening on one specific port number in the TCP/IP protocol. When you assign port numbers it is recommended that you check the port number and see if it is available.

Use the `NETSTAT *CNN` command and see if your desired port is free for use. One port can only serve one server at the time. You can reuse the port number; however, you have to stop the server on that particular port before you start another.

Description

Enter the description for the server into this field.

Application library name

When the JIT compiler kicks in the output will be a genuine I5/OS program object. This program is placed in the application library. Since that is the case and the max length of a object in I5/OS i 10 char. you must ensure that the final object name is unique despite your IFS source is longer or is placed in subdirectories.

HTTP IFS path

This is the IFS location where all your resources for your applications and web sites "lives". By default the root path will be a subdirectory to the "/www" path also used by the Apache server. This allows you to share resources like .css . gif .html files between IceBreak applications and Apache Web server.

Referring to the root path with in applications:

The file name and path can be absolute or relative when you build applications i.e. using the `ResponseWriteTag(Filename : TagName);`
This reference is according to the following rule:

- /path/filename.ext This absolute from the IFS root
- ./Filename.ext This is relative to the Server instance root path
- Filename.ext This is relative to the browser relative "refer location"

The above applies to application only. From the url to top level root is always the this root path for security reasons. If you want to make applications that serves other directories which is above the root, then make an application that includes this resource. then you have programmatic control over the security.

10.9 Display current server

This list is all the available server variables and their corresponding values. This is achieved by the build-in function `getServerVar();`

You can learn how that program is build here: servervars.

10.10 Display current header info

The list here is the information sent from you browser to the IceBreak server.

Click here to learn more about the HTTP headers within a IceBreak program.

10.11 Display all servers

IceBreak can be installed in different libraries on your system i.e. if you want to run it in several versions or if you want a dedicated production / development environment.

Each version also gives you a dedicated subsystem along with a dedicated administration.

The "Display all servers" gives you a complete list of server on the system despite which library it is configured in. You can sort the on library , port number or server name.

A trailing list shows you the libraries that contains IceBreak servers - this includes also products where IceBreak runs as OEM.

10.12 Display joblog

IceBreak supports a dedicated i5/OS job for each "browser" job. That means that you also a dedicated joblog for each browser job.

This joblog is very convenient when you want to debug an applications. This is where you programs exceptions will be logged.

However, when you run the servers with the application pool activated, the joblog gives less meaning since you will not know which job was serving the request - unless you set the pool size to one.

10.13 Display job

IceBreak supports a dedicated i5/OS job for each "browser" job. The "display job" feature is very convenient when you want to debug an applications.

You can see the current program stack and which files are open etc.

However, when you run the servers with the application pool activated, the display job gives less meaning since you will not know which job was serving the request - unless you set the pool size to one.

The "Display job" is equivalent to the "DSPJOB" i5/OS command.

10.14 Display trace

The trace contains - if activated on the server instance - all HTTP traffic between the client (browser) and the IceBreak server instance.

The trace files is located on the IFS as a stream file.

10.15 Authorization

This is where you configure the access to the administration menu in IceBreak.

The configuration parameters are activated after you restart the server instance.

10.16 Subsystem configuration

The behaviour of IceBreak subsystem can be configured here.

Autostart the IceBreak subsystem after IPL.

If you select this checkbox, an IceBreak auto-start job entry will be added to your controlling subsystem.

If you prefer to start IceBreak - i.e. in the QSTRUP program - you can just click "NO" in the check box and use the following code:

```
ICEBREAK/STRICESBS
```

10.17 Your information

The information on this form will be used when you upgrade or register you licence of IceBreak.

10.18 License information

Before you can use IceBreak you need a valid licence key.

The Licence key can be obtained from the web site www.icebreak.org simply by clicking "order Licence code" from where you can select which service contract you will purchase and which type of IceBreak server you will be using depending on your needs.

Shortly after you will receive an e-mail with the Licence key, which you have to enter here before you can use the IceBreak server.

The licence key will determine which features will be available in you copy of IceBreak and what kind of service you will expect.

10.19 Build history log

Each time a new version or IceBreak is created by IceBreak develop team - they will increment the build number.

This log show a historical list of changes and additions to the IceBreak system.

The build log is based on the data from the knowledge base which you can find on icebreak.org - and you can post request for enhancements and inform us about bugs.

10.20 Open projects log

Open projects log is a list of ongoing projects in the IceBreak community. It covers:

- Reported bugs which are not fixed yet
- Changes
- New features
- Enhancements

The "open project log" is based on the data from the knowledge base which you can find on icebreak.org - and you can post request for enhancements and inform us about bugs.

If you have requests or problems with IceBreak please consult this list before post a new request at icebreak.org.

Part XI

11 Appendix

11.1 Intergration to BlueSeries

Appendix:

BluesSeries is a Back office system for mailing, faxing, short messages service and other B2B communication. You can access components in BlueSeries by the BlueSeries Bind-directory and header file.

If You utilize the BlueSeries - you need to change the library list for the serverinstance to include the BlueSeries library.

BlueSeries has a object model called a "Business Objects" which can be converted from and to XML documents, spool files, TXT files, CSV files and PDF files. You can acces the Business Object with the following:

SetXvar

Syntax:

SetXvar (Variable : Value);

Field Name	Data type	Description
Variable	VARCHAR(64)	The Name and path to an node in the business Object
Value	VARCHAR(256)	The value you want to set

GetXvar

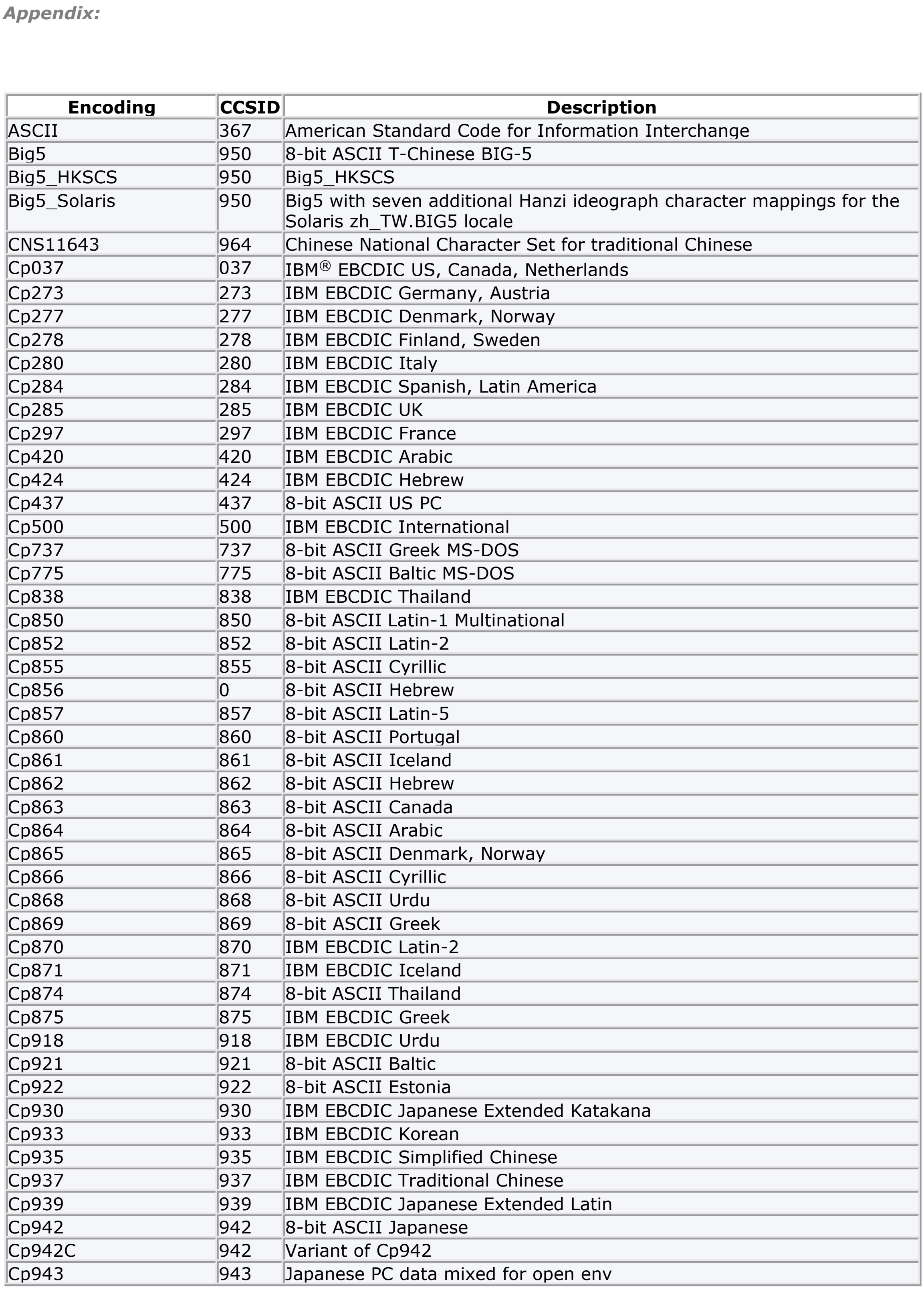
Syntax:

Value = GetXvar (Variable);

Field Name	Data type	Description
Variable	VARCHAR(64)	The Name and path to an node in the business Object
Value	VARCHAR(256)	The value you want to set

The Following is updating and retrieve a custom number in the object:

11.2 Internal relation between CCSID and encoding schemes



Appendix:

