http://www.lansa.com

LANSA

Newsletter
September 2006

RAMP from LANSA

RAMP goes way beyond conventional refacing and screen scraping technologies. It is an entirely new approach to iSeries application modernization and innovation.

LANSA's **Rapid Application Modernization Process** lets you run refaced legacy iSeries (AS/400) applications inside a LANSA Application Framework to give both modernized application navigation and a staged path for system replacement or deployment to other platforms.

Give existing RPG and COBOL applications a Windows rich-client or Web browser user interface with enhanced navigation in weeks or months rather than years – with no change to iSeries (AS/400) server logic.
Only RAMP lets you integrate reanimated 5250 screens with new GUI components that can also run on iSeries, Windows, UNIX or Linux. Moving your applications into the LANSA Application Framework lets you use the full power of the LANSA 2005 development suite and new technologies like XML, SOA and Web services.

RAMP is the Fastest Way to Modernize iSeries Applications

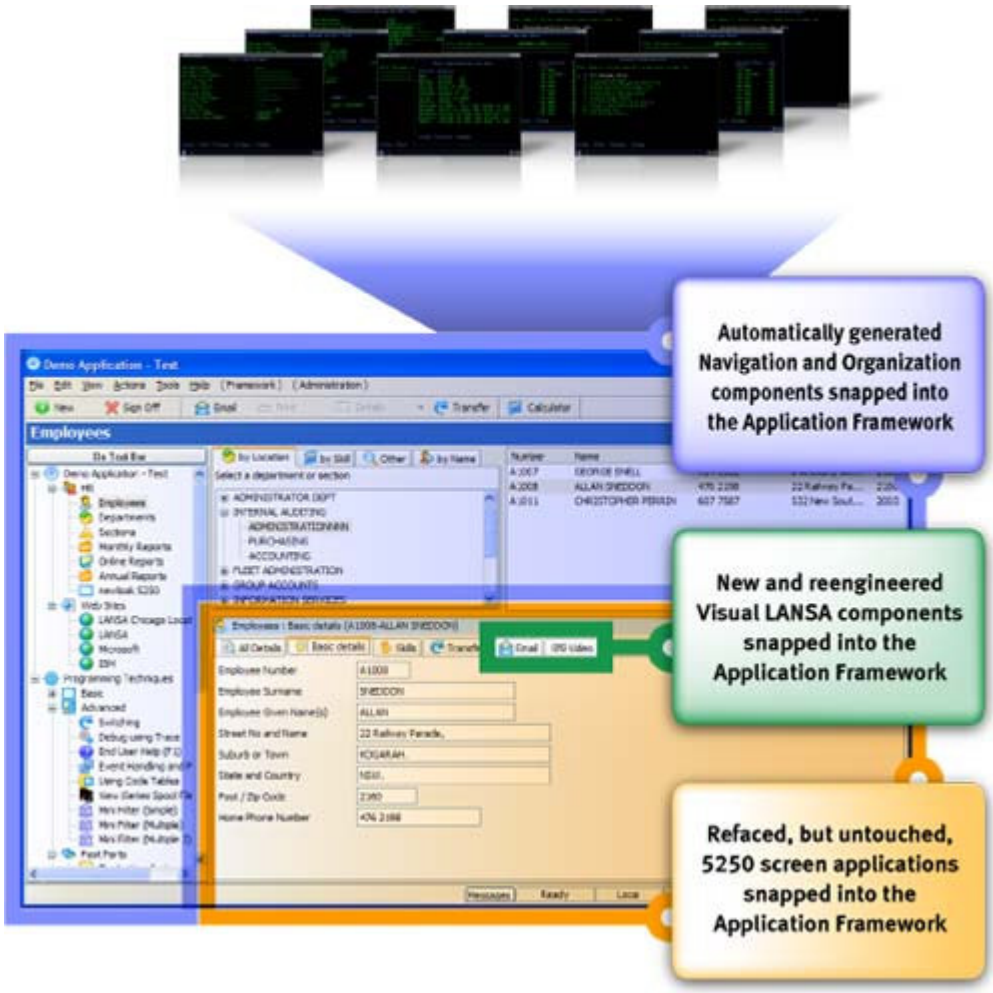
Rapid Application Modernization Process or RAMP is a new offering from LANSA that delivers both today's "must-have" tactical enhancements and a smooth path to long-term modernization of iSeries applications using LANSA's rapid prototyping and intelligent refacing tools.

RAMP provides a development and execution Application Framework with a Microsoft Outlook-style "look-and-feel." You can "mix-and-match" your existing iSeries 5250 screens and batch-based jobs with Visual LANSA components that can execute against iSeries, Windows, UNIX or Linux servers in a Web browser or as Windows rich-client applications.

RAMP goes way beyond conventional refacing technologies that only provide presentation enhancements. RAMP generates Navigation, Filter and Organization components for deployment on your choice of platforms, with your choice of user interface.

In This Issue

From Green Screen to GUI in Weeks



Modernize at Your Own Pace

Short-term Imperatives Versus Long-term Plans

- The RAMP approach recognizes that you must address the tactical issues of functionality shortcomings in existing applications while your organization works on its modernization plans. Business must continue, it can't stop while your systems are modernized.

- RAMP meets this challenge by delivering tactical solutions in parallel with long-term strategic modernization using the same development tools.

- You can Web-enable parts of your application to provide a self-service portal, consume and publish Web services or deliver new functionality to meet business requirements — all within the same Application Framework that serves as a platform for full modernization.

- With RAMP, you avoid duplication of effort and throw-away quick fixes that are a step sideways at best.

- With LANSA, adding product functionality or a new “face” to existing applications can be a step forward to full modernization.

Stage 1: Creation of a True Modernization Framework

- This step defines your modernization goals and rollout plan in a very short time, in a practical and pragmatic way.

- Using LANSA’s Instant Prototyping Assistant, you develop an unencumbered vision of your modernized application and plan what does and does not require significant reengineering.

- Most importantly, the result is a fully working Application Framework that is not thrown away — it evolves through Stages 2 and 3 of your modernization journey.

Stage 2: Navigation, Integration and Initial Enrichment

- Stage 2 uses LANSA’s Application Navigation Assistant to reanimate existing 5250 programs and snap them into the Application Framework produced in Stage 1.

- RAMP lets you “mix-and-match” your existing iSeries 5250 screens and batch jobs with Visual LANSA components that can execute against iSeries, Windows, UNIX or Linux servers in a Web browser or as Windows rich-client applications — all in the same Application Framework.

- RAMP raises the modernization bar to a new level and delivers much more than conventional refacing technologies that simply provide Presentation enhancements. RAMP adds a “touch of magic” and creates Navigation, Filter and Organization components for iSeries, Windows, UNIX or Linux.

Stage 3: Ongoing Reengineering and Enrichment

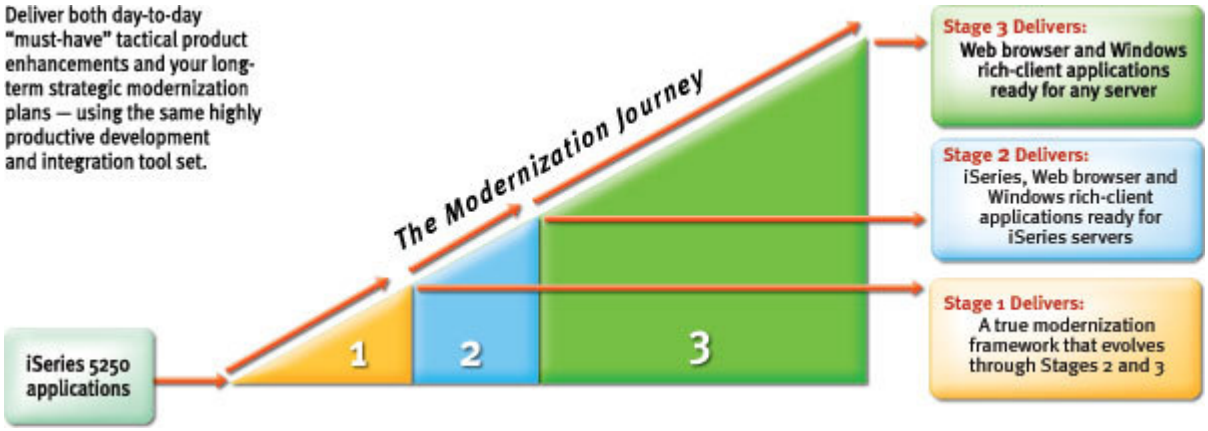
- This stage builds on the Application Framework developed in Stages 1 and 2. You decide which programs will be enriched and enhanced and the depth of modernization.

- If server platform independence is a vital component of your overall modernization strategy, then applications with RPG/DDS dependencies can be modernized with the full power of the LANSA 2005 development suite.

- At your own pace, you can progressively redevelop your old 5250-based application into a modern repository-based LANSA application that supports modern techniques such as Web services and SOA.

- The final result is a fully modernized application, built with your vision, to your plan and ready for your platform of choice.

RAMP Raises the iSeries Application Modernization Bar to a New Level



Cannot get past Existing LANSA Communications Directory prompt in Setup during upgrade to 11.0

Description

During an upgrade from LANSA V10.0 to V11.0, one of the install screens will prompt for the Existing LANSA Communications Directory. For most installs this value will be pre-filled and does not need to be changed. However in some cases no default value will appear for this prompt. Furthermore, even if the correct value is entered the Install will reject it and the install cannot continue.

