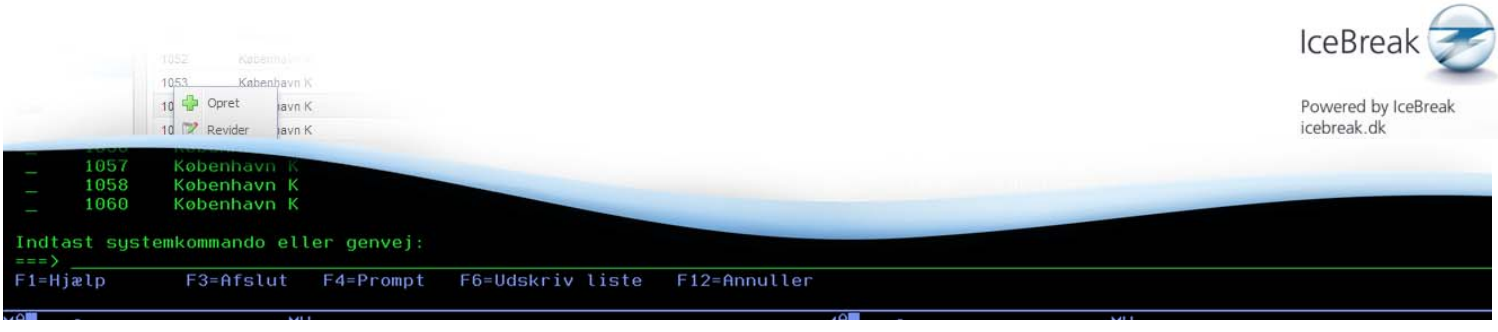
IceCap™

Developed for IBM i™. Powered by IceBreak™.

Getting Started

Version 1.51



First edition for IceCap™ version 1.51

Version 1.51 printed March 2017

System & Method
Håndværkersvinget 8
DK-2970 Hørsholm

www.system-method.com
www.IceBreak.org

IceCap™

Welcome to a new world of using IBM i™. IceCap will provide a smooth integration with modern application infrastructures on the private net or the internet.

What is IceCap™?

IceCap is a number of tools that is required to power applications and present them in a standard web browser – all powered from an IBM i™ platform. So basically IceCap is:

1) New menu structure.
2) User credential system.
3) 5250 web emulator
4) “Scripting” engine
5) Session management system.

So let me first go through all the steps from installing icecap – to what welcomes the end user and then drill down into details later in this document.

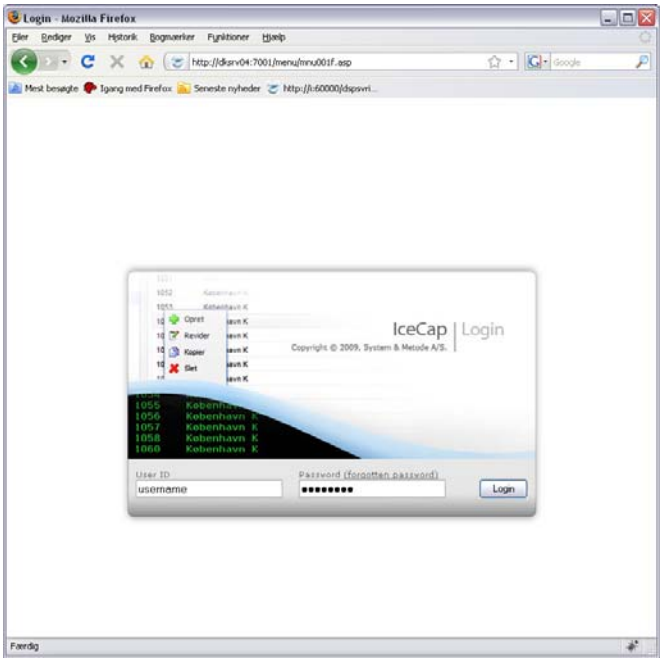
Installing IceCap™

IceCap it self is an application that runs under the IceBreak server. So if you don’t have IceBreak installed I suggest that you start with that right now. When you have installed IceBreak, just open the **application store** in the administration menu HTTP://MyIbmi:7000 and locate and download IceCap. I suggest you download the image to your desktop and install it on your IBM i from there. After installation you can simply delete it from your desktop.