Encoding	CCSID	Description
ASCII	367	American Standard Code for Information Interchange
Big5	950	8-bit ASCII T-Chinese BIG-5
Big5_HKSCS	950	Big5_HKSCS
Big5_Solaris	950	Big5 with seven additional Hanzi ideograph character mappings for the Solaris zh_TW.BIG5 locale
CNS11643	964	Chinese National Character Set for traditional Chinese
Cp037	037	IBM® EBCDIC US, Canada, Netherlands
Cp273	273	IBM EBCDIC Germany, Austria
Cp277	277	IBM EBCDIC Denmark, Norway
Cp278	278	IBM EBCDIC Finland, Sweden
Cp280	280	IBM EBCDIC Italy
Cp284	284	IBM EBCDIC Spanish, Latin America
Cp285	285	IBM EBCDIC UK
Cp297	297	IBM EBCDIC France
Cp420	420	IBM EBCDIC Arabic
Cp424	424	IBM EBCDIC Hebrew
Cp437	437	8-bit ASCII US PC
Cp500	500	IBM EBCDIC International
Cp737	737	8-bit ASCII Greek MS-DOS
Cp775	775	8-bit ASCII Baltic MS-DOS
Cp838	838	IBM EBCDIC Thailand
Cp850	850	8-bit ASCII Latin-1 Multinational
Cp852	852	8-bit ASCII Latin-2
Cp855	855	8-bit ASCII Cyrillic
Cp856	0	8-bit ASCII Hebrew
Cp857	857	8-bit ASCII Latin-5
Cp860	860	8-bit ASCII Portugal
Cp861	861	8-bit ASCII Iceland
Cp862	862	8-bit ASCII Hebrew
Cp863	863	8-bit ASCII Canada
Cp864	864	8-bit ASCII Arabic
Cp865	865	8-bit ASCII Denmark, Norway
Cp866	866	8-bit ASCII Cyrillic
Cp868	868	8-bit ASCII Urdu
Cp869	869	8-bit ASCII Greek
Cp870	870	IBM EBCDIC Latin-2
Cp871	871	IBM EBCDIC Iceland
Cp874	874	8-bit ASCII Thailand
Cp875	875	IBM EBCDIC Greek
Cp918	918	IBM EBCDIC Urdu
Cp921	921	8-bit ASCII Baltic
Cp922	922	8-bit ASCII Estonia
Cp930	930	IBM EBCDIC Japanese Extended Katakana
Cp933	933	IBM EBCDIC Korean
Cp935	935	IBM EBCDIC Simplified Chinese
Cp937	937	IBM EBCDIC Traditional Chinese
Cp939	939	IBM EBCDIC Japanese Extended Latin
Cp942	942	8-bit ASCII Japanese
Cp942C	942	Variant of Cp942
Cp943	943	Japanese PC data mixed for open env

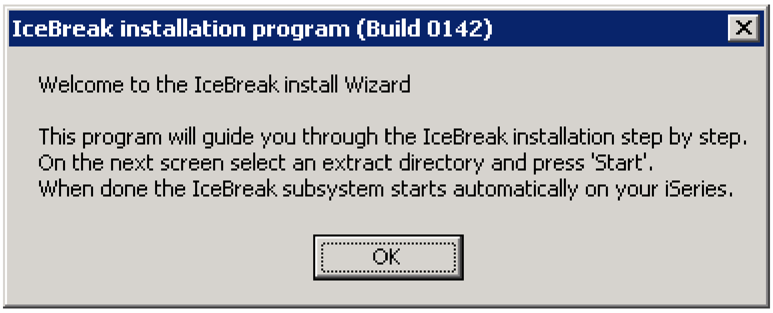
Cp943C	943	Japanese PC data mixed for open env
Cp948	948	8-bit ASCII IBM Traditional Chinese
Cp949	944	8-bit ASCII Korean KSC5601
Cp949C	949	Variant of Cp949
Cp950	950	8-bit ASCII T-Chinese BIG-5
Cp964	964	EUC Traditional Chinese
Cp970	970	EUC Korean
Cp1006	1006	ISO 8-bit Urdu
Cp1025	1025	IBM EBCDIC Cyrillic
Cp1026	1026	IBM EBCDIC Turkey
Cp1046	1046	8-bit ASCII Arabic
Cp1097	1097	IBM EBCDIC Farsi
Cp1098	1098	8-bit ASCII Farsi
Cp1112	1112	IBM EBCDIC Baltic
Cp1122	1122	IBM EBCDIC Estonia
Cp1123	1123	IBM EBCDIC Ukraine
Cp1124	0	ISO 8-bit Ukraine
Cp1140	1140	Variant of Cp037 with Euro character
Cp1141	1141	Variant of Cp273 with Euro character
Cp1142	1142	Variant of Cp277 with Euro character
Cp1143	1143	Variant of Cp278 with Euro character
Cp1144	1144	Variant of Cp280 with Euro character
Cp1145	1145	Variant of Cp284 with Euro character
Cp1146	1146	Variant of Cp285 with Euro character
Cp1147	1147	Variant of Cp297 with Euro character
Cp1148	1148	Variant of Cp500 with Euro character
Cp1149	1149	Variant of Cp871 with Euro character
Cp1250	1250	MS-Win Latin-2
Cp1251	1251	MS-Win Cyrillic
Cp1252	1252	MS-Win Latin-1
Cp1253	1253	MS-Win Greek
Cp1254	1254	MS-Win Turkish
Cp1255	1255	MS-Win Hebrew
Cp1256	1256	MS-Win Arabic
Cp1257	1257	MS-Win Baltic
Cp1258	1251	MS-Win Russian
Cp1381	1381	8-bit ASCII S-Chinese GB
Cp1383	1383	EUC Simplified Chinese
Cp33722	33722	EUC Japanese
EUC_CN	1383	EUC for Simplified Chinese
EUC_JP	5050	EUC for Japanese
EUC_JP_LINUX	0	JISX 0201, 0208 , EUC encoding Japanese
EUC_KR	970	EUC for Korean
EUC_TW	964	EUC for Traditional Chinese
GB2312	1381	8-bit ASCII S-Chinese GB
GB18030	1392	Simplified Chinese, PRC standard
GBK	1386	New simplified Chinese 8-bit ASCII 9
ISCII91	806	ISCII91 encoding of Indic scripts
ISO2022CN	965	ISO 2022 CN, Chinese (conversion to Unicode only)
ISO2022_CN_CNS	965	CNS11643 in ISO 2022 CN form, Traditional Chinese (conversion from Unicode only)
ISO2022_CN_GB	1383	GB2312 in ISO 2022 CN form, Simplified Chinese (conversion from Unicode only)
ISO2022CN_CNS	965	7-bit ASCII for Traditional Chinese
ISO2022CN_GB	1383	7-bit ASCII for Simplified Chinese
ISO2022JP	5054	7-bit ASCII for Japanese
ISO2022KR	25546	7-bit ASCII for Korean

ISO8859_1	819	ISO 8859-1 Latin Alphabet No. 1
ISO8859_2	912	ISO 8859-2 ISO Latin-2
ISO8859_3	0	ISO 8859-3 ISO Latin-3
ISO8859_4	914	ISO 8859-4 ISO Latin-4
ISO8859_5	915	ISO 8859-5 ISO Latin-5
ISO8859_6	1089	ISO 8859-6 ISO Latin-6 (Arabic)
ISO8859_7	813	ISO 8859-7 ISO Latin-7 (Greek/Latin)
ISO8859_8	916	ISO 8859-8 ISO Latin-8 (Hebrew)
ISO8859_9	920	ISO 8859-9 ISO Latin-9 (ECMA-128, Turkey)
ISO8859_13	0	Latin Alphabet No. 7
ISO8859_15	923	ISO8859_15
ISO8859_15_FDIS	923	ISO 8859-15, Latin alphabet No. 9
ISO-8859-15	923	ISO 8859-15, Latin Alphabet No. 9
JIS0201	897	Japanese industry standard X0201
JIS0208	5052	Japanese industry standard X0208
JIS0212	0	Japanese industry standard X0212
JISAutoDetect	0	Detects and converts from Shift-JIS, EUC-JP, ISO 2022 JP (conversion to Unicode only)
Johab	0	Korean composition Hangul encoding (full)
K018_R	878	Cyrillic
KSC5601	949	8-bit ASCII Korean
MacArabic	1256	Macintosh Arabic
MacCentralEurope	1282	Macintosh Latin-2
MacCroatian	1284	Macintosh Croatian
MacCyrillic	1283	Macintosh Cyrillic
MacDingbat	0	Macintosh Dingbat
MacGreek	1280	Macintosh Greek
MacHebrew	1255	Macintosh Hebrew
MacIceland	1286	Macintosh Iceland
MacRoman	0	Macintosh Roman
MacRomania	1285	Macintosh Romania
MacSymbol	0	Macintosh Symbol
MacThai	0	Macintosh Thai
MacTurkish	1281	Macintosh Turkish
MacUkraine	1283	Macintosh Ukraine
MS874	874	MS-Win Thailand
MS932	943	Windows® Japanese
MS936	936	Windows Simplified Chinese
MS949	949	Windows Korean
MS950	950	Windows Traditional Chinese
MS950_HKSCS	NA	Windows Traditional Chinese with Hong Kong S.A.R. of China extensions
SJIS	932	8-bit ASCII Japanese
TIS620	874	Thai industry standard 620
US-ASCII	367	American Standard Code for Information Interchange
UTF8	1208	UTF-8 (IBM CCSID 1208, which is not yet available on the System i5 platform)
UTF-16	1200	Sixteen-bit UCS Transformation Format, byte order identified by an optional byte-order mark
UTF-16BE	1200	Sixteen-bit Unicode Transformation Format, big-endian byte order
UTF-16LE	1200	Sixteen-bit Unicode Transformation Format, little-endian byte order
UTF-8	1208	Eight-bit UCS Transformation Format
Unicode	13488	UNICODE, UCS-2
UnicodeBig	13488	Same as Unicode

11.3 Manual Installation

If you have problems with the standard installation program for IceBreak you can use the following method to install it manually:

Start up the normal installation process as always. Press OK

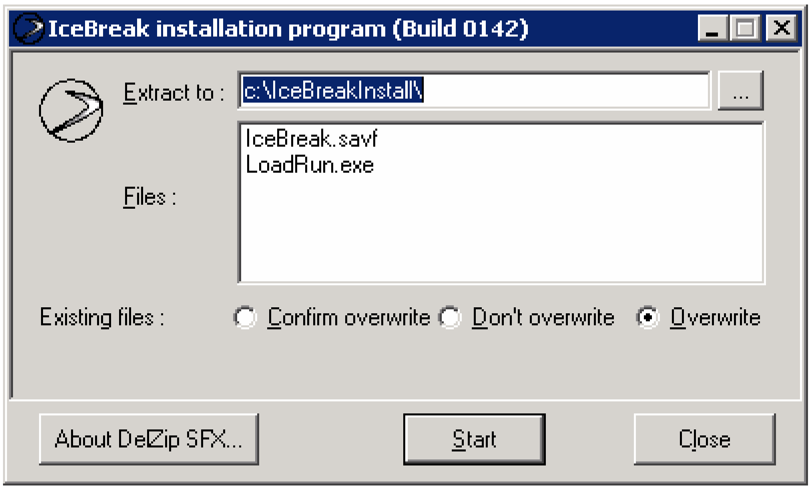


IceBreak installation program (Build 0142)

Welcome to the IceBreak install Wizard

This program will guide you through the IceBreak installation step by step.
On the next screen select an extract directory and press 'Start'.
When done the IceBreak subsystem starts automatically on your iSeries.