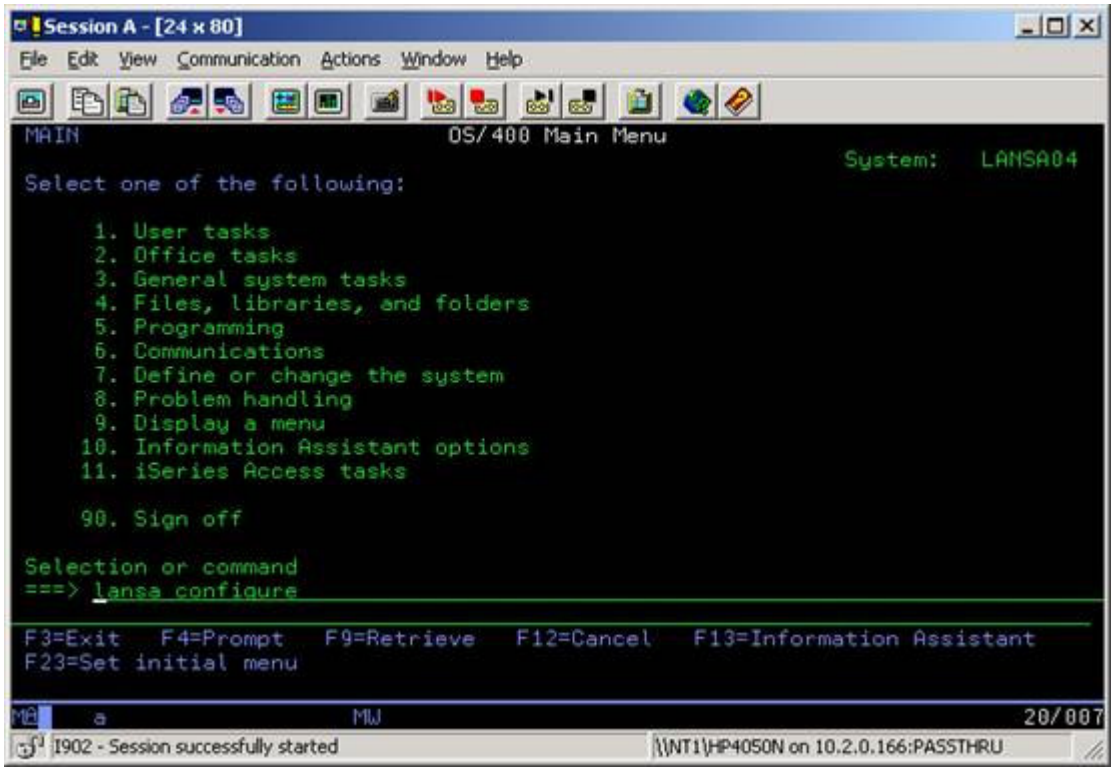
Cause

This is caused by the communications library path being blank in the data areas that the install uses. The install cannot complete until the data areas contain the correct value. The steps below should be followed to update the communications entry which will allow the install to complete.

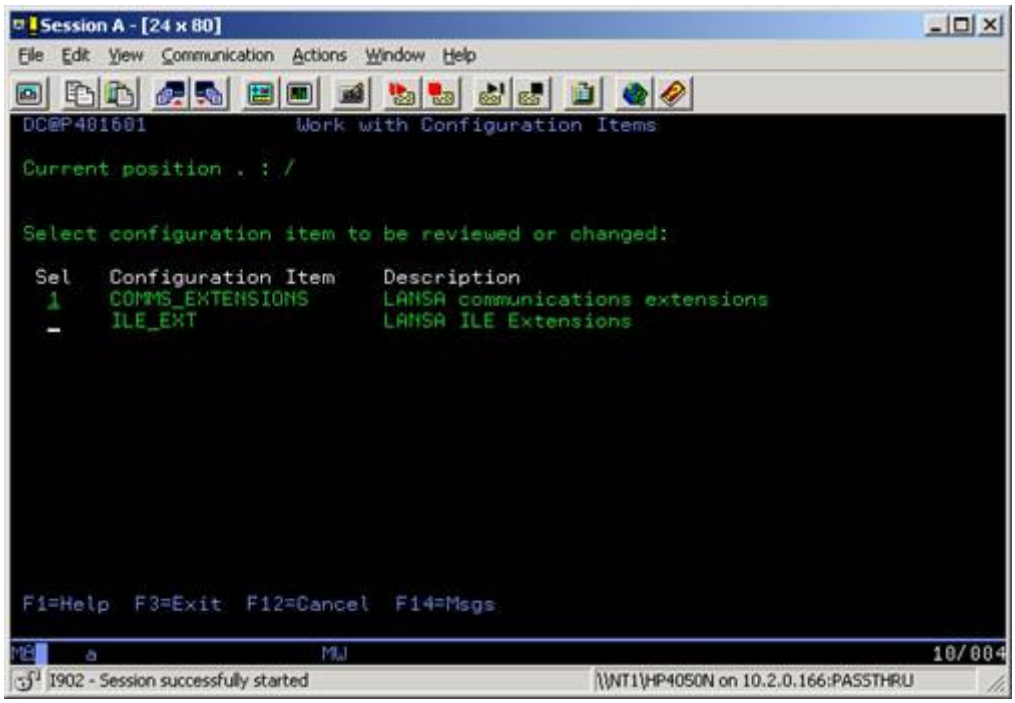
Solution

To correct the data areas, you should use following steps to ensure that the data is updated correctly.

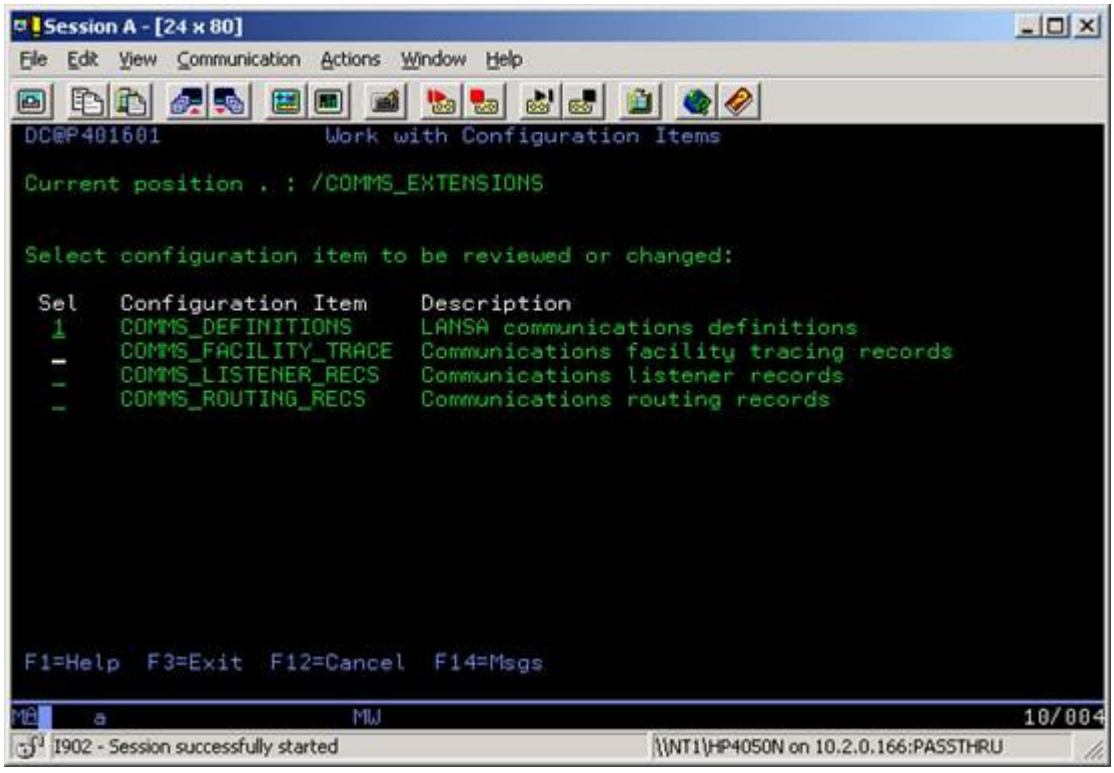
1. From the command prompt, run LANSA Configure



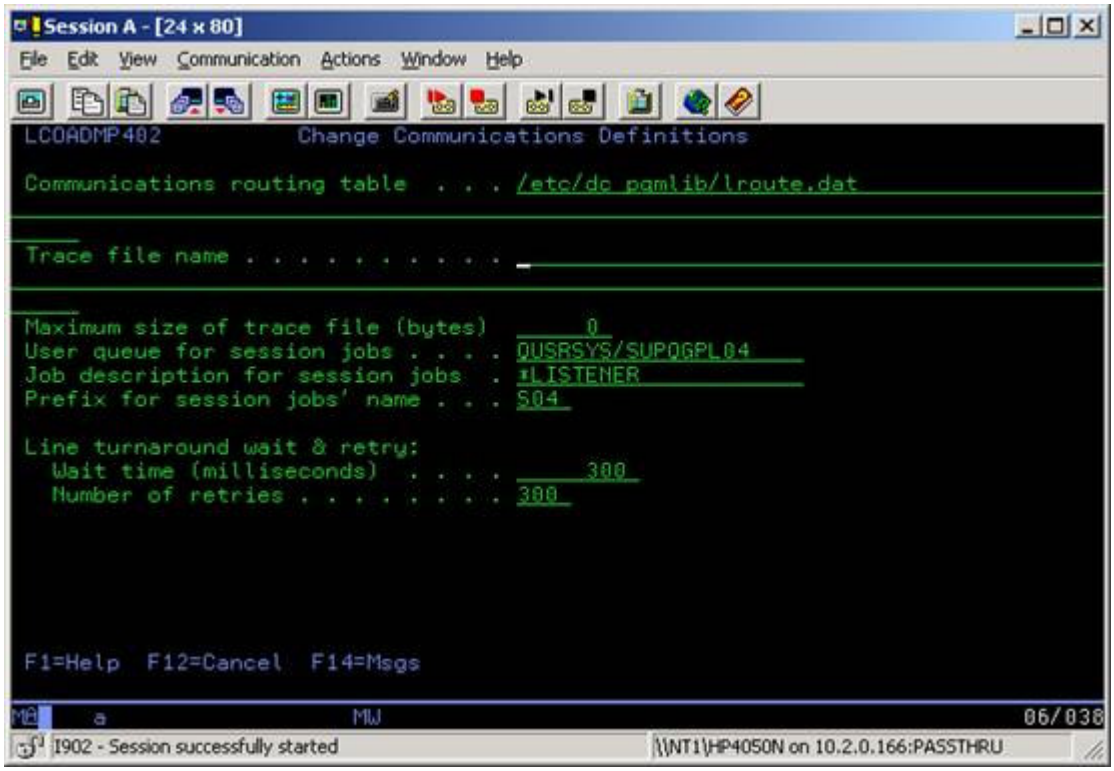
2. From the resulting menu choose COMMS_EXTENSIONS