Running IceCap for the first time.

When Icecap is installed you can start it by opening your browser and enter the name of your IBM i and the port number for IceCap in the URL line like:

HTTP://MyIbmi:7050



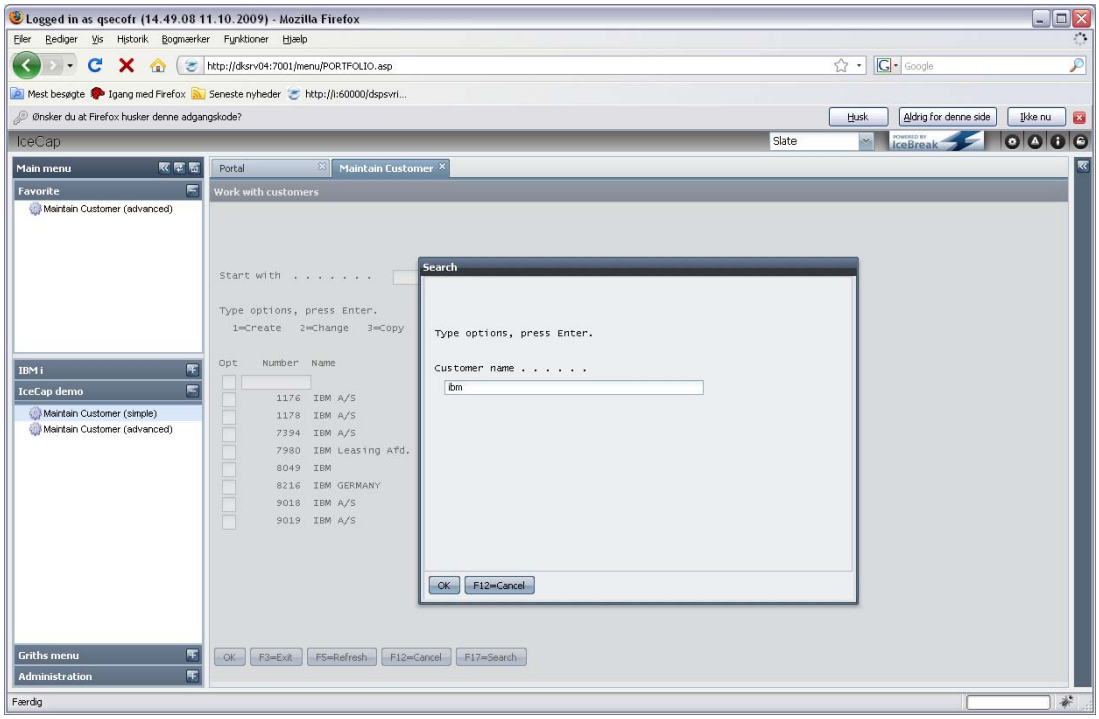
Now the login prompt occurs: One thing to notice is that is can be both real IBM i™ user profile credentials OR mapped profiles – that is profiles that points to a real IBM i™ user profile. We will talk more on mapped user profile in the user administration. But this first time just use QSECCOFR or any other profile with *SECADM.

One important thing to notice: This should be the only time you see a login screen in this session. From now IceCap has logged you into the IBM i™ and IceCap will spawn new applications in subsequent windows with that profile. Just like windows or Mac will have done it. Each window can be a 5250 window or it can be a regular Web 2.0 application, simple html based applications, links to the internet – google. Windows remote desktops etc. That only depends on your ambitions and wishes.

One key feature is that each window or tab can communicate with each other in the common IceCap session. We will drill down in inter process communication later.

After the logon in prompt:

The main window will appear. To the left you have the navigation pane, in the middle the actual application. To the right, we have the context based help system.



Let's investigate the components.