Press the Start and files needed will be unzipped and places in the directory entered into ”Extract to”.



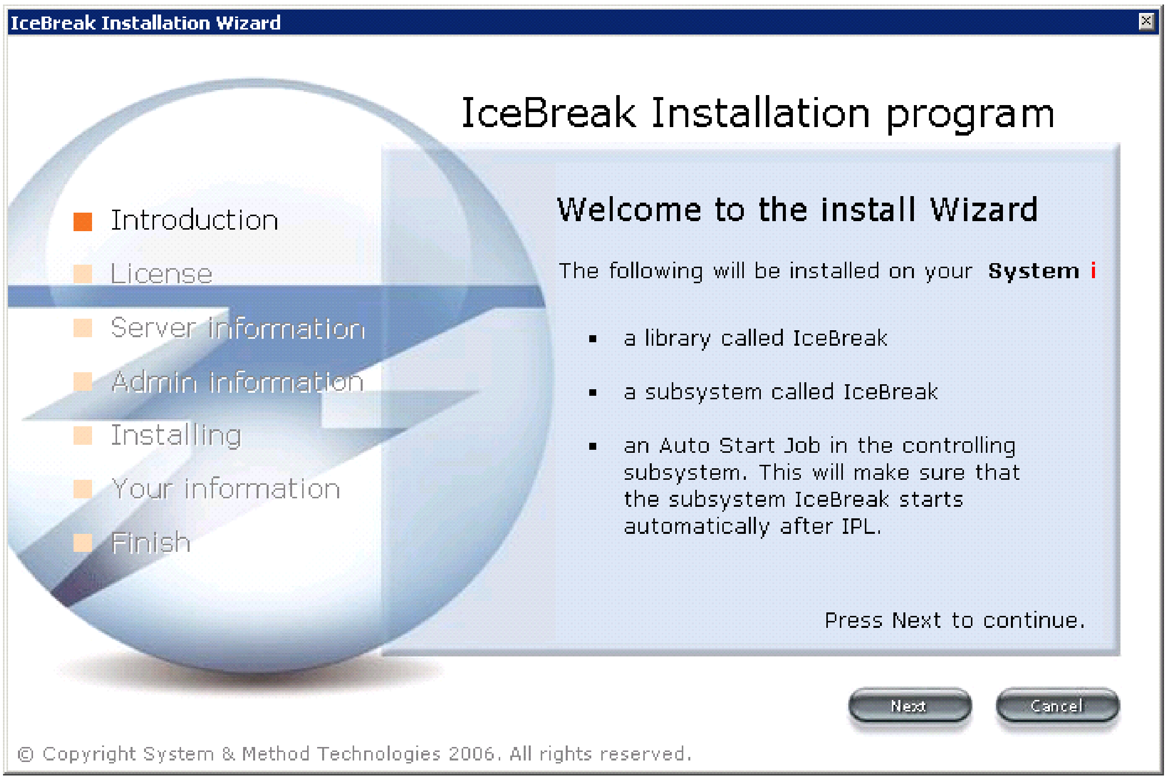
IceBreak installation program (Build 0142)

Extract to : c:\IceBreakInstall\

Files :
IceBreak.savf
LoadRun.exe

Existing files : ○ Confirm overwrite ○ Don't overwrite ● Overwrite

After the extract has finished the following panel will show up. Press the Cancel and Yes to leave the installation program.



IceBreak Installation program

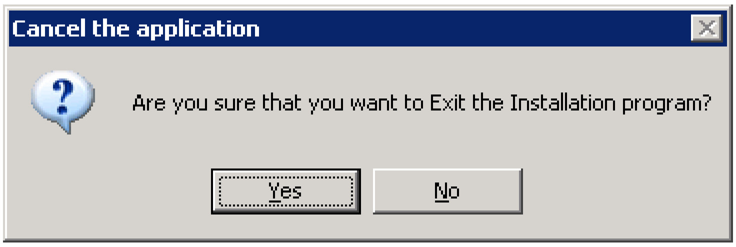
Welcome to the install Wizard

The following will be installed on your **System i**

- a library called IceBreak
- a subsystem called IceBreak
- an Auto Start Job in the controlling subsystem. This will make sure that the subsystem IceBreak starts automatically after IPL.

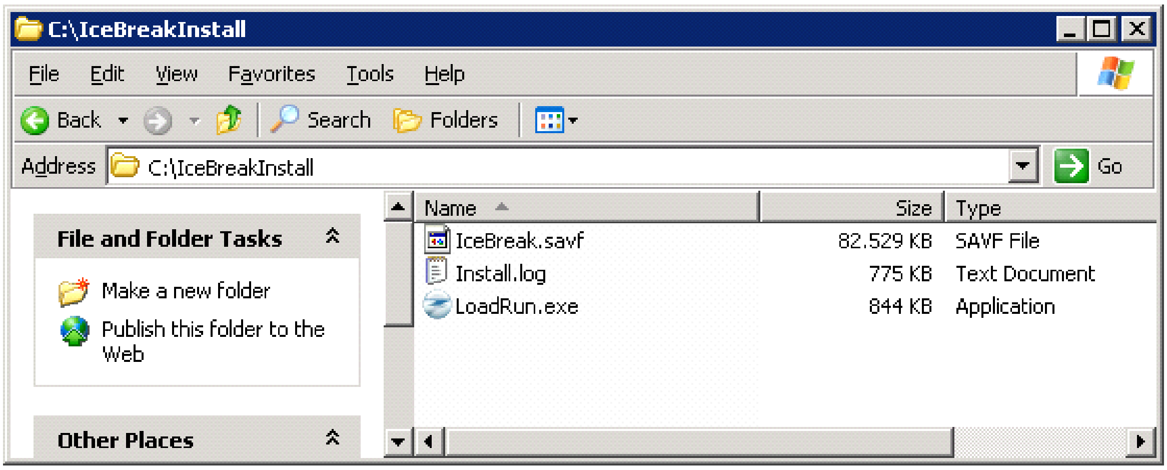
Press Next to continue.

© Copyright System & Method Technologies 2006. All rights reserved.



Are you sure that you want to Exit the Installation program?

Locate the Directory where you just extracted the installation files to:

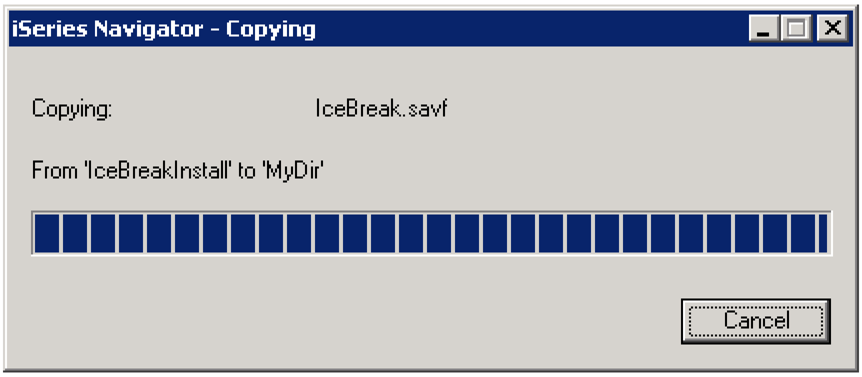


Address: C:\IceBreakInstall

Name	Size	Type
IceBreak.savf	82.529 KB	SAVF File
Install.log	775 KB	Text Document
LoadRun.exe	844 KB	Application

Copy the “IceBreak.savf” file to a directory in the IFS on the target System. Use Windows explorer or the

IBMi Navigator.



iSeries Navigator - Copying

Copying: IceBreak.savf

From 'IceBreakInstall' to 'MyDir'

Sign on to a 5250 session and run the following commands as *SECOFR

- CPYFRMSTMF FROMSTMF('/Mydir/IceBreak.savf') TOMBR('/qsys.lib/qtemp.lib/IceBreak.file') MBROPT(*REPLACE)
- RSTLIB SAVLIB(BLUEICE) DEV(*SAVF) SAVF(QTEMP/ICEBREAK) RSTLIB(INSTBOX)
- CHGCMD CMD(INSTBOX/INSTALL) PRDLIB(INSTBOX)

If needed you can change the installation parameters, you can do so by change the data area call INSTBOX/INSTALLDFT. The layout of that data area looks like:

Pos. 1 length 10 - The IceBreak installation library
Pos. 11 length 10 – The database library
Pos. 21 length 128 – The IceBreak root directory
Pos. 149 length 128 – The ADMIN root directory
Pos. 278 length 10 – ADMIN server name
Pos. 288 length 5 – ADMIN port number
Pos. 293 length 1 – '1' indicate that the IceBreak menu should be duplicated into library QGPL

The installation program is now ready to be executed with the following command:

- INSTBOX/INSTALL

11.4 Technical decumentations

11.4.1 What is IceBreak

IceBreak is a combined HTTP and application server for the Integrated Language Environment (ILE) on the System i platform.

The purpose of IceBreak is not to have a cross platform solution but rather an optimized application server that runs even on the smallest System i.

IceBreak is written in ILE (C/C++ and CLP) which eliminates any conversion when making procedure calls from the user application into the server core. In addition, because of IceBreak's "platform dependent” concept, it can utilize MI interface functions, user indexes and user spaces, to gain the most power out of the System i platform.

IceBreak is a server platform for bringing the System i native languages RPG, COBOL, CLP, C and C++ to the web.

IceBreak is modeled as a **web servlet container for the ILE environment** that brings you similar functionality to your application as a serverlet container for JAVA like Tomcat or WebSphere.

11.4.2 Application Server Programs as a concept

IceBreak is based on the ASPX syntax developed by Microsoft for the .NET framework used in IIS (Internet Information Server) environment. ASPX is also inspired by other environments like JSP (Java Server Pages).

The ASPX syntax has two components which consist of the presentation layer and the program logic. You switch between presentation and program logic by escaping back and forth with the <% and %> character sequence. See later in ”ASPX Source Parsing”.