3. Then from the menu choose COMMS_DEFINITIONS



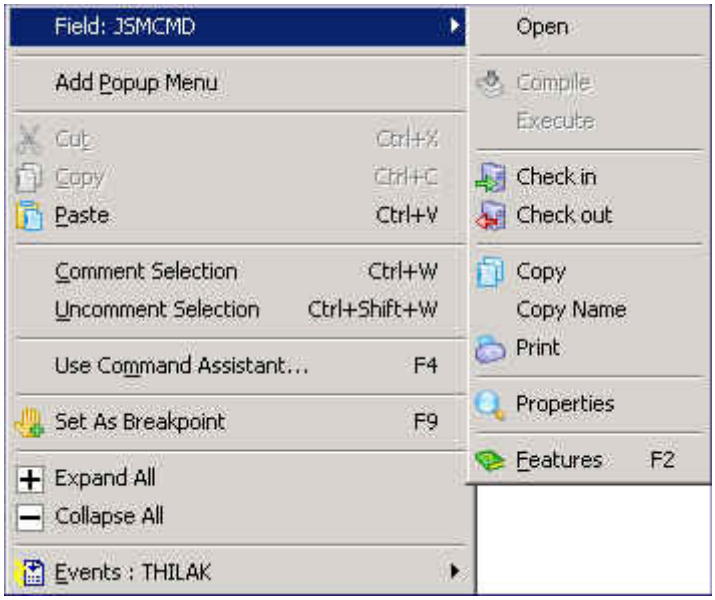
4. Press F21 to enter change mode and enter the correct communications routing table detail. For example, if LANSA is installed to the default location, you might enter "/etc/dc_pgmlib/lroute.dat" (note this may vary and you are advised to determine which directory is used in your configuration before continuing).



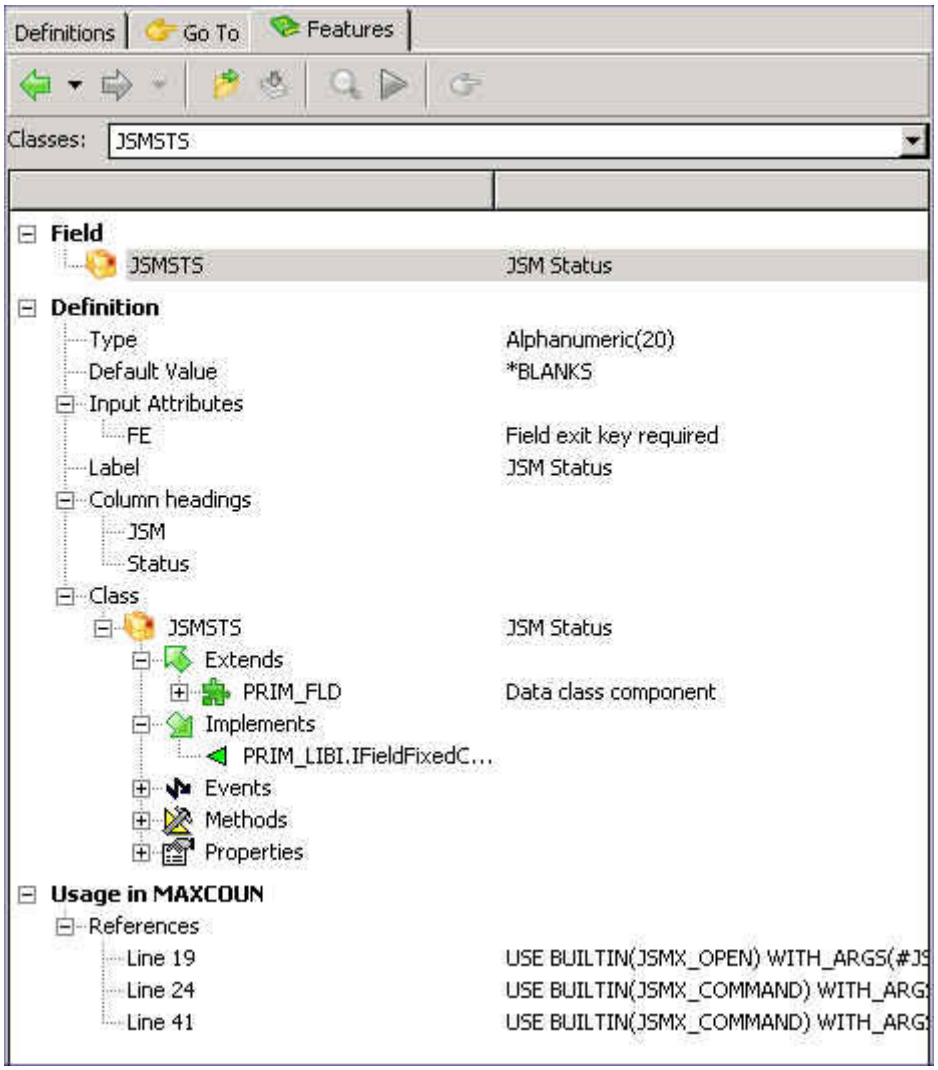
5. Press enter to accept and then exit the LANSA Configure program.

6. If you display the data area lcoa01 you should see the change has already taken place. The above steps should also create the lroute.dat table in the /etc/dc_pgmlib folder. After logging off and signing back on again as QSECOFR, you should be able to run the upgrade without any further problems.

Context sensitive features including HELP have vastly improved for the IDE for CU3



As can be seen above there are many more options available to the user from the Editor now. The F2 feature help has been vastly improved.



It is now very much context sensitive; put the cursor on a section of your code and press F2, or right click on any bit of code and the feature help will display the help contents for that property, component, method, etc.

*Using WEB_MAP FOR(*NONE) and WEB_MAP FOR(*INPUT) in WAMs*

The WEB_MAP command is used to declare the field and list data to be mapped between the Presentation Layer and your Webroutines. The FOR parameter defines the direction of these mappings.

Fields and lists declared with WEB_MAP FOR(*NONE) defines that their values are not to be mapped in or out of the Webroutine. This attribute is used for session state data where the value is only used in application logic.

Fields and lists declared with WEB_MAP FOR(*INPUT) defines that their values are only to be mapped in to the Webroutine and are not to be mapped out of the Webroutine.

Fields and lists declared with WEB_MAP FOR(*OUTPUT) defines that their values are not to be mapped in to the Webroutine and are only to be mapped out of the Webroutine.

Fields and lists declared with WEB_MAP FOR(*BOTH) defines that their values are to be mapped both in and out of the Webroutine.

Prior to V11.3, the *NONE and *INPUT values of the FOR parameter were not enforced correctly so that fields and lists declared with these FOR parameters had their data values mapped out of the Webroutine.

Once V11.3 is applied, the FOR parameter values are strictly enforced so that only fields and lists with WEB_MAP FOR(*OUTPUT) and FOR(*BOTH) have their data values mapped out of the Webroutine so that they are available to the Presentation Layer. This may have an impact on your existing WAM applications if you have declared fields and/or lists with WEB_MAP FOR(*NONE) or WEB_MAP FOR(*INPUT) but the values are required to be mapped out from your Webroutines.

Refer to the *Web Application Modules (WAMs)* Guide section 1.4.2 for a further explanation of the WEB_MAP command.

How to avoid MCH3402 errors when repeatedly running RDMLX functions on iSeries after checking and compile

Situation

You are writing and testing an iSeries application which contains RDMLX functions which are run and tested on an iSeries.

This will involve writing/amending the RDMLX function in Visual LANSA, checking in the RDMLX function to the iSeries, compiling on iSeries, then testing on iSeries. This will be repeated several times.

Attempting to run an RDMLX function from the iSeries LANSA menus when the RDMLX function has been changed will result in an MCH3402 error message ("Tries to refer to all or part of an object that no longer exists") for the function. This can be avoided by exiting from LANSA between executions of the RDMLX function.

A recommended way to run the function is by using the command
LANSA RUN PROCESS(…….) FUNCTION(…….) PARTITION(…)
or
LANSA X_ RUN PROCESS(…….) FUNCTION(…….) PARTITION(…)

Please also see the following Link on how JSMDirect in LANSA Integrator is impacted.
www.lansa.com/support/notes/p0286.htm

LANSA Customer day

10 October - Beurs van Berlage, Amsterdam

LANSA will organize a customer day at Tuesday 10 October.