The Navigator:

The Navigator is divided into sections which we call **systems**. You can expand and collapse each system by the button next to the system name. Each system has its own menu with subsequent menu options. And also each system can be presented by the menu structure within IceCap – or it can be powered by its own custom tailored menu loader. More on that in "Writing my own menu driver"

For each system you have a search feature and a "**drag to favourites**" feature.

The search feature is enabled by clicking the spy-glass. Now enter your search keyword. All menu options in that particular system that satisfies the search criteria will be listed below.

Now you can just click on an item to start it up – or you can drag it into your favourites.

The items from any of the menus in any of the systems can be arranged and organizer after the end user needs in the favourites. Also this is independent of the menu items

origin from which system. And finally if you don't need the favourite longer, simply right click and delete it

Click or double click?
If you click on a menu item it will be opened in the active tab in the application pane (the middle) but if you double click it will add a new tab with that application.

Double click is an easy an intuitive way to start new 5250 sessions and have old stuff and new stuff running side by side in a productive way.

The navigator is self can be hidden by the **<<** in the top to provide more space for applications

The application pane

All applications started will run in the application pane. Each application has its own tab, and you can close a application when you need it longer by the **[x]** in the upper right corner of the tab. Or you can right click on the tab to close all others. If you have opened a lot of tabs you can right and left scroll through them by the small arrows left and right off the tabs.

The context help pane

If the application in the middle pane has an associated help file. The content will automatically be shown here. If the user has super user credentials this contents can be edited by pressing the **[#]** in the upper right corner.

The context help is self can be hidden by the **>>** in the top to provide more space for applications

Configuring IceCap:

You will find the IceCap administration system in lower left side of the Navigator. Open the "Administration" and you will be able to configure systems, menus and users.

Working with systems:

IceCap ships with a "demo" system and the IBM i operating system by default. The easiest way to get started is by looking at the properties of the demo system and then builds your own based on these properties.

Working with menu items:

You will need to be able to start your application from a command line or call a program to start your application since IceCap is launching applications like entering commands from a command prompt.

Let’s say you have a customer inquiry program you want to add to the menu. In my case I can launch it like:

```
CALL CUSTOMERA
```

Now adding the this function to IceCap emulator which is called IceCap.aspx. So the in the url field you will enter:

```
IceCap.aspx?func=CALL CUSTOMERA
```

If you had extra parameter you can add them as you like. Also nosiest that you can use IBMI commands like:

```
IceCap.aspx?func=WRKSPLF
```

Finally just ad a description you would like to associate with the menu item and you are done.

You don’t have to manually add your complete application to IceCap, rather if you already have your own menu system you can just make your own menu loader.

Coding with IceCap:

You can reuse your skills - IceCap let you use the RPG language as the “scripting” language for navigation and extension of your original application.

Writing your own menu loader:

Your new menu loader will be an AJAX RESTfull webservice component that returns your menu in JSON format. It sounds pretty fancy but is in fact super simple in fact.

What the IceCap need to show a menu outline is a list of URL’s to load in the applications pane.

A simple menu program written in IceBreak RPG looks l something like this:

```
<%
/free
%>
[{
     id:"I1.1",
     text:"Customers ",
     leaf:true,
     icon:"/imgs/app.gif ",
     path:"/demo"
     link:" IceCap.aspx?func=CALL CUSTOMERA"
     helplink:"/customer.html"
  },{
     id:"I1.2",
     text:"Work with spoolfiles",
     leaf:true,
     icon:"/imgs/app.gif ",
     link:" IceCap.aspx?func=WRKSPLF "
  }
]
<%
*INLR = *ON;
```

Now let me walk you though the code:

The first line contains **<%** tells the IceBreak pre-compiler to go into code mode.
Now the /free is a native RPG statement that let us enter code the free-format-syntax. You can omit the if you prefer fixed format.

The **%>** Escapes the IceBreak pre-compiler back to *place-data-in-the-resulting-document mode* – or just doc mode.

The next twelve lines is the JSON document that is retuned to the IceCap menu. JSON is short for JavaScipt Object Notation and all communication in IceCap is done with JSON.

