

SILVERDEV

ADVANCED TECHNICALS

R P G I L E programming

Notice relative aux droits d'auteurs.

Les informations contenues dans ce document pourront faire l'objet de modifications sans préavis et ne sauraient en aucune manière engager **EXPERIA**. La fourniture du progiciel est régie par un octroi de licence ou un accord de confidentialité. Le progiciel ne peut être utilisé, copié ou reproduit sur quelque support que ce soit que conformément aux termes de cette licence ou de cet accord de confidentialité. L'acheteur ne peut effectuer des copies que dans le but de sauvegarde ou d'archivage.

Aucune partie du manuel et du progiciel ne peut être reproduite ou transmise par quelque moyen que ce soit, électronique ou mécanique, y compris par photocopie, enregistrement ou tout autre procédé de stockage, de traitement et de récupération d'informations, pour d'autres buts que l'usage personnel de l'acheteur sans permission expresse et écrite de la société **EXPERIA**.

IBM, AS/400, iSeries, System i, i5, Power I sont des marques déposées de International Business Machines Corporation. Windows est une marque déposée de Microsoft.

Tous les autres produits sont des marques déposées de leur société respective.

EXPERIA Europe
4, rue L.Beridot
Les jardins d'Epione
38500 VOIRON
FRANCE

Table des matières

Chapitre 1. Optimisation.....	6	Program Child2	39
Functions with response	6	Chapitre 8. Multiple window application	40
Field cache	6	with tabs	40
Window cache.....	8	Introduction	40
List type functions	8	Placing a component in a container	40
Function sdGetSfl	8	Sources.....	41
Re-using windows	9	PGM1 program.....	41
Compression.....	9	PGM2 program.....	42
Local events.....	10	Chapitre 9. Multi occurencies	44
sdAddNode	10	Introduction	44
Chapitre 2. Functions of the List category11		Form handle	44
Introduction	11	Events	44
For example:	11	Storing data	45
sdGetControls, sdGetComponents	12	Example	48
Chapitre 3. SFL: Copying data locally.....	13	Chapitre 10. Multi occurencies, sdsrvlst	49
sdGetSfl.....	13	service program	49
sdGetSflRow.....	14	Introduction	49
sdGetSflCol.....	14	Sdsrvlst service program.....	49
Chapitre 4. Dynamic creation of		srvchld service program	52
 components	16	Child program.....	54
Chapitre 5. Local events.....	19	Prototypes.....	56
Introduction	19	Exportation member	56
Language.....	19	Binding directory	57
Structures and key words	19	Chapitre 11. Dialog box functions	58
Editor	21	SdDialog.....	58
Interactions with server events.....	22	sdShowMessage	58
Windows	23	sdMsgDlg	59
Dynamic addition of local events.....	23	First solution:.....	59
Examples	23	Second solution:.....	59
Chapitre 6. Multi-window applications.....	29	sdSelectFile.....	61
Introduction	29	sdSelectDir.....	63
Windows organization	29	sdPrintDlg.....	63
Example of floating windows:.....	29	sdSelectColor	63
Stack.....	30	Chapitre 12. PC file category functions.....	65
Interaction between programmes.....	30	sdGetDir.....	65
Case 1 : On an action in screen1, you want to modify		SdDelFile.....	66
something in screen2.....	31	sdDownLoad	66
Case 2 : On an action in screen2, you want modify		SdForceDir	67
something on screen1.....	32	SdGetFileSize	68
Parameters	33	SdSaveToFile	68
Chapitre 7. MDI applications	35	SdSelectFile	69
Introduction	35	SdSetXferBar	69
For example:	36	SdUpload	69
Program MDIFORM.....	36	SdSelectDir	70
Program Child1	38		

SdFileExists	70
SdDirExist	71
sdCopyFile	71
SdGetRealFolder	71
SdRenameFile	73

Chapitre 13. Collections category functions 74

sdCollecClear	74
sdCollecAdd	74
sdCollecDelete	74
sdCollecIns	74

Chapitre 14. TStrings category functions .. 76

Introduction	76
Functions with response	76
Field cache	76
Window cache	78
List type functions	78
Function sdGetSfl	79
Re-using windows	79
Compression	80
Local events	80
sdAddNode	80
sdSIAdd	80
sdSIClear	81
sdSIDelete	81
sdSISet	81
sdSIInsert	82

Chapitre 15. Set category functions..... 83

sdSetSet	83
sdAddSet	84
sdDelSet	84
SdGetSet	85
SdUpdSet	86
SdIsInSet	86

Chapitre 16. Miscellaneous functions..... 89

SdHideMainForm	89
SdCursor	89
SdApplySet	90
sdEnd	90
sdInnerEnabled	90
sdInnerReadOnly	91
sdGetClientIp	91
SdGetEnvVar	91
sdMsgDebug	92
sdSetKeepAlive	92

SdBringApplicationToFront	93
SdFlashApplication	93

Chapitre 17. Tips94

"Position to" in the sub-files	94
Sub-files and F11	94
Wait window	97
Introduction:	97
Window	97
Code:	97
Caller program:	98
Activate/deactivate items of the OnPopup menu	99
Selection windows	99
Compilation control	101
Keyboard shortcuts	101
Modification of a sub-file column	102
Adjusting component size	102
Align	103
Anchors	103
Current user profile	104
Retrieval of the current profile in the programs	105
RPG	105
CLP	105
Triggering an event programmatically	105
Sending a message between windows	106
Opening a document on the server	106
Dynamic CSfl:	107
Sfl mouseover	107
Spare a call to sdGet	107

Chapitre 18. Screen translation 109

Langs property	109
String translation	109
PSVDDLO file	111
Changing the language during execution	112
Moving a screen	112
Translation menu	113
SdRtvMsg	113

Chapitre 19. Image category functions 115

sdSetImg	115
sdGetImg	117
sdLoadFromFile	118
sdAssignTo	119
sdAddImg	119

Chapitre 20. List of windows and events . 121

sdGetFormCount	121
sdGetFormHandle	121
sdGetFormName	121
sdGetFormLib	122

sdGetFormObj	122	CScreen component.....	143
sdGetEventCount.....	123	onResize event.	144
sdGetEventName.....	124		
sdGetEventProc.....	124	Chapitre 25. ADO.....	145
Chapitre 21. Colors.....	125	Introduction	145
Designer	125	CADODConnection	145
Modification during execution.....	125	CAdoQuery	146
Choice of colour	126	Chapitre 26. Exit points.....	148
Pre-defined colours.....	126	SVDCTRLCPL.....	148
Chapitre 22. Anchoring components	130	SVDEXCEPT.....	148
Introduction	130	Chapitre 27. Interaction with external	150
Basic example	130	programs.	150
Floating window	131	Launch a silverdev applicatoin from a pc	
Improvement.....	131	program.....	150
Chapitre 23. Drag and drop operations....	133	Launch a silverdev application from a 5250	
Introduction	133	application.....	151
DragMode	133	Launch a silverdev application from a html	
OnDragOver	134	page	154
OnDragDrop	135	Launch a silverdev application from a silverdev	
Examples	135	application.....	154
Drag and drop of a ClistBox to a ClistBox.....	135	Launch an external program from a silverdev	
Drag and drop of a ClistView to a ClistView.....	136	application.....	155
DragCursor and CCustomCursor	137	Launch a web page from a silverdev	
OnDragOverEx, OnDragDropEx	138	application.....	157
Chapitre 24. Screen size adaptation	140	Insert a 5250 application in a silverdev	
Introduction	140	program.....	157
Size changeable by user.....	140	Insert a web page in a silverdev program.....	158
Align, anchors	140	Chapitre 28. sdsrvjson service program ..	159
Anchors.....	141	Introduction :	159
Nested containers	141	Functions.....	159
Splitter component	142	Chapitre 29. WebBrowser	160
Gridpanel, flowPanel, relativePanel	143	Chapitre 30. Web services	161

Chapitre 1. Optimisation

In all client-server applications, response time is essential.

This section presents the means used by SilverDev to optimise response times and advises you on how to improve your applications.

Functions with response

Calls to functions requiring a response are slower since they need to go to and from the network.

Functions that return no value are grouped in a buffer.

This is the case of the sdSet, sdSetCell etc. functions for example.

This buffer is sent to the client at the end of the event manager or during a function call that returns a value (e.g. sdGet) or during a the sdApplySet function call.

For example:

On a test machine, we measured:

1000 sdGet calls: approx. 3 seconds (3000 milliseconds).

1000 sdset calls: approx. 15 milliseconds

Field cache

To improve the performance of sdGet type functions, for each event, certain property values are sent to the server.

These values are placed in cache.

The sdGet type functions start by searching this cache.

This mechanism is transparent for the developer, but it enables a significant improvement in performance.

Only the properties with the highest probability of being queried are placed in cache.

For example, only the text property of the CEdit component is placed in cache.

In the debugger:

Send event

Event: _2.BtnOK.OnClick

2009/10/12 15:01:59:218

Cache transmission

_2.TITLE.text=Les robots

_2.PRICE.value=4

_2.STOCK.value=0

_2.ORDER.value=Y

_2.Themes.keySelected=2

_2.Themes.text=Fantastique

_2.IDAUT.value=17

_2.NOMAUT.text=Dostoïevski

_2.NOMEDI.text='ai Lu

_2.IDEDI.value=4

_2.DATEPUB.value=20040513

_2.DATEPUB.DateIso=2004-05-13

2009/10/12 15:01:59:234

Get execution

_2.DATEPUB.Validate

Result: True

2009/10/12 15:01:59:343

Script execution

function Main() {

_2.ModalResult=1;

}

C	Eval	TITLE= sdGet(F1:'TITLE':'Text')
C	if	sdGetBool(F1:'DATEPARU':'ValidDate')=*off
C	eval	Message =Message + 'Invalid date'
C		+ X'0D'
C	endif	
C	Eval	PRICE=sdGetNum(F1:'PRICE':'Value')
C	Eval	STOCK=sdGetInt(F1:'STOCK':'Value')
C	Eval	ORDER=sdGet(F1:'ORDER':
C		'Value')
C	Eval	IDTHEME=sdGetNum(F1:'Themes':
C		'KEYSELECTED')
C	Eval	IDAUT=sdGetInt(F1:'IDAUT':'Value')
C	Eval	IDEDI=sdGetInt(F1:'IDEDI':'Value')
C	eval	DATEPUB=sdGetDate(F1:'DATEPUB')
C	callp	sdSetInt(F1:'*FORM':'ModalResult':Mrok)

Window cache

SilverDev windows have to be transmitted from the server to the client. If they contain images, the transfer can take rather a long time. The windows created are therefore saved on the client's disk. When a window is created, the client and server start by finding out if the window already exists on the client workstation. The windows are stored in the /cache directory.

The cache file name is the window's code md5. When a window is modified, and therefore placed in cache once again, the old cache of this window is deleted. The correspondence between the origin of a window and its cache file is saved in the register database.

List type functions

As we saw earlier, sdGet type functions take a non-negligible time.

Re-reading all the items of a treeview, memo, listbox, checklistbox, etc. type component can therefore take rather a long time.

The List category functions enable a considerable amount of time to be saved. The principle is to pass on all the information of interest about a component and copy it into the memory to be able to query it locally. For more information, see (Chapitre 1)

Function sdGetSfl

For display of a directory in MyDesk, various pieces of data are sent to MyDesk. The information is about the application and possibly an icon. Text type information only represents a few bytes. The icons are much larger, representing 90% of the data transferred.

A mechanism based on an md5 hashing code is used to send the icons once only.

For each application, there is a file with a .app extension on the server and possibly a .ico file too.

The MD5 code of the icon is stored in the .app file.

This file is sent first.

MyDesk checks to see if the icon exists on the disk.

If the icon already exists, it is used. If not, it is downloaded then saved on the disk under the name of the icon's hashing code.

Note:

These icons are also used for the creation of shortcuts on the desktop.

Re-using windows

When you create a window, resources are used on the client side to create the components of the window.

When the user closes a window, it is hidden but not destroyed.

This means that the components of the window and the window itself use memory on the PC side.

The window will be destroyed when the program is closed (on the PC).

If you want to free the window when it is closed, add the following code to the window's onclose event:

```
p ONCLOSE...
p          B
d          PI
d PevtInf          *   const options(*nopass)
C          callp    sdFree (F1:'*FORM')
```

Whether the window is destroyed or not, remember to test that a window does not already exist before creating it:

```
C          If      F1 <> 0
C          eval    F1 = sdcreateForm(F1REF)
C          endif
```

If it already exists, you will create several identical windows that use resources unnecessarily.

Compression

During the first data exchanges, exchange time is measured.

Depending on the time calculated, the server determines whether or not the client is distant or on the local network.

If the client is distant, the data are compressed before transmission.

Local events

All processing can be carried out in the server events, but it is quicker to process events locally rather than on the server. When an event occurs frequently and processing does not require objects on the server, prefer local events.

sdAddNode

The sdAddNode2 function should be preferred to the sdAddNode function as it is much faster and easier to use.

Chapitre 2. Functions of the List category

Introduction

Certain components have a list of values and it can take a long time to browse all these values using traditional methods. Querying the value of a property takes approximately 10ms.

The sdGetList function enables querying of these components and storage of the data on the server. This function returns a pointer that must then be used in the sdGetListLabel, sdGetListKey, sdGetListSelected, sdGetListState and sdGetListCount functions.

List type functions enable querying of components such as CCheckListBox, CListBox, CComboBox, CTreeView, CListView or CMemo.

It is much faster to browse all the values of these components in this way than with the traditional sdGet and sdSGet functions.

For example:

```
DList      s      *
DLabel     s      256   Varying
DKey       s      256   Varying
DSelected  s      N
DState     s      5u 0
Di         s      10u 0 inz (0)
C          eval    List=sdGetList (Fl: 'CheckListBox1')
C          if      List <> *NULL
C          DOW      I < sdgetListCount(list)
C          eval     Label=sdGetListLabel (List:i)
C          eval     Key=sdGetListKey (List:i)
C          eval     Selected=sdGetListSelected (List:i)
C          eval     State=sdGetListState (List:i)
C          enddo
C          callp    sdFreeList (List)
C          endif
```

Use the sdFreeList function to free the memory allocated by the sdGetList function.

sdGetControls, sdGetComponents

The sdGetControls and sdGetComponents functions also return a pointer to an object of the same type as that returned by sdGetList.

The sdGetControls function returns the list of all the controls contained in the control passed as a parameter.

The sdGetComponents function returns the list of all the components whose parameter component is responsible for destruction.

For a window created with Designer, all the components on the window are in this list.

The pointer can therefore be used as a parameter in functions sdGetListLabel and sdGetListKey.

sdGetListLabel returns the name of the component and sdGetListKey returns the type of component.

For example:

```
DList      s      *
DControlName s      256    Varying
DTypeControl s      256    Varying
Di         s      10u 0 inz(0)
C          eval    List=sdGetControls(F1: '*FORM')
C          if      List <> *NULL
C          DOW      I < sdgetListCount(list)
C          eval     ControlName=sdGetListLabel(List:i)
C          eval     TypeControl=sdGetListKey(List:i)
C          eval     i=i+1
C          enddo
C          callp    sdFreeList(List)
C          endif
```

Chapitre 3. SFL: Copying data locally

sdGetSfl

The sdGetCell, sdGetCellNum, etc. functions enable querying of the content of the cells in an CSFL component.
Querying all the cells of an sfl comprising several columns and lines can take rather a long time.

To optimise response time, use the sdGetSfl function.
This function copies the data of the CSfl component locally.
It is then possible to access the copied data using the sdGetSflStr, sdGetSflNum, sdGetSflDate, sdGetSflTime, sdGetSflNull, sdGetSflModified, sdGetSflLines, sdGetSflColumns and sdGetSflColName functions.

The Filter parameter enables selected rows to be chosen.

The values possible for Filter are:

Value	Constant defined in H,SILVERDEV	Description
0	ALL_ROWS	All rows
1	DIRTY_ROWS	Only modified rows
2	SELECTED_ROWS	Only selected rows

The sdGetSfl function allocates memory. To free this memory, use the sdFreeSfl function.

For example:

```
*/EVENT Button2_OnClick
Dsfl          s          *
D i           s          10i 0
D MyDate      s          d
D MyNum       s          10 0
D MyStr       s          100 Varying
C             eval      sfl=sdgetsfl(f1:'sfl1')
```

```

C      if      Sfl <> *NULL
C      for      i=1 to sdGetSflLines(Sfl)
C      if      sdGetSflModified(Sfl:i)
C      if      not sdGetSflNull(Sfl:'MyDate':i)
C      eval      MyDate=sdgetSflDate(sfl:'MyDate':i)
C      endif
C      eval      MyNum=sdgetSflNum(sfl:'MyNum':i)
C      eval      MyStr=sdgetSflStr(sfl:'MyStr':i)
C      endif
C      endfor
C      callp      sdFreeSfl(Sfl)

```

sdGetSflRow

The sdGetSfl function returns all the data from the client machine to the server. If you only need the data from a single line, you can use the sdGetSflRow function instead of sdGetSfl.

The sdGetSflRow function returns a pointer that you can use in the sdGetSflStr, sdGetSflNum, etc. functions.

The last parameter of these functions must then be 1.

For example:

```

*/EVENT Button2_OnClick
Dsfl      s      *
D Row      s      10i 0
D MyDate    s      d
D MyNum      s      10 0
D MyStr      s      100 Varying
C      eval      Row = sdGetInt(Fl:'sfl1':'RowSelected')
C      if      Row > 0
C      eval      sfl=sdgetsflRow(fl:'sfl1':Row)
C      if      Sfl <> *NULL
C      eval      MyNum=sdgetSflNum(sfl:'MyNum':1)
C      eval      MyStr=sdgetSflStr(sfl:'MyStr':1)
C      endif
C      endif
C      callp      sdFreeSfl(Sfl)

```

sdGetSflCol

The sdGetSflCol function enables all the values of one column of a CSFL component to be retrieved.

The values of the line are copied into a local structure.
This structure is the same as that obtained via the sdGetSfl function.

Example:

```
Dsfl      s      *
D Row     s      10i 0
D MyDate  s      d
D MyNum   s      10 0
D MyStr   s      100 Varying
C         eval    sfl=sdgetsflCol (f1:'sfl1':'MyColumn')
C         if      Sfl <> *NULL

C         for     i=1 to sdGetSflLines(Sfl)
C         eval    MyNum=sdgetSflNum(sfl:'MyNum':i)
C         eval    MyStr=sdgetSflStr(sfl:'MyStr':i)
C         endfor
C         callp    sdFreeSfl(Sfl)
C         endif
```

Chapitre 4. Dynamic creation of components

Dynamic creation of components is done with the `sdCreate` function.

For create of a form with a lot of comonents, you can improve the speed, by using the `sdLoadComponent` function.

This function uses a xml file that you create.

Schema of the file is :

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Control">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Control" type="Control" />
        <xs:element name="Data" type="xs:string" />
        <xs:simpleType>
          </xs:simpleType>
        </xs:element >
        <xs:element name="Event" >
          <xs:simpleType>
            <xs:attribute name="Name" type="xs:string"/>
          </xs:simpleType>
        </xs:element >
        <xs:element name="Script" >
          <xs:simpleType>
            <xs:attribute name="Language" type="xs:string"/>
            <xs:restriction base="xs:string">
              <xs:enumeration value="DelphiScript"/>
              <xs:enumeration value="JavaScript"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element >
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Root node must be 'Control'

Include nodes 'Control' contains include controls.

'Data' node contains properties of the component

'Event' nodes contains component events.

'Script' node contains script to execute

Service program SDSRVCMP is usefull to create the file.