We invite you all for this day, so please put this day into your agenda. We will send you a complete agenda later on this month.

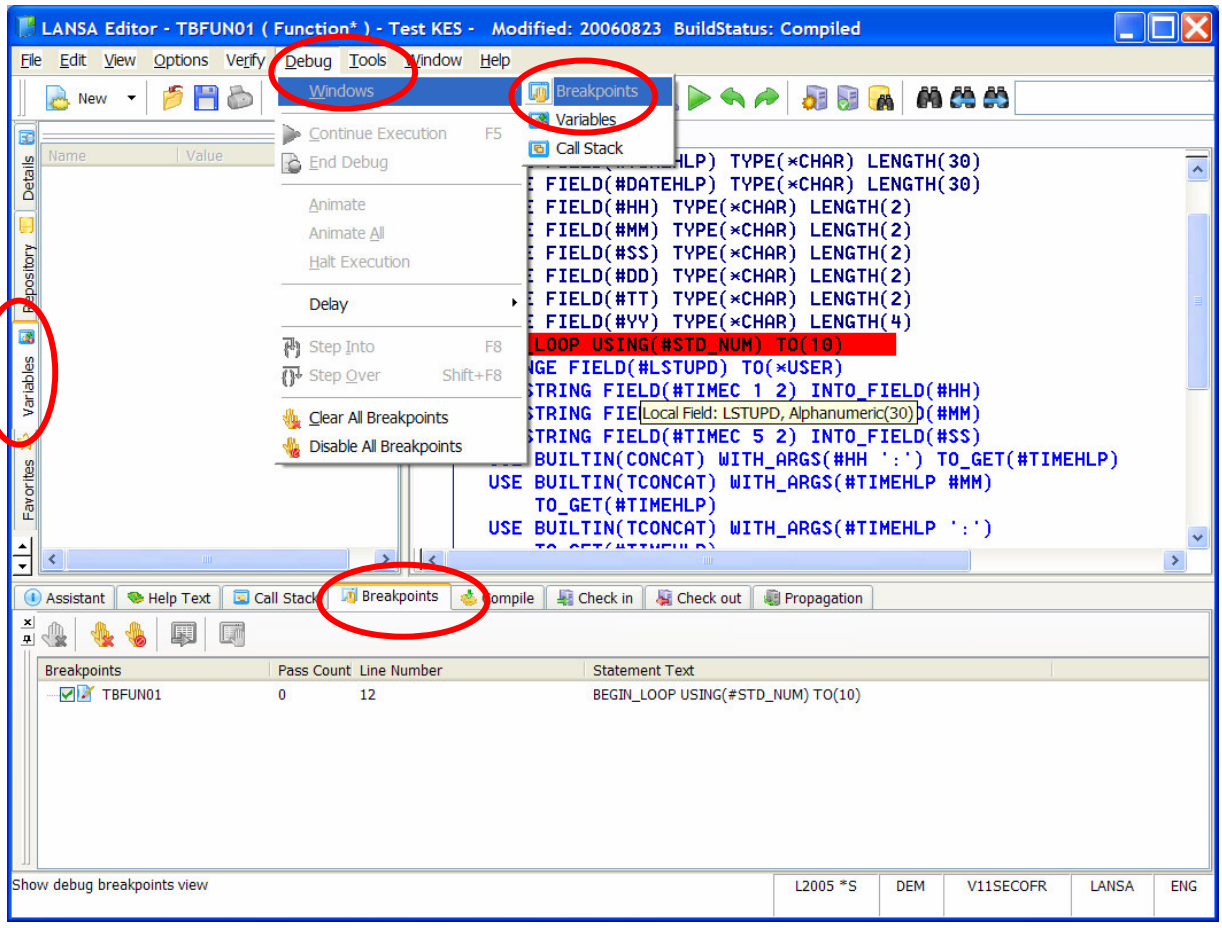
Guest speaker during this day **Diane Joester**, LANSA Senior Architect from Sydney Australia.

Martin Fincham, General Manager EMEA, will also be available during this day:
"LANSA is not just the name of a company or a product. LANSA is a community, a feeling of fellowship that results from sharing common attitudes, interests, and goals. The team at LANSA may be the nucleus of this community but we are not its only members. Our partners, customers, developers and end-users form a virtuous circle that strengthens the more everyone unites. Show your support for the LANSA community by attending our User Day; and don't just send the person that always attends events and conferences! Instead consider the value to your company, and the community, of sending a mixed group that represents the different roles and functions in your organisation - there will surely be something on the agenda for everyone."

Breakpoint Property Break On Condition for Debugging

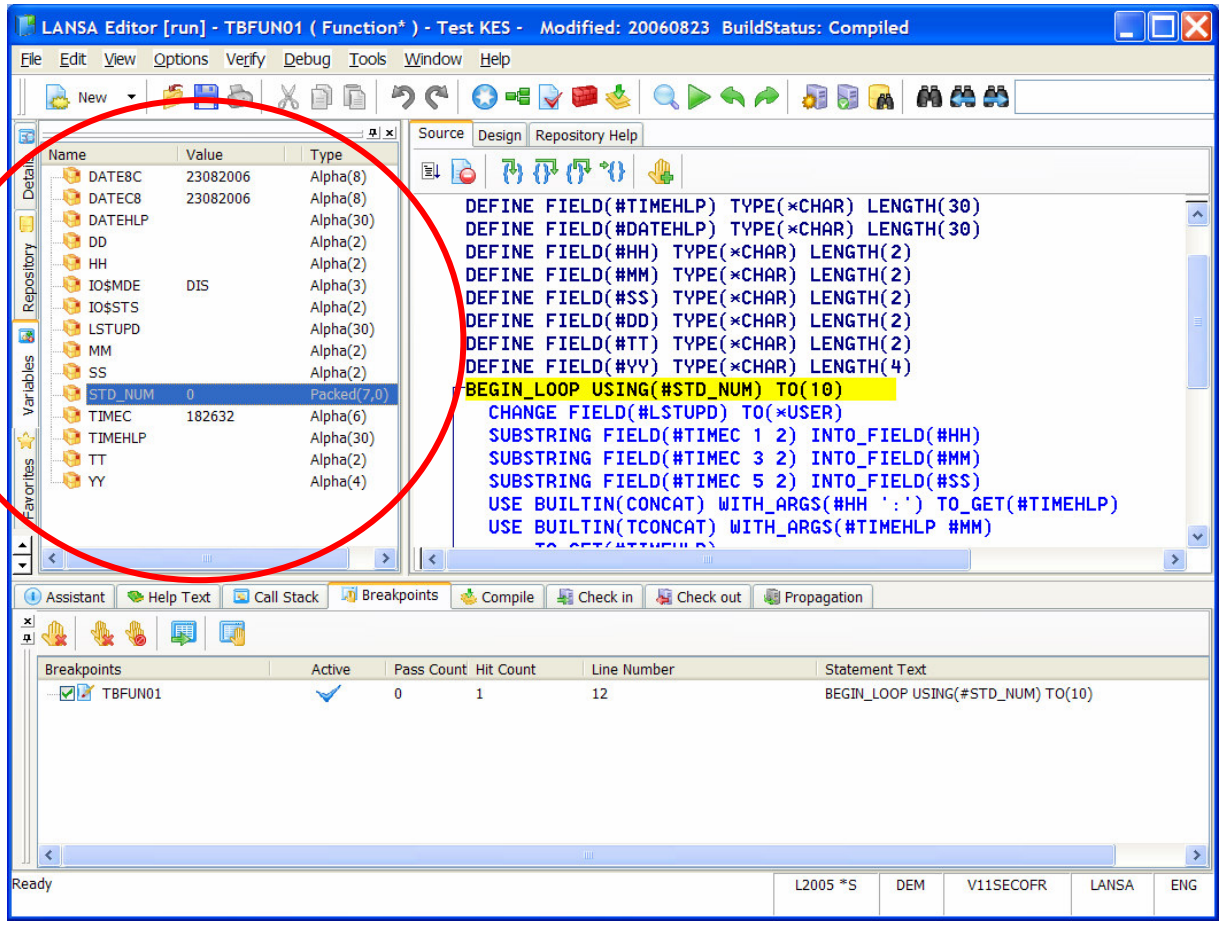
The Visual LANSA development environment has a feature which you are probably not aware of!