In short:

```
[   Starts an array
 {   Strarts an object

… Now comes a list of Properties and their values

 },{ ends an object and starts a new one

  … The next list of Properties and their values

 }  Ends the last object
]  Ends the array
```

Each menu object contains:

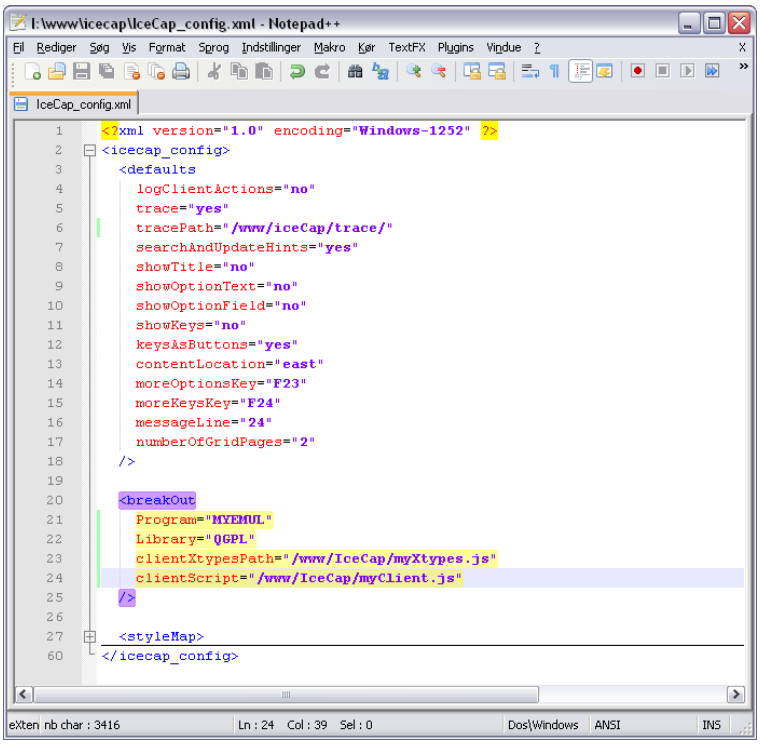
id:	A unique token for locating the menu item.
text:	The description displayed for the menu item.
leaf:	true if it is a menu item. false if it is a sub-menu
icon:	path to an image file used as the menu icon
link:	the URL to be loaded into the application pane.
helplink:	the URL to be loaded into the help pane.
path:	If a hive or subfolder is used you can specify the path.

A real menu loader would typically load the items from a DB/2 table and check access right against the user credentials etc. An example is found in appendix A.

Writing your own break outs:

Break-outs are a concept where IceCap leaves the original program logic / application flow and uses either client side or server side components. A client side break out might be using Google map to show the customers location. A server side break out might be to show a calendar picker for a date field.

You need to register your break out's in the IceCap configuration XML file which is found in the IceCap root directory on the IFS. Typically that would bee //myIBMI/qroot/www/IceCap/icecap_config.xml



Now locate the breakout section, and enter the program name and library of your choice for you server side break outs. In my example ill call it MYEMUL in library QGPL. Client

Server side Breakouts

IceCap contains a long list of API's you can use to automate and enhance you application to give it a modern look and feel.

Your server side break out program will extend the IceCap emulator and will be called each time an IBMI screen is ready to be displayed. You will be able to peek into the 5250 display buffer, manipulate input fields and press any function key.

The target language is plain RPG, and you simply create and compile an ILE program object from within your favourite development environment.

In my little sample MYEMUL I have changed all input fields on any screen that appears to be libraries to be an x-type DDLIB. X-types is data types with some code that changes behaviour in the given context. The DDLIB xtype is a Drop-Down of libraries on the IBMI.