Example :


```
path = '/home/user1/' + getQuallJobFileName+'.xml';
fd = openfile(path);
writeInFile(fd:'<Control Type="CPanel">');
writeInFile(fd:'<Data>');
writeInFile(fd:'object Panel1:CPanel' +CRLF);
writeInFile(fd:'caption = ' + encodeString('Panel1') +CRLF);
writeInFile(fd:'end' +CRLF);
writeInFile(fd:'</Data>');
writeInFile(fd:'<Control Type="CButton">');
writeInFile(fd:'<Data>');
writeInFile(fd:'object Button3:CButton' +CRLF);
writeInFile(fd:'caption = ' + encodeString('Button3') +CRLF);
writeInFile(fd:'end' +CRLF);
writeInFile(fd:'</Data>');
writeInFile(fd:'<Event Name="onClick"/>');
writeInFile(fd:'<Script Language ="JavaScript">');
writeInFile(fd:'self.button3.caption = "Hello";');
writeInFile(fd:'</Script>');
writeInFile(fd:'</Control>');
writeInFile(fd:'</Control>');
closeFile(fd);
sdLoadComponent (F1:'*FORM': '*FORM':path);
deleteFile(fd);
```

Chapitre 5. Local events

Introduction

A SilverDev program can be written in full without using local events. This function was actually only added at the very late stages of the software's development.

The use of local events is an advanced function.

However, this function makes SilverDev more powerful and enables optimisation of response time.

It is important to understand when to use a local event and when to use a server event.

When an event manager does not need to query a server object (mainly the database), it may be advantageous to use a local event. Processing will be quicker because there is no exchange between the server and the client.

Some events occur very often and it is more logical to avoid using server events for such cases.

Language

Some events have modifiable parameters.

We therefore had to use a language that enabled function parameters to be passed by reference. JavaScript does not enable this, therefore we chose to use Pascal.

Structures and key words

Field declarations

```
var  
i:integer;  
chaine :string;  
Reel :Float ;
```

```
begin  
...
```

Operators

Quality	Different	Strictly greater than	Strictly less than	Greater than or equal to	Less than or equal to
=	<>	>	<	>=	<=

Allocation
:=

Logic Or	Logic And
Or	And

Conditions

```
if (var1=0) or(var2 <> 5)then
begin
...
end
else
...
end;
```

Loops

```
for i:=0 to 10 do
begin
...
end;
```

```
while(i <10)do
begin
...
end ;
```

```
repeat
...
until (i >2);
```

```
break : quit the loop
continue : move on to the next iteration
```

Comments:

```
//comment on one line
(*comment on several lines*)
{comments on several lines}
```

Editor

Designer enables entry of the event code.

To enter the code of a local event, double-click on the line to display a code editor.

If the code is associated with a local event, the name of this event will be shown in bold, in blue.

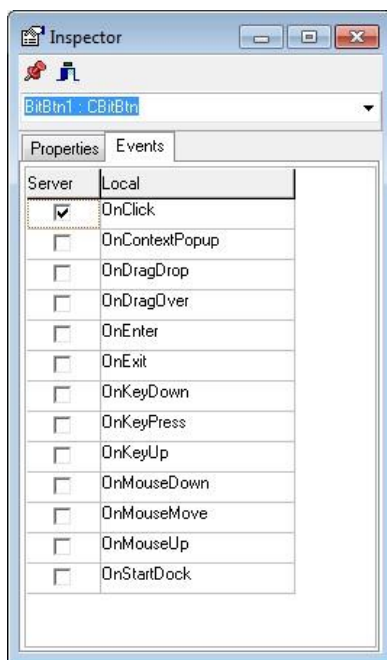


Figure 1

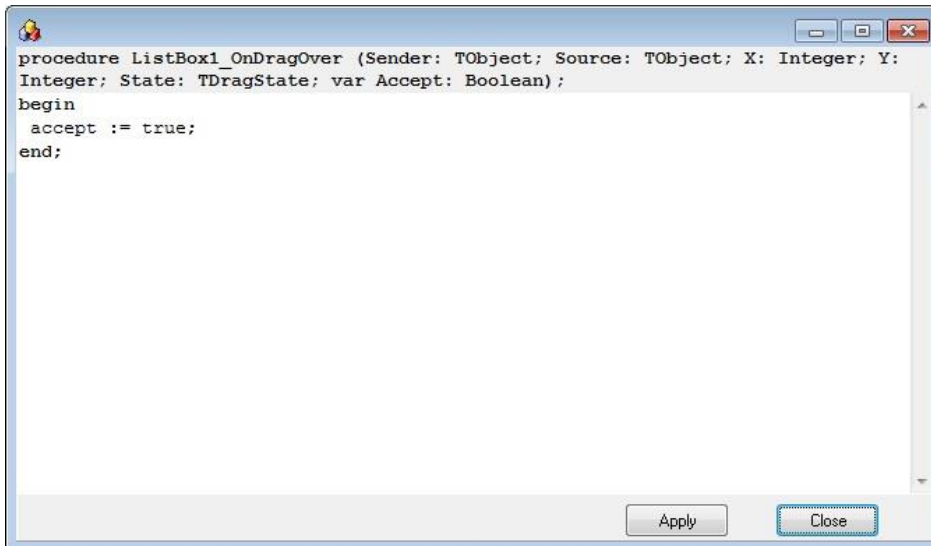


Figure 2

The function prototype is not modifiable.

If the editor background is white, this means you have clicked on "Apply".

If the editor background is yellow, this means you have changed the code, but not yet clicked on "Apply".

To delete a local event, delete all the code and click "Apply".

Interactions with server events

You can use local processing or server processing for the same event .

It is also possible to choose to process the event both locally and on the server. In this case, you can cancel server processing from the local processing by modifying the Boolean field SendEvt.

The SendEvt field is an overall field.

Do not declare a local field to the function with this name as it would hide the overall field SendEvt.

Note: If you write a local event manager and a server event manager, always assign a value to the SendEvt field in the local event manager.

Note: If you do not ask for event notification to the server (tick box in Designer), setting the SendEvt field to true will have no effect.

Windows

In the function bodies, the objects must be preceded with the name of the window containing the object. The identifier to be used for the windows is "This". For the other objects, the identifier used should be the Name property.

Dynamic addition of local events

If you want to add events upon execution for components created dynamically for example, use the sdAddLocalEvent function.

Prototype:

```
d sdAddLocalEvent...
D                               pr
d  Form                        5u 0 const
d  Component                   30   varying value
d  Event                       30   varying value
d  Code                        1000  varying value
```

Examples

Example 1:

Description:

The ondragover event is used to determine whether or not the user can move an elt with drag and drop

Code

```
procedure ListBox1_OnDragOver (Sender: TObject; Source: TObject; X:
Integer; Y: Integer; State: TDragState; var Accept: Boolean);
begin
```

```
Accept := (Source = This.listbox2);  
end;
```

Example 2:**Description:**

The onchange event of a CEdit is sent when the content exceeds 10 characters (useful for barcode reader entry).

Code

```
procedure Edit1_OnChange (Sender: TObject);  
begin  
    SendEvt := (length(This.edit1.text) >= 10);  
end;
```

Example 3:**Description:**

The onKeyPress event is sent to the server only if the Enter key is used.

Code

```
procedure Edit1_OnKeyPress (Sender: TObject; var Key: Char);  
begin  
    SendEvt := (Key = #13); // touche enter  
end;
```

Example 4:**Description:**

The entry zone changes colour.

Code

```
procedure Edit1_OnEnter (Sender: TObject);  
begin
```



```
This.edit1.color := clred;  
end;
```

```
procedure Edit1_OnExit (Sender: TObject);  
begin  
    This.edit1.color := clwhite;  
end;
```

Example 5:**Description:**

Check zone entry in a form.

Code

```
procedure Button1_OnClick (Sender: TObject);  
var  
    msg:string;  
begin  
    msg:='';  
    if (This.edit1.text='') then  
        begin  
            msg:=msg+'edit1 is required'+#13#10;  
        end;  
    if (This.edit2.text='') then  
        begin  
            msg:=msg+'edit2 is required'+#13#10;  
        end;  
    if (msg='') then  
        begin  
            SendEvt := true;  
        end  
    else  
        begin  
            messageDlg(msg,mtinformation,mbokcancel,0);  
        end;  
end;
```

Example 6

Description:

A tick box is entered when the user enters an edit zone.

Code

```
procedure NBOPER_OnEnter (Sender: TObject);  
begin  
  This.chk4.checked:=true;  
end;
```

Example 7**Description:**

A CChart component makes a rotation according to the movement of a trackbar.

Code

```
procedure TrackBar1_OnChange (Sender: TObject);  
begin  
  This.series1.RotationAngle:= This.TrackBar1.Position;  
end;
```

Example 8**Description:**

Continuous rotation of a CChart component.

Code

```
procedure Timer1_OnTimer (Sender: TObject);  
begin  
  if(This.Series1.RotationAngle =359)then  
  begin  
    This.Series1.RotationAngle :=0;  
  end  
  else  
  begin  
    This.Series1.RotationAngle := This.Series1.RotationAngle+1;  
  end;  
end;
```

Example 9

Erreur ! Source du renvoi introuvable.

Example 10

```
procedure TreeView1_OnCheckNode (Sender: TObject; Node: TNode; Value:
Integer; var Allow: Boolean);
begin
    // The user is allowed to check node if the node has no child
    Allow := node.count = 0;
end;
```

Example 11

“Position to” in the sub-files

Example 12

Activate/deactivate items of the OnPopup menu

Example 13

```
if (MessageDlg('Confirmation ? ', mtConfirmation, mkset (mbYes,
mbNo), 0) = MrYes) then
begin
    ...
end;
```

The following values are possible for the buttons:

mbYes, mbNo, mbOK, mbCancel, mbAbort, mbRetry, mbIgnore, mbAll,
mbNoToAll, mbYesToAll, mbHelp

The following values are possible for the return:

mrYes, mrNo, mrNone, mrOk, mrCancel, mrAbort, mrRetry, mrIgnore, mrAll,
mrNoToAll, mrYesToAll

Example 14

For sfl mouseover, see **Sfl**

Example 15

Scroll sfl upon dragover:

```
procedure SFL1_OnDragOver (Sender: TObject; Source: TObject; X: Integer;
Y: Integer; State: TDragState; var Accept: Boolean);
begin
  if (( X + 10) > This.SFL1.ClientWidth) then
  begin
    SendMessage(This.SFL1.Handle, 276, 1, 0)
  end
  else if( x < 10) then
  begin
    SendMessage(This.SFL1.Handle, 276, 0, 0)
  end;
  if (( Y + 10) > This.SFL1.ClientHeight) then
  begin
    SendMessage(This.SFL1.Handle, 277, 1, 0)
  end
  else if( y < 10) then
  begin
    SendMessage(This.SFL1.Handle, 277, 0, 0)
  end;
end;
```

Chapitre 6. Multi-window applications

Introduction

When a Silverdev program calls another Silverdev program, two windows are displayed on the screen.

The number of windows displayed is not limited.

Unlike a 5250 application, the user can access several windows at once.

The first window displayed is the main window.

When the main window is closed by the user, the process on the client part and the job on the server are terminated.

Windows organization

The default behavior is that each window is independent.

In this case, the windows are said to be floating.

Example of floating windows:

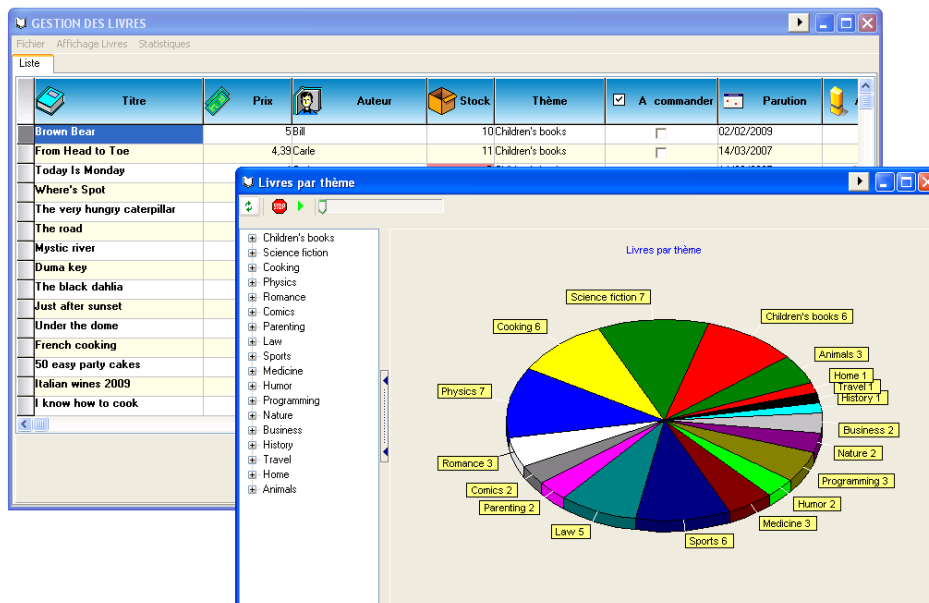


Figure 3

Stack

When the first program is launched, it calls the `sdStart` function that enters in a loop to listen for server events.

When the second program is called, this program is loaded on memory, the the program goes out of the stack. It is still loaded on memory, and event handlers are called when corresponding events occur.

Interaction between programmes

User can access to several screens at the same time. This brings a new situation compare to 5250 programming. You may want that an action on a screen leads to a modification on another screen.

If you follow the rule one screen/on program, this means that you need to execute in a program a function that is in another program.

Pgm1 and pgm2 are two programs, with screens screen1 and screen2, program pgm1 calls program pgm2.

Case 1 : On an action in screen1, you want to modify something in screen2.

Program pgm2 must have a procedure pointeur type parameter and assign this pointer with the address of the function of pgm2 that must be executed in pgm1.

Pgm 1 :

```

D ptrfnc      s          *   procptr

D fnc         pr          extproc(ptrfnc)
...

C             call      'PGM2'
C             parm      ptrfnc
...

/Free
  Callp fnc();
/end-free

```

Pgm 2 :

```

D ptrfnc      s          *   procptr

D fnc         pr
D parm1       10u 0
D parm2       10

C  *entry     plist
C             parm      ptrfnc

/Free
  ptrfnc = %paddr('FNC');
/end-free

P fnc         B
D             pi
...

P fnc         E

```

Case 2 : On an action in screen2, you want modify something on screen1.

Program pgm1 must pass the adress of the function to be executed as a parameter in pgm2.

Pgm 1 :

```

D ptrfnc      s          *   procptr

D fnc         pr          extproc(ptrfnc)

/Free
  ptrfnc = %paddr('FNC');
/end-free

C             call      'PGM2'
C             parm      ptrfnc

P fnc         B
D             pi
D parm1       10u 0
D parm2       10
...
P fnc         E

```

Pgm 2 :

```

D ptrfnc      s          *   procptr

D fnc         pr          extproc(ptrfnc)

C *entry      plist
C             parm      ptrfnc

/Free
  Callp fnc();
/end-free

```

Note 1 : The argument passed to the built in function %paddr must be in uppercase.

This is due to the fact that the rpg compiler change the source to uppercase.

Note 2 : The two previous solutions can be mixed.

Note 3 : You may think it would be a good idea to pass as a parameter between programs only screen handles (F1) and to code component names of screen1 in pgm2. This is a bad idea, because pgm1 and pgm become strongly bound.

Parameters

The extern function can have some parameters. You need to define correctly the prototype.

Case 1 becomes :

Pgm 1 :

```
D ptrfnc      s          *   procptr

D fnc        pr          extproc(ptrfnc)
D parm1      10u 0
D parm2      10
...

C            call      'PGM2'
C            parm      ptrfnc
...

/Free
Callp fnc(arg1:arg2) ;
/end-free
```

Pgm 2 :

```
D ptrfnc      s          *   procptr

D fnc        pr
D parm1      10u 0
D parm2      10

C *entry     plist
C            parm      ptrfnc

/Free
```

```
ptrfnc = %paddr('FNC');  
/end-free
```

```
P fnc          B  
D              pi  
D parm1        10u 0  
D parm2        10  
...  
P fnc          E
```

Warning, rpg does not have typed pointers, so the compiler will not detect an error if the address assigned to ptrfnc is the address of a function with different parameters.

An error will occur at runtime.

Chapitre 7. MDI applications

Introduction

An MDI application has a parent window and child windows.

The child windows cannot leave the framework of the mother window and the mother window menu is always accessible.

Functions are available to arrange to the windows in a mosaic.

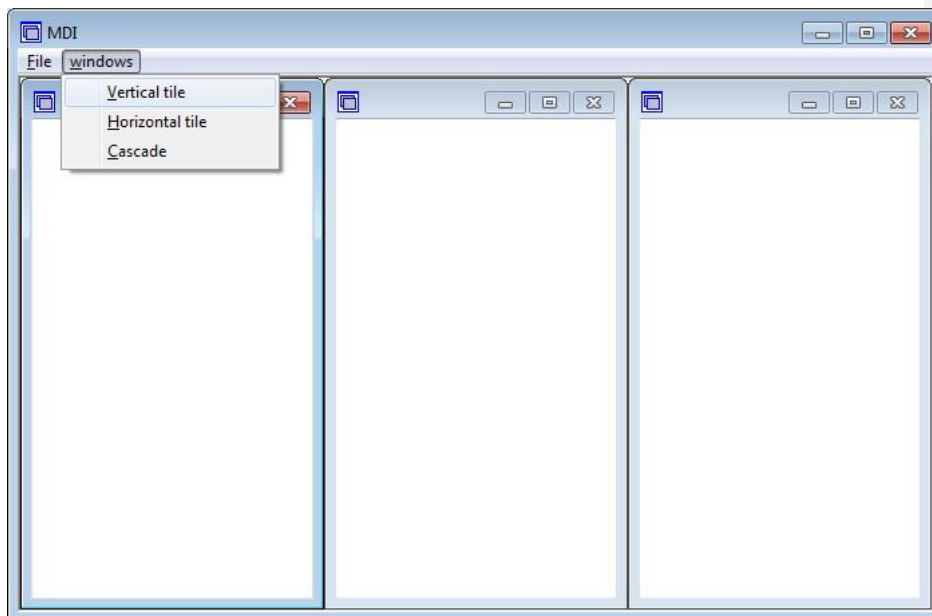


Figure 4

To make a window a parent, change its Formstyle property to fsMDIFORM.

To make a window a child, change its Formstyle property to fsMDICHILD.

A child window cannot exist if there is no parent window.

A child window cannot be invisible. If the user clicks on the cross of a child window, the window is minimised instead of closed. To actually close it, the window must be freed in its onclose event.

For example:

The following example is composed of 3 programs each containing one window.
The two programs, Child1 and Child2, cannot be called alone, since their window is a fsMdiChild type window.