- Start the LANSA IDE
- Select the Function , or component to Debug.
- Set breakpoint (F9) on line with counter
- Select Debug, Windows, Breakpoints
- Select Debug, Windows, Variables

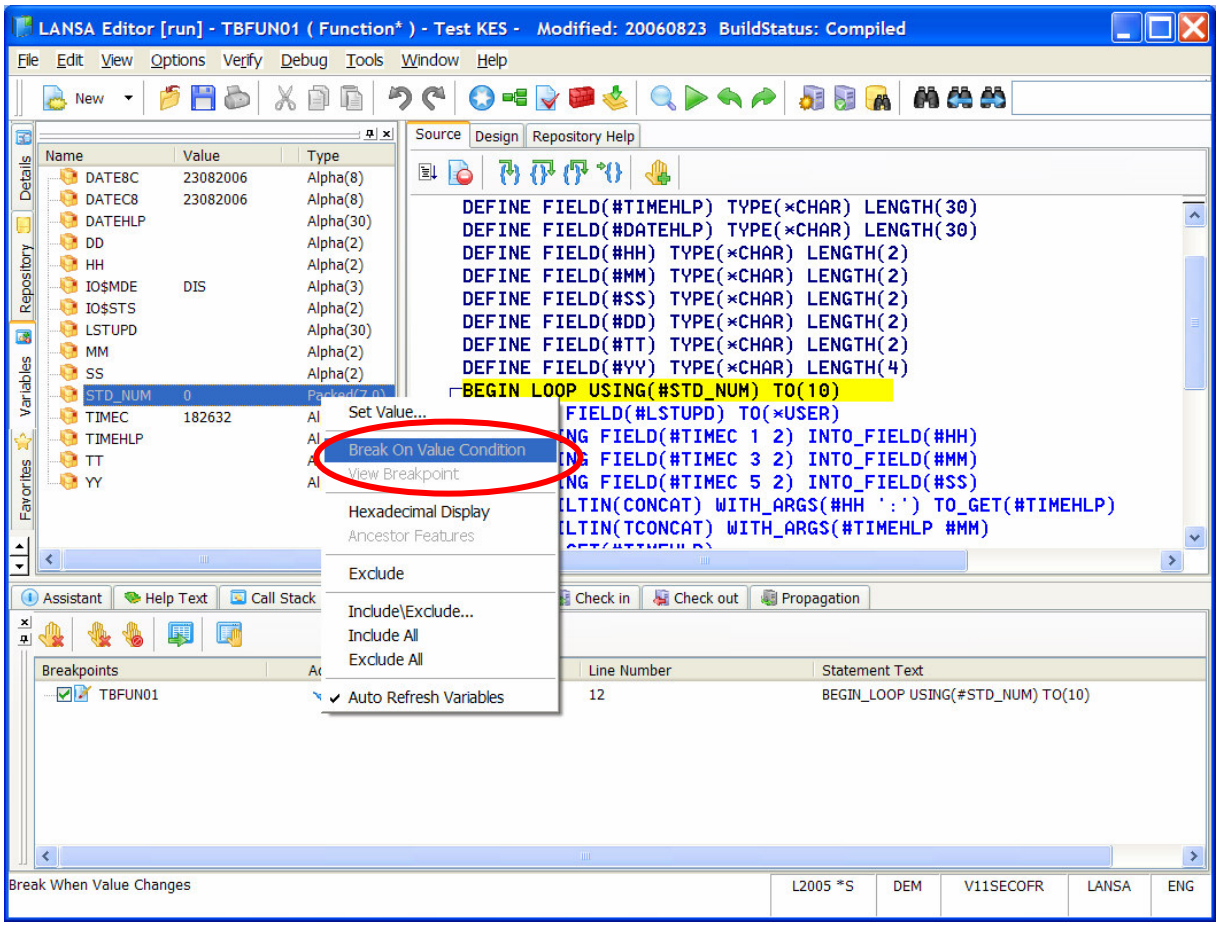


You can see the Breakpoint in the source shown in Red background color, the Variables-panel at the left side , and the Breakpoint-panel at the bottom.

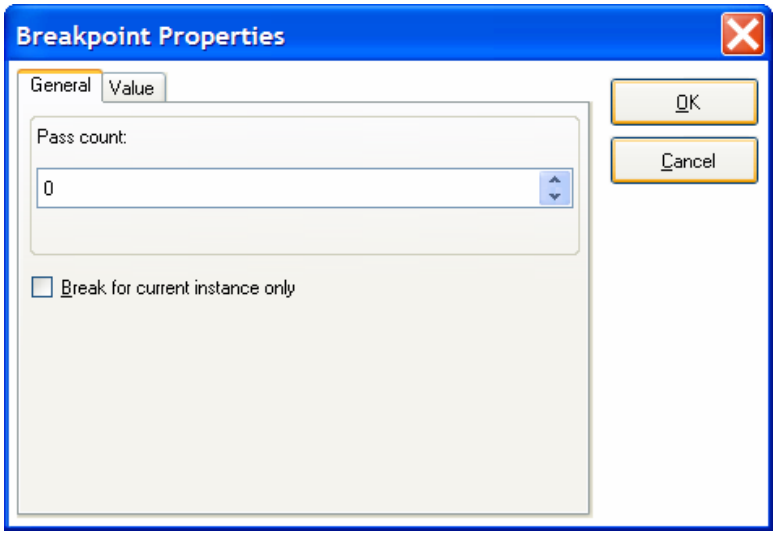
After running it the first time, the variables are shown:



Select the field to use with a condition , i.e. STD_NUM , and right click on it.

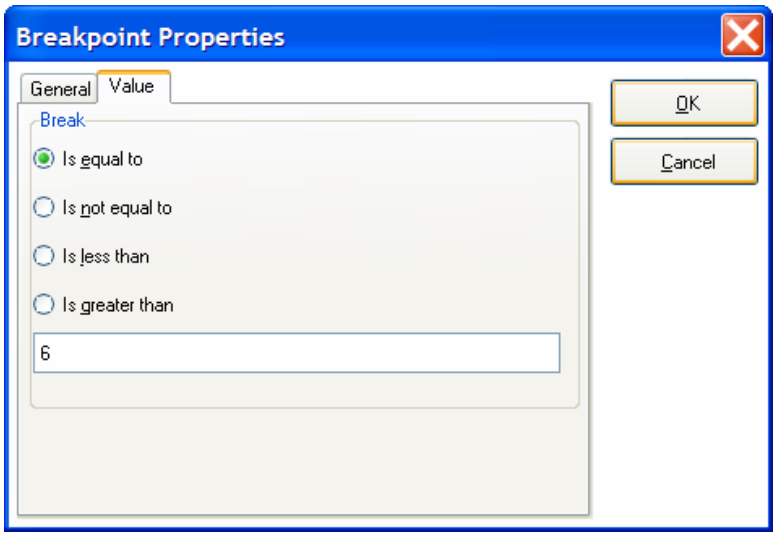


Select the option “Break on value Condition” and this small window will appear:

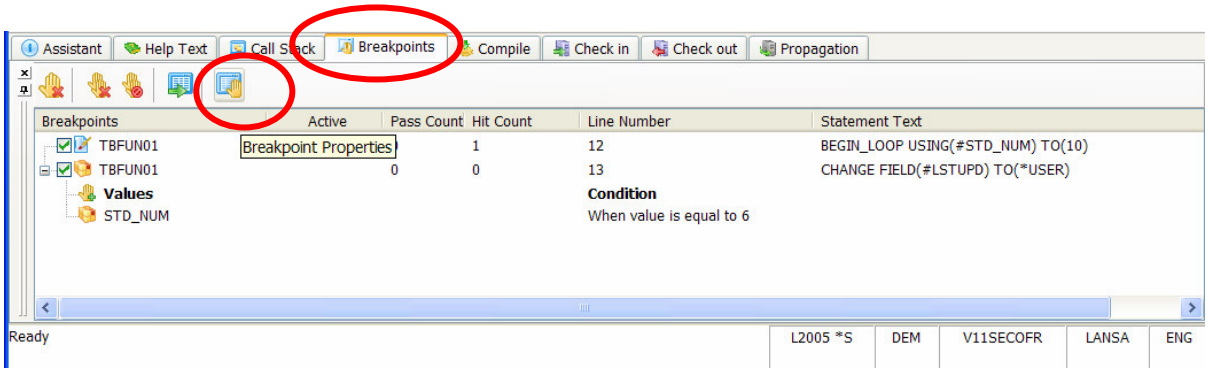


Here you can specify the number of times the program should pass this line before halting at this breakpoint, but you can also…….

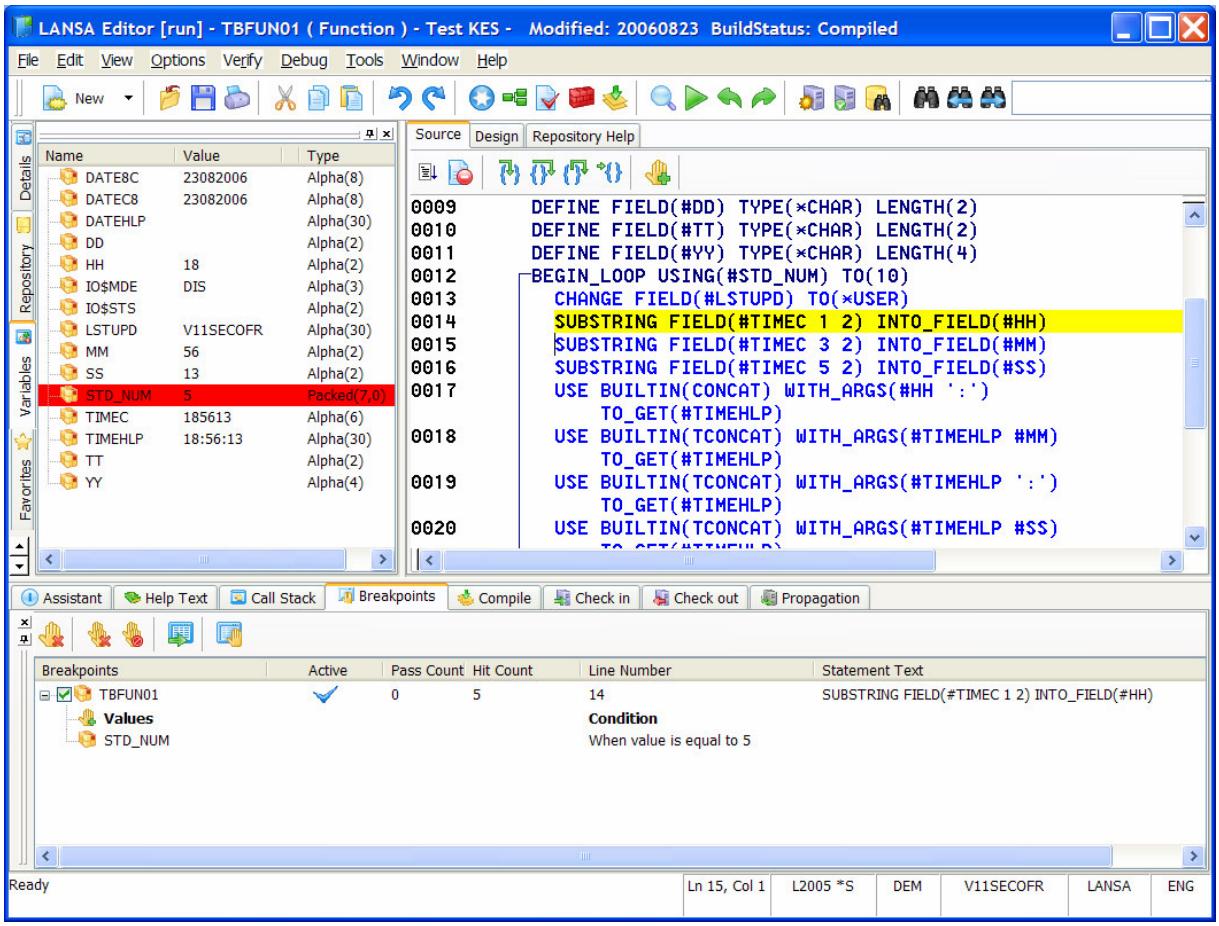
….. select the second tabsheet, “Value”, and define the condition when to halt at this line:



After specification , you can examine all the special breakpoint settings, using “Breakpoint Toolbar” ,in the lower pane tabsheet “Breakpoints” , for the details click on the button “Breakpoint properties”:



Start debug , press F5 and you can see the execution will stop after the number of cycles that you have specified (Breakpoint Properties - sheet 1) , or when the condition is true (Breakpoint Properties - sheet 2).



The value of the field #STD_NUM is 5 (See Top-left-side pane – Variables sheet).

When you press F5 again the function will continue onto the next breakpoint.

Explorer Sample Application

Use the explorer component to view files and directories either on local hard disks or across the network.

The events of the explorer component pass the file name, path and path type as parameters. The following sample code assigns this information to LANSA fields as an item gets the focus in an explorer component:

```
EVTROUTINE HANDLING(#EXPLORER.ItemGotFocus) Path(#PathName)
PathType(#PathType) Name(#FileName)

  Change #Std_Descs #PathType.Value

  Change #Std_Desc #PathName.Value

  Change #Std_Texts #FileName.Value

ENDROUTINE
```

If you want to use two explorer components, one for showing directories and paths and one for showing the files in them (similar to Windows Explorer), you can implement the communication between the two components using the NotifyComponent property of the first explorer component.

To see how the explorer component works, copy the following code and paste it into a form, compile and execute it.