I'll go into details about coding x-types I moment. But first let's have a look at the nature of the breakout program and the API's.

```
h actgrp(*caller)
h dftactgrp(*no)
h bnddir('ICECAP')
 /Include qAspHdr,ICECAP

d pVts            s               *
d vts             ds                  likeds(vtsDS)
d loc             ds                  likeds(vtLocationDS)
d vtShow          s               N

c     *entry        plist
c                   parm                    vts
c                   parm                    vtShow
 * ---------------------------------------------------------------------- */
 /free

  // Continue after and show the emulator screen
  // (Set to *OFF if you want to reload next panel)
  vtShow = *ON;
  pVts = %addr(vts);

  // Find all libraries and replace the field type to a DDLIB
  // a dropdown of libraries
  loc = vtLocate(pVts: VT_INPUT: 1: 1: VT_EAST: 'library');
  dow (loc > *LOVAL );
    vtSetMetaData (pVts:loc:'{xtype:"DDLIB"}');
    loc = vtLocate(pVts: VT_INPUT: loc.Lin : loc.Col: VT_EAST: 'library');
  enddo;

  return;
```

The first lines; h-specs is definig that we will use that API's that IceCap is supporting and we will hook into the activations group that IceCap already provides.

The include is simply defining the prototypes and data types we can use. E.g. the virtual terminal session structure (the VTS). Please notice that ALL IceCap API's refers to a pointer of the VTS, so in the first section we store a pointer to the VTS by the `%addr()` building. Also notes that all data structures is templates. So you need to construct you own with the `likeds` keyword.

All IceCap API's is prefixed by `vt` and the emulator API's is prefixed with `eml`

```
loc = vtLocate(pVts: VT_INPUT: FromLin: FromCol: VT_EAST: 'library');
```

vtLocate returns the position (line and column) of the first input field found east of (to the right of) the keyword "library". The search starts at FromCol and FromLin.

If the keyword was not found, it will return 0/0 in the location.

Now we need to add some extra features for that input field. We need to add the X-type of the drop down of libraries:

```
vtSetMetaData (pVts:loc:'{xtype:"DDLIB"}');
```

`vtSetMetaData` supplies the extra information needed for the emulator to present a given input field at the specified line/column in the current context, but as a different type.

In my little sample it will simply change all inputs after the keyword ‘library’ however in a real world application I would consider it a rather sloppy solution. You would rather do something like detecting the panel name and the do some action:

```
if vtSaaTitleContains (pVts: ‘Work with Members Using PDM’);
   vtSetMetaData (pVts:loc:'{xtype:"DDLIB"}');
endIf;
```

According to the IBM Systems Application Architecture – the title will be centred in the first line of the screen. So if the application complies with SAA you can detect the panel title with `vtSaaTitleContains()`However many applications has their own standards. In that case you can determine the application name and which particular screen is displayed with a combination if API calls like:

```
if  vtGetScreen(pVts: Lin1 :  Col1 : Len1) = ‘CUSTOMER01’
and vtGetScreen(pVts: Lin2 :  Col2 : Len2) = ‘Search’;
…
```

And maybe you event don’t want so show a particular screen to the end user – or maybe combine two or more screens in to one. In that case you net to navigate through the application with function keys like ENTER, page-up, F3 etc. Sometimes you need to fill in data to the application The API for that looks like:

```
vtSetField(pVts: lin : col : ‘MyValue’);
vtPressKey(pVts: vtENTER);
```

All the API's are defined and described in library ICEBREAK file QASPSRC member ICECAP.

Ex the template structure the virtual terminal session looks like:

```
The vts contains:
d vtsDS           ds                  based(I_NULL) qualified
 * total screen buffer as string or array
d   scnBuf                     3564
d     scn80x24                   80    dim(24) overlay(scnBuf)
d     scn132x27                 132    dim(27) overlay(scnBuf)
 * witdth and height of the current screen typical W:80 or W:132 and H:24 or W:27
d   width                        5i 0
d   height                       5i 0
 * current cursor location line and column  number 1=First
d   cursorLin                    5i 0
d   cursorCol                    5i 0
 * Number of input capable fields on the current
d   inputFields                  5i 0
 * input field that has focus 1=First
d   focus                        5i 0
 * Line number where the errror or errorsubile is
d   errLin                       5i 0
 * Each save/restore display can be used as system. 0=No window, 1=first window
d   winLevel                     5i 0
 * Width / Height of the current window in number of chars will be 80x24 or 132x27 if no window.
d   winWidth                     5i 0
d   winHeight                    5i 0
 * Starting line and column of the first char in the windo
d   winCol                       5i 0
d   winLin                       5i 0
 * PC-organizer command ( is the STRPCCMD ) was issued
d   pcoCmd                     256    varying
 * *ON if messages is pending on display msg queue
d   messages                     N
 * Status code: 0=Ok; -1=Timeout  >0 Automatic Sign-on Reason
d   status                     10i 0
 * Name of the connected device
d   deviceName                 10
 * If an error messages, this has the text else ''
d   errMsg                     128    varying
```