Program MDIFORM

```
h dftactgrp(*no)
h bnkdir('SILVERDEV')
/copy h,silverdev
*
*
d F1          S          5u 0
d F1REF       S          21A  INZ('LIBL/MDIFORM')

d Init        pr

d MnuChild1_OnClick...
d              pr
d  PevtInf          *  const options(*nopass)

d MnuChild2_OnClick...
d              pr
d  PevtInf          *  const options(*nopass)

d MnuVert_OnClick...
d              pr
d  PevtInf          *  const options(*nopass)

d MnuHor_OnClick...
d              pr
d  PevtInf          *  const options(*nopass)

d MnuCascade_OnClick...
d              pr
d  PevtInf          *  const options(*nopass)

c              eval      *inRT = *on
c              callp     sdStart(%paddr('INIT'))

p Init        B
d              PI
c              If        F1 = 0
c              eval      F1 = sdcreateForm(F1REF)
c              callp     sdSetCallBack(F1
```

```

c          : 'MnuChild1.OnClick'
c          : %paddr (
c          'MNUCHILD1_ONCLICK'))
c          callp sdSetCallBack(F1
c          : 'MnuChild2.OnClick'
c          : %paddr (
c          'MNUCHILD2_ONCLICK'))
c          callp sdSetCallBack(F1
c          : 'MnuVert.OnClick'
c          : %paddr (
c          'MNUVERT_ONCLICK'))
c          callp sdSetCallBack(F1
c          : 'MnuHor.OnClick'
c          : %paddr (
c          'MNUHOR_ONCLICK'))
c          callp sdSetCallBack(F1
c          : 'MnuCascade.OnClick'
c          : %paddr (
c          'MNUCASCADE_ONCLICK'))
C          Endif
c          callp sdShow(F1)
p          E
pMnuChild1_OnClick...
p          B
d          PI
d PevtInf          * const options(*nopass)
C          call    'CHILD1'
p          E

pMnuChild2_OnClick...
p          B
d          PI
d PevtInf          * const options(*nopass)
C          call    'CHILD2'
p          E
pMnuVert_OnClick...
p          B
d          PI
d PevtInf          * const options(*nopass)
C          callp   sdSet(F1: '*FORM'; 'TileMode': 'tbVertical')
C          callp   sdTile(F1)
p          E
pMnuHor_OnClick...
p          B
d          PI
d PevtInf          * const options(*nopass)
C          callp   sdSet(F1: '*FORM'; 'TileMode': 'tbHorizontal')
C          callp   sdTile(F1)
p          E

```

```

pMnuCascade_OnClick...
p          B
d          PI
d PevtInf          * const options(*nopass)
C          callp    sdCascade(F1)
p          E

```

Program Child1

```

h dftactgrp(*no)
h ACTGRP('SILVERDEV')
h bnmdir('SILVERDEV')
/copy h,silverdev
d F1          S          5u 0
d FIREF       S          21A INZ('*LIBL/CHILD1')

d OnClose     pr
d PevtInf          * const options(*nopass)
c
c          eval      *inRT = *on
C          If        F1 = 0
c          eval      F1 = sdcreateForm(F1REF)
c          callp     sdSetCallBack(F1
c                  : 'OnClose'
c                  : %paddr(
c                  'ONCLOSE'))
C          endif
C          Callp     sdShow(F1)
pOnClose      B
d          PI
D Parameters  ds          based(pevtinf) D Win
5u 0
D Evt          48a
,* Variable
D Action      10i 0
,* 0 : ne rien faire
,* 1 : Cacher la fenetre
,* 2 : Liberer la fenetre
,* 3 : Minimiser la fenetre
,*d PevtInf          * const options(*nopass)
c          eval      action = 2
c          callp     sdFreeForm(Win)
p          E

```

Program Child2

Same as Child1, replace

d F1REF	S	21A	INZ ('*LIBL/CHILD1')
---------	---	-----	----------------------

with

d F1REF	S	21A	INZ ('*LIBL/CHILD2')
---------	---	-----	----------------------

Chapitre 8. Multiple window application with tabs

Introduction

An alternative solution to an mdi application is to place the windows in tab sheets, as shown below:

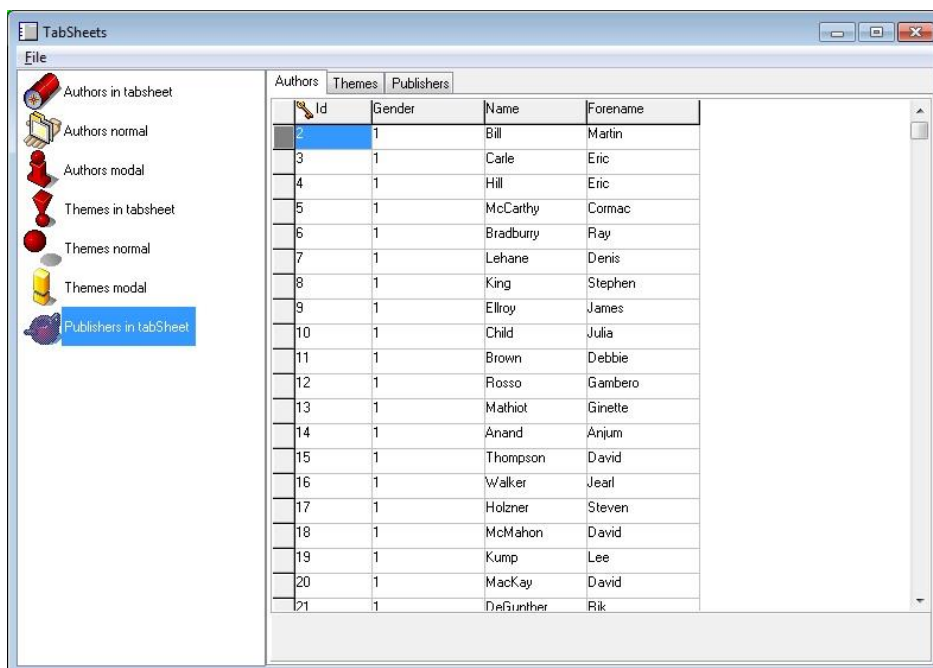


Figure 5

Placing a component in a container

To place a component in another component upon execution, use the `sdSetCtrl` function and the parent property.

Example 1:

To place a component in a panel:


```
c          callp      sdSetCtrl(F1: 'ListBox1': 'parent': F1: 'Panel1')
```

If you want the component to take up the whole panel:

```
C          callp      sdSet(F1: '*FORM': 'align': 'alclient')
```

Example 2:

To place a window (F1) in a panel of another window F2:

```
c          callp      sdSetCtrl(F1: '*FORM': 'parent': F2: 'Panel1')
```

Example 3:

Creating a tab sheet in window F1 and inserting a window F2 in the tab sheet:

```
c          callp      sdcreate(F1: 'TTabSheet': 'Tab1')
c          callp      sdSetCtrl(F1: 'Tab1': 'pagecontrol':
c                      F1: 'pagecontrol1')
c          callp      sdSetCtrl(F2: '*FORM': 'parent': F1: 'Tab1')
C          callp      sdSet(F2: '*FORM': 'BorderStyle': 'bsNone')
C          callp      sdSet(F2: '*FORM': 'align': 'alclient')
```

Sources

The principle of the method is shown below:

PGM1 program

```
*/EVENT ListView1_OnDbClick
* -----*
* Description :                               *
* -----*
D Parameters      ds          based(pevtnf)
D Win              5u 0
D Evt              48a
*
D pgm              s          20      inz (*LIBL/PGM2')
c                  call      pgm
C                  parm      F1
*/EVENT MnuClose_OnClick
```

```

* -----*
* Description : *
* -----*
Parameters      ds              based(pevtnf)
Win              5u 0
Evt              48a
*
nameTab          s              30    varying
DnumWin          s              5u 0
C                eval          NumWin = sdGetInt(F1:'Pagecontrol1':
C                'ActivePage.Tag')
C                callp         sdClose(numWin)

```

PGM2 program

```

*/BLOCK RPGSPCID
* ----- RPGSPCID : Data descriptions (D Spec.)
D tab            s              30    varying
DFParent         s              5u 0

*/BLOCK RPGPARM
* ----- RPGPARM : Program parameters
C  *entry        plist
C                parm          FParent

*/BLOCK RPGINITEND
* ----- RPGINITEND : End of Initialization procedure
c                eval          tab = 'Tab'+%char(F1)
c                callp         sdcreate(FParent:'TTabSheet':tab)
C                callp         sdSetInt(FParent:Tab:'Tag':F1)
c                callp         sdSetCtrl(FParent:tab:'pagecontrol':
c                                fParent:'pagecontrol1')
c                callp         sdSetCtrl(F1: '*FORM': 'parent': FParent:
c                                Tab)
C                callp         sdSet(F1: '*FORM': 'BorderStyle': 'bsNone')
C                callp         sdSet(F1: '*FORM': 'align': 'alclient')
*/BLOCK RPGBEFORESHOW
* ----- RPGBEFORESHOW: Before show(F1)
C                callp         sdSetBool(Fparent:Tab:'TabVisible':*on)
C                callp         sdSetCtrl(Fparent:'pageControl1':
C                'ActivePage':Fparent:Tab)

*/EVENT OnClose
* -----*
* Description : *
* -----*
D Parameters      ds              based(pevtnf)
D Win              5u 0

```

EXPERIA Europe

SilverDev

```
D Evt                                48a
* Can be changed
D Action                            10i 0
* 0 : Do nothing
* 1 : Hide the window
* 2 : Free the window instance
* 3 : Minimize the window
*
C                                callp    sdSetBool(FParent:Tab:'TabVisible':*off)
```

Comment: A similar, slightly more complex program can be found in the demo programs (see SDDMTAB1, SDDMTAB2, SDDMTAB3, SDDMTAB4 sources).

Chapitre 9. Multi occurencies

Introduction

This chapter explains how to create an application with a form having many occurencies. This means that for each call of a program, a new form is created. The same program handles several forms.

The forms may be floating or organized in mdi windows or in tab sheets as indicated in the two previous chapters

Form handle

Silverdev generates the following code:

```
/Free
  If F1 = 0;
    F1 = sdCreateForm(F1REF);
/End-Free
```

A form is created only at the first call.

We are going to add in the afterShow moment the following code :

```
/BLOCK RPGAFTERSHOW
// ----- RPGAFTERSHOW : After show(F1)

C          eval      F1 = 0
```

Thus, for each call , F1 will have the value "zero", and the sdCreateForm function will be called.

Events

In the previous chapter, we have seen thate the value of the F1 variable is not stored.

Thus you must not use the F1 variable in events.

Each event has some parameters. These parameters depend on the type of the event, but two parameters are always are always passed in an event.

It's the parameters Win and evt.

```

~/EVENT OnClose
, * -----*
, * Description :
, * -----*
D Parameters      ds          based(pevtinf)
D Win              5u 0
D Evt              48a

```

Always use the win parameter in the events for a multi form occurrences kind of program.

Storing data

Let's say we need to store some values for each form. A client number, a path on the ifs etc..

Since the form is instanced several times, we have to store these values as many times as there are any forms. We also need to be able to retrieve these values from the form handler, the win parameter given in the event.

We can imagine a lot of implementations. a static array (the number of occurrences will be limited), a bdd file, a dynamic array, a linked list..

The implementation that we are going to show you here is the one with a dynamic array.

Let's start by declaring the variables :

```

D NbChildren      s          10u 0 inz(0)

D child           ds          qualified based(ptrChild)
D fChild          5u 0
*fileDescriptor
D fd              10i 0
D path            250        varying

D children        s          *      inz(*NULL)

```

Children is a pointer giving the address of the dynamic array.

child is an element of the array "children". All the information you want to store for a form have to be in the data structure child.

NbChildren stores the number of items in the dynamic array.

Each time a form is created, we call a function named ensureChild. This function searches in the dynamic array for an available item. If no item is available, the function ensureChild will add 10 new items to the dynamic array

An other function, getChild allows to retrieve the values associated to a form.

```

P EnsureChild      B
D                  pi
Dfx                5u 0
D i                s      10i 0
D size             s      10u 0
D saveNbChildren   s      10u 0

c                  for      i= 1 to NbChildren
c                  eval      ptrChild = Children +
c                             %size(child)* (i-1)
c                  if        child.fChild = 0
C                  clear      child
c                  eval      child.fChild = fx
c                  return
c                  endif
c                  endfor

* no free entry found in dynamic array
* we add 10 new entries
c                  eval      saveNbChildren = nbChildren
c                  eval      nbChildren = nbChildren + 10
c                  eval      size = nbChildren * %size(child)
c                  if        Children = *null
C                  alloc      size      Children
c                  else
C                  REALLOC     size      Children
c                  endif

* initialization of allocated storage
c                  for      i= saveNbChildren to NbChildren-1
c                  eval      ptrChild = Children +
c                             %size(child)* (i)
C                  clear      child
c                  endfor

```

```

* Pointing on first new entry
c          eval      ptrChild = Children +
c          saveNbChildren * %size(child)
C          clear      child
c          eval      child.fChild = fx
P EnsureChild      E

P getChild      B
D          pi      N
D fChild      5u 0
D i          s      10i 0
c          for      i = 1 to nbChildren
c          eval      ptrChild = Children +
c          %size(child)* (i-1)
c          if      child.fChild = fChild
c          return    *on
c          endif
c          endfor
c          return    *off
P getChild      E

```

EnsureChild must be called in the AfterCreate moment.

The function getChild must be called in each event.

In the next example, the window is freed in the event onClose.

EnsureChild doit être appelée dans le moment AfterCreate.

La fonction getChild doit être appelé pour chaque événement.

Dans l'exemple suivant, la fenêtre est libérée dans l'événement onClose.

In the same time, we assign the value zero to the fChild field so that item can be used in a future call.

```

~/EVENT OnClose
,* -----*
,* Description :
,* -----*
D Parameters      ds      based(pevtinf)
D Win              5u 0
D Evt              48a
,* Variable
D Action          10i 0
,* 0 : ne rien faire
,* 1 : Cacher la fenêtre
,* 2 : Libérer la fenêtre
,* 3 : Minimiser la fenêtre

```

```
,*
c      eval      ptrChild = getChild(fChild)
c      if        ptrChild <> *null
c      if        sdMsgDlg(boxConfirm:btnOK + btnCnl:
c                  'Fermer ' + child.path + '? ') =
c                  btnok
c      eval      action = 2
c      eval      child.fchild = 0
...
c      endif
c      endif
```

Example

See the programs sddmmdi1 and sddmmdi2 in silverdemo.

Chapitre 10. Multi occurencies, sdsrvlst service program

Introduction

We will create a child program that will display data of a client.
Several forms created by the child program can exist at the same time.

A service program that we will call `srvchld` will allow to have all the informations on existing forms. This service program will store informations specific to each form.

The `srvchld` service program will use a service program provided with `silverdev`, the `sdsrvlst` service program.

`Sdsrvlst` is similar to class list in java or c++

Programmation with `sdsrvlst` is similar to object oriented programmation.

Fields of structure `DsList` are similar to properties, functions are similar to methods.

« Objects » are created with `%alloc` and added to an « object » of type `DsList` the same way you would create an object in c++ to add it to a object of type list.

It is just an analogy, not real poo, so no inheriting or polyphormism.

Sdsrvlst service program

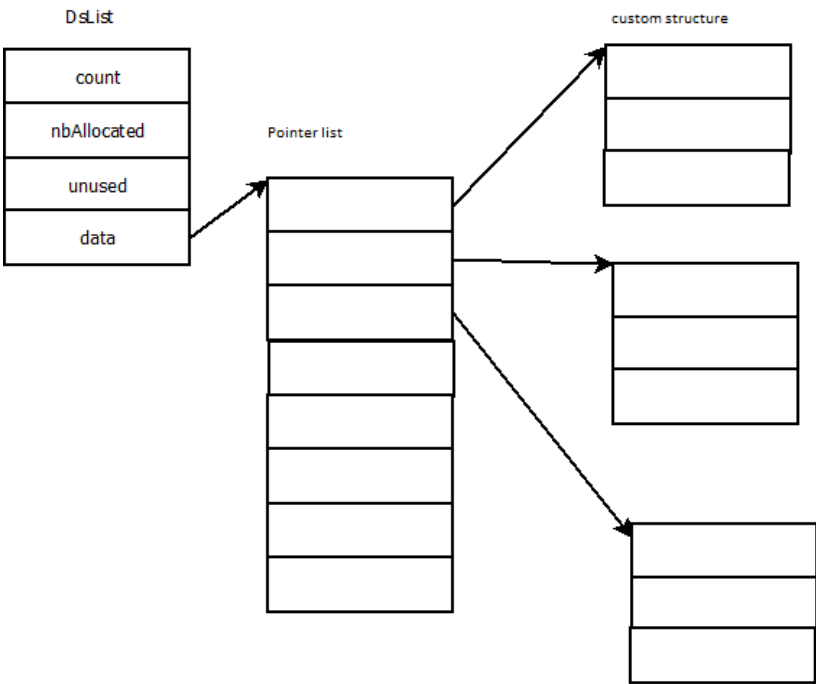
`Sdsrvlst` service program is provided with `silverdev`. A bound directory `sdsrvlst` is provided too, and prototypes are in `silverdev,h,sdsrvlst`.

This service program allows to handle lists of pointers. It contains all necessary function to add, delete an item, go through all items, exchange items, sort, etc...

Function of `sdsrvlst` service program :

<code>addItem</code>	Allows to add an item in a list
<code>clearList</code>	Allows to delte all items in a list. The <code>freeltems</code> parameter allows to precise if memory pointed by pointers is deallocated by the function.
<code>DeleteItem</code>	Allows to remove an item from the list. The <code>freeltem</code> parameter allows to precise if pointed memory must be deallocated by the function.
<code>DeleteItemIdx</code>	Allows to remove an item from the with its index.
<code>exchangeItems</code>	Allows to exchange two items in the list.
<code>exchangeItemsIdx</code>	Allow to exchange two items in the list with their index.

getItem	Allows to get an item from its index. With count field of DsList, this function allows to go through all items in the list.
indexOf	Allows to retrieve index of an item. Returns -1 if the pointer is not in the list.
SortList	Allows to sort a list. This function takes a callback type parameter (see following example)



Count field can be read directly. (do not modify it)
Allocated field is for intern use (do not modify it)
Unused field allows to align data field on 16 bytes
Data field points on dynimacally allocated data.

The pointer list is a data area dynamically allocated. If needed, addItem reallocs a bigger area.

In pointer list, there is every 16 bytes a pointer pointing on a data area dynamically allocated.

The clearList function reset the count field to zero, frees the memory pointed by data field, and if second parameter is *on, memory allocated for each item is freed.

Example :

Custom structued declared :

D dsClient	ds	qualified template
D id	10	0
D name	50	varying
D adress	250	varying

Client type variable declared (pointed by ptrClient)

D client	ds	likeDs(DsClient)
D		based(ptrClient)

Client list declared :

D clients	ds	likeDs(dsList) inz
-----------	----	--------------------

Adding a client in the list

```

/free
ptrClient = %alloc(%size(DsClient));
clear client;
client.name = NAME;
addItem(clients:ptrClient);
/end-free

```

Reading client list:

```

D i          s          10u 0
/free
for i = 1 to clients.count;
  ptrClient = getItem(clients:i);
  sdSetCell(Fl:'sf1Clients':'name':i:client.name);
endfor;
/end-free

```

sortList example :

The sortList function allows to sort a list.

This function takes a callback function as a parameter.

```
P procCompare      B
d                  pi          N
D ptr1             *          value
D ptr2             *          value

D child1           ds          likeDs (DsChild)
D                  based(ptr1)

D child2           ds          likeDs (DsChild)
D                  based(ptr2)
/free
return child1.numCli > child2.numCli;
/end-free
```

Calling the sortList function :

```
callp sortlist(children:%paddr(procCompare)) ;
```

srvchld service program

The srvchld service program will contain a member for prototypes, h/srvchld, a member for functions definition qrpglesrc/srvchld, a member for exportation définition qsrvsrvc/srvchld and a binding directory srvchld.

The qrpglesrc/srvchld contains a variable of type DsList.

```
D children         ds          likeDs (DsList)
```

The DsList type is defined in sdsrvlst.

This list will contain items of type DsChild :

```
D DsChild          ds          qualified based(ptrDummy)
D handle           5u 0
D numClient        10 0
```

Here, structure DsChild is really simple, but it could be more complex for your needs.

It can contain strings, file descriptors, (if the window opens an ifs file for example), pointers, structures and even DsList structure type.

The first function that we will define in `srvchld` is the one allowing creation of a new form.

```
P CreateNewChild...
P      B
D      pi      *
D child      ds      likeDs (DsChild)
d      based(ptrChild)

/free
eval ptrChild = %alloc(%size(DsChildQuery));
clear child;
call addItem(children:ptrChild);
return ptrChild;

/end-free
P CreateNewChild...
P      E
```

We used `addItem` function of `sdsrvlst` service program to add the structure created in the list.

If we continue analogy with `poo`, `createNewChild` function is kind of a constructor for `DsChild` structure.

The second function we will define in `srvchld` is the one allowing opening of a new client form.

```
P openChild      B      export
D      pi
D numCli      10 0
D i      s      10u 0
D pWin      s      5u 0
D child      ds      likeDs (DsChildQuery)
D      based(ptrChild)

/free
for i = 1 to children.count;
eval ptrChild = getItem(children:i);
if child.numCli = numCli;
callp sdShow(child.handle);
return ;
endif;
endfor;
callp CHILD(pWin);
eval ptrChild = createNewChild;
eval child.handle = pWin;
eval child.numCli = numCli;
/end-free
c
P openChild      E
```

We used count field of DsList and the getItem function of sdsrvlst to go through all items in the list.

A function in srvcld will allow to retrieve the pointer allowing to get informations in the form .

```

P getChild      B      export
D      pi      *
D handle      5u 0
D i      s      10i 0
D child      ds      likeds (DsChildQuery)
d      based(ptrChild)
/free
  for i = 1 to childrenQuery.count;
    eval ptrChild = getItem(children:i);
    if child.handle = handle;
      return ptrchild;
    endif;
  endfor;
  return *NULL;
/end-free
P getChild      E

```

Child program

The child program will be a silverdev program. It allows to create the form.