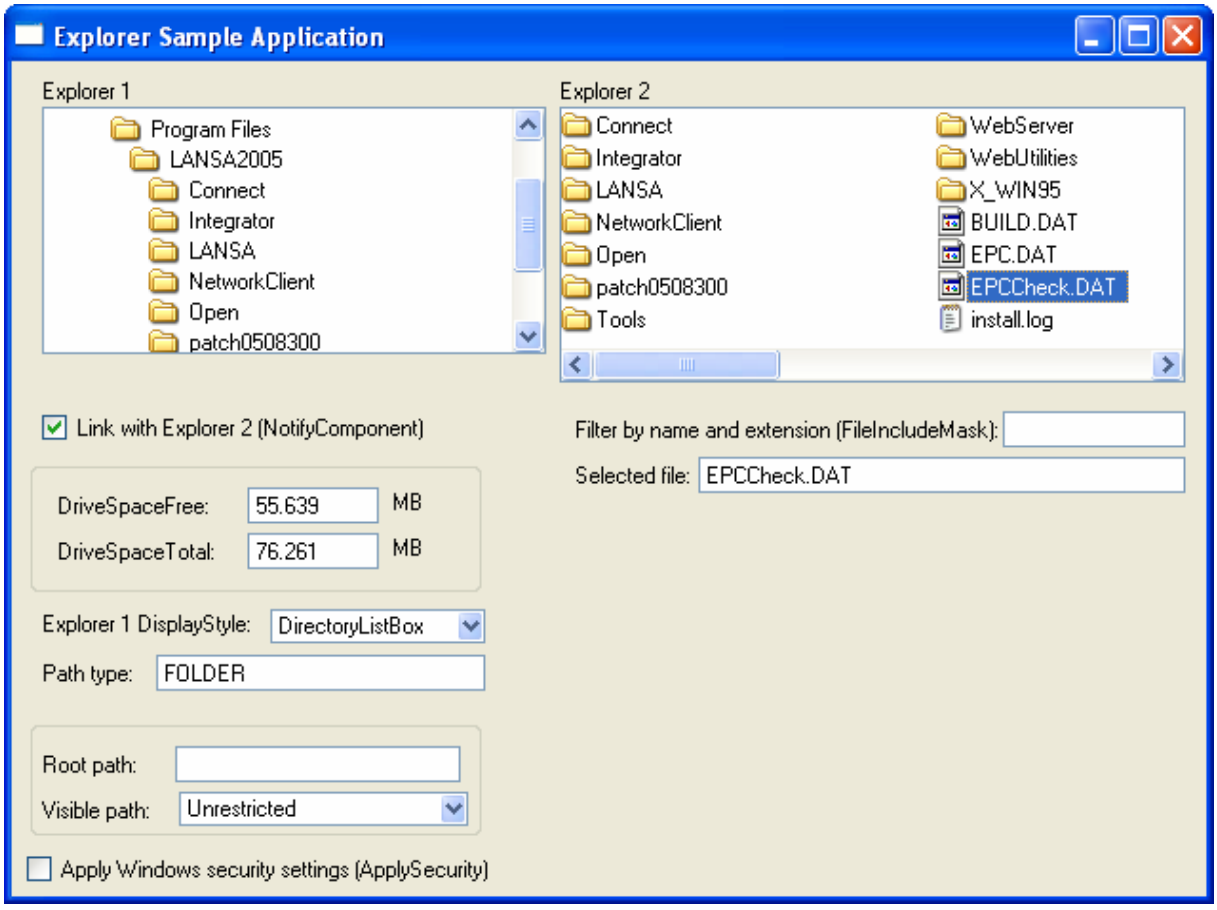
Source:

```
Function Options(*DIRECT)

Begin_Com Role(*EXTENDS #PRIM_FORM) Caption('Explorer Sample Application') Clientheight(439)
Clientwidth(628) Height(473) Left(404) Top(141) Width(636)

Define_Com Class(#PRIM_DCBX) Name(#DCBX_1) Displayposition(1) Fileincludemask('"*.*"')
Filename('Desktop') Height(145) Left(16) Parent(#COM_OWNER) Tabposition(1) Tabstop(False)
Top(24) Width(265)
Define_Com Class(#PRIM_DCBX) Name(#DCBX_2) Displayposition(2) Displaystyle(GeneralListView)
Fileincludemask('"*.*"') Filename('Desktop') Height(145) Left(288) Parent(#COM_OWNER)
Tabposition(2) Tabstop(False) Top(24) Width(329)
Define_Com Class(#PRIM_CMBX) Name(#CMBX_1) Componentversion(1) Displayposition(3)
Height(18) Left(136) Parent(#COM_OWNER) Showselection(False) Showselectionhilight(False)
Tabposition(3) Top(288) Width(113)
Define_Com Class(#PRIM_LABL) Name(#LABL_1) Caption('Explorer 1 DisplayStyle:')
Displayposition(4) Height(20) Left(16) Parent(#COM_OWNER) Tabposition(4) Tabstop(False)
Top(288) Width(113)
Define_Com Class(#PRIM_CBCL) Name(#CBCL_1) Displayposition(1) Parent(#CMBX_1)
Source(#STD_TEXT)
Define_Com Class(#PRIM_LABL) Name(#LABL_2) Caption('Explorer 1') Displayposition(5) Height(17)
Left(16) Parent(#COM_OWNER) Tabposition(5) Tabstop(False) Top(8) Width(98)
Define_Com Class(#PRIM_LABL) Name(#LABL_3) Caption('Explorer 2') Displayposition(6) Height(17)
Left(288) Parent(#COM_OWNER) Tabposition(6) Tabstop(False) Top(8) Width(98)
Define_Com Class(#PRIM_CKBX) Name(#CKBX_1) Caption('Link with Explorer 2 (NotifyComponent)')
Displayposition(7) Height(17) Left(16) Parent(#COM_OWNER) Tabposition(7) Top(184) Width(233)
```

```
Define_Com Class(#PRIM_CKBX) Name(#CKBX_2) Caption('Apply Windows security settings
(ApplySecurity)') Displayposition(8) Height(17) Left(8) Parent(#COM_OWNER) Tabposition(8)
Top(416) Width(257)
Define_Com Class(#STD_NUM.Visual) Name(#STD_NUM) Caption('DriveSpaceFree:')
Displayposition(9) Height(19) Labeltype(Caption) Left(24) Marginleft(100) Parent(#COM_OWNER)
Tabposition(9) Top(224) Usepicklist(False) Width(169)
Define_Com Class(#PRIM_LABL) Name(#LABL_4) Caption('MB') Displayposition(10) Height(17)
Left(200) Parent(#COM_OWNER) Tabposition(10) Tabstop(False) Top(224) Width(25)
Define_Com Class(#STD_NUM.Visual) Name(#STD_NUM_1) Caption('DriveSpaceTotal:')
Displayposition(11) Height(19) Labeltype(Caption) Left(24) Marginleft(100) Parent(#COM_OWNER)
Tabposition(11) Top(248) Usepicklist(False) Width(169)
Define_Com Class(#PRIM_LABL) Name(#LABL_5) Caption('MB') Displayposition(12) Height(17)
Left(200) Parent(#COM_OWNER) Tabposition(12) Tabstop(False) Top(248) Width(25)
Define_Com Class(#STD_TEXTS.Visual) Name(#STD_TEXTS) Caption('Filter by name and extension
(FileIncludeMask):') Displayposition(13) Enabled(False) Height(19) Labeltype(Caption) Left(296)
Marginleft(225) Parent(#COM_OWNER) Tabposition(13) Top(184) Usepicklist(False) Width(321)
Define_Com Class(#STD_DESC.Visual) Name(#STD_DESC) Caption('Selected file:')
Displayposition(14) Enabled(False) Height(19) Labeltype(Caption) Left(296) Marginleft(65)
Parent(#COM_OWNER) Tabposition(14) Top(208) Usepicklist(False) Width(321)
Define_Com Class(#STD_DESCL.Visual) Name(#STD_DESCL) Caption('Path type:')
Displayposition(15) Height(19) Labeltype(Caption) Left(16) Marginleft(60) Parent(#COM_OWNER)
Tabposition(15) Top(312) Usepicklist(False) Width(233)
Define_Com Class(#PRIM_GPBX) Name(#GPBX_1) Displayposition(16) Height(73) Left(8)
Parent(#COM_OWNER) Tabposition(16) Tabstop(False) Top(208) Width(241)
Define_Com Class(#PRIM_GPBX) Name(#GPBX_2) Displayposition(17) Height(65) Left(8)
Parent(#COM_OWNER) Tabposition(17) Tabstop(False) Top(344) Width(241)
Define_Com Class(#STD_INST2.Visual) Name(#STD_INST2) Caption('Root path:') Displayposition(1)
Height(19) Labeltype(Caption) Left(8) Marginleft(70) Parent(#GPBX_2) Tabposition(1) Top(16)
Usepicklist(False) Width(220)
Define_Com Class(#PRIM_CMBX) Name(#CMBX_3) Componentversion(1) Displayposition(2)
Height(18) Left(80) Parent(#GPBX_2) Showselection(False) Showselectionhilight(False)
Tabposition(2) Top(40) Width(152)
Define_Com Class(#PRIM_LABL) Name(#LABL_7) Caption('Visible path:') Displayposition(3)
Height(17) Left(8) Parent(#GPBX_2) Tabposition(3) Tabstop(False) Top(42) Width(98)
Define_Com Class(#PRIM_CBCL) Name(#CBCL_3) Displayposition(1) Parent(#CMBX_3)
Source(#STD_INSTR)

Evtroutine Handling(#com_owner.Initialize)
* Fill Explorer Displaystyles
Change Field(#std_text) To('''DirectoryListBox''')
Add_Entry To_List(#CMBX_1)
Change Field(#std_text) To('''DirectoryListView''')
Add_Entry To_List(#CMBX_1)
Change Field(#std_text) To('''DirectoryTreeView''')
Add_Entry To_List(#CMBX_1)
Change Field(#std_text) To('''DriveComboBox''')
Add_Entry To_List(#CMBX_1)
Change Field(#std_text) To('''FileListBox''')
Add_Entry To_List(#CMBX_1)
Change Field(#std_text) To('''FileListView''')
Add_Entry To_List(#CMBX_1)
Change Field(#std_text) To('''GeneralListView''')
Add_Entry To_List(#CMBX_1)
Get_Entry Number(4) From_List(#CMBX_1)
Set Com(#CMBX_1.CURRENTITEM) FOCUS(TRUE)

* Fill VisiblePaths
Change Field(#std_instr) To('''LocalDrivesOnly''')
Add_Entry To_List(#CMBX_3)

Change Field(#std_instr) To('''RootPath''')
```

```
Add_Entry To_List(#CMBX_3)
Change Field(#std_instr) To("'Unrestricted"')
Add_Entry To_List(#CMBX_3)
Get_Entry Number(3) From_List(#CMBX_3)
Set Com(#CMBX_3.CURRENTITEM) FOCUS(TRUE)
Endroutine


* Set DisplayStyle for Explorer 1
Evtroutine Handling(#CMBX_1.ItemGotSelection) Options(*NOCLEARMESSAGES
*NOCLEARERRORS)
Set Com(#DCBX_1) DISPLAYSTYLE(#cmbx_1.text)
Endroutine


* Set VisiblePath for Explorer 1
Evtroutine Handling(#CMBX_3.ItemGotSelection) Options(*NOCLEARMESSAGES
*NOCLEARERRORS)
Set Com(#DCBX_1) VISIBLEPATH(#cmbx_3.text)
Endroutine


* Specify NotifyComponent Property for Explorer 1
Evtroutine Handling(#CKBX_1.Click)
If Cond('#ckbx_1.ButtonState *eq Checked')
Set Com(#DCBX_1) NOTIFYCOMPONENT(#DCBX_2)
Set Com(#STD_DESC #STD_TEXTS) ENABLED(True)
Else
Set Com(#DCBX_1) NOTIFYCOMPONENT(*null)
Set Com(#STD_DESC #STD_TEXTS) ENABLED(False)
Endif
Endroutine


* Set the ApplySecurity property for Explorer 1 and Explorer 2
Evtroutine Handling(#CKBX_2.Click)
If Cond('#ckbx_1.ButtonState *eq Checked')
Set Com(#DCBX_1 #DCBX_2) APPLYSECURITY(True)
Else
Set Com(#DCBX_1 #DCBX_2) APPLYSECURITY(False)
Endif
Endroutine


* When a new path is selected, get the total and available space on the drive and the path type
Evtroutine Handling(#DCBX_1.PathChanged) Options(*NOCLEARMESSAGES *NOCLEARERRORS)
Change Field(#std_num) To(#DCBX_1.DriveSpaceFree)
Change Field(#std_num_1) To(#DCBX_1.DriveSpaceTotal)
Change Field(#std_descl) To(#DCBX_1.PathType)
Endroutine


* Set the IncludeMask for Explorer 2
Evtroutine Handling(#STD_TEXTS.Changed) Options(*NOCLEARMESSAGES *NOCLEARERRORS)
Set Com(#DCBX_2) FILEINCLUDEMASK(#std_texts)
Endroutine


* Get the file name selected in Explorer 2

Evtroutine Handling(#DCBX_2.ItemGotFocus) Options(*NOCLEARMESSAGES *NOCLEARERRORS)
```

```
Change Field(#std_desc) To(#DCBX_2.FileName)
Endroutine


* Get the RootPath name
Evtroutine Handling(#STD_INST2.Changed) Options(*NOCLEARMESSAGES *NOCLEARERRORS)
Set Com(#DCBX_1) ROOTPATH(#std_inst2.value)
Endroutine
End_Com
```