x-types

x-types is a concept of a javascript class the emulator will instantiate client side and enhance the layout or behaviour of an input field.

By default IceCap ships with a number of predefined x-types:

DATEFIELD	Date picker
CHECKBOX	Check box
BUTTON	Button
DDFIELD	Drop down – by simple array
DDSQL	Drop down – based on an SQL statement
DDLIB	Drop down – All libraries on the IBMI

These components are shipped as open-source. You can extend these or use them as origin for your own x-types. You will find them in the IceCap IFS path as which typically will be:

```
'/www/IceCap/dftXtypes.js'
```

Also you need to register your xtypes in the `icecap_confi.xml` in the `xtypepath`
If you want to write your own, you might find it helpful to visit the homepage of ExtJS which is the framework used in the IceCap emulator.

Inter process communication

Maybe the most important feature of IceCap is the possibility to communicate between IBMI sessions that are encapsulated in the IceBreak session. That is: you have full access to the IceBreak session, so from within the set and get session variables, run webservices, build ILOB's (Internal large objects) etc.

All you need is to add the IceBreak bind directory to your program. If your program is run in a normal 5250 emulator these API's will however have no effect.

Example: Lets say you have a good old 5250 program, where you work with a list of customers. Now you want to show a pie-chart on the portal for the current selected customer. Your pie-chart need just the customer number to do the trick:

All you need to add to add the following code to the 5250 program:

```
H BNDDIR('ICEBREAK')
  /include qasphdr,httpxserv
…
C                    EXFMT  MAIN  // Original code with custno
C                    CALLP  sesSetVarNum('custno' : CUSTNO)
...
```

But also – now your "old" applications can communicate with each other using the IceBreak session.

Ie. In and other 5250 program you can retrieve the customer number and show the invoices for that customer, without entering the customer number:

```
H BNDDIR('ICEBREAK')
  /include qasphdr,httpxserv
…
C                    eval custNo = sesGetVarNum ('custno');
C                    EXSR   CHAINCUST     // Original code now using the custno
C                    EXFMT  MAIN          // Original code
...
```