The child program contains a variable F1 as any silverdev program, but this variable is not usable in events.

F1 contains value of last created form, but several forms created by this program can exist at the same time in the application.

When the child program creates a form, the caller must get id created to assign it to DsChild structure associated.

We add F1 variable in program parameters.

```

š*/BLOCK RPGPARM
// ----- RPGPARM :
c  *entry      plist
C      parm      F1

```

The child program will use the srvcld service program to get informations about forms.

Program source will contain following instruction :

```
h bnddir ('SRVCHLD')
```

Using srvchld binding directory

And :

```
/copy H,srvchld
```

Copy of srvchld service program prototypes

For each event , we will not use F1 variable but win parameter in the event.

Example:

```
~/EVENT Button1_OnClick
,* -----*
,* Description :
,* -----*
D Parameters      ds          based(pevtnf)
D Win              5u 0
D Evt              48a
,*
,* based variables
D child           ds          likeDs(DsChild)
D                 based(ptrChild)
/Free
eval ptrChild = getchild(win);
if ptrChild <> *NULL;
// you can use child.numCli
endif;
/end-free
```

When a form is deleted, in onclose event for example, use the sdDeleteItem functino in sdsrvlst to delete informations about the form :

```
~/EVENT OnClose
,* -----*
,* Description :
,* -----*
D Parameters      ds          based(pevtnf)
D Win              5u 0
D Evt              48a
,* Variable
D Action           10i 0
D child           ds          likeds(DsChild)
d                 based(ptrChild)
/free
eval ptrChild = getchild(Win);
```

```

if ptrChild = *NULL;
  eval action = 2;
  callp sdFreeForm(child.handle);
  callp deleteItem(childrenQuery:ptrChild:*on);
endif;
/end-free

```

We used deleteItem function from sdsrvlst service program.

Note: The third parameter indicates whether the function must deallocate memory of DsChild structure. When this parameter is false, pointer is removed from the list but memory allocated for this pointer is not freed.

Two cases to put this parameter to false.

Case 1 : If an item is in several lists, when destroying this items, you have to remove it from all lists containing it, but deallocation must be done only once.

Case 2 : If DsChild structure is more complex and contains pointers, you will need to write a specific function to deallocate memory pointed by these pointers.

Prototypes

A member for prototypes is not mandatory, but it is advised for a nice programming.

Member will be included with a /copy in programs that will use the service program .
Member will be also included with a /copy in member qrpglesrc/srvchld.

Exportation member

Exportation member of srvchld service program is not mandatory. If you don't create it , you have to specify EXPORT *ALL in the crtsrvpgm command. (default value is *srcfile)

Exportation member user allows to handle signatures of the service program.
Here what can look like an exportation source :

```

STRPGMEXP  PGMLVL(*CURRENT) SIGNATURE('SRVCHLD')
export(createNewChild)
export(OpenChild)
export(getChild)
export(FreeChild)
ENDPGMEXP

```

Binding directory

Create a srvcld binding directory with crtbnmdir command, then add srvcld service program in the binding directory with the command WRKBNDIRE BNDDIR(SRVCHLD) ou AADBNDIRE.

Binding directory can be added in programs that use service program in H card.

Chapitre 11. Dialog box functions

SdDialog

Prototype:

d	sdSaveToFile	pr	N
d	pform		5u 0 const
d	Component		30 varying value

Description:

The sdDialog function enables display of a dialogue box.
The function returns *on if the user clicks OK.

This function applies to the following components:
CfontDialog, CcolorDialog, COpenDialog, CsaveDialog,
COpenPictureDialog, CSavePictureDialog.

For example:

DName	s	200	varying
C	if	sdDialog(f1:'fontdialog1')	
C	eval	name=sdget(f1:'fontdialog1':'font.name')	
C	callp	sdshowmessage(name)	
C	endif		

DCouleur	s	10i 0	
C	if	sdDialog(f1:'ColorDialog1')	
C	eval	Couleur= sdGetInt(f1:'ColorDialog1':'Color')	
C	callp	sdshowmessage(%char(Couleur))	
C	endif		

sdShowMessage

If you simply want to display a message in a modal form, you do not need to create a form.

Instead, use the sdShowMessage function.

This function calls an operating system dialogue box.
The size of the dialogue box is adapted to the text that you want to display.

To display a message over several lines, add the code x'0d' in the text for each new line.

sdMsgDlg

If you want to create a dialogue sheet with the user, there are two possible solutions.

First solution:

Create a form.
Add a label and buttons with different modalResult values.
Display this form in modal and query the form's modalResult property to know which button the user clicked.

Second solution:

Use the sdMsgDlg function.
This function uses several parameters.
These parameters enable you to modify the text contained in the dialogue box, to indicate the buttons that you want to display and possibly to add an icon.
This function returns a value. It is therefore possible to know which button the user clicked.

Note: The sdMsgDlg function can be used instead of the sdShowMessage function by using a single button.

Prototype:

d sdMsgDlg	pr	4	0	
DIcone		2	0	value
DBoutons		4	0	value
dTexte		3000		varying value

Parameters:

Return value: To know which button the user clicked.

Buttons: Buttons displayed in the dialogue box.

Possible button values and return value:
Values predefined in H,SILVERDEV:

DBtnNONE	C	0
----------	---	---

EXPERIA Europe

SilverDev

DBtnOK	C	1
DBtnCn1	C	2
DBtnAbort	C	4
DBtnRetry	C	8
DBtnIgnore	C	16
DBtnYes	C	32
DBtnNo	C	64
DBtnAll	C	128
DBtnNoToAll	C	256
DBtnYesToAll	C	512

Icon: Image inserted in the dialogue box.

Possible icon values:

Values predefined in H,SILVERDEV:

DBoxWarning	C	0
DBoxError	C	1
DBoxInfo	C	2
DBoxConfirm	C	3
DBoxNone	C	4

Text: Text displayed in the dialogue box.

To display several buttons, just add the values required.

For example:

```
C      if      sdMsgDlg(boxConfirm:btnOK + btnCn1:
C      'Do you really want to delete? ') = btnOK
C      endif
```

sdSelectFile

Note:

Prefer to use the sdDialog function with the COpenDialog or CSaveDialog component.

Prototype:

d sdSelectFile	pr	N	
DFilter		1000	varying value
DFileName		1000	varying
DTypeSelect		1 0	value
DInitialDir		1000	varying value options(*nopass)

Description:

This function opens a dialogue box that enables the user to select a file from his/her hard disk.

Return value:

The Boolean returned is *ON if the user selects a file and *OFF if the user abandons file selection.

Parameters:

Filter:

Determines the file masks (filters) available in the dialogue box.

The file selection dialogue box includes a scroll list of file types under the entry field. When the user selects a file type from this list, only the files of that type are shown in the dialogue box.

Assign a value comprising a description and a mask to the Filter, separated by a vertical bar characters. The vertical bar must not be surrounded by spaces. For example: 'Text files (*.txt)|*.TXT'

Several filters must be separated by as many vertical bars. For example:

```
'Text files (*.txt)|*.TXT|RTF Files(*.rtf)|*.RTF'
```

To include several masks in a single filter, separate the masks with semi-colons.
For example:

```
'Image files |*.BMP;*.JPG;*.ICO'
```

If no value is assigned to Filter, the dialogue box displays all types of files.

FileName:

Name of the file selected by the user. (Check that the user has selected a file by testing the return value).

TypeSelect:

Determines the type of dialogue box.

Possible values:

DSeOpen	C	0
DSeSave	C	1
DseOpenPicture	C	2
DseSavePicture	C	3

InitialDir:

This parameter is optional; it specifies the initial directory displayed in the dialogue box.

For example:

```
PSelectImg      B
D               PI
D path          s      1000    varying
C               if      sdSelectFile('*.jpg |*.jpg':
C                   Path:seOpenPicture)
C               callp    sdLoadFromFile(FI:'Image1.picture':
C                   path)
C               endif
P               E
```

sdSelectDir

Prototype:

D sdSelectDir	pr	N	
DFileName		1000	varying
DInitialDir		1000	varying value options(*nopass)

Note:

You can create your own sdSelectDir function using the CShellTreeView component of the System tab.

sdPrintDlg

Note:

Prefer to use the sdDialog function with the CPrintDialog component.

This function enables a dialogue box to be opened to select the printer.

Prototype:

d sdprintDlg	pr	N
--------------	----	---

For example:

C	if	sdPrintDlg
C	callp	sdPrint(F2: 'Rep1')
C	endif	

sdSelectColor

Note:

Prefer to use the sdDialog function with the CColorDialog component.

This function enables a dialogue box to be opened to select a colour.

Prototype:

d sdSelectColor	pr	N
DColor		10i 0

Parameters:

Return value:

The Boolean returned is *ON if the user selects a colour and *OFF if the user abandons colour selection.

Color:

Value of the colour selected by the user.

Equal to 0 if the user does not select a colour.

For example:

```
pButton1_OnClick...
P          B
d          PI
d PevtInf          * const options(*nopass)
D MyColor      s          10i 0
C          if          sdSelectColor(MyColor)
C          callp      sdsetInt(F1:'edit1':'color': MyColor)
C          endif
P          E
```

Note:

The `sdSelectColoret` `sdPrintDlg` functions have become obsolete since creation of the `sdDialog` function.

Chapitre 12. PC file category functions

The following functions enable file handling on the client side.

sdGetDir

Prototype

```
d sdgetDir      pr      *
d Dir          256    varying value
```

Description

The sdGetDir function enables listing of the files corresponding to a filter in a directory on the client workstation.

The sdGetListLevelpermet function tells you the size of the file.
The sdGetListState function tells you the type of file or directory:

Possible values for this type of file:

- 1 read-only file
- 2 hidden file
- 4 system file
- 8 id volume file
- 16 directory
- 32 archive
- 64 file

The values can be combined. For example: 65=64 + 1 means a read-only file

For example:

```
Dlist      s      *
D i        s      10i 0
D Size     s      10u 0
D Type     s      5u 0
D Name     s      50    varying
C          eval    list = sdgetdir('c:/*')
C          for     i=0 to sdgetlistcount(list)-1
```

```
C      eval      Name = sdgetListLabel(list:i)
C      eval      Type = sdgetListstate(list:i)
C      eval      Size = sdgetListLevel(list:i)
C      endfor
C      callp      sdFreelist(list)
```

SdDelFile

Prototype:

```
d sdDelFile      pr      10u 0
d Dir      1000      Varying Value
```

Description:

The sdDelFile function enables deletion of a file on the client workstation.

Return value:

- 0: The file has been deleted.
- 1: The file does not exist.
- 2: The file exists but could not be deleted.

sdDownLoad

Prototype:

```
d sdDownLoad      pr      10i 0
d pathAs      1000a      varying value
d pathPC      1000a      varying value
d Erase      N      Options( *nopass) Value
```

Description:

Enables a file to be copied from the ifs to the PC.

Parameters:

The Erase parameter enables forced overwriting of the file on the PC if the file exists.

If Erase is *OFF or if it is omitted and the file exists on the PC, a dialogue box asks the user to confirm before overwriting the existing file.

Return value:

- 0: All OK.
- 1: The file does not exist on the IFS.
- 2: A file with the same name exists and the user has refused to overwrite it.
- 3: A problem occurred while saving.

Note:

If a file with the same name already exists on the client PC, a dialogue box asks for confirmation before overwriting the file. If the user refuses, the function's return code is -2.

For example:

```
*/EVENT Button1_OnClick
D pathPc      s      1000    varying
D pathAs      s      1000    varying
D retour      s      10i 0

C      eval      pathAs=sdget(f1:'edit1':'text')
C      if      sdSelectFile('*. * |*. *':
C              PathPc:seSave)
C      eval      Retour = sdDownload(pathAs:
C              pathPc)
C      endif
```

SdForceDir

Prototype:

d	sdForceDir	pr	10u 0
d	Dir	1000	Varying Value

Description:

The sdForceDir function enables creation of a directory on the client workstation.

Return value:

- 0: The directory has been created.
- 1: The directory was not created.

SdGetFileSize

Prototype:

D	sdGetFileSize	pr	10u	0
D	FilePath		1000	Varying Value

Description:

Enables query of the size of a file on the client workstation.

For example:

C		eval	Taille=sdGetFileSize (FileFrom)
---	--	------	---------------------------------

SdSaveToFile

Prototype:

d	sdSaveToFile	pr	10u	0
d	pform		5u	0 const
d	Component		30	varying value
D	File		1000	Varying Value

Description:

Enables an image or the content of a TStrings to be saved in a file.

Return values:

- 0: All OK
- 1: The component passed as a parameter does not seem to manage this function
- 2: The save failed (the file may be in use)
- 3: The directory does not exist

For example:

C		callp	sdSaveToFile (f1: 'memo1.lines':
C			'c:/test.txt')

```
C      callp      sdSaveToFile(f1:'Image1.picture':  
C      'c:/test.bmp')
```

```
C      callp      sdSaveToFile(f1:'Chart1':  
C      'c:/test. bmp ')
```

SdSelectFile

See sdSelectFile

SdSetXFerBar

Prototype:

```
d sdSetXFerBar      pr  
d pform              5u 0 const options(*nopass)  
d Component          30   varying value options(*nopass)
```

Description:

Enables design of a CProgressBar type component to show progression of the sdUpload and sdDownLoad functions.

Note:

To delete the assignment, call the function without parameters.

For example:

```
C      callp      sdSetXFerBar(F1:'Progressbar1')  
...  
C      callp      sdSetXFerBar
```

SdUpload

Prototype:

```
d sdUpload          pr              10i 0  
d pathPC            1000a   varying value  
d pathAs            1000a   varying value
```

Description:

Enables a file to be copied from the client PC to the IFS.

Parameters:**Return value:**

- 0: All OK.
- 1: The PC file does not exist.
- 2: The IFS file name is incorrect or you do not have the required rights.
- 3: The PC file appears to be in use.

For example:

```
P EvtUpload      B
D               PI
D FichierPC      s      1000    varying
D FichierAs      s      1000    varying
D Retour        s      10i 0

C               eval      FichierAs='/home/images/testUpload.jpg'
C               if        sdSelectFile('*.jpg |*.jpg':
C                       FichierPc:seOpenPicture)
C               eval      retour =sdUpload(FichierPc:FichierAs)
...
C               endif
P               E
```

SdSelectDir

See sdSelectDir

SdFileExists**Prototype:**

```
d sdFileExists  pr      N
```

d	File		1000	Varying	Value
---	------	--	------	---------	-------

SdDirExist

Prototype:

d	sdDirExists	pr		N	
d	Dir		1000	Varying	Value

sdCopyFile

Prototype:

d	sdRenameFile	pr		N	
D	OldFile		1000	Varying	Value
D	NewFile		1000	Varying	Value

Description:

The sdCopyFile function enables a file to be copied from the PC.

Return value:

- *On: The function succeeded.
- *OFF: The function failed (OldFile non-existent or in use, destination directory non-existent, NewFile exists already, etc.)

For example:

D	ret	S		N	
C		eval		ret=sdCopyFile('c:/test.txt':'c:/toto/'	
C				'test.txt')	

SdGetRealFolder

Prototype:

d	sdGetRealFolder...				
d		pr		1000	Varying
d	File			1000	Varying Value

Description:

The `sdGetRealFolder` function returns the real name of a file from an expression containing an environment field like `%TEMP%`

For example:

```
DPath      s      1000    varying
C          eval    Path   =' %TEMP%\test.txt'
C          eval    Path   =sdGetRealFolder(Path )
C          callp   sdDownload('/home/Doc.txt':
C                  Path)
C          callp   sdExecutePc('open':Path:
C                  '' :W_NORMAL)
```

To find out the system fields, right click on MyComputer:

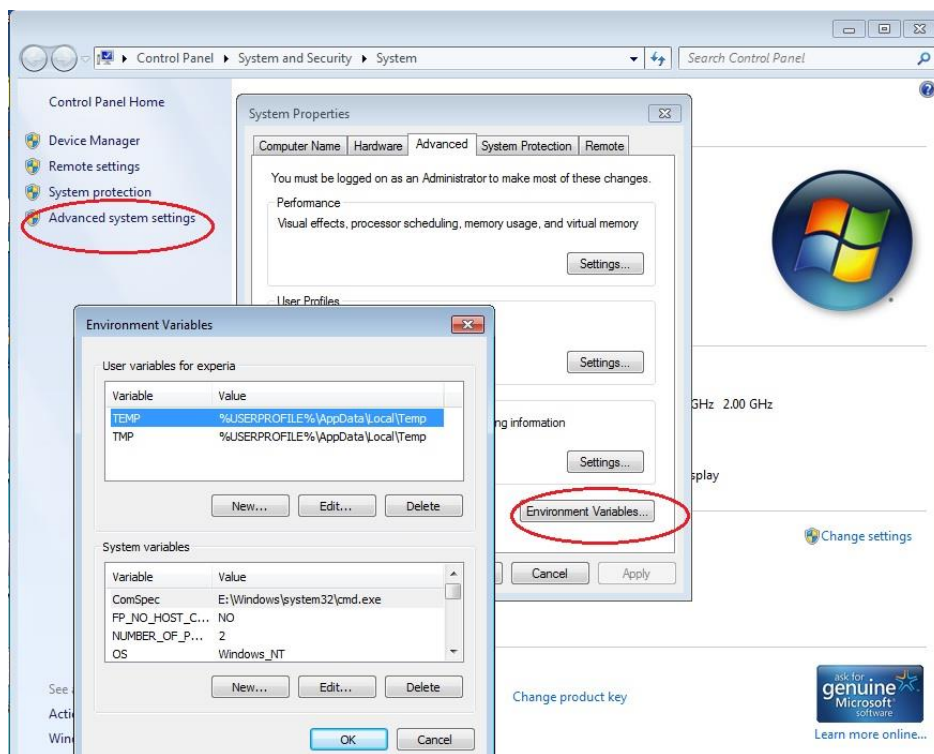


Figure 6

SdRenameFile

Prototype:

```
d sdRenameFile      pr          N
D OldFile           1000      Varying Value
D NewFile           1000      Varying Value
```

Description:

The sdRenameFile function enables a file to be renamed on the PC.

Return value:

- *On: The function succeeded.
- *OFF: The function failed (OldFile non-existent or in use, destination directory non-existent, NewFile already exists, etc.)

For example:

```
D ret      S      N
C          eval    ret=sdRenameFile('c:/test.txt':'c:/Foo/'
C                               'test.txt')
```

Chapitre 13. Collections category functions

The CDateEdit.ValidDates, CSFL.columns and CSFL.ExAttributes properties are collections.

The following functions enable the collections to be handled by programming.

sdCollecClear

Prototype:

```
d sdCollecClear  pr
d  pform                5u 0
d  component            30   varying value
d  Collection            30   varying value
```

sdCollecAdd

Prototype:

```
d sdCollecAdd    pr
d  pform                5u 0
d  component            30   varying value
d  Collection            30   varying value
```

sdCollecDelete

Prototype:

```
d sdCollecDelete pr
d  pform                5u 0
d  component            30   varying value
d  Collection            30   varying value
d  Elt                   10 0 value
```

sdCollecIns

Prototype:

```
d sdCollecIns    pr
```

EXPERIA Europe

SilverDev

```
d pform          5u 0
d component      30   varying value
d Collection     30   varying value
d Elt            10  0 val
```

For example:

```
C          callp      sdCollecClear(F1:'listview1':'items')
C          callp      sdCollecAdd(F1:'listview1':'columns')
C          callp      sdSetString(F1:'listview1':
C          'columns(0).caption':'Column 1 ')
C
```

Chapitre 14. TString category functions

Introduction

A TString type property is a list of character strings.
The following properties are TString type properties:
CMemo.lines, CListBox.items, CListBox.keys, CComboBox.items, CComboBox.Keys

Note: For TString type properties, you can use the List category functions. See Optimisation

In all client-server applications, response time is essential.
This section presents the means used by SilverDev to optimise response times and advises you on how to improve your applications.

Functions with response

Calls to functions requiring a response are slower since they need to go to and from the network.

Functions that return no value are grouped in a buffer.
This is the case of the sdSet, sdSetCell etc. functions for example.
This buffer is sent to the client at the end of the event manager or during a function call that returns a value (e.g. sdGet) or during a the sdApplySet function call.