The feature of ASPX is the fact that you can isolate the presentation layer form the business logic – providing a Model View Control (MVC) structure on your design.

On the other hand, the ASPX approach can also provide the combination ILE logic and the presentation layer which in turn can be a dangerous cocktail. Blending RPG logic with the HTML presentation layer is far from modern design concepts like Model-View-Control (MVC). However IceBreak has enhanced the ASPX syntax to “include” or refer to the presentations logic from the program logic to an external described presentation layer in a separate (X)HTML / XML / JSON file. This is very similar to the way that the ILE developer has used externally described display files for years.

ASPX is for the same reason ideal for “quick-and-dirty” solutions and basic educational teaching:

```
<%@ language=”RPGLE” %>
<html>
  <p>Hello world</p>
</html>
<% return; %>
```

Ninety percent of Microsoft powered internet solutions are based on ASPX. As a result, there is a large base of ASPX knowledge. Although IceBreak is a dedicated server platform for the System i ILE developers, it has attracted developers from the Microsoft .NET community by redefining RPG as “just another scripting language” for ASPX.

11.4.3 The Object model

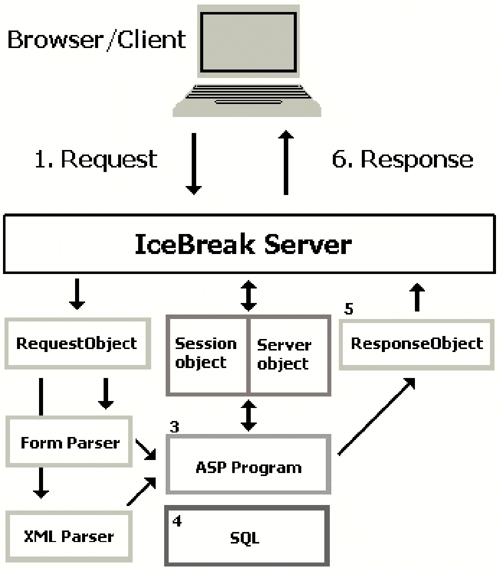
The host languages in IceBreak are all ILE, namely RPG, COBOL, C and CLP. These languages do not have an object model as we know it from the Object Oriented programming (OOP) world. IceBreak provides an interface so you are able to access the IceBreak object model directly from RPG, COBOL, C and CLP. This is done by processing the .ASPX application source with a precompiler that creates object wrapper functions to the object model. This can only be done by the fact that each object in the model only occurs once so an explicit object reference in not necessary.

The object model is hidden from the application; however IceBreak allows the user via an API interface to communicate directly with all objects instantiated by the IceBreak server from within RPG, COBOL, C or CLP.

Take a look at the communication flow between the IceBreak server and client web browsers:

- The client places a request to the IceBreak server which creates a request object (1)
- The request object is processed. Dependent on the content type either the form-parser for HTML or the XML-parser is executed for XML input.
- ASPX Program initiates (3). You can retrieve data from the request object (2) and render the response object (5)
- You can use imbedded SQL to make database I/O, call legacy code, use IBM API's or call IceBreak API's etc.
- The Response object is prepared to the clients codepage and encoding scheme
- IceBreak sends the response back to the client (6)

The Objects used in the IceBreak server



When the client (the browser) requests a (X)HTML document from a web server, it will send a request to the server. The server then receives the response and renders the result in the browser canvas. Let us follow each of these objects involved in the process from the request to the response.

11.4.3.1 The Request Object:

The Request object contains all data sent from the client. In the HTTP protocol that is the **header** and **content**. The header describes the content which can be either a "Form Object" or an "XML-Object".

- `MyVar=QryStr('URLParameter')`
- `MyVarNum=QryStrNum('NumericURLParameter')`
- `GetHeader(KeyWord)`
- `GetHeaderList(Name:Value:'*FIRST' | '*NEXT')`
- `Form2Ilob(IlobPtr : ‘FormField’);`

The Form parser and xml parser are working on the request object:

11.4.3.2 The Form Parser:

The Form Parser initiates automatically when it is a HTML Form-object. The following functions are available in your ASPX-program:

- `Myvar=Form('FormVar');`
- `MyNumvar=FormNum('FormNumVar');`

11.4.3.3 The Session Object

Fact: The Internet is "stateless" and that might be the biggest challenge when migrating existing 5250 applications to browser-based applications. Programs must be designed to process input, produce output, and then terminate. You cannot have open files or static memory between panels.

The IceBreak server maintains a session with the client by creating a "session cookie." Even if the client browser is blocking "cookies" a session cookie is normally allowed. Sometimes you have to change the "allow session cookies" flag in the browser for the applications to work correctly.

If session cookies are not an option, IceBreak also supports “Session stability by URL redirection.” In this case, the browser is redirected to a specific URL containing a pseudo sub-path which is the session id. Internally this session id can only be requested from the IP address that created the session. The session sub-path is purely virtual and is removed from any resource reference by the IceBreak server.

Sessions are maintained as long as you want, and you can configure the duration of a session using the WRKICESVR or WRKXSVR command or from IceBreak's browser ADMIN page. A session has a very small "footprint" and uses little system resources.

- `Session object functions:`
- `MyVar = SessGetVar('MyVariable');`
- `MyVarNum = SessGetVarNum('MyNumericVariable');`
- `SessSetVar(''MyVariable');`
- `SessSetVarNum(''MyNumericVariable');`
- `IlobPtr = SessgetILOB(IlobName);`

11.4.3.4 The XML parser:

The XML parser is invoked automatically when the content in the request is a XML-document. That is when the content-type is set to "text/xml". Now you are ready to access any element or attribute in the request object by the X-Path syntax.
Example:

- `Myvar=XmlGetValue('/element1/element2/element[elementnumber]@anattribute' : 'Defaultvalue');`

The Build-in XML parser can also be used directly on stream files, strings and ILOB's. This is discussed later.

11.4.3.5 The Server Object:

The server object contains static server information such as the configuration parameters and dynamic information from the client.

Server object functions:

- `MyVar = GetServerVar('AserverVariable');`
- `PointerToMyVar = GetServerVarPtr('AserverVariable');`

See the appendix I for complete list of header values available.

All i5/OS system values are also available by prefixing the system value with "**SERVER_SYSVAL_**". like:

```
DayOfWeek = GetServerVar('SERVER_SYSVAL_QDAYOFWEEK');
```

The complete list of system values can be found in the i5/OS command WRKSYSVAL. Here are samples values

Server variable	Sysval	Description	Sample Value
SERVER_SYSVAL_QDAYOFWEEK	QDAYOFWEEK	The day of week	*THU
SERVER_SYSVAL_QTIME	QTIME	The current time	153856763

11.4.3.6 The Application Object:

One static application object is maintained for each server instance. This object is accessible from all applications running within this server instance. The Application object is thread safe.

This is implemented as a UserIndex placed in the application library call "ICEAPPLVAR"

Since it is located in the application library, values are available when deployed.

Application object functions:

- `MyVar = ApplGetVar('MyVariable');`
- `MyVarNum = ApplGetVarNum('MyNumericVariable');`
- `ApplSetVar('MyVariable':’Text’);`
- `ApplSetVarNum('MyNumericVariable':1234);`
- `ApplClrVar('MyVariable');`

11.4.3.7 The Response Object:

The Response object contains all data sent from the server to the client. In HTTP protocol this is the header and content. The header describes the content which can be of any type HTML, XML, GIF, TIF, PDF, etc. This is the place where you render your dynamic output data with the ASPX syntax.

When the ASPX program quits, IceBreak sends the response to the client.

Response object functions:

- `ResponseWrite(Value)`
- `ResponseWriteNL(Value)`
- `ResponseNLWrite(Value)`
- `ResponseEncTrim(Value)`
- `ResponseWriteILOB(IlobPtr)`
- `ResponseWriteLanguageMsg(msgid)`
- `ResponseWriteBLOB(LobLocator )`
- `ResponseWriteCLOB(LobLocator )`
- `SetContentType(MimeType)`
- `SetCharset(CharSet)`
- `AddHeader('Name' : 'Value')`
- `SetHeader('Name' : 'Value')`
- `Redirect('ToUrl')`
- `SetStatus(Code)`
- `SetCacheTimeout(Minutes)`
- `SetEncodingType(*HTML (default) | *XML | *NONE )`
- `SetMarker (MarkerName :  MarkerValue);`
- `ParseMarker ();`
- `ResponseRelease();`

11.4.3.8 Other function

Some of the IceBreak API functions are not related to any object in the object model. They are, however; useful when making web-applications:

Miscellaneous atomic function

- `MyNum= Num(’12-34/456/789’);`
- `SetDecimalPoint  (  DecimalPoint   );`
- `Fld = UrlDecode  (    UrlField     );`
- `Fld  = UrlEncode (    UrlField     );`
- `Ok = PutStreamString  (  Filename: Filemode : Value : Xlate );`
- `Ok = GetStreamString (Filename:Value : Offset  : Length  : Xlate  );`
- `Ok = Include (‘FileName ‘);`
- `Ok = IncludeUrlEncode(‘FileName’);`
- `Trace (‘TraceData’);`
- `Mimevar= GetMimeType  (‘FileName or extention’ );`
- `SQL_Execute_HTML(Sqlstmt : MaxRows );`
- `SQL_Execute_XML(Sqlstmt : MaxRows );`
- `SQL_Execute_JS(Sqlstmt : MaxRows );`
- `Ok = HttpRequest( ReqUrl:ReqTimeOut:ReqMethod:ReqData:ReqContentType:ReqHeader:eqXlate: RespXl`

11.4.3.9 Internal Large Objects - ILOB's

Host languages like RPG and COBOL have a limit in program variable size. IceBreak breaks that limit by utilizing userspaces and associates them to a session. The transfer of data back and forth between Request object or Response object can be done with help from ILOBS. Also SQL CLOBS and BLOBS can be mapped to ILOB's.

Consider a file upload application. When the user hits the "upload" button on a form this data can be placed in an ILOB up to 2Gbytes. This ILOB can now save itself as a stream file. The XML parser can parse it or it can be placed into DB/2 CLOB field.