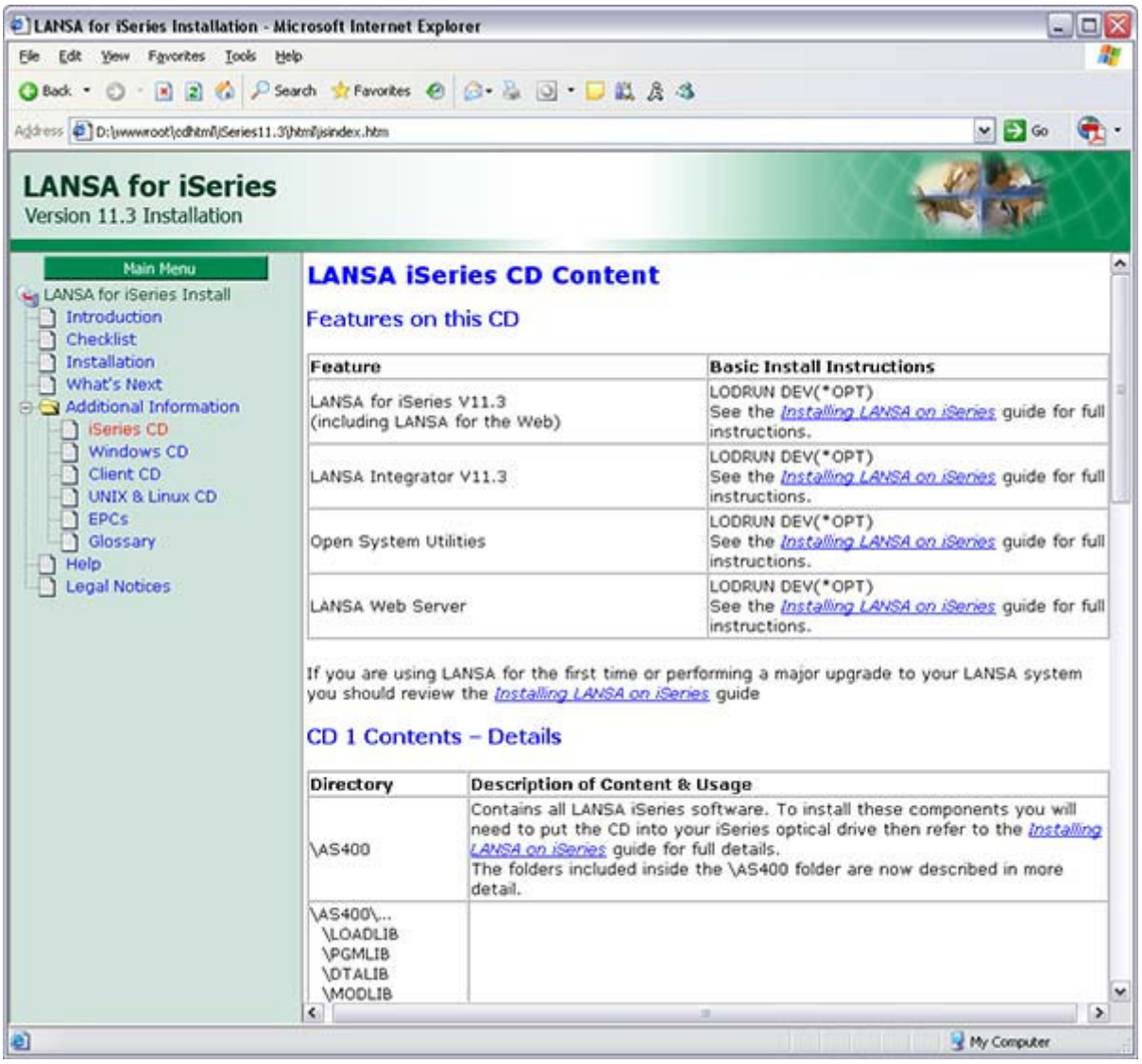


How to view the contents of the 2 LANSA iSeries 11.3 CDs

Prior to V11.3, LANSA for the iSeries software has been contained within 1 CD. Now in version 11.3 LANSA iSeries spans over 2 CD's. The contents of the second CD includes a set of standard imports. When installing or upgrading LANSA for iSeries 11.3 you will automatically be prompted to insert the second CD. This will typically occur when the install or upgrade is performing the partition initializations.

To view a list of the contents of the iSeries CD's you can perform the following:

1. Put the iSeries CD1 of 2 in a Windows drive, the auto-load will then load the index page
2. Select the "LANSA for iSeries install" on the main menu
3. In the left menu, select Additional Information and then iSeries CD. This lists the contents of the 2 iSeries CDs



Imbedded Interface Points (IIPs) VLF

(Thanks to Jack Moerman from West Brabantse Delta, the Netherlands)

The Visual LANSA Framework is shipped with a large number of imbedded interface points (IIPs). IIPs are places at which externally exposed code is invoked to perform specific internal (or imbedded) logic while the Framework is executing.

For example, in Windows applications there is an IIP method named avConnectFiles that defines how files are to be connected up to a server system by the Framework.

This standard shipped IIP version does this:

```
Mthroutine avConnectFiles options(*Redefine)
* ==> Define_map *input  #std_obj  #UserProfile
* ==> Define_map *input #vf_elnum #DftBlockSize
* ==> Define_map *input #vf_elnum #DftMaxRecSel

USE BUILTIN(CONNECT_FILE) WITH_ARGS('*' *SSERVER_SSN #DftBlockSize.Value
                          #DftMaxRecSel.Value)
Endroutine
```

If you want you can modify this shipped IIP logic to do something different for your Framework.
In Windows applications the IIPs are defined as *methods* in the shipped *component* UF_SYSTM.
In Web browser applications the IIPs are defined *as RDML functions* in the shipped *process* UF_SYSBR.

If you want to learn more about IIPs then a good place to start is by looking at the source code shipped in component UF_SYSTM (for Windows applications) and in the functions contained in the process UF_SYSBR (for Web browser applications).

West Brabantse Delta has iSeries 5250 LANSA applications and Windows applications created in LANSA's Visual LANSA Framework (it uses LANSA SuperServer to use the iSeries file data). When an end user starts an application in the VLF, they want to use another librarylist then when the same user starts a 5250 application on the iSeries.

They use the Imbedded Interface Point technique of the VLF to handle this.

Jack Moerman:

ALGLIBL is a function that handles the setting of the libarylist. In our development partition we want to use a separate data set for each of our developers, but they all make use of the same file definitions.

You can expand the X_RUN.EXE command with the UDEF parameter to complete this. It makes the IIP universal.

The avConnectFiles source method routine:

```
MTHROUTINE NAME(avConnectFiles) OPTIONS(*Redefine)
* ==> Define_map *input  #std_obj  #UserProfile
* ==> Define_map *input #vf_elnum #DftBlockSize
* ==> Define_map *input #vf_elnum #DftMaxRecSel

USE BUILTIN(CONNECT_FILE) WITH_ARGS('*' *SSERVER_SSN #DftBlockSize.Value
#DftMaxRecSel.Value)
* >>>> start of manually added source <<.
DEFINE FIELD(#dataset) TYPE(*CHAR) LENGTH(3) DESC('Name dataset')
DEF_LIST NAME(#WL_LIBL) FIELDS(#dataset) TYPE(*WORKING) ENTRYS(0000001)
* Read UDEF which is part of the X_RUN cmd
USE BUILTIN(GET_SESSION_VALUE) WITH_ARGS(UDEF) TO_GET(#DATASET #RETURNCD)
* call ALGLIBL
ADD_ENTRY TO_LIST(#WL_LIBL)
USE BUILTIN(CALL_SERVER_FUNCTION) WITH_ARGS(*SSERVER_SSN ALGLIBL N N
#WL_LIBL) TO_GET(#returncd)
IF COND('#returncd *ne OK')
CHANGE FIELD(#msgdesc) TO('ALGLIBL ended abnormally, check report')
USE BUILTIN(message_box_add) WITH_ARGS(#MSGDESC)
USE BUILTIN(message_box_show) WITH_ARGS(*Default *Default ERROR ALGLIBL)
ABORT
ENDIF
* >>>> end of manually added source <<.
ENDROUTINE
```

This is the source of the ALGLIBL function on the iSeries:

```
FUNCTION OPTIONS(*NOMESSAGES *DEFERWRITE *DIRECT) RCV_LIST(#WL_LIBL)
* Program interface
DEFINE FIELD(#DATASET) TYPE(*CHAR) LENGTH(3) DESC('Name dataset')
DEF_LIST NAME(#WL_LIBL) FIELDS(#DATASET) TYPE(*WORKING) ENTRYS(0000001)
* Workfields
DEFINE FIELD(#CMD) TYPE(*CHAR) LENGTH(80)
DEFINE FIELD(#CMD2) TYPE(*CHAR) LENGTH(80)
* Get the name from dataset
GET_ENTRY NUMBER(1) FROM_LIST(#WL_LIBL)
IF COND('#io$sts *eq OK')
* built cmd, syntax:
* CALL PGM(*LIBL/ALGLIBL) PARM(&DATASET &PARTITIE)
IF COND('#dataset *eq *blanks')
USE BUILTIN(CONCAT) WITH_ARGS('CALL PGM(*LIBL/ALGLIBL) PARM(' *QUOTE '   ' *QUOTE)
TO_GET(#CMD)
USE BUILTIN(CONCAT) WITH_ARGS(*QUOTE *PARTITION *QUOTE ')') TO_GET(#CMD2)
USE BUILTIN(BCONCAT) WITH_ARGS(#CMD #CMD2) TO_GET(#CMD)
ELSE
USE BUILTIN(CONCAT) WITH_ARGS('CALL PGM(*LIBL/ALGLIBL) PARM(' #DATASET ' '
*PARTITION ')') TO_GET(#CMD)
ENDIF
```

```
* Call of CL program ALGLIBL
EXEC_OS400 COMMAND(#CMD) IF_ERROR(ERR)
GOTO LABEL(END)
* When error: print #CMD and Abort
ERR: DEF_LINE NAME(#PRT) FIELDS(#CMD) IDENTIFY(*NOID)
PRINT LINE(#PRT)
ABORT MSGTXT('ALGLIBL ended abnormally, check report')
ELSE
ABORT MSGTXT('Incorrect call of CL program ALGLIBL. No entries in Interface')
ENDIF
* Exit
END: RETURN
```

Finally you need a CL application called ALGLIBL to handle the setting of the librarylist (not part of this newsletter).

RRNO on Logicals

In CU3 (LANSA version 11.3) the setting for logical files getting an rrno field was intended behaviour. This was implemented due to requests from customers who suggested that this would improve performance.

So, essentially, we have turned on CREATE_RRNO_INDEXES=YES (in x_dbmenv.dat) for all databases in V11.0.

If required this can be reversed by setting CREATE_RRNO_INDEXES=NO in x_dbmenv.dat.

This feature is supposed to improve performance of SELECT/FETCH and DELETE/UPDATE WITH_KEY/WHERE, as building up the result set is faster.

However, additional indexes slow down INSERT, UPDATE, DELETE too, as indexes have to be updated.

Open Query File not in RDMLX

Using OPNQRYF is not supported in RDMLX objects. Although this has not been properly documented in the guides, this is not supported with RDMLX anymore.

This is logged as CCSID 125953 to modify the documentation where necessary to reflect this unavailability.

The option and in fact better solution, is to use the SELECT_SQL command as this has been improved and in fact now works better than the OPNQRYF option.

Instead of using the %XLATE option as in the SET example SET183A,
you can use one of these three functions: **UPPER**, **UCASE** or **TRANSLATE** function.

SELECT_SQL * FROM IKNDTA06/pslmst WHERE (UPPER(ADDRESS1) LIKE UPPER('%STREET%'))

or
SELECT_SQL * FROM IKNDTA06/pslmst WHERE (UCASE(ADDRESS1) LIKE UCASE('%STREET%'))

or
SELECT_SQL * FROM IKNDTA06/pslmst WHERE (TRANSLATE(ADDRESS1) LIKE TRANSLATE('%AVENUE%'))

This should achieve the same functionality as for the %XLATE with the OPNQRYF command.