For example:

On a test machine, we measured:
1000 sdGet calls: approx. 3 seconds (3000 milliseconds).
1000 sdset calls: approx. 15 milliseconds

Field cache

To improve the performance of sdGet type functions, for each event, certain property values are sent to the server.
These values are placed in cache.
The sdGet type functions start by searching this cache.

This mechanism is transparent for the developer, but it enables a significant improvement in performance.

Only the properties with the highest probability of being queried are placed in cache.

For example, only the text property of the CEdit component is placed in cache.

In the debugger:

Send event

Event: _2.BtnOK.OnClick

2009/10/12 15:01:59:218

Cache transmission

_2.TITLE.text=Les robots

_2.PRICE.value=4

_2.STOCK.value=0

_2.ORDER.value=Y

_2.Themes.keySelected=2

_2.Themes.text=Fantastique

_2.IDAUT.value=17

_2.NOMAUT.text=Dostoïevski

_2.NOMEDI.text='ai Lu

_2.IDEDI.value=4

_2.DATEPUB.value=20040513

_2.DATEPUB.DateIso=2004-05-13

2009/10/12 15:01:59:234

Get execution

_2.DATEPUB.Validate

Result: True

2009/10/12 15:01:59:343

Script execution

function Main() {

_2.ModalResult=1;

}

C	Eval	TITLE= sdGet(F1:'TITLE': 'Text')
C	if	sdGetBool(F1:'DATEPARU': 'Validate')=*off
C	eval	Message =Message + 'Invalid date'

```
C                                     + X'OD'
C
C      endif
C      Eval      PRICE=sdGetNum(F1: 'PRICE': 'Value')
C      Eval      STOCK=sdGetInt(F1: 'STOCK': 'Value')
C      Eval      ORDER=sdGet(F1: 'ORDER':
C                  'Value')
C      Eval      IDTHEME=sdGetNum(F1: 'Themes':
C                  'KEYSELECTED')
C      Eval      IDAUT=sdGetInt(F1: 'IDAUT': 'Value')
C      Eval      IDEDI=sdGetInt(F1: 'IDEDI': 'Value')
C      eval      DATEPUB=sdGetDate(F1: 'DATEPUB')
C      callp      sdSetInt(F1: '*FORM': 'ModalResult':Mrok)
```

Window cache

SilverDev windows have to be transmitted from the server to the client. If they contain images, the transfer can take rather a long time. The windows created are therefore saved on the client's disk. When a window is created, the client and server start by finding out if the window already exists on the client workstation. The windows are stored in the /cache directory.

The cache file name is the window's code md5. When a window is modified, and therefore placed in cache once again, the old cache of this window is deleted. The correspondence between the origin of a window and its cache file is saved in the register database.

List type functions

As we saw earlier, sdGet type functions take a non-negligible time.

Re-reading all the items of a treeview, memo, listbox, checklistbox, etc. type component can therefore take rather a long time.

The List category functions enable a considerable amount of time to be saved. The principle is to pass on all the information of interest about a component and copy it into the memory to be able to query it locally. For more information, see (Chapitre 1)

Function sdGetSfl

For display of a directory in MyDesk, various pieces of data are sent to MyDesk.
The information is about the application and possibly an icon.
Text type information only represents a few bytes.
The icons are much larger, representing 90% of the data transferred.

A mechanism based on an md5 hashing code is used to send the icons once only.

For each application, there is a file with a .app extension on the server and possibly a .ico file too.

The MD5 code of the icon is stored in the .app file.

This file is sent first.

MyDesk checks to see if the icon exists on the disk.

If the icon already exists, it is used. If not, it is downloaded then saved on the disk under the name of the icon's hashing code.

Note:

These icons are also used for the creation of shortcuts on the desktop.

Re-using windows

When you create a window, resources are used on the client side to create the components of the window.

When the user closes a window, it is hidden but not destroyed.

This means that the components of the window and the window itself use memory on the PC side.

The window will be destroyed when the program is closed (on the PC).

If you want to free the window when it is closed, add the following code to the window's onclose event:

```
p ONCLOSE...
p          B
d          PI
d PevtInf          *   const options(*nopass)
C          callp    sdFree (F1:'*FORM')
```

Whether the window is destroyed or not, remember to test that a window does not already exist before creating it:

```
C          If      F1 <> 0
C          eval    F1 = sdcreateForm(F1REF)
C          endif
```

If it already exists, you will create several identical windows that use resources unnecessarily.

Compression

During the first data exchanges, exchange time is measured.

Depending on the time calculated, the server determines whether or not the client is distant or on the local network.

If the client is distant, the data are compressed before transmission.

Local events

All processing can be carried out in the server events, but it is quicker to process events locally rather than on the server. When an event occurs frequently and processing does not require objects on the server, prefer local events.

sdAddNode

The sdAddNode2 function should be preferred to the sdAddNode function as it is much faster and easier to use.

|

sdSIAdd

Prototype:

DsdSIAdd	pr	
d Fl	5u 0	const
D component	30	varying value
D property	30	varying value
d value	999a	varying value
d Trim	N	value options(*nopass)

Description:

The sdSIAdd function adds a string to the end of the list.

sdSIClear

Prototype:

d sdSIClear	pr		
d Fl		5u 0	const
D component		30	varying value
D property		30	varying value

Description:

Deletes all strings of the TString object.

sdSIDelete

Prototype:

D sdSIDelete	pr		
D Fl		5u 0	const
D component		30	varying value
D property		30	varying value
D Index		10i 0	value

Description:

Deletes a string from a list.

sdSIGet

Prototype:

D sdSIGet	pr	999a	varying
D Fl		5u 0	const
D component		30	varying value
D property		30	varying value
D Index		10 0	value

Description:

Returns the Line index string.

sdSIInsert

Prototype:

```
D sdSIInsert      pr
D Fl              5u 0 const
D component       30   varying value
D property        30   varying value
D Index          10i 0 value
D value          999a  varying value
D Trim           N    value options(*nopass)
```

Description:

Inserts a string in a TString object in the position indicated by the index.

Chapitre 15. Set category functions

Some properties are of the set type, i.e. they can contain zero, one or several values.

For example: the font.style property is a set that can contain the following values: fsBold, fsItalic, fsUnderLine, fsStrikeOut

These values can be assigned in Designer or during execution.

All the functions that handle these values can be arranged in two categories:

The first category operates with values (1,2,4,8 etc)

These are the functions:

SdSetSet

SdAddSet

SdDelSet

The second category operates with rows ((1,2,3,4 etc)

In this category, a string of 32 characters represents the set.

For example: the string '101100...' means bold, not italics, underline, strikeout.

These are the functions:

SdGetset

SdUpdSet

sdIsInSet

sdSetSet

Prototype:

```
d sdSetSet      pr
d pform                5u 0
d component          30  varying value
d Property           30  varying value
d P4                  30  varying value options(*nopass)
d P5                  30  varying value options(*nopass)
d P6                  30  varying value options(*nopass)
d P7                  30  varying value options(*nopass)
d P8                  30  varying value options(*nopass)
d P9                  30  varying value options(*nopass)
d P10                 30  varying value options(*nopass)
d P11                 30  varying value options(*nopass)
d P12                 30  varying value options(*nopass)
d P13                 30  varying value options(*nopass)
```

d	P14	30	varying value options(*nopass)
d	P15	30	varying value options(*nopass)

Description:

The sdSetSet function enables modification of a "Set" type property, like Font.Style

Example 1:

C	callp	sdSetSet(form:'button1':'font.style':
C		'fsBold':'fsUnderline')

Example 2:

To delete all the values of a set:

C	callp	sdSetSet(form:'Button1':'font.Style')
---	-------	---------------------------------------

sdAddSet**Prototype:**

d	sdAddSet	pr	
d	pform	5u 0	const
d	component	30	varying value
d	Property	30	varying value
d	Value	30	varying value

Description:

The sdAddSet function enables items to be added to the "Set" type properties, like Font.Style

Example 1:

C	callp	sdAddSet(F1:'Label1':'Font.style':
C		'fsBold')
C	callp	sdAddSet(F1:'Label1':'Font.style':
C		'fsUnderLine')

sdDelSet

Prototype:

d sdDelSet	pr	
d pform		5u 0 const
d component	30	varying value
d Property	30	varying value
d Value	30	varying value

Description:

The sdDelSet function enables items to be removed from the "Set" type properties, like Font.Style.

For example:

C	callp	sdDelSet(F1: 'Label1': 'Font.style':
C		'fsBold')
C	callp	sdDelSet(F1: 'Label1': 'Font.style':
C		'fsUnderLine')

SdGetSet**Prototype:**

d sdGetSet	pr	32
d pform		5u 0 const
d component	30a	varying value
d Property	50a	varying value

Description:

Use this function to query a Set type property.
The result is a string of up to 32 characters.
Each character corresponds to a value of the set.
For the font.style property, for example, the string '1011' corresponds to:

FsBold	True
FsItalic	False
FsUnderline	True
FsStrikeOut	False

For example:

```
D Style      s      32
C            eval    style=sdgetset(f1:'label1':'font.style')
C            if      sdIsInSet(Style:2)
C            callp    sdShowmessage('2')
C            endif
```

SdUpdSet

Prototype:

```
d sdUpdSet      pr      32
d pform          5u 0 const
d component      30a    varying value
d Property       50a    varying value
d Value          32a    varying value
```

Description:

This function enables a set to be updated.

The Value parameter is a string of up to 32 characters.

Each character corresponds to a value of the set.

For the font.style property, for example, the string '1011' corresponds to:

FsBold	True
FsItalic	False
FsUnderline	True
FsStrikeOut	False

For example:

```
C            callp    sdUpdSet(f1:'label1':'font.style':'1111')
```

SdIsInSet

Prototype:

```
d sdIsInSet      pr      N
d Chaine          32
d position        10i 0 value
```

Description:

Use this function to find out if a value is in a set.
Retrieve the value of a set using the `sdGetSet` function or query a Set type parameter of an event directly.

For example:

```
pButton1_OnMouseDown...
P          B
d          PI
d PevtInf          *   const options(*nopass)
DParams          ds   based(PevtInf)
D Win          5u 0
D Event          48
D Button          10i 0
D Shift          32
C          if      sdIsInSet(Shift:3)
C          callp   sdShowMessage('3')
C          endif
```

Note:

The font.style property composed of the following values, fsBold, fsItalic, fsUnderline and fsStrikeOut,

```
C          callp   sdUpdSet(f1:'label1':'font.style':'1010')
```

is equivalent to:

```
C          callp   sdSetSet(form:'button1':'font.style':
C          'fsBold':'fsUnderlIne')
```

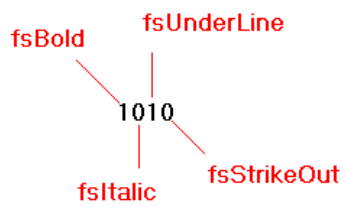


Figure 7

Chapitre 16. Miscellaneous functions

SdHideMainForm

Prototype:

```
d sdHideMainForm pr
```

Description:

By default, SilverDev makes the first window received visible even if the Visible property of this window is false.

This behaviour is acceptable in most cases, but may be problematic in certain applications. If you want to make the first window invisible, call the sdHideMainForm function.

This function must of course be called before the first call to the sdCreateForm function.

For example:

```
C      callp      sdHideMainForm
c      eval      F1 = sdCreateForm(F1REF)
c      callp      sdSetCallBack(F1
c                  : 'MenuItem1.OnClick'
c                  : %paddr(
c                  'MENUITEM1_ONCLICK'))
...
```

SdCursor

Prototype:

```
d sdCursor      pr
dModif          N value
```

Description:

When a SilverDev program notifies an event to the server, the cursor becomes an egg-timer while waiting for the server's response.

To activate/deactivate this function, use the sdCursor function.

For example:

```
C          callp      sdCursor(*OFF)
```

SdApplySet

The SilverDev functions generate code that is understandable by the client part, i.e. the launcher.

If the functions do not return a value, the code generated is placed in a buffer. This buffer is sent either when called by a function that returns a value (sdGet, sdshowmodal, sdSelectFile, etc.) or at the end of the event manager (i.e. at the end of the procedure associated with the event) or when called by the sdApplySet function.

In normal operation, this function does not have to be called; SilverDev calls it when necessary, for example in the two above-mentioned cases.

However, there may be certain cases that SilverDev's development team has not envisaged.

sdEnd

```
d sdEnd          pr
```

Description:

Tells the client part to close.

Caution: This function tells the client part to close but does not quit the RPG program. If instructions follow the call to the sdEnd function, these instructions will be executed.

sdInnerEnabled

```
d sdInnerEnabled pr
d form              like(tWHandle) const
d component          30  varying value
d value              n    value
```

Description:

The sdInnerEnabled function changes the Enabled property of the components contained in the component to which the function applies.

Note 1:

The function applies recursively.

sdInnerReadOnly

```
d sdInnerReadOnly pr
d form                like(tWHandle) const
d component            30    varying value
d value                n    value
```

Description:

The sdInnerReadOnly function changes the readOnly property of the components contained in the component to which the function applies.

Note 1:

The function applies recursively.

sdGetClientIp

Prototype:

```
D sdgetClientIp...
D                pr            20    varying
```

Description:

This function enables retrieval of the IP address of the connected PC.

```
g value
```

SdGetEnvVar

Prototype:

```
d sdEnvVar        pr            *
```

d Variables	256	options(*string) value
-------------	-----	------------------------

Description:

The sdGetEnvVar function enables query of the environment fields on the client machine. The fields queried must be separated by commas.
It is thus possible to retrieve several values at the same time.

For example:

```
Dlist      s      *
D i        s      10i 0

D Size     s      10u 0
D Value    s      500   varying
D Name     s      500   varying
C          eval    list = sdgetEnvVar('path,clientName')
C          for     i=0 to sdgetlistcount(list)-1
C          eval    Name = sdgetlistLabel(list:i)
C          eval    Value = sdgetlistKey(list:i)

C          endfor
C          callp    sdFreelist(list)
```

sdMsgDebug**Prototype:**

d sdMsgDebug	pr	
dTexte	3000	varying value

Description:

The sdMsgDebug function enables transmission of a message in the client side debug window.

For example:

C	callp	sdmsgdebug('ix is equal to : `+%char(ix)`')
---	-------	---

sdSetKeepAlive

Prototype:

```
D sdSetKeepAlive pr
D Interval 10u 0 value
```

Description:

The sdSetKeepAlive function enables activation of a client side timer which sends a frame regularly to the server. The interval parameter determines the number of seconds between frame transmissions.

This function enables detection of jobs for which the connection was broken and also enables prevention of disconnections due to a proxy or firewall.

For an interval greater than zero, if the server receives no data for a period equal to the interval parameter + 60 seconds, the job changes to MSGW (message SVD0208)

If the interval parameter is zero, the frames are no longer sent and the server will wait indefinitely for client action.

SdBringApplicationToFront

Prototype:

```
D sdBringApplicationToFront...
D pr
```

Description:

The sdBringApplicationToFront function places the application in the foreground.

SdFlashApplication

Prototype :

```
D sdFlashApplication...
D pr
```

Description :

The sd sdFlashApplication function informs the user that the application requieres attention by changing the applicaiton button in the taskbar.

Chapitre 17. Tips

“Position to” in the sub-files

To reach a line of a sub-file when entered text in a CEdit, use the local events:

```
procedure Edit1_OnChange (Sender: TObject);
var
cpt:integer;
temp:string;
begin
  for cpt:= 1 to This.SFL1.Lines do
  begin
    temp:= This.SFL1.CellValue['title',cpt];
    if(UpperCase(temp) >= UpperCase(This.Edit1.Text)) or (cpt =
This.SFL1.Lines) then
    begin
      This.SFL1.GotoCell('title',cpt,false);
      break;
    end;
  end;
end;
```

Sub-files and F11

In 5250 applications, you were probably accustomed to modifying the view of the sub-files using F11.

SilverDev's sub-file has a horizontal scroll box, it is therefore possible to do without these different views entirely.

However, some users still want the different views of the sub-file.

The solution that we recommend is to use a single sub-file. When the user changes views, you can make the columns visible or hidden.

```
C          callp      sdSetBool(F1:'sf11':
C          'colbyname('col2').visible':*ON)
```

The view can be changed for example, using one of the following operations:

_Popup menu on the sub-file
_Shortcut F11 (use a Taction)
_Buttons
_Tab sheets

If using tab sheets, use a CPageControl.

Reduce its height so that only the tab sheets are visible, set its align property to alTop, then place the sub-file in alclient just below.

On the tab sheet onshow event, change the column visibilities.

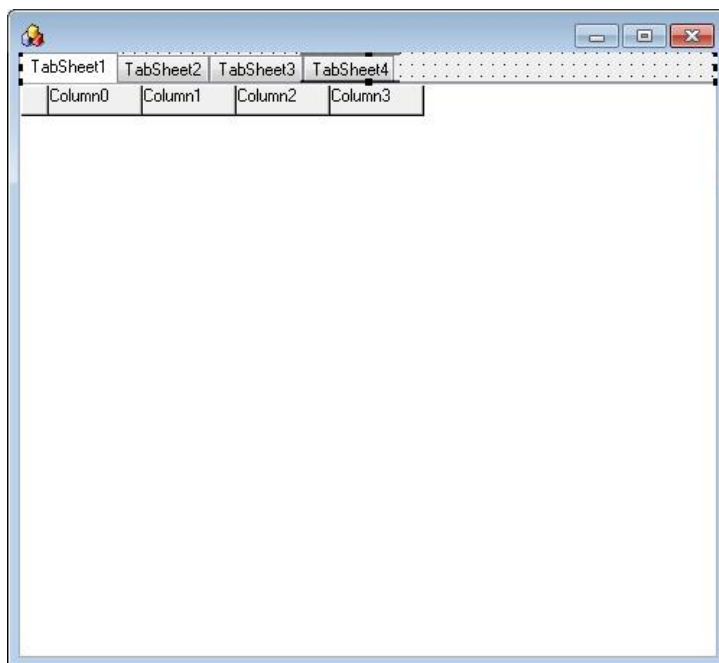


Figure 8

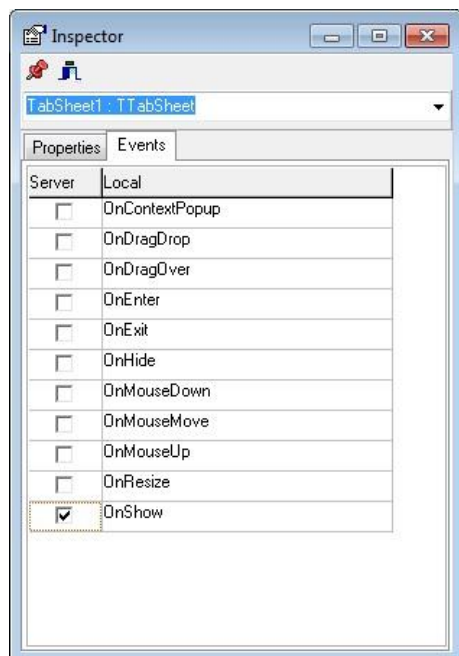


Figure 9

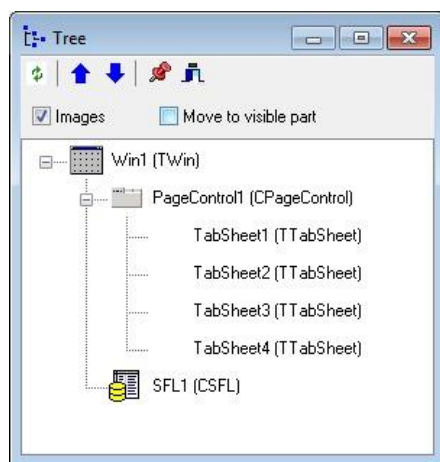


Figure 10

Wait window

Introduction:

In the example below, we are going to make a wait window that can be used by several programs.

Window

Start by creating a window containing a progressBar and a label.

Change the Name property of the progressBar to Bar1.

Change the BorderIcons property to [] and Position to poScreenCenter.

Change the properties of Label1: align = alClient, alignment = taCenter and Layout = tlCenter

Tick the window's OnAfterShow event.