The access to IOB functions are available by including:

```
/include QASPhdr,ilob
```

ILOB Functions:

- `IlobPtr = ILOB_OpenPersistant(Library : Name);`
- `Ok = ILOB_DeletePersistant(IlobPtr);`
- `Ok = ILOB_Read(IlobPtr: String: Offset : Length);`
- `Ok = ILOB_ReadNext(IlobPtr: String :Length);`
- `Ok = ILOB_Write(IlobPtr :  String :Offset);`
- `Ok = ILOB_LoadFromBinaryStream (IlobPtr: FileName);`
- `Ok = ILOB_LoadFromTextStream  (IlobPtr: FileName);`
- `Ok = ILOB_SaveToBinaryStream (IlobPtr: FileName);`
- `Ok = ILOB_SaveToTextStream(IlobPtr: FileName);`
- `Ok = ILOB_Append(IlobPtr : String);`
- `Ok = ILOB_Clear (IlobPtr);`
- `DataPtr= ILOB_GetDataPtr(IlobPtr);`
- `Len = ILOB_GetLength(IlobPtr);`
- `ILOB_SetWriteBinary(IlobPtr; *ON|*OFF);`
- `ILOB_Close(IlobPtr);`
- `ILOB_Ilob2Clob( ClobLocator :  IlobPtr : Offset :  Length );`
- `ILOB_Clob2Ilob( IlobPtr : ClobLocator  : Offset :  Length );`

ILOB's are also used for response and request objects in some application server session dispatcher modes (see later)

11.4.4 Creating Application Server Programs (ASPX):

When an IceBreak server instance is configured, it is assigned a "root path" in the IFS and a "applications" library. Compiler and recompiles are based on Just-In-Time (JIT) compilation as follows:

1. The "hello.ASPX" applications is placed in the root path.
2. The browser is invoked using an URL that hits the server instance and the hello.ASPX application. For example, http://MySeries:7001/hello.ASPX
3. If the first file extension is ".ASPX" or ".ASMX", IceBreak will determine if "HELLO" as an program object is in the application library.
   a. If not, the JIT compiler compiles the hello.ASPX application and runs the program object from the application library
   b. If the hello.ASPX application exists, the system checks to see if the source has been modified since this object creation timestamp
      i. If Yes, the JIT compiler re-compiles the hello.ASPX and runs the program object from the application library.
      ii. If No, the hello.ASPX program object is just run from the application library.

If the compiler fails to produce an executable, IceBreak formats the compiler post list to HTML and returns the page to the browser. Since the compile failed, IceBreak does not run the application.

If the IceBreak server instance is a "production" instance, it will call the application immediately and ignore the JIT compiler.

11.4.4.1 Compile and run

The precompiler looks up the first file extension in build-in MIME table (Physical file MIM00).

There are three types of extensions that will trigger a program: .RPGLE .ASPX and .ASMX.

The .RPGLE and .ASPX will be compiled into a "plain" *PGM object.

The .ASMX is used as WebServices and result in a *SRVPGM (service program). See WebServices later.

11.4.4.2 ASPX program files - Normal web application programs

When the precompiler is invoked it goes through the .ASPX source stream file and separates the code and the response object data from each other. The code fragments are left (almost) unchanged but the response data is placed in method call on the response object namely Response.Write("..");

Since RPG and COBOL do not understand the Response object, it is accessed by the ResponseWrite("…"); API wrapper function in IceBreak. The new source is placed in a source physical file called "QASPSRC" in the application library with a member name the same as the source stream file (first 10 characters).

A program module is then created with the corresponding native CRTRPGMOD, CRTSQLRPG etc. commands depending on the "language=" compiler directive. (see later) The final program is created with the CRTPGM command where the service program SVC200, IceBreak server core, and call back procedures are bound to the final program resulting in the response data being placed as literal constants in the *PGM object. For this reason, the only object required at runtime is the *PGM object and the IceBreak server.

11.4.4.3 ASMX program files - WebServices

When the precompiler is invoked it goes through the .ASMX source stream file and separates the code and the response object data from each other. For each procedure a prototype is created and the special parameter keywords "Input", "Output" or "InOut" are the removed from the procedure interface. The "Input", "Output" and "InOut" keywords are IceBreak extensions to the parameters in the RPG procedure interface declarations.

A Special procedure wrapper is generated that:

1) Calls the X-path functions and pulls all input parameters from the SOAP body.
2) Calls the exported WebServices procedure.
3) Constructs the response XML SOAP document with response.write wrapper calls.

This wrapper procedure has the same name as the target procedure suffixed by a "_". The IceBreak server is calling this wrapper procedure when hit by a WebService request.

The new source is placed in a source physical file called "QASPSRC" in the application library with a member name the same as the source stream file (first 10 characters).

The procedure headers are placed in a file called "QSOAPHDR" in the application library with a member name the same as the source stream file (first 10 characters). So the WebService can be used as a "normal" service program as well.

A WSDL file is produced containing the complete WebService definition for all exported procedures in the RPG program. The name is the same as the .ASMX file but with the .WSDL extension. This file is served when the WebService is called with ?WSDL from and URL like:

```
http://MySeries:7001/MyWebService.ASMX?WSDL
```

A program module is then created with the corresponding native commands such as CRTRPGMOD, CRTSQLRPG etc. depending on the "language=" compiler directive. (See later)

The final service program is created with the CRTSRVPGM where the service program SVC200, IceBreak server core, and call back procedures are bound to the user written WebService service program. This means that all response data, SOAP header, SOAP body, is actually placed as literal constants in the *SRVPGM object and for that reason the only object required at runtime is the *SRVPGM object and the IceBreak server.

IceBreak is dynamic loading, linking and procedure resolving the service program/ procedure containing the webservice to the IceBreak server core at runtime. This is similar to DLL under Windows.

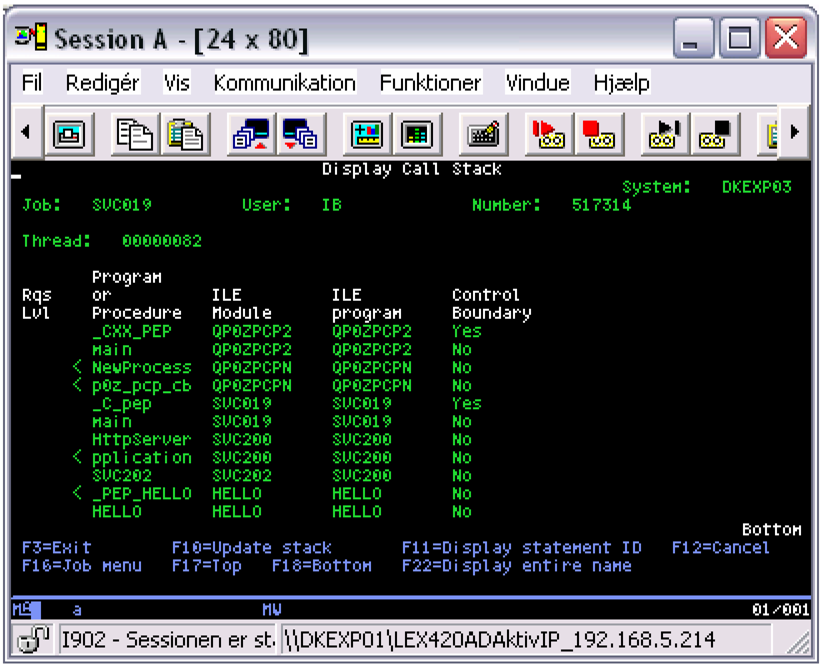
WebServices created in this manner can also be accessed by Microsoft DCOM by creating an IceBreak server instance for the same application library with a server type *SVC and installing the IceBreak DCOM proxy on any accessible windows server in the network. I.e. Any System i service program built that way can act as a Microsoft DCOM object.

11.4.4.4 Program activation group

When an ASPX program runs, by default it inherits the IceBreak server default activation group. So if the user application terminates uncontrolled, IceBreak is able to recover from that error and send the joblog back to the user.

When such error occurs the response status is automatically set to 501 – “Internal server error.” Otherwise the response status is set to 200, which is “OK”. The Status can however be overridden by the SetStatus(); response object function.

The server instance program SVC019 is bound to the application server service program SVC200. Also the user application is also bound to SVC200 which makes the call stack very thin.



```
                              Display Call Stack
                                                            System:   DKEXP03
 Job:   SVC019          User:   IB             Number:   517314

 Thread:    00000082

        Program
 Rqs    or            ILE          ILE          Control
 Lvl    Procedure     Module       program      Boundary
        _CXX_PEP      QP0ZPCP2     QP0ZPCP2     Yes
        main          QP0ZPCP2     QP0ZPCP2     No
      < NewProcess    QP0ZPCPN     QP0ZPCPN     No
      < p0z_pcp_cb    QP0ZPCPN     QP0ZPCPN     No
        _C_pep        SVC019       SVC019       Yes
        main          SVC019       SVC019       No
        HttpServer    SVC200       SVC200       No
      < pplication    SVC200       SVC200       No
        SVC202        SVC202       SVC200       No
      < _PEP_HELLO    HELLO        HELLO        No
        HELLO         HELLO        HELLO        No
                                                                        Bottom
 F3=Exit         F10=Update stack       F11=Display statement ID   F12=Cancel
 F16=Job menu    F17=Top    F18=Bottom  F22=Display entire name

MS     a               MW                                              01/001
```

I902 - Sessionen er st. \\DKEXP01\LEX420ADAktivIP_192.168.5.214

.

11.4.5 Dispatcher methods

IceBreak has four different ways to run an application. Each server instance can be configured to run one of these:

Dispatcher Method	Scalability	Design considerations	Application isolation
Application pooling	High	Yes – Clearing of fields / resetting files required	Low
Session persistent process	Low	None – File pointers / SQL commit boundaries are preserved	High
Multi process	Medium	Yes - Session may continue in a new process	High
Single session	High/Low	For Debug and data-queue like WebServices	Low

The one you pick depends on the requirements for your application.

11.4.5.1 Session persistent process

Each client request results in one process that remains active until the session times out or is disconnected. The process remains active even if the socket connection is physically terminated.

File pointers and variables are preserved when using "return." The explicit need of interaction with the session object by SessSetVar and sessGetvar is not necessary.

Session persistent process

11.4.5.2 Application pooling

The pool is loaded when the server instance is started, that is the number of pre-established process is started. When the client connects, simple stream files are served by the connection thread. When it is an application, the first available application server instance in the pool receives the request. If none is available a new is started.

Programs written for connection pooling have to re-initialize variables and reset file pointers between each request since the client may change for the next request.