Appendix A

Example – menu loader

```
<%@ language="SQLRPGLE" sqlopt="COMMIT(*NONE) SRTSEQ(*LANGIDUNQ)" modopt="SRTSEQ(*LANGIDUNQ)" %>
<%
H DECEDIT('0,') DATEDIT(*YMD.)  COPYRIGHT('System og Metode (c), 2009')
H BNDDIR('MNUUTL')
H BNDDIR('ICEUTILS')
 * ------------------------------------------------------------------------------------ *
 * Program ....: sample menu
 *               Code is provided as is - this is only an example
 * Function ...: Build User Menu
 * Author .....: Claus Rytter Larsen, System og Metode A/S - June 2009
 * ------------------------------------------------------------------------------------ *
FXMREG     IF   E           K Disk
FXEREG     IF   E           K Disk
 * ------------------------------------------------------------------------------------ *
d i               s              3  0
d wID             s                   like(XMXEID)
d wChildID        s                   like(XMXEID) dim(14)
d sep             s              1
d len             s              3  0

d func            s            256    varying
d path            s             24    varying
d url             s            256    varying
d search          s            256    varying
d sqlCmd          s           5000    varying

d dftSelIco       s             48    varying
d                                     inz('folder_open.gif')
d dftUSlIco       s             48    varying
d                                     inz('folder_close.gif')
 * ------------------------------------------------------------------------------------ *
 /Include qSrc,mnuUtlP
 /include qasphdr,iceUtils
 * ------------------------------------------------------------------------------------ *
 * Initialize
 * ------------------------------------------------------------------------------------ *
/free
  SetContentType('application/XML; charset=windows-1252');

  wID = form('node');
  path = form('path');
  search = form('search');

  len = %len(%trim(path));
  if %subst(%trim(path):len:1) = '/';
    path = %subst(%trim(path):1:len-1);
  else;
    path = %trim(path);
  endif;

  if %subst(wID:1:1) = 'y';
    wID = qrystr('node');
  endif;

%>[<%

  sep = '';
  if search <> '';
    exsr sqlFetch;
    exsr sqlOpn;
    dow (sqlCod = 0);
      exsr addMNU;
      exsr sqlFetch;
    enddo;
    exsr sqlClose;
```

```
  else;
    exsr buildMenu;
  endif;

%>]<%

  return;
 // ------------------------------------------------------------------------ *
 // Build Menu
 // This system has that menu items in a record with one array
 // ------------------------------------------------------------------------ *
Begsr buildMenu;
  SetLL (wID) XMREC;
  ReadE (wID) XMREC;
  Dow not %eof(XMREG);
    wChildID = '';
    wChildID(1) = XMXE01;
    wChildID(2) = XMXE02;
    wChildID(3) = XMXE03;
    wChildID(4) = XMXE04;
    wChildID(5) = XMXE05;
    wChildID(6) = XMXE06;
    wChildID(7) = XMXE07;
    wChildID(8) = XMXE08;
    wChildID(9) = XMXE09;
    wChildID(10) = XMXE10;
    wChildID(11) = XMXE11;
    wChildID(12) = XMXE12;
    wChildID(13) = XMXE13;
    wChildID(14) = XMXE14;
    for i = 1 to 14;
      if wChildID(i) <> '';
        chain wChildID(i) XEREC;
        if %found;
          exsr addMNU;
        endif;
      endif;
    endfor;
    ReadE (wID) XMREC;
  enddo;
endsr;
 // ------------------------------------------------------------------------ *
 // Add Menu item
 // ------------------------------------------------------------------------ *
Begsr addMNU;
%><%=sep%>
{
  id:"<%=XEXEID%>",
  text:"<%=encodeJSONstr(XEXMTX)%>",
  icon:"/icemnu/system/images/icons/list.gif",
  leaf:true,
  path:"<%responseWrite(encodeJSONstr(path));%>",
  link:"<%responseWrite(encodeJSONstr(XEXCLP));%>",
  helplink:"<%=XEXCLP%>"
}<%
 sep = ',';
endsr;
 // ------------------------------------------------------------------------ *
 // Open sql cursor
 // ------------------------------------------------------------------------ *
begsr sqlOpn;
  sqlCmd  = 'select XEXMTX, XEXTYP, XEXHLP, XEXEID, XEXCLP'
          + ' from XEREG'
          + ' where upper(XEXMTX) like ''%'
          + %trim(uppercase(search)) + '%'''
          + ' order by XEXMTX';
/end-free
c/Exec SQL
c+    Prepare pList from :sqlCmd
c/End-Exec
/free
/end-free
C/Exec SQL
c+    Declare List cursor for pList
```

```
c/End-Exec
/free
/end-free
c/Exec SQL
c+     Open List
c/End-Exec
/free
endsr;
 // ------------------------------------------------------------------------ *
 // Fetch row fom sql cursor
 // ------------------------------------------------------------------------ *
begsr sqlFetch;
/end-free
c/Exec SQL
c+      Fetch list into :XEXMTX, :XEXTYP, :XEXHLP, :XEXEID, :XEXCLP
c/End-Exec
/free
endsr;
 // ------------------------------------------------------------------------ *
 // Close sql cursor
 // ------------------------------------------------------------------------ *
begsr sqlClose;
/end-free
c/Exec SQL
c+      Close list
c/End-Exec
/free
endsr;
/end-free
%>
```