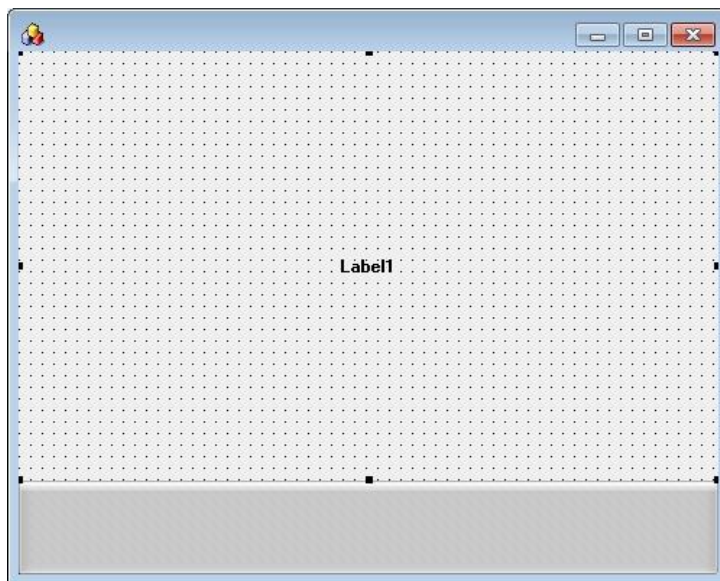


Figure 11

Code:

```
*/BLOCK RPGSPCID
* ----- RPGSPCID : Data descriptions (D Spec.)
Dpfonc          s          *   procptr
```

```

DFonc          pr          extproc(pfunc)
DFx            5u 0
C      *entry      plist
C            parm          PFONC
*/BLOCK RPGBEFORESHOW
* ----- RPGBEFORESHOW: Before show(F1)
C            callp      sdCursor(*off)
*/BLOCK RPGAFTERSHOW
* ----- RPGAFTERSHOW : After show(F1)
C            callp      sdSetInt(F1:'Bar1':'Position':0)
C            callp      sdCursor(*on)
*/EVENT OnAfterShow
* -----*
* Description : *
* -----*
D Parameters    ds          based(pevtnf)
D Win           5u 0
D Evt           48a
C            callp      func(F1)
C            callp      sdClose(F1)

```

Caller program:

```

*/BLOCK RPGSPCID
* ----- RPGSPCID : Data descriptions (D Spec.)
DMyProc        pr
DWait         5u 0

*/BLOCK RPGPROCDEF
* ----- RPGPROCDEF : User procedures
PMyProc        B
D            pi
DWait         5u 0
D i           s          10i 0
C            callp      sdSetInt(FWait:'Bar1':'Max':10)
C            for        i = 1 to 10
C                ...
C            callp      sdSetInt(FWait:'Bar1':'position':i)
C            callp      sdSetString(FWait:'Labell':'caption'
C                               %char(i) +'/10')
C            callp      sdRefresh(FWait:'Labell')
C            callp      sdApplySet
C            endfor
PMyProc        E

*/EVENT Button1_OnClick
D Parmameters  ds          based(pevtnf)

```

```

D Win                5u 0
D Evt                48a
D pgm                s    20    inz('WAIT')
D pMyProc            s    *    procptr
*
C                eval    pMyProc = %paddr(MYPROC)
C                call    pgm
C                parm    pMyProc

```

Note : You can also use `sdStrLongProcess` and `sdEndLongProcess`.

This method is easier, the long process is done in the window containing `CActivityIndicator` or `CProgressBar`, but you have to create a window in each application

Activate/deactivate items of the OnPopup menu

```

procedure TForm1.PopupMenu1Popup(Sender: TObject);
var
Statut:string;
row:integer;
begin
row:= This.CSFL1.RowSelected;
Statut:='';
if(row >=1)then
begin
Statut:= This.CSFL1.GetCellValue('statut',row);
end;
This.menuitem1.Enabled := (row >=1) and ((Statut ='1') or (Statut =
'2'));
end;

```

Selection windows

When a window is used to allow the user to make a selection, create a program with a modal window.

To do so, on the program name, choose option 8 and the `*MODAL` window display mode.

SilverDev	Silverdev programs	23/04/10 15:27:53
Program name	SDADDGRP	Creation ADUVAL 2008-01-25
Type	RPGLE	
Description	Group creation	
Main window user space	SDADDGRP	Library : *LIBL
Display mode	*MODAL	(*NOMODAL, *MODAL, *NOSHOW)
Program end indicator	RT	(LR, RT)

The generator creates a field called \$F1ModRes.

Add this field to the list of the parameters of the program to be called along with the values to be retrieved.

```

*/BLOCK RPGFARM
* ----- RPGFARM : Program parameters
C  *ENTRY      PLIST
C              PARM          $F1ModRes
C              PARM          PIDAGE          5 0

```

The generator generates the following code:

```

c              Eval      $F1ModRes = sdShowModal(F1)

```

On the event of the user's choice (double-click on a sfl or button), change the window's Modalresult property.

```

*/EVENT btnOk_OnClick
* -----
* Description :
* -----
DRow          s          10i 0
DWIDAGE        s          5 0
DWdate         s          8 0
C              eval      Row = sdGetInt(F1:'sfl1':
C                      'RowSelected')
C              if        Row > 0
C              eval      WIdAge=sdGetCellNum(F1:'sfl1':
C                      'IDAGE':row)
C              eval      PIDAGE=WIDAGE
C              callp      sdSetInt(F1:'*FORM':
C                      'ModalResult':Mrok)
C              endif

```

In the caller program, use the ModRes parameter to know whether or not the user has made a choice or left the modal window by clicking the cross at the top right.

DPmodRes	s	10u	0	
DNewIDAGE	s	5	0	
DNbAges	s	10	0	
C	K1	klist		
C		kfld		WUSER
C		kfld		WIDAGE
C		call	'SVCHXAGES'	
C		parm		PmodRes
C		parm		NBAGES
C		if	PModRes=MrOk	

Note: If a sub-file is to be loaded into a modal window, call the loading in the BEFORESHOW moment. The AFTERSHOW moment occurs once the window has closed.

Note: a button can be added with a modalResult property of mrCancel. This button automatically closes the modal window and the caller program will retrieve the fact that the user has cancelled.

Compilation control

When a screen is compiled, the SilverDev server searches for an exit program associated to the exit point SVDCTRLCPLSVD

To add an exit program to an exit point, use WRKREGIN command.

This program is not supplied. The server does nothing if the program does not exist. You can therefore write this program.

It is a program whose parameters have to be as follows:

PGM	PARM(&USER &LIB &USRSPC &RET &TEXTRET)
DCL	VAR(&USER) TYPE(*CHAR) LEN(10)
DCL	VAR(&LIB) TYPE(*CHAR) LEN(10)
DCL	VAR(&USRSPC) TYPE(*CHAR) LEN(10)
DCL	VAR(&RET) TYPE(*CHAR) LEN(1)
DCL	VAR(&TEXTRET) TYPE(*CHAR) LEN(256)

The User parameter is the connected profile.

The Lib parameter is the library where the screen will be compiled.

The Ustrspc parameter is the name of the screen that will be compiled.

The Ret parameter must return Y or N (N = compilation will be blocked)

The TextRet parameter is a text that will be displayed in Designer if Ret is N

Keyboard shortcuts

5250 applications offer the possibility of using shortcuts like F1, F2, etc.

You will probably be tempted to offer such shortcuts in your applications for your users.

The best solution is to use a TAction component.
This component has a property called shortcut.
It is the combination of keys that triggers the action.
The use of the shortcut triggers the onexecute event.

Modification of a sub-file column

To change a property of a sub-file column during execution, use the code below as an example:

```
C          callp      sdSetBool(F1:'sfl1':  
C          'colByName('Titre').ColumnField.ReadOnly':  
C          *ON)
```

or:

```
C          callp      sdSetBool(F1:'sfl1':  
C          sdCol('Titre')+'.ColumnField.ReadOnly':  
C          *ON)
```

Adjusting component size

If you allow the user to maximise the windows, the position and size of the components will no longer be suitable.

For example:

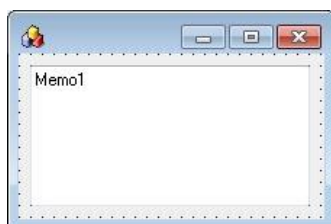


Figure 12

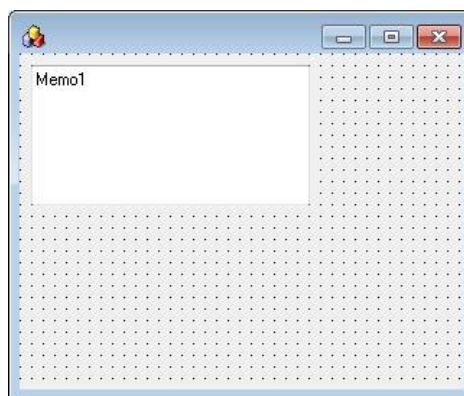


Figure 13

Align

This property, common to all components, can have the following values:

alClient	The control fills the client zone of its parent. If another control already occupies part of this zone, the new control will be resized to take up the remainder of the zone.
alLeft	The control is placed on the left edge of its parent and takes all the height of its parent. Its width is not altered.
alRight	The control is placed on the right edge of its parent and takes all the height of its parent. Its width is not altered.
alBottom	The control is placed at the bottom of its parent and takes all the width of its parent. Its height is not modified.
alTop	The control is placed at the top of its parent and takes all the width of its parent. Its height is not modified.
alNone	The control remains in its original position. Default value.

Note:

It is possible to have several components with the alLeft property (for example).

They will be next to one another. In this case, use the CSplitter component in the Additional tab sheet to allow the user to modify the width of the components.

Anchors

If you want your component to remain 1cm from the edge, regardless of the size of the window, the Align property is not sufficient.

In this case, you need the Anchors property.

The Anchors property is a set that can include the four following values: akLeft, akRight, akTop, akBottom.

By default, the components include the akLeft and akTop values.

This means that the components always remain the same distance from the top and from the right of the parent control.

If you want the buttons to remain at the bottom right of a window, you will need to set the Anchors property to [akRight,akBottom]

If you include all the values, the component will be enlarged with the window.

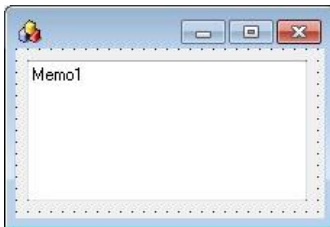


Figure 14

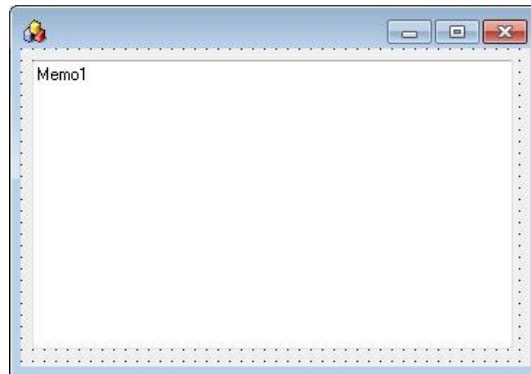


Figure 15

Current user profile

SilverDev jobs operate in a pre-started TCP/IP server under a standard profile, therefore the user profile indicated in the jobname does not necessarily correspond to the profile actually used by the job.

When the connection is made, the job profile is changed (by system API) to that indicated by the end user.

The job user profile does not reflect this modification.

The user profile actually used is called: “**current user profile**” (or Current profile)

You can see this profile using option 1 (task status) of the DSPJOB command (or WRKJOB):

Display Job Status Attributes			
Job:	SDWRKBCH	User: QPGMR	Number: 33663
Status of job	ACTIVE		
Current user profile	JEANMICHEL		
Job user identity	JEANMICHEL		
Set by	*DEFAULT		

Here, although the job name indicates user QPGMR, the profile actually used for rights' management is JEANMICHEL

Retrieval of the current profile in the programs

RPG

The SDS (System Data Structure) enables retrieval of the name of the current user in positions 358 to 367:

d	psds	sds	
d	pgmStatus	*status	
d	pgmProc	*proc	
d	pgmRoutine	*routine	
d	pgmExcpId	40	46
d	pgmLibrary	81	90
d	pgmMsgDta	91	170
d	pgmJob	244	269
d	pgmJobName	244	253
d	pgmJobUser	254	263
d	pgmJobNumber	264	269
d	pgmQJob	244	269
d	pgmCurUser	358	367

Caution: field pgmJobUser returns the name of the user who started SilverDev.

CLP

The CURUSER parameter of command RTVJOBA enables retrieval of the current user profile name in CLP:

```
RTVJOBA CURUSER(&CURUSER)
```

Triggering an event programmatically

To trigger the onclick of a button:

```
C          callp      sdMethod(F1: 'Button1': 'Perform':
C          'BM_CLICK': '1': '0')
```

Note: Use of this method is probably a sign that your program structure is inadequate.

Sending a message between windows

Messages can be sent from one window to another using the `sdSendMsgWin` function, for example, to trigger a procedure contained in another program.

Program 1:

```
C          callp      sdSendMsgWin(F1:1:10:'REFRESH')
```

Program 2:

```
*/EVENT OnMsgWin
D Parameters      ds          based(pevtnf)
D Win              5u 0
D Evt              48a
* Non modifiable
D From             5u 0
D Code             10u 0
D Data            100    varying
C          select
C          when      Data ='REFRESH'
C          callp      Refresh
...
C          ends1
```

Comment 1: This event is sent from the server to the server. It does not pass via the client part.

Opening a document on the server

To open a document on the ifs, copy it into the PC's temp directory and open it as follows:

```
DPath      s          1000    varying
C          eval        Path   ='%TEMP%\test.txt'

C          eval        Path   =sdGetRealFolder(Path )
C          callp        sdDownload('/home/Doc.txt':
C          Path)
C          callp        sdExecutePc('open':Path:
C          '' :W_NORMAL)
```

Dynamic CSfl:

The sddmsql program in silverdemo uses dynamic sql queries and creates columns according to the query.

Sfl mouseover

The following example enables display of a text in a memo when the mouse passes over a cell of a sub-file. **This example uses local events.**

The window has a memo called memo1 and an sfl called sfl1.

The sfl has a column col1 and a hidden column col2.

When the mouse passes over col1, the content of col2 is displayed in the memo.

The local code of the sfl onMouseMove event is:

```
procedure SFL1_OnMouseMove (Sender: TObject; Shift: TShiftState; X:
Integer; Y: Integer);
var
row:integer;
col:string;
begin
  This.memo1.lines.clear();
  This.sfl1.getCellAt(x,y,row,col);
  if(row>0) and (col='col1')then
  begin
    This.memo1.lines.add(screen1.sfl1.getCellValue('col2',row));
  end;
End;
```

Spare a call to sdGet

Instead of writing :

D path	s	250	varying
c	eval	path = sdGet(F1:'IfsDialog1':'fileName')	
c	callp	sdSetString(F1:'edtPath':'text':path)	

You can write :

c	callp	sdSetCtrl(F1:'edtPath':'text':F1:
---	-------	-----------------------------------

c

'ifsDialog1.fileName')

Chapitre 18. Screen translation

Silverdev proposes a feature for creating multi-lingual screens, and a function to search for a message in a message file.

Langs property

Select the OptionGeneration.Langs collection type property.

Add an element to this collection (or several elements if you want to translate into several languages).

Assign the LangId property to this element. This property should have three characters.

String translation

After compiling the screen, use the Tools/Localisation menu.



Figure 16

If there are several elements in the OptionGeneration.Langs collection, the langId list will be displayed in the window.

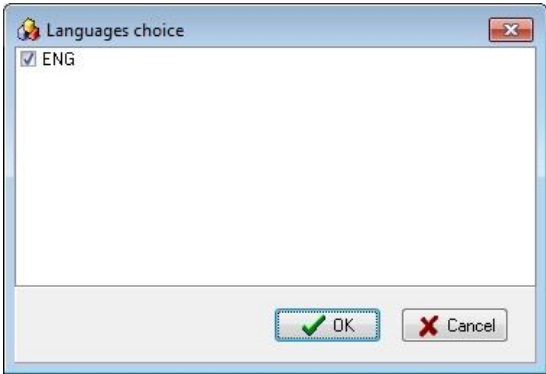


Figure 17
Tick the languages you want to display in the following window:

Translations				
SILVERDEMO/SDDMLANG				
Property	String	General translation ENG	Specific translation ENG	Translated string ENG
Caption	Démonstration de traduction	Translation demonstration		Translation demonstration
Label1.Caption	Bonjour	Hello		Hello
ComboBox1.Items(0)	janvier	january		january
ComboBox1.Items(1)	février	february		february
ComboBox1.Items(2)	mars	march		march
ComboBox1.Items(3)	avril	april		april
ComboBox1.Items(4)	mai	may		may
ComboBox1.Items(5)	juin	june		june
ComboBox1.Items(6)	juillet	july		july
ComboBox1.Items(7)	août	august		august
ComboBox1.Items(8)	septembre	september		september
ComboBox1.Items(9)	octobre	october		october
ComboBox1.Items(10)	novembre	november		november
ComboBox1.Items(11)	décembre	december		december
ComboBox1.Text	mars	march		march
Button1.Caption	Anglais	English		English
Button2.Caption	Français			French
ListBox1.Items(0)	lundi	monday		monday
ListBox1.Items(1)	mardi	tuesday		tuesday
ListBox1.Items(2)	mercredi	wednesday		wednesday
ListBox1.Items(3)	jeudi	thursday		thursday
ListBox1.Items(4)	vendredi	friday		friday
ListBox1.Items(5)	samedi	saturday		saturday
ListBox1.Items(6)	dimanche	sunday		sunday

Figure 18

The General translation and Specific translation columns can be modified; the others are read-only.

The Property column lists all the string type properties that may require translation.

The original strings in the window are displayed in the String column.
The translations are displayed in the Translated string column.
The strings are translated automatically on the basis of the PSVDDLO file.

The Property column lists all the string type properties that may require translation.

The original strings in the window are displayed in the String column.

The translations are displayed in the Translated string column.
The strings are translated automatically on the basis of the PSVDDLO file.

PSVDDLO file

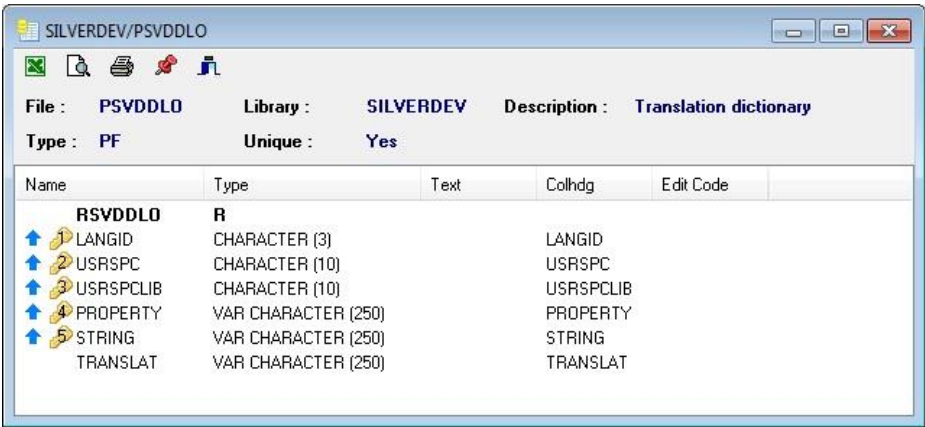


Figure 19

The string to be translated is searched first with the usrspc, usrspclib and property fields corresponding to the values on the screen, then with these values blank.


The next two columns are for text entry and enable modification of the psvddlo file.

The General translation column supplies the psvdl file with blank usrspc, usrspclib and property fields. These values will therefore be used for all the screens.

The Specific translation column supplies the psvddlo file with the current screen values in the usrspc, usrspclib, and property fields. These values will therefore be used for this screen.

After modifying the psvddlo file, the screen must be recompiled.

Press  to recompile the screen.

Press  to change languages.

Changing the language during execution

To change a screen language during execution, use the sdTranslate function.

```
D sdTranslate    pr
D F1            5u 0 const
D LangId        3    value
```

The LangId parameter corresponds to the value specified in the window's Langs collection.

```
c                callp    sdTranslate(F1: 'ENG')
```

To return to the original language, change the langId parameter to *blanks.