Explicit need of interaction with the session object by SessSetVar and sessGetvar is required to keep tract of the session.

Application pooling

11.4.5.3 Multi Process

When the client connects, all requests are served by the application process. This process is alive as long as the socket is connected. The session object, however is maintained until the session times out or the session is disconnected

Programs written for "multi process" pooling have to re-initialize variables and reset file pointers between each request since the socket may be closed by network equipment or the browser does not support "keep connection."

Explicit need of interaction with the session object by SessSetVar and sessGetvar is required to keep tract of session.

Multi process

11.4.5.4 Single Session

When the client connects, the connection is established by the dispatcher directly. The dispatcher does not dispatch anything but calls the application directly.

This is useful for debugging purposes since the job number does not change for the server. It is also suitable for WebServices that use the “dataqueue” style. The client requests are queued in the socket layer and processed one at a time.

File pointers, variables and SQL commit boundaries are preserved but explicit need of interaction with the session object by SessSetVar and sessGetvar is required to keep tract of session. For the same reason you might need to reset file pointer, variables, etc between request depending on the application type.

Single session

11.4.6 Programming

11.4.6.1 ASPX File Identification

1. An .ASPX file, like a .JSP file, is a blend of response document in XHTML, XML or HTML and program code.
2. You toggle between ASPX code and render the response document with the escape sequence <% and %>.
3. The ASPX code is inside the escape sequence tags <% and %> and the rendered mode is outside the escape sequence tags.
4. The programming language (script language) can not be determined from the file extension .ASPX. An .ASPX files is interpreted as an XHTML file until the <% %> escape sequence tags are encountered.
5. If the first character of the escape sequence is the @ sign, it is interpreted as a compiler/runtime directive:

```
<%@ language=”RPGLE” %>
```

6. Each compilation step has its own parameter string:
   a. SQLOPT – if imbedded SQL is used
   b. MODOPT – for the CRTxxMOD command
   c. PGMOPT – for CRTPGM or SRTSRVPGM command

7. If the “language” directive is omitted, the default language is “RPG”
8. Following values are valid for parameter language:
   a. RPGLE
   b. SQLRPGLE
   c. CBLLE
   d. SQLCBLLE
   e. CLLE
   f. CLE
   g. SQLC

11.4.6.2 ASPX File parsing

A simple ASPX program:

```
<%@ language=”RPGLE” %>
<html>
  <p>Hello world</p>
</html>
<% return; %>
```

1. The parsing starts in render mode but the first character encountered is the <% escape sequence for ASPX code. The first non-blank character is the @ sign which identifies it as a compiler directive. This means that any keyword and value are valid.
2. Next the compiler directive is terminated by the %> escape sequence which puts the code back into render mode. The rest is plain XHTML syntax.
3. On the last line, the <% escape sequence toggles back to ASPX code mode until a %> escape sequence is encountered which ends the ASPX file.

11.4.6.3 How to place dynamic data at runtime

There are two types of variables in IceBreak and the syntax is different.

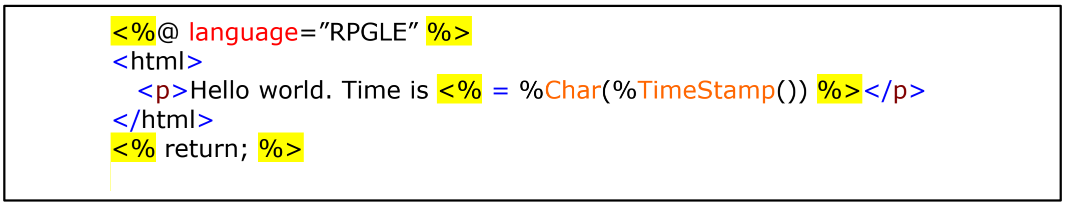
1. **Direct program variables** are identified by the "=" sign as the first non-blank character in the ASPX code stream:

```
<% = MyProgramVar %>
```

The value must be in character format so packed, zoned, time and timestamp etc. data types must be cast into character format using the %char() RPG build-in-function.

This syntax also applies for return values from function calls.

Example:



```
<%@ language="RPGLE" %>
<html>
  <p>Hello world. Time is <% = %Char(%TimeStamp()) %></p>
</html>
<% return; %>
```

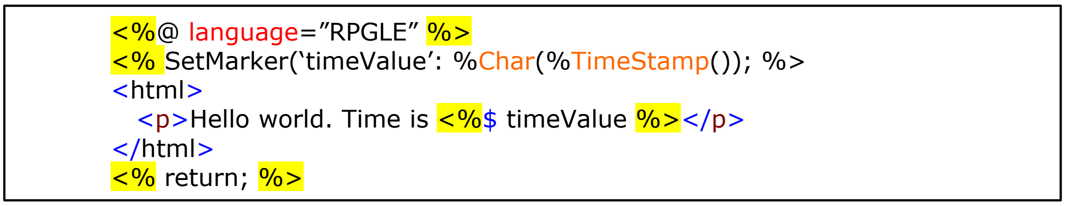
2. **Marker variables** are identified by the "$" sign as the first non-blank character in the ASPX code stream:

```
<% $ MyMarkerVar %>
```

The value must be in character format so packed, zoned, time and timestamp etc. data types must be cast into character format using the %char() RPG build-in-function.

Markers are dynamic variables defined and used at runtime.

Example:



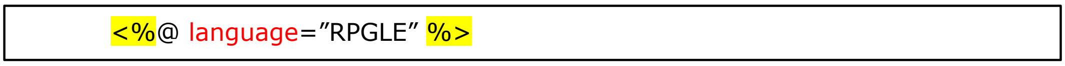
```
<%@ language="RPGLE" %>
<% SetMarker('timeValue': %Char(%TimeStamp()); %>
<html>
  <p>Hello world. Time is <%$ timeValue %></p>
</html>
<% return; %>
```

11.4.6.4 Remarks

Remarks can be stated in 3 ways:

- If the parser is in ASPX CODE mode and finds a /*, any following code and XHTML is rendered in remark color and not syntax checked until a */ is found again
- If the parser is in ASPX CODE mode and finds //, the rest of the line including XHTML is rendered in remark color and not syntax checked.
- If the first char in the line is a * or blank + *, the rest of the line including XHTML is rendered in remark color and not syntax checked.

Example:



```
<%@ language="RPGLE" %>
```

```
<html>
  <p>Hello world. Time is <% /* = %Char(%TimeStamp()) */ %></p>
</html>
<% // Now terminate the program
  Return; %>
```

11.4.6.5 IceBreak language enhancements

There have been a few powerful enhancements made to the ILE languages. These include the following:

RPG:

- Code is by default in free format if selected on the server instance
- Free format code may start in position 1
- Fixed format spec. can start in any position. IceBreak will align the code to column 6.
- Block comments start with /* and ending with */
- Keywords “Input”, “output” and “inout” are allowed for procedure interfaces and used with WebSevices.
- “Var” macro for building D-spec in free format
- SQL can be written from free format even on I5OS/v5.1 like:

```
//' -----------------------------------------------------------------------
//' Fetch - is retrieving the next row from the result set
//' -----------------------------------------------------------------------
Begsr Sql_Fetch;

  Exec SQL
     Fetch list into :recds;

   Exsr Sql_Monitor;
Endsr;
```

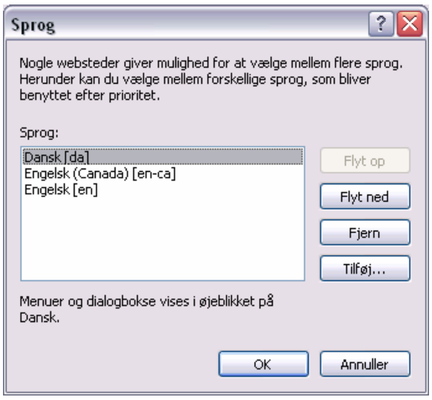
11.4.6.6 I18N - internationalization

IceBreak supports internationalization (‘I’ + 18 letters + ‘N’). The precompiler searches for the ^ character in XML/HTML/XHTML and .ASPX source. When found, the rest of the string until a “ or an ‘ is found is stored in the I18N_DFT message file in the application library.

A string sequence for xhtml/xml/html can be made by encapsulate the text into brackets [and].

IceBreak stores the unique message id for that string instead of the text string itself. By creating new message files suffixed by the selected language from the browser, any .ASPX page will select the according language. However, when the message file or message id is not found, the text from the default message file I18N_DFT is displayed

i.e if the browser is set to use the following languages:



Nogle websteder giver mulighed for at vælge mellem flere sprog. Herunder kan du vælge mellem forskellige sprog, som bliver benyttet efter prioritet.

Sprog:

Dansk [da]
Engelsk (Canada) [en-ca]
Engelsk [en]

Menuer og dialogbokse vises i øjeblikket på Dansk.

That is Danish, then Canadian English and final generic English. The IceBreak will search the messages in the following order:

```
I18N_DA Danish
I18N_EN_CA     Canadian English
I18N_EN generic English
I18N_DFT       Default language
```

Example:

```
<%@ language="RPGLE" %>
<%
//' Translation can occure at paragraphs or any other html/xml tag
//' the translation continues until next tag occures ... like:
%>
<p>^Text</p>
<%
    /*' Also Transtaltion can occure at attribute values between ".." and '..'*/
%>
<input type="Button" value="^Ok">
<input type="Button" value='^Submit'>
<%
    /*' Finally you can let the translation span over many tags starting with ^[ and end with ]
      ' however, expantion of ASPX-variavbes are not valid in translation */
%>

<p>^[This text contains <b>bold</b> tag, but i'll like to have it all in the
translation messages file]</p>

<% return %>
```

11.4.6.7 AJAX

All server instances have a “/system” directory. This directory includes “ajax.js” which is a simple yet power full AJAX implementation which allows the user to create AJAX applications with a minimum of JavaScript knowledge.