To change all the screens, use the sdTranslateAll function.

```
D sdTranslateAll pr
D LangId        3    value
```

All the application's screen will then be translated. The screens created after calling sdTranslateAll will be translated automatically.

Caution: if you use sdTranslateAll, it is best to call it at the start of the programme. If the value of a string has been changed by the programme, the string assigned during screen creation will be translated and not the new string.

Moving a screen

If you rename a screen or move a screen from one library to another, the translation information will be preserved, but the next time the screen is compiled, it will not be found in psvddlo. Remember to change or copy the PSVDDLO records.

Examples:

Screen moved from the SILVERDEMO library to the DEMO library.

```
UPDATE SILVERDEV/PSVDDLO SET USRSPCLIB = 'DEMO' WHERE USRSPC
= 'SDDMLANG' and USRSPCLIB ='SILVERDEMO'
```

Screen copied from the SILVERDEMO library to the DEMO library

```
INSERT INTO PSVDDLO (LANGID, USRSPC, USRSPCLIB, PROPERTY,
STRING, TRANSLAT) SELECT LANGID, USRSPC,'DEMO', PROPERTY,
STRING, TRANSLAT FROM PSVDDLO WHERE USRSPC= 'SDDMLANG' and
USRSPCLIB='SILVERDEMO'
```

Translation menu

Users can access the translation menu by right-clicking the task bar.

The list of all possible translations is displayed.

When the user selects a language, the result is the same as calling sdTranslateAll.

SdRtvMsg

Prototype:

```
d sdRtvMsg...
d          pr          1000a  varying
d msgfile          20a  const
d msgid           7a  const
d msgdta          1000a  const options(*varsize:*nopass)
d msgdtalen        10i 0  const options(*nopass)
```

Description:

The sdRtvMsg function enables the 1st level text of a message to be obtained, possibly by using substitute data. The function uses the QMHRTVM API for this purpose.

The compulsory MsgFile parameter, is a qualified object name, object name in 10 characters, then library name in 10 characters, representing a message file (type *MSGF). SdRtvMsg, because it uses QMHRTVM, is liable to return a message from

another message file if message file substitution (OVRMSGF) is active. The special values *CURLIB and *LIBL can be used as library names.

The compulsory MsgId parameter specifies the identifier of the message to be extracted.

The optional MsgDta parameter contains the substitution values for the message. The format of this parameter depends on the message extracted.

The optional MsgDtaLen parameter indicates the length of the MsgDta parameter. If this parameter is not specified or if its value is zero, the MsgDta parameter is ignored.

The value returned contains the formatted 1st level text.

If the message identifier is not found, the value returned contains the 1st level text of the QSYS/QCPFMSG CPF2419 message.

Any other error encountered during the QMHRTVM call is returned to the caller as an exception.

Chapitre 19. Image category functions

sdSetImg

To load an image from the IFS to a component able to receive an image (CImage, CBitBtn, CspeedButton, etc.).

Prototype:

d sdSetImg	pr	10i 0	
d form			like(tWHandle)
d component		30	varying value
d propertyPath		256a	varying value
d TypeImage		3	value
d path		256a	varying value

Description:

The sdSetImg function enables an image to be loaded to a TBitmap, TPicture or TIcon type object.

TypeImage accepts the following values: 'bmp', 'jpg' or 'ico' .

The file indicated by the Path parameter must be of the type indicated by the TypeImage parameter.

Type of object	Possible values of TypeImage
TBitmap	'bmp'
TIcon	'ico'
TPicture	'bmp', 'jpg' or 'ico'

Return values:

0: All OK

-1: The file specified does not exist or you do not have the rights required. The image is not loaded.

-2: The file sent does not appear to be of the type indicated or the component cannot load this type of image.

(You are sending an image whose format does not correspond to the format indicated by the TypeImage parameter.)

Parameters

Component: component containing the image

PropertyPath: property of the component able to contain an image. This property must be one of the following types: TBitmap, TPicture or TIcon.

TypeImage: indicates the type of image sent.

path: IFS path of the image to be loaded

Example 1:

```
C          callp      sdSetImg (form:'BitBtn1':'Glyph':  
C          'bmp': '/Images/test.bmp')
```

Example 2:

```
PLoadImage      B  
  
D              PI  
DBookID          4  0  
DImageName       s      1000    varying  
D ret            s      10i  0  
C              eval      ImageName='/Home/Images/'+  
C              SdIntToStr(BookID)+' .jpg'  
C              eval      Ret  = sdSetImg(F1:'Image1':  
C              'Picture':'jpg':ImageName)  
*  
C              if        ret <> 0  
C              callp      sdSet      (F1:'Image1':  
C              'Picture.graphic':'Null')  
C              endif
```

Example 3:

Loading of an icon in a form:

```
C          callp      sdSetImg(form:'*FORM':'Icon':  
C          'ico': '/Images/test.ico')
```

Example 4:

Loading of a bmp in a CImage:

```
C          callp      sdSetImg (form:'Image1':'picture.Bitmap':  
C          'bmp':'/Images /test.bmp')
```

or

```
C          callp      sdSetImg (form:'Image1':'picture':  
C          'bmp':'/Images /test.bmp')
```

Example 5:

Loading of a jpg in a CImage:

```
C          callp      sdSetImg (form:'Image1':'picture':  
C          'jpg':'/Images /test.bmp')
```

sdGetImg

Prototype:

```
d sdGetImg      pr          10i 0  
d pform          5u 0  
d component      30a  varying value  
d propertyPath   256a  varying value  
d path           256a  varying value
```

Description:

Enables an image stored in a component to be retrieved and saved in a file on the IFS.

The propertyPath parameter is a property that must be of one of the following types: TBitmap, TPicture or TIcon.

Return value:

0: All OK

-1: The image in the component is empty and therefore the file was not created/modified on the IFS.

-2: The name of the IFS file is incorrect or you do not have the required rights

Parameters:

pForm: Identifier of the window containing the "component".

Component: Component containing the image.

propertyPath: TPicture, TBitmap or TIcon type property containing the image.
 path: IFS path to store the retrieved image

For example:

```
PWriteImage      B
D                PI
D Ret            s          10i 0
D NameImg        s          500   varying
C                eval      nameImg = '/Home/Images/' +
C                                sdIntToStr(IDBOOK) + '.jpg'
C                eval      Ret = sdGetImg(Fl: 'Image1':
C                                'picture': NameImg)
C                *
C                if        (ret = -1) and access(NameImg:W_OK)=0
C                callp      UnLink(NameImg)
C                endif
```

sdLoadFromFile

Prototype:

```
d sdLoadFromFile pr          10i 0
d pform          5u 0 const
d Component       30   varying value
d FileName       1000  varying value
```

Description:

Enables an image to be loaded from the client's disk.
 Applies to Tpicture and TString type objects.

For example:

```
PSelectImg      B
D                PI
D path          s          1000   varying
C                if        sdSelectFile('*.jpg |*.jpg':
C                                Path:seOpenPicture)
C                callp      sdLoadFromFile(Fl: 'Image1.picture':
C                                path)
C                endif
P                E
```

sdAssignTo

Prototype:

```
d sdAssignTo      pr
d  pform1          5u 0
d  Object1         50a  varying value
d  pform2          5u 0
d  Object2         50a  varying value
```

Description:

Copies the content of the Object1 object into Object2. Object1 and Object2 must be of the same type.

For example:

Copies the TPicture object of a CImage into another.

```
PEvtGdeImg      B
D               PI
* -- Insert your code here ...
C               if      FicheImg = 0
C               eval    FicheImg =sdCreateform('*LIBL/FICHEIMG')
C               endif
C               callp    sdAssignTo(Fiche: 'Image1.picture':
C                               FicheImg: 'Image1.picture')
C               callp    sdShowModal(FicheImg)
```

Note:

This function can be applied to objects of the following types:

*TChartSeries, TBitmap, TBrush, TFont, TIcon, TListColumn,
TListColumns, TListItems, TNode, TNodes, TPicture, TString*

sdAddImg

Prototype:

d	sdAddImg	pr	10i	0
d	form		like(tWHandle)	const
d	component		30	varying value
d	TypeImage		3	value
d	path		256a	varying value

Description:

Enables an image to be added to a CReplImage component or to a CimageList component.

For example:

C	callp	sdAddImg(F2:'Img1':'jpg':
C		BaseImage+'Img'+%char(A_IDBOOK)+' .jpg')

Return values:

0: All OK

-1: The specified file does not exist or you do not have the required rights. The image is not loaded.

-2: The file sent does not appear to be of the type specified, or the component cannot load this type of image.

(You are sending an image whose format does not correspond to the format specified by the TypeImage parameter.)

Chapitre 20. List of windows and events

sdGetFormCount

Prototype:

d sdGetFormCount	pr	5u 0
------------------	----	------

Description:

This function tells you the number of current windows in the application.

sdGetFormHandle

Prototype:

d sdGetFormHandle...		
d	pr	5u 0
D ind		5u 0

Description:

This function tells you the value returned by the sdCreateForm function during window creation.

sdGetFormName

Prototype:

d	sdGetFormName	pr	11a	Varying
d	form		5u	0

Description:

This function tells you the name of the window on the client side.

sdGetFormLib**Prototype:**

d	sdGetFormLib	pr	10	
d	form		5u	0

Description:

This function tells you the library from which the window originates.

sdGetFormObj**Prototype:**

d	sdGetFormObj	pr	10	
d	form		5u	0

Description:

This function tells you the name of the usrspc from which the window originates.

sdGetEventCount

Prototype:

d	sdGetEventCount...	
d	pr	5u 0
D	PForm	5u 0

Description:

This function tells you the number of events for a window.

sdGetEventName

Prototype:

d sdGetEventName	pr	48
D PForm		5u 0
D EventNumber		5u 0

Description:

This function tells you the name of an event in a window.

sdGetEventProc

Prototype:

d sdGetEventProc	pr	*	procptr
D PForm		5u 0	
D EventNumber		5u 0	

Description:

This function tells you the address of the function associated with an event.

For example:

```
D i          s          5u 0
D j          s          5u 0
D handle     s          5u 0
D Lib        s          10
D Obj        s          10
D name       s          8
D eventcount s          5u 0
D eventname  s          48   varying
D eventProc  s          *   procptr
C           for      i = 1 to sdGetFormCount
C               eval  Handle=sdWinHandle(i)
C               eval  Lib=sdGetFormLib(Handle)
C               eval  Obj=sdGetFormObj(Handle)
C               eval  Name=sdGetFormName(Handle)
C               for   j = 1 to sdEventCount(handle)
C                   eval  eventname = sdEventName(handle:j)
C                   eval  eventProc = sdEventProc(handle:j)
C               endfor
C           endfor
```

| *Note :These functions do not query the client part.*

Chapitre 21. Colors

Designer

You can modify colour type properties in Designer using a dialogue box, as shown below.



Figure 20

The result is either a pre-determined colour (clBlue, clRed, etc.) or a hexadecimal value of 4 bytes.

The first three bytes represent the RGB intensity of blue, green and red respectively. Value 00FF0000 represents a full intensity pure blue, 0000FF00 a full intensity pure green and 000000FF a full intensity pure red. 00000000 represents black and 00FFFFFF white.

If the most significant byte is zero (00), the colour obtained is the closest to it in the system palette. If the most significant byte is (01), the colour obtained is the closest to it in the currently produced palette. If the most significant byte is two (02), the colour obtained is the closest to it in the logic palette of the current peripheral context.

Modification during execution

To modify a colour type property during execution, you have to use RGB values.

The three examples below assign the colour blue:

C	<code>callp</code>	<code>sdset(f1: 'Tree1': 'color': '0xFF0000')</code>
C	<code>callp</code>	<code>sdSetInt(F1: 'tree1': 'color': X'FF0000')</code>
C	<code>callp</code>	<code>sdSetInt(F1: 'tree1': 'color': 16711680)</code>

The first two solutions are recommended since they are the most legible.

Choice of colour

To allow the user to choose the colour, use the `sdSelectColor` function.

This function displays the same dialogue box as in Designer.

The `sdSelectColor` function returns a Boolean indicating whether or not the user has chosen a colour. The colour chosen by the user is then indicated in the function parameter. See (**`sdSelectColor`**).

Pre-defined colours

Some colours can be indicated by name.

In this case, during execution, use the following code:

C	<code>callp</code>	<code>sdset(f1: 'editnum1': 'color': 'clblue')</code>
---	--------------------	---

Note:

If you query the value of a colour property, you will obtain the numeric value and not the associated text, i.e. 16711680 for clBlue.

Caution: colours like `clInfoBgk`, `clBtnFace`, etc. depend on PC configuration.

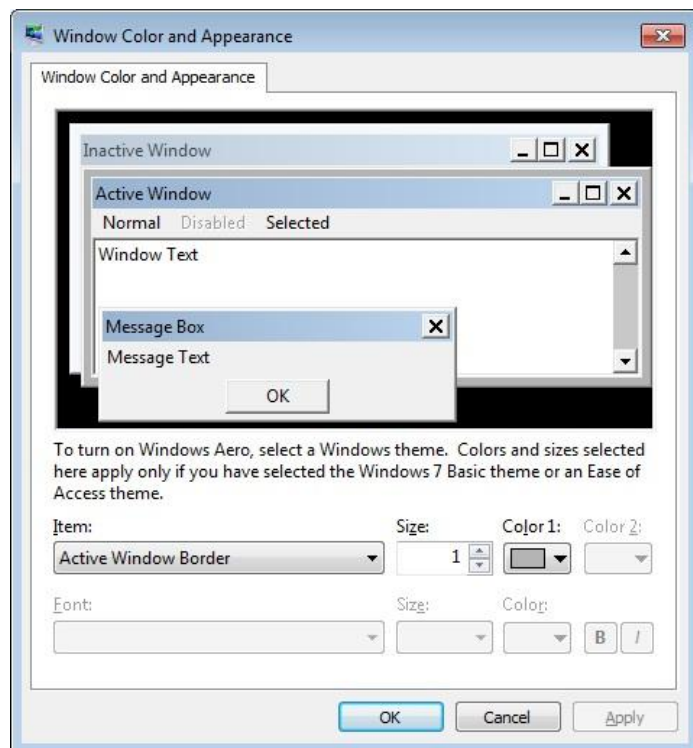


Figure 21

Summary table:

Value	Meaning
ClAqua	Water
ClBlack	Black
ClBlue	Blue
ClCream	Cream
ClDkGray	Dark grey
ClFuchsia	Fuchsia
clGray	Grey
clGreen	Green
clLime	Lime green
clLtGray	Light grey
clMaroon	Maroon
clMedGray	Medium grey
clMoneyGreen	Mint green

clNavy	Navy blue
clOlive	Olive green
clPurple	Purple
clRed	Red
clSilver	Silver
clSkyBlue	Sky blue
clTeal	Teal
clWhite	White
clYellow	Yellow
clInfoText	Windows 95 or NT 4.0 only: text colour of the tip windows
clInfoBk	Windows 95 or NT 4.0 only: background colour of the tip windows
clGradientActiveCaption	Windows 98 or 2000 only: colour of the right side in the colour gradient of the title bar of an active window. clActiveCaption specifies the colour of the left side
clGradientInactiveCaption	Windows 98 or 2000 only: colour of the right side of the colour gradient of the title bar of an inactive window. clInactiveCaption specifies the colour of the left side
clDefault	Default colour for the control to which the colour is assigned
clBackground	Background colour of the Windows desktop
clActiveCaption	Colour of the title bar of the active window
clInactiveCaption	Colour of the title bar of inactive windows
clMenu	Background colour of menus
clWindow	Background colour of windows
clWindowFrame	Colour of window frames
clMenuText	Colour of menu text
clWindowText	Colour of text in windows
clCaptionText	Colour of text in the title bar of the active window
clActiveBorder	Colour of the border of the active window
clHighlight	Background colour of the selected text
clAppWorkSpace	Colour of the application work space
clHightlightText	Colour of the selected text

clBtnFace	Colour of the button face
clBtnShadow	Colour of the button shadow
clGrayText	Colour of a grey text
clScrollBar	Colour of the scroll bar
clBtnText	Colour of the button text
clInactiveCaptionText	Colour of the title bar text of inactive windows
clBtnHighlight	Colour of a selected button
cl3DDkShadow	Windows 95 or NT 4.0 only: dark shadow of 3D items
cl3DLight	Windows 95 or NT 4.0 only: light colour of 3D items (for the sides facing the light source)

Chapitre 22. Anchoring components

Introduction

This chapter shows how to anchor a window in a panel upon execution.

Basic example

Create a window called SDDMDOCK1.
Add a panel called panelLeft.
Set the panelLeft component's align property to allLeft.
Set the panelLeft component's DockSite property to true.

Create a second window called SDDMDOCK2.

Change SDDMDOCK2's dragMode property to dmAutomatic.
Change SDDMDOCK2's dragKind property to dkDock.

Call SDDMDOCK2 from SDDMDOCK1.
The SDDMDOCK2 window can be anchored or un-anchored in the panelLeft component.

Add a CPanel type component called panelRight, aligned on the right in SDDMDOCK1.
Change panelRight's DockSite property to true.

It is now possible to move SDDMDOCK2 from one panel to the other.

| *Comment 1:*

| You can anchor all *TControl* type components in the same way.

| *Comment 2:*

| *All TWinControl type components can be used as anchoring sites (CPanel, CPageControl, CScrollBar, etc.)*

Floating window

In the previous example, the `SDDMDOCK2` window can be moved from one panel to the other, but the window can also be unanchored.

If the unanchored component is not a window but a panel for example (`dockablePanel`), it will then be in an autonomous window. If this window is closed, you just have to change the `dockablePanel`'s `visible` property to `true` to make it reappear.

You can prevent the anchorable component from becoming a floating component using the `onUnDock` local event.

The example below prevents the user from unanchoring a component from `panelLeft` and leaving it to float

```
procedure PanelLeft_OnUnDock (Sender: TObject; Client: TControl;
NewTarget: TWinControl; var Allow: Boolean);
begin
    allow := newTarget <> nil;
end;
```

Improvement

We now want to make it possible to anchor a component to the left of the `SDDMDOCK1` window, but without the `panelLeft` component appearing if no components are anchored in it.

Set the `panelLeft` `width` property to zero.

Change the `panelLeft` `onDockOver` local event as follows:

```
procedure PanelLeft_OnDockOver (Sender: TObject; Source:
TDragDockObject; X: Integer; Y: Integer; State: TDragState; var Accept:
Boolean);
var
  ARect: TRect;
begin
  Accept := true;
  if (Source.DockRect.Right = Source.DockRect.left) then
  begin
    ARect := Source.DockRect;
    ARect.Right := ARect.left + 100;
    source.dockRect := ARect;
  end;
end;
```

Change the panelLeft onDockDrop local event as follows:

```
procedure PanelLeft_OnDockDrop (Sender: TObject; Source:
TDragDockObject; X: Integer; Y: Integer);begin
  if(exdock1.panelLeft.width = 0) then
  begin
    exdock1.panelLeft.width := source.dockRect.right -
source.dockRect.left;
  end;
end;
```

Change the panelLeft onDockDrop local event as follows:

```
procedure PanelLeft_OnUnDock (Sender: TObject; Client: TControl;
NewTarget: TWinControl; var Allow: Boolean);
begin
  allow := newTarget <> nil;
  if(allow) and (exdock1.panelLeft.DockClientCount =1) then
  begin
    exdock1.panelLeft.width := 0;
  end;
```

end;

Chapitre 23. Drag and drop operations

Introduction

A drag and drop operation involves interaction between two components: the source component which starts the drag and drop and the target component which receives the drag and drop.

The same component can be both source and target.

The source and target can be in two different windows.

The source and target must be in the same process.


A drag and drop can be operated from and to any component, however, it is more common to operate drag and drops with components having a list of values (ClistBox, CtreeView, ClistView, CSFL etc.).

DragMode

The DragMode property concerns the source component.

There are two possible values for the DragMode property: dmManual and dmAutomatic.

If the DragMode property is dmAutomatic, then as soon as the user clicks the component, a drag and drop operation is commenced.

The cursor automatically changes as follows: 

The drag and drop operation stops when the operator releases the left mouse button.