This example shows an html file that request “ajaxserver.ASPX” when you click the button:

- Line 2 : the AJAX script is loaded;
- Line 7 : ASPX program “ajaxServer.ASPX” is called. We want the response in “MyResult”
- Line 8 : “MyResult” is the id of an “Div” tag. The inner HTML is replaced asynchronous by the response of “ajaxServer.ASPX”

```
<html>
<link rel="stylesheet" type="text/css" href="/System/Styles/IceBreak.css"/>
<script language="JavaScript1.2" src="/System/Scripts/ajax.js"></script>
<body>
  <input type=button
        value='Click here to run the Ajax request'
        onclick="adAjaxCall('myresult','ajaxServer.ASPX?manuid=SONY');"/>
  <div id=myresult></div>
</body>
</html>
```

This is the “ajaxserver.ASPX” that returns a table content depending on the URL parameter. The Ajax will replace the “div” tag with id “myresult” with the result of the ASPX response object which will be a table component.

```
<%
F*Filename+IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++
+++++++++Comments++++++++
Fproduct1  if   e           k disk
D pmanuid         s                   like(manuid)

/free
*inlr = *on;
pmanuid = qrystr('manuid');
%>
<table>

<%
  chain pmanuid productr;
  dow not %eof(product1) and manuid = pmanuid;
%><tr>
    <td><% = prodid %></td>
    <td><% = desc  %></td>
  </td>
<%
    read productr;
  enddo;
%>
</table>
```

11.4.7 IceBreak Macros

IceBreak Macros can be simple macros that just expand at compile time. But macros can also be REXX-scripts which are run at compile time.

COBOL:
Since COBOL has a poor implementation of user function, macros are used. Macros are one line of code terminated by and “;” . Macros are stored in /system/Macros/macros.xml. End users can write generic macros by adding the macro to this XML file.

Example:

```
      Main section.

      Do.

           MyCustNo = Request.Form("CustNo");
           Move MyCustNo to CustNo of CustRec.

      Enddo.
           exit.
```

This will expand to:

Example:

```
      Main section.

      Do.

           call procedure "Request_Form" using
             reference  MyCustNo
             content    “CustNo”
           end-call

           Move MyField to CustNo of CustRec.

      Enddo.
           exit.
```

11.4.7.1 Using the simple macros

The IceBreak precompiler predefines the following REXX variables:

- `$RETURNPARM`
  - The name of the left side parameter used in function like macros: x=MyMacro(). here $RETURNPARM will be "X"
- `$PARM01` to `$PARM99`
  - The parameter from the macros e.g.: `MyMacro(Customer,Item)` `$PARM01` is "Customer" and `$PARM02` is "Item"
- `$STMTNO`
  - The ASPX-Source statement where the macro was found
- `$FILENAME`
  - The ASPX-Source filename

11.4.7.2 Using script macros

Script Macros are expanded into the ASPX-source by using "expand" IceBreak-REXX-build-in-function”.

The IceBreak precompiler predefines the following REXX variables:

- `RETURNPARM`
  - The name of the left side parameter used in function like macros: `x=MyMacro()`. here `RETURNPARM` will be "X"
- `PARM01` to `PARM99`
  - The parameter from the macros e.g.: `MyMacro(Customer,Item)` `PARM01` is "Customer" and

`PARM02` is "Item"

- `STMTNO`
  - The ASPX-Source statement where the macro was found
- `FILENAME`
  - The ASPX-Source filename

When using if/while comparison always embed the script in CDATA tags to avoid XML misinterpretation.

Syntax
The generic syntax of the macro is:

```
returnparm = macroname(parm01, parm02 , parm03 ... parm99);
```

where ";" is the macro identifier

Macros are NOT case sensitive.
All programming languages have a separate pool of macros. However SQL versions of the language shares the same pool as their host language.
A "Macroidentifier" can be specific for each language.

The "Macroidentifier" must be one char that is unique and not used in the language context.

11.4.7.3 Macro list

Default Macro definitions.

```
<MacroDefinitions>
 <language type="RPGLE" MacroIdentifier="." ParameterMarker="$">

  <macro name="var" type="buildin"/>

  <macro name="SmallScriptMacro" type="script"><![CDATA[
      expand( RETURNPARM '=' PARM01 '+' PARM02);
  ]]></macro>

 </language>

 <language type="CBLLE" MacroIdentifier=";" ParameterMarker="$">

  <macro name="Request.Form" type="simple">
   call procedure "Request_Form" using
      reference  $RETURNPARM
      content    $PARM01
   end-call
  </macro>

  <macro name="Request.FormNum" type="simple">
   call procedure "Request_FormNum" using
      reference  IcebreakFloat
      content    $PARM01
   end-call
   move IcebreakFloat to $RETURNPARM
  </macro>

  <macro name="Request.QueryString" type="simple">
   call procedure "Request_QueryString" using
      reference  $RETURNPARM
      content    $PARM01
  end-call
  </macro>

  <macro name="Request.QueryStringNum" type="simple">
  call procedure "Request_QueryStringNum" using
      reference  IcebreakFloat
      content    $PARM01
  end-call
  move IcebreakFloat to $RETURNPARM
  </macro>

  <macro name="Response.Include" type="simple">
  call procedure "Response_Include" using
      reference  $RETURNPARM
      content    $PARM01
  end-call
  </macro>

  <macro name="Response.SetNoHeader" type="simple">
  call procedure "SetNoHeader"
  end-call
  </macro>

  <macro name="Response.Write" type="simple">
  call procedure "Response_Write" using
     content    $PARM01
  end-call
  </macro>

  <macro name="Response.SetCharset" type="simple">
  move $PARM01 to IceBreakVarCharString
  move 16      to IceBreakVarCharLen
  call procedure "SetCharset" using
     IceBreakVarChar
  end-call
  </macro>

 </language>

 <language type="DEMO" MacroIdentifier=";" ParameterMarker="$">
 <!-- implementing the formnum as a script - this has however some performance overhead
-->
  <macro name="formnum" type="script">
   expand('     call procedure "Request_FormNum" using');
   expand('         reference  IcebreakFloat');
   expand('         content    ' PARM01);
   expand('     end-call');
   expand('     move IcebreakFloat to ' RETURNPARM);
  </macro>
 </language>
</MacroDefinitions>
```

11.4.8 Security Considerations in IceBreak

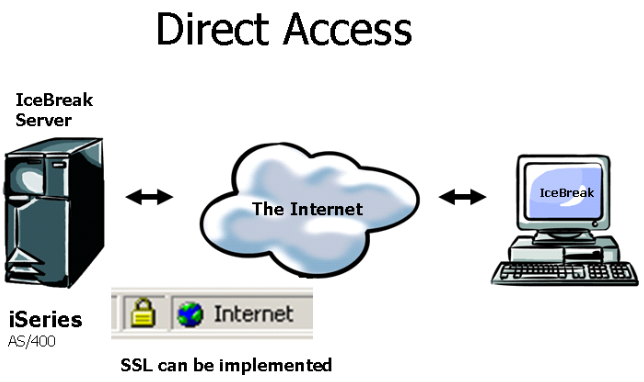
There are basically 3 ways you can present IceBreak applications to the WEB.

What solution to select, is a normally a matter of company policy in relation to security issues.

1) The application has direct access to the WEB
2) The application will communicate through an Apache Server (Internal or External)
3) The application will communicate through a Secure Web server

Confidentiality is a major goal in security. Using encryption functions, such as Secure Sockets Layer (SSL), you can ensure that network traffic cannot be read by an unauthorized user while in transit. See 7.4

11.4.8.1 Direct Access



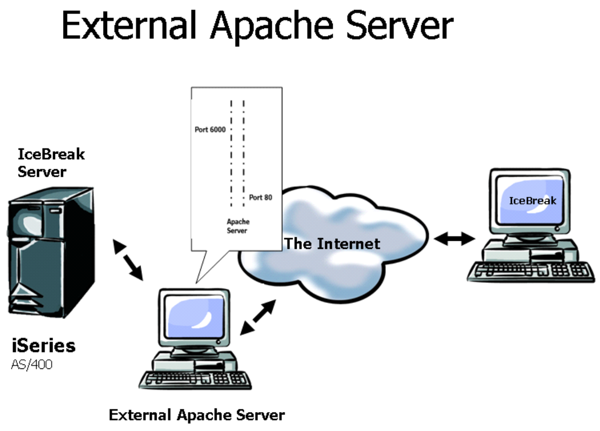
Considerations
All networked systems should be secured
to control remote access

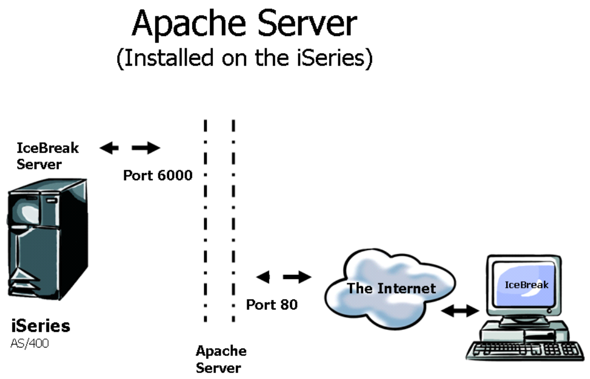
- Access to IFS
- Telnet
- FTP
- ODBC/JDBC
- Remote commands

- Standard I5/OS (OS/400) security features can be used
- IceBreak provides user log on features
- All jobs in the session then run under the signed on user's profile allowing standard i5/OS (OS/400) security features to apply
- Dynamic menus with options dependent upon the logged on user can be devised
- ASPX programs run on the server, not in the browser
- Access to data is controlled by ASPX program
- End users do not have direct access to data on the server
- Only the HTML created by the ASPX program is sent to the browser, users cannot see the program code, file names, or other elements of the business
- Compiled objects on the IBMi cannot be manipulated.

11.4.8.2 Apache Server

The application can communicate through an Apache Server (Internal or External)





Secure your IceBreak application by use of the well proven Apache Technology.

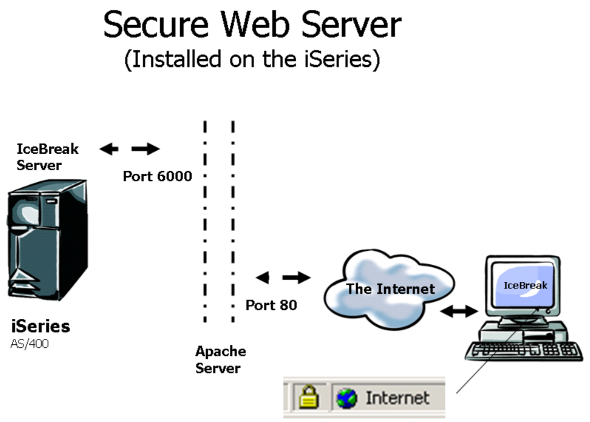
IBM® embraced the widely popular open-source Apache server several years ago as the Hypertext Transfer Protocol (HTTP) server of choice for its Web products.

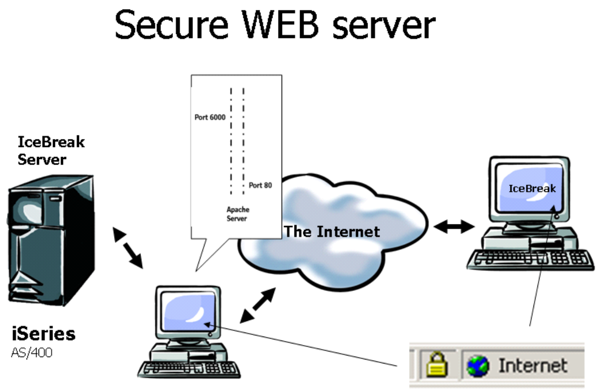
The foundation of any On Demand Business application is the Web server, and IBM has made a significant investment in Apache to be that foundation. The broad investment in Apache across IBM's product offerings allows Web developers to leverage existing Apache skills and software to build applications for commercial use. This does also apply for IceBreak

Considerations
See the IBM Redbook:

http://www.redbooks.ibm.com/redbooks/pdfs/sg246716.pdf

11.4.8.3 Secure Web Server





Communicate through a Secure Web server.
Set up a server on the Web that supports one or more of the major security protocols such as SSL, HTTPS and PCT.

This means that e.g. order form data from your browser is encrypted before being sent (uploaded) to the Web site, making it extremely difficult for a third party to decipher credit card numbers and other sensitive data that it might fraudulently capture.

Considerations:
See the IBM Redbook:

http://www.redbooks.ibm.com/redbooks/pdfs/sg246716.pdf

11.4.8.4 Secure Sockets Layer (SSL)

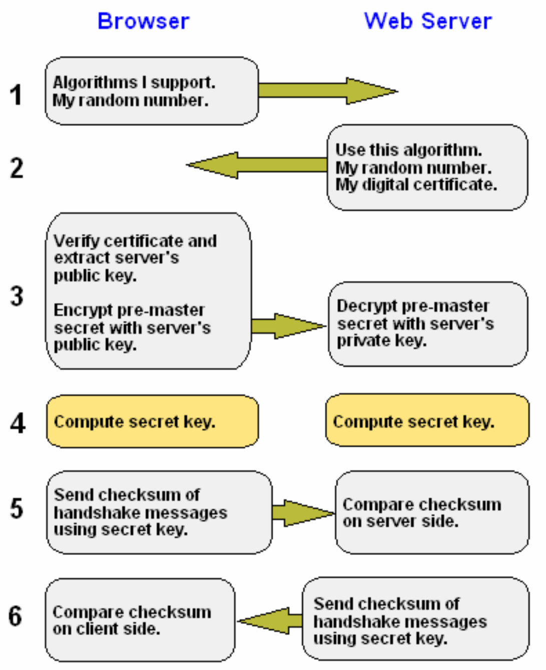
Confidentiality is a major goal in security. Using encryption functions, such as Secure Sockets Layer (SSL), you can ensure that network traffic cannot be read by an unauthorized user while in transit.

SSL is widely used to do two things: to validate the identity of a Web site and to create an encrypted connection for sending credit card and other personal data. Look for a lock icon at the bottom of your browser when you order merchandise on the Web. If the lock is closed, you are on a secure SSL.

HTTPS and Port Number 443
An SSL session is started by sending a request to the Web server with an HTTPS prefix in the URL, which causes port number 443 to be placed into the packets. Port 443 is the number assigned to the SSL application on the server.

The Handshake
After the two sides acknowledge each other, the browser sends the server a list of algorithms it supports, and the server responds with its choice and a signed digital certificate. From an internal list of certificate authorities (CAs) and their public keys, the browser uses the appropriate public key to validate the signed certificate. Both sides also send each other random numbers. Contact you local dealer for more details on certificates.

Data for Secret Keys Is Passed
The browser extracts the public key of the Web site from the server's certificate and uses it to encrypt a pre-master key and send it to the server. At each end, the client and server independently use the pre-master key and random numbers passed earlier to generate the secret keys used to encrypt and decrypt the rest of the session.


Browser
Web Server
1
Algorithms I support. My random number.
2
Use this algorithm. My random number. My digital certificate.
3
Verify certificate and extract server's public key.
Encrypt pre-master secret with server's public key.
Decrypt pre-master secret with server's private key.
4
Compute secret key.
Compute secret key.
5
Send checksum of handshake messages using secret key.
Compare checksum on server side.
6
Compare checksum on client side.
Send checksum of handshake messages using secret key.

From Computer Desktop Encyclopedia
© 2005 The Computer Language Co. Inc.

The SSL Handshake

These steps take place to negotiate an SSL session before any user data is transmitted. Steps 5 and 6 verify the integrity of the handshake, ensuring that nobody tampered with any messages. These checksums are called "message authentication codes" (MAC).

Appendix I – Server variables available.

Server variable	Description	Sample Value
SERVER_SOFTWARE	The name and version of the running server	IceBreak/V1R11BLD0081
SERVER_ID	Name of current server instance	ICEUDV
SERVER_DESCRIPTION	Description of the server instance	Test for pre installation
SERVER_LOCAL_PORT	The TCP/IP port number (from 1 to 65535) where the server is polling for requests	60001
SERVER_INTERFACE	The TCP/IP interface where the server is listening for requests. (Interface created by OS/400 command ADDTCPIFC)	*ANY
SERVER_JOB_NAME	The underlying OS/400 job name	FAXUDV
SERVER_JOB_USER	The underlying OS/400 job user name (not the active user)	IB
SERVER_JOB_NUMBER	The underlying OS/400 job internal job number	509556
SERVER_JOB_MODE	The state of the server: "*PROD" or "*DEVELOP"	*DEVELOP
SERVER_LOGON_REQUIRED	An authentication prompt is automatically displayed when the user request the page and are "unknown" (Not available yet)	*NO
SERVER_STARTUP_TYPE	Whether the server should start when the subsystem is activated or must be started by STRICESVR or STRXSVR	*AUTO
SERVER_DEFAULT_USERPROFILE	The user profile used when starting a server instance process	IB
SERVER_ROOT_PATH	The IFS path used for web document. This can not be relative but is fixed to the IFS-root	/www/Icebreak
SERVER_DFT_DOC	The web document displayed when no specific document is requested. This is relative to the SERVER_ROOT_PATH	Index.ASPX
SERVER_CACHE_TIMEOUT	Number of seconds before a document expires. 0=immediately. Servers in *DEVELOP mode always overrides this value to 0 so the cache always is refreshed	240
SERVER_INPUT_BUFFER_SIZE	Number of bytes in the input buffer (the Request Object). When zero a default of 1Mbytes is used	0
SERVER_OUTPUT_BUFFER_SIZE	Number of bytes in the output buffer (the Response Object). When zero a default of 1Mbytes is used	0
SERVER_COOKIE_BUFFER_SIZE	Number of bytes in the output buffer (the Response Object). When zero a default of 64Kbytes is used	0
SERVER_INITIAL_PGM_NAME	Name of program to set extra libray list etc. It is called when the server instance is initiate - Not each time a new client connects or *NONE	
SERVER_INITIAL_PGM_LIB	Name of library where the initial program exists	
SERVER_JOBQ_NAME	The jobqueue from where the server	

Server variable	Description	Sample Value
	process is started	
SERVER_JOBQ_LIB	Name of library where the jobqueue exists"	
SERVER_APPLICATION_LIB	This is where all your ASPX-programs are placed when they are compiled. This is where the QASPSRC file is created with your precompiled ASPX-program sources	ICEDEV
SERVER_TGTRLS	OS/400 version	V5R1M0
SERVER_TRACE_FILE	The name of the trace file created when TRACE=*ON When blank, the file name defaults to Trace.txt in the SERVER_ROOT_PATH	trace.txt
SERVER_SESSION_TIMEOUT	Number of seconds before the session automatically is terminated Default is 1440 seconds	1440
SERVER_SESSION_ID	The Unique Session Timestamp-id; also it is the time when the session was started	2006-07-27-15.37.29.516500
SERVER_SESSION_NUMBER	The Unique Session number; Also it is the job number of the first lightweight job that initiated the session	516500
SERVER_SYSTEM_NAME	The system name from the network attribute	DKEXP03
REMOTE_ADDR	The remote TCP/IP address of the client web browser	192.168.5.3
REMOTE_PORT	The remote TCP/IP port number negotiated by the TCP/IP layer	1892
REQUEST_HOST_NAME	The TCP/IP address or name for the requested server	dkexp03
REQUEST_METHOD	The method the document was requested: GET parameters is parsed along the URL POST Parameters are parsed in the form object	GET
REQUEST_HEADER	The complete header string	Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, */* Referer: http://dkexp03:60001/tutorials/Tutorials.ASPX?topic=ex01server.ASPX&desc=GetServerVar() Accept-Language: da,en;q=0.5 Accept-Encoding: gzip, deflate User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322; .NET CLR 2.0.50727) Host: dkexp03:60001 Connection: Keep-Alive Cookie:

Server variable	Description	Sample Value
		sys_sesid="2006-07-27-15.37.29.516500"; sys_sesid="2006-07-27-15.37.29.516500"; iNavigate__2=1
REQUEST_RAW	The complete request, excluding the content	GET /tutorials/ex01server.ASPXHTTP/1.1 Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, */* Referer: http://dkexp03:60001/tutorials/Tutorials.ASPX?topic=ex01server.ASPX&desc=GetServerVar() Accept-Language: da,en;q=0.5 Accept-Encoding: gzip, deflate User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322; .NET CLR 2.0.50727) Host: dkexp03:60001 Connection: Keep-Alive Cookie: sys_sesid="2006-07-27-15.37.29.516500"; sys_sesid="2006-07-27-15.37.29.516500"; iNavigate__2=1
REQUEST_CONTENT	The content string	
QUERY_STRING	Parameters sent along the GET or POST request after the document. The URL in the browser	

Part XII

12 Acknowledgements

IceBreak core

IceBreak core also ships with the following third party library. The IceBreak team acknowledges the Licence terms and conditions

ZLIB

Jean-loup Gailly jloup@gzip.org
Mark Adler madler@alumni.caltech.edu

Read more on http://zlib.net/

Index

ICE BREAK