If the DragMode property is dmManual, the sdBeginDrag function must be called in the onMouseDown event.

```
d sdBeginDrag      pr
d pform             5u 0 const
d component         30   varying value
d Immediate         N   value
d TreshOld          10i 0 value
```

Commenté [M1]: Threshold ?

The Immediate parameter enables the user to indicate whether the drag and drop operation is launched immediately or if it is launched when the user starts to move the mouse.

If the Immediate parameter is *on, the Threshold parameter enables the user to indicate the number of pixels that the mouse must be moved before starting the drag and drop operation. The -1 value enables the default value configured on the PC to be used.


OnDragOver

Once the drag and drop operation has started, the onDragOver event is triggered for all the components that the mouse passes over.

The event parameters are:

d	Win	5u	0
d	evt	48a	
d	drag	n	
d	name	100a	varying
d	PosX	10i	0
d	PosY	10i	0
d	WinSrc	5u	0

Drag	Modifiable parameter. Set to *ON to enable triggering of the OnDragDrop event (and the cursor will be a drag and drop cursor)
Name	Name of the component starting the drag and drop
PosX,PosY	Mouse position
WinSrc	Number of the window of the component starting the drag and drop

Setting the drag parameter to false changes the cursor to  to inform the user that drag and drop operations are not possible on this component.
The other parameters offer further information about the component starting the drag and drop and the location of the dragover.

OnDragDrop

The onDragDrop event is started during a drag and drop operation when the user releases the left mouse button and the drag parameter is set to *on in the component's previous onDragOver event.

d	PevtInf		*	const options(*nopass)
d	parm	ds		based(pevtinf)
d	Win		5u	0
d	evt		48a	
d	name		100a	varying
d	PosX		10i	0
d	PosY		10i	0
d	WinSrc		5u	0

	Name	Name of the component starting the drag and drop
	PosX, PosY	Mouse position
	WinSrc	Number of the window of the component starting the drag and drop.

The event parameters offer further information about the component starting the drag and drop and the drop location.

Examples

Drag and drop of a ClistBox to a ClistBox

```
~/EVENT ListBox2_OnDragOver
,* -----*
,* Description :                               *
,* -----*
D Parameters      ds              based(pevtinf)
D Win              5u 0
D Evt              48a
,* Modifiable
D Drag              n
,* Non modifiable
D Name              100a  varying
D PosX              10i 0
D PosY              10i 0
,*
c                  eval      drag = *on
```

```

~/EVENT ListBox2_OnDragDrop
, * -----*
, * Description : *
, * -----*
D Parameters      ds          based(pevtnf)
D Win              5u 0
D Evt              48a
D Name            100a  varying
D PosX            10i 0
D PosY            10i 0
, *
d sel             s          10i 0
d Dest            s          10i 0
d chaine          s          30a
d cle             s          30a
d List            s          *
d i               s          10i 0
C                 eval      Dest = sdItemAtPos(Win:'ListBox2':
C                 PosX:PosY:*ON)
C                 eval      List=sdGetList(Win:Name)
C                 for       i =0 to sdGetListCount(List)-1
C                 if        sdGetListSelected(List:i)
C                 eval      chaine = sdgetListLabel(List:i)
C                 eval      cle = sdgetListKey(List:i)
C                 if        Dest ==-1
C                 callp      sdSlAdd(F1:'ListBox2':'Keys':cle)
C                 callp      sdSlAdd(F1:'ListBox2':'Items':chaine)
C                 else
C                 callp      sdSlInsert(F1:'ListBox2':'Keys':Dest:Cle)
C                 callp
sdSlInsert(F1:'ListBox2':'Items':Dest:Chaine)
C                 eval      Dest=Dest+1
C                 endif
C                 endif
C                 endfor
C                 callp      sddelselected(Win:name)
C                 callp      sdFreeList(List)

```

Drag and drop of a ClistView to a ClistView

```

~/EVENT TreeView1_OnDragOver
, * -----*
, * Description : *
, * -----*
D Parameters      ds          based(pevtnf)
D Win              5u 0
D Evt              48a

```



```
,* Modifiable
D Drag          n
,* Non modifiable
D Name          100a   varying
D PosX          10i 0
D PosY          10i 0
,*
C              eval    drag = (Name ='TreeView1')

~/EVENT TreeView1_OnDragDrop
,* -----*
,* Description :
,* -----*
D Parameters    ds          based(pevtnf)
D Win          5u 0
D Evt          48a
D Name          100a   varying
D PosX          10i 0
D PosY          10i 0
,*
DNode1         s          10i 0
DNode2         s          10i 0
C              eval    node1=sdgetInt(F1:'TreeView1':
C                      'FirstSelected')
C              eval    node2=sdNodeAt(F1:'treeView1':PosX:PosY)
C              if      Node1 = -1  or Node1=Node2
C              return
C              endif
C              if      Node2=-1
C              callp    sdMoveNode(F1:'TreeView1':Node1:0:
C                      'naAdd')
C              else
C              callp    sdMoveNode(F1:'TreeView1':Node1:Node2:
C                      'naAddChild')
C              endif
```

DragCursor and CCustomCursor

The cursor associated with the drag and drop operation can be modified.
In this example, the cursor is changed when the user presses Control at the same time as the onMouseDown:

```
D Parameters    ds          based(pevtnf)
D Win          5u 0
D Evt          48a
D Button       10i 0
```

```

D Shift          32
D X              10i 0
D Y              10i 0
, *
c                if      sdIsInSet(shift:3)
c                callp    sdSetInt(F1: 'TreeView1': 'dragCursor':1)
c                else
c                callp    sdSetInt(F1: 'TreeView1': 'dragCursor':-12)
c                endif
c                callp    sdBeginDrag(F1: 'TreeView1': *off:-1)

```

The value assigned to the dragCursor property must be between -1 and -22 (predefined cursors) or a defined value for a customized cursor. A customized cursor is defined using the CCustomCursor component.

By interrogating the dragCursor value at the time of the drop, the user can determine how the drag and drop was initiated.

It is thus possible to distinguish between a movement or a copy, depending on whether or not Control was activated on onMouseDown.

OnDragOverEx, OnDragDropEx

To simplify the drag and drop operation to a CSFL component, the CSFL component has two extra events.

These two events work in the same way as the OnDragOver and OnDragDrop events, but with different parameters.

The OnDragOverEx and OnDragDropEx parameters enable the user to find out the cell over which the mouse was dragged very rapidly.

OnDragOverEx parameters

```

d PevtInf          *      const options(*nopass)
d parm            ds      based(pevtinf)
d Win              5u 0
d evt              48a
d drag             n
d WinSrc           5u 0
d name             100a   varying
d Row              10i 0
d Col              100a   varying

```

Drag	Modifiable parameter. Set to *ON to enable triggering of the OnDragDrop event (and the drag and drop cursor)
WinSrc	Number of the window of the component initiating the drag and drop
Name	Name of the component initiating the drag and drop.
Row	Row of the drag
Col	Column of the drag

OnDragDropEx parameters

```
d PevtInf          *    const options(*nopass)
d parm            ds    based(pevtinf)
d Win              5u 0
d evt             48a
d WinSrc          5u 0
d name            100a   varying
d Row             10i 0
d Col             100a   varying
```

WinSrc	Number of the window of the component initiating the drag and drop
Name	Name of the component initiating the drag and drop
Row	Row of the drop
Col	Column of the drop

Chapitre 24. Screen size adaptation

Introduction

Screen size adaptation is an important and complex matter. This subject deserves a chapter of its own in order to see every possibility.

Size changeable by user

To enable the user to change a form size, the `borderStyle` property must be `bsSizeable` (default value)

The `borderIcons` property, if it contains `biMaximize` value allows the user to display a full screen form.

Align, anchors

The simplest way to adapt to screen size modifications is to use `align` and `anchors` properties.

When `align` property is `alBottom`, component is displayed in all bottom of its container.

When `align` property is `alTop`, component is displayed in all top of its container.

When `align` property is `alRight`, component is displayed in all right of its container.

When `align` property is `alLeft`, component is displayed in all left of its container.

When `align` property is `alClient`, component is displayed in all space left in the container.

You can for example have a panel with `align = alBottom` and a grid with `align = alClient`.

Panel will fill the bottom of the form, and the grid will fill the space left.

Anchors

In previous example, we can add two buttons in the bottom panel.

By default, these two buttons have their anchors property with values [akLeft,akTop]

This means that the component will stay à the same distance of left edge and top edge of its container.

You can change to [akRight,akBottom] so the component stay at the same distance from bottom and right edges.

If you change anchors so that it contains for example akRight and akLeft, the size of the component will change so that it can stay at the same distance of right and left edges.

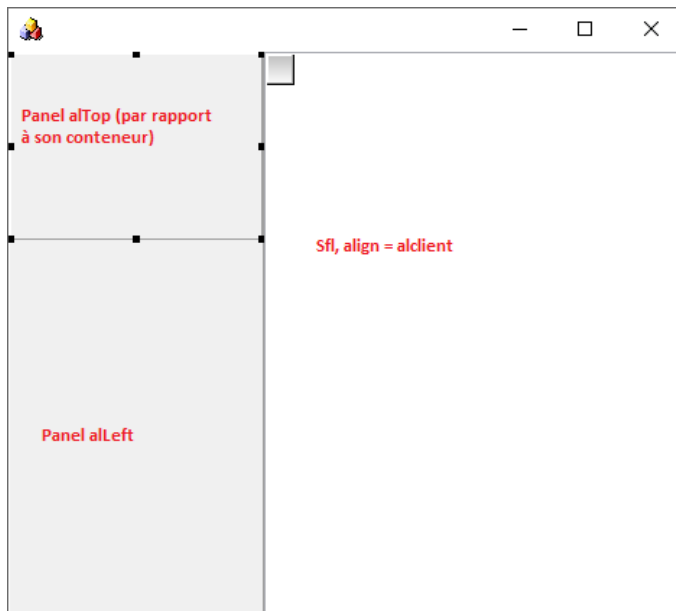
This can be a solution to have a component that grow in size without touching edges.

For a component, use align or anchors, but not both.

You can use align for some components and anchors for some others (like in our example)

Nested containers

To create a complex display, you can for example have a panel with align = left, and inside this panel put a panel with align = alTop for example.

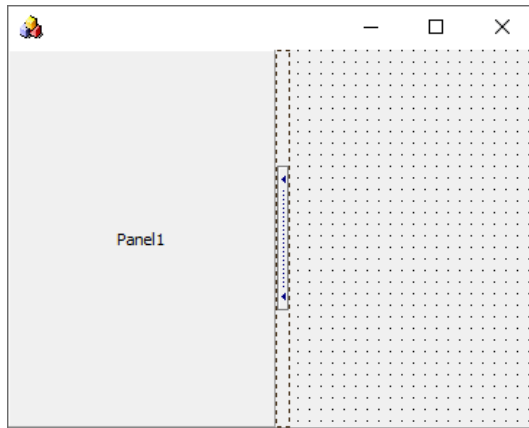


Splitter component

When you use align property to split a screen , for example with an area to the top, an area to the bottom, and an area to the left, it's the développeur that decides size of top, bottom and left areas.

So that the user can change size of this areas (and the left space area with align = alClient) you can add a CSplitter component.

If align property of CSplitter is alLeft, the splitter will be next to the area with align = alLeft. The users will be able to change the size of the left area.



Gridpanel, flowPanel, relativePanel

Another possibility to adapt to size changes, is to use components GridPanel, FlowPanel et RelativePanel

With this technic, it's panels that handle display of the components.
Left and top values of the components inside are not used.

The panel changes display of the components each time it is resized.

This three components can use nested technics.

CScreen component

CScreen component allows to retrieve screen size.

You can depending of the size of the user's screen, move/resize components at runtime by program.

onResize event.

onResize event allows to detect that the user has changed a form's size. You can then move/resize components by program.

Chapitre 25. ADO

Introduction

ActiveX Data Objects is a Microsoft technology supplying an access interface to data in the Windows environment.

This technology enables access to a remote database.

Data access is made from the client and not from the SilverDev server.

OLE DB drivers must therefore be installed on the client workstation running the SilverDev application.

The set of components is in the ADO palette.

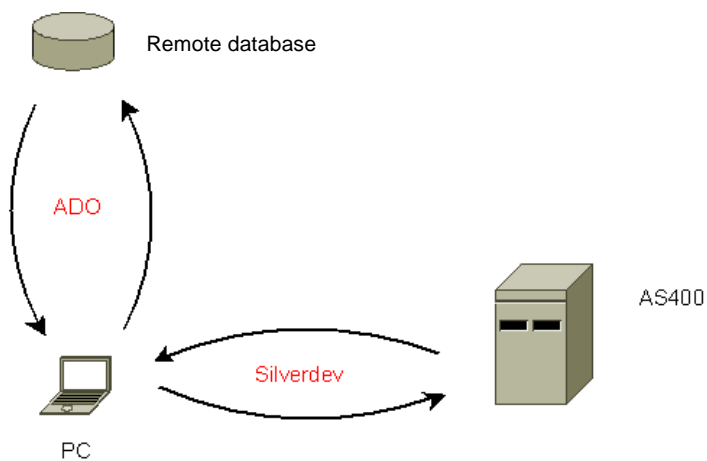


Figure 22

CADOConnection

The CADOConnection component enables definition of a connection to an OLE driver.

The component must then be referenced by the other components.

The key property of this component is the ConnectionString property.

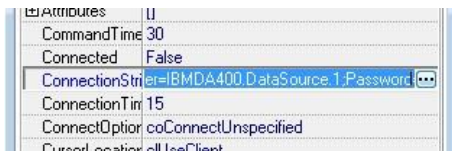


Figure 23

A specific editor is provided for this property.

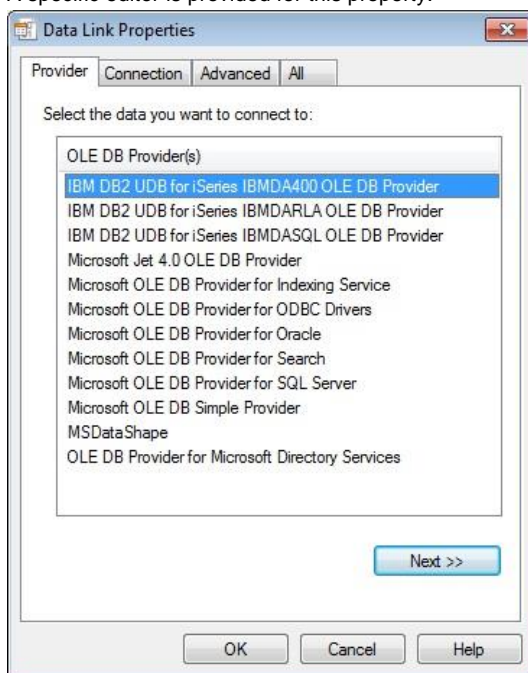


Figure 24

CAdoQuery

The other components of the ADO palette have a ConnectionString property but it is easier to refer them to a CAdoConnection component via the Connection property.

The CAdoQuery component enables a query to be made on the remote database.

The result of this query can be obtained using the sdGetSFL function.

This function, created for the CSFL component, also works for ADO type components. The Column parameter will be the name of a field.

For example:

```

Dsfl          s          *
Dnomcli       s          40
D I           s          10i 0
* open connection
C              callp      sdSetBool(F1:'adoconnection1':'connected'
C                               *on)
* populate the query properties
C              callp      sdSlClear(F1:'adoquery1':'sql')
C              callp      sdSlAdd(F1:'adoquery1':'sql':
C                               'select * from EXAMPLE.CUSTOMERS')
* Activate query
C              callp      sdSetBool(F1:'adoquery1':'active'
C                               *on)
* Get the result set returned by the query
C              eval       sfl=sdGetSfl(F1:'adoquery1')
C              for        i= 1 to sdGetSflLines(sfl)
C              eval       nomcli=sdGetsflStr(sfl:'CUSTNAME':i)
C              endfor
* Free ressources previously allocated by sdGetSFL
C              callp      sdFreeSfl(sfl)

```

Example 2:

```

DMsg          s          65000    varying
Dret          s          10i 0
C              callp      sdSetBool(F1:'adoconnection1':'connected'
C                               *on)
C              callp      sdSlClear(F1:'adoquery1':'sql')
C              callp      sdSlAdd(F1:'adoquery1':'sql':
C                               'insert into EXAMPLE.CUSTOMERS ' +
C                               'values(50,1,''test'',NULL,NULL,'+
C                               'NULL,NULL,NULL,NULL)')
C              eval       ret=sdExecSql(F1:'AdoQuery1':msg)
C              callp      sdshowmessage(%char(ret))
C              callp      sdshowmessage(msg)

```

Chapitre 26. Exit points

To associate a program with an exit point, use the WRKREGINF command, then option 8.

SilverDev provides the following exit points:

SVDCTRLCPL

Enables screen compilation to be checked.

Exit point format:

CTRCPL01

Parameters:

PGM	PARM(&USER &LIB &USRSPC &RET &TEXTRET)
DCL	VAR(&USER) TYPE(*CHAR) LEN(10)
DCL	VAR(&LIB) TYPE(*CHAR) LEN(10)
DCL	VAR(&USRSPC) TYPE(*CHAR) LEN(10)
DCL	VAR(&RET) TYPE(*CHAR) LEN(1)
DCL	VAR(&TEXTRET) TYPE(*CHAR) LEN(256)

The User parameter is the current login profile.

The Lib parameter is the library in which the screen will be compiled.

The Ustrpc parameter is the name of the screen that will be compiled.

The Ret parameter should return Y or N (N will block compilation).

The TextRet parameter is a text to be displayed in the designer if Ret is N.

SVDEXCEPT

Enables interception of a exception/error.

Exit point format:

EXCEPT01

Parameters:

C	*entry	plist		
C		parm	jobName	10
C		parm	userName	10
C		parm	jobnumber	6
C		parm	stopJob	1

Set the stopJob parameter to 'Y' to stop the job.

If you do not change the stopJob parameter, job status will probably remain MSGW.

Chapitre 27. Interaction with external programs.

Several solutions exist to launch a silverdev application from another program or to launch another program from a silverdev application. This chapter resumes all these solutions.

For every solution, Silverdev and the external program run independently.

Launch a silverdev applicatoin from a pc program.

To launch a silverdev application, you have to call the launcher.exe program. During silverdev installation, the environment variable SVDHOME is set. The environment variable contains the the directory of the launcher.exe program.

You can (and you must)pass a parameter to the launcher.exe program. This parameter has to be formed as an url with key/value pairs.

Example (Values are in green) :

```
silverdev:host=192.168.0.80&port=7003&application=/examples/books/sddmbks  
&user=TOTO&pwd=1234
```

Les clefs possibles sont :

Key	Mandatory	Description
Host	Yes	Server ip address
Port	Yes	Server tcp address
Application	Yes	Silverdev application to launc
User	No	User to connect
Pwd	No	Password
Caption	No	Title displayed in task bar
Index	No	Internal image index
Md5	No	Md5 hash code of the icon
UserData	No	Value you can retrieve with sdGetUserData

| Not :

When a value has characters such as space or equal, you have to encode the value as a url.

Example :The value tic & tac has to be encoded as : tic%20%26%20tac

Launch a silverdev application from a 5250 application

To launch a silverdev application from a 5250 application, you can use the strpcmd command.

If you need to launch a silverdev application with parameters, you can do as follows:

The idea is to create dynamically a file on the ifs with all the informations of the program to launch, and to use the strpcmd command to launch the silverdev application.

Here the full example :

```
Hbnddir('QC2LE')
Hdftactgrp(*no)
d O_RDONLY      S          10i 0 inz(1)
d O_WRONLY      S          10i 0 inz(2)
d O_RDWR        S          10i 0 inz(4)
d O_CREAT        S          10i 0 inz(8)
d O_EXCL         S          10i 0 inz(16)
d O_TRUNC        S          10i 0 inz(64)
d O_TEXTDATA     S          10i 0 inz(16777216)
d O_CODEPAGE     S          10i 0 inz(8388608)
d O_INHERITMODE  S          10i 0 inz(134217728)

d S_IRUSR        S          10i 0 inz(256)
d S_IWUSR        S          10i 0 inz(128)
d S_IXUSR        S          10i 0 inz(64)
d S_IRWXU        S          10i 0 inz(448)
d S_IRGRP        S          10i 0 inz(32)
d S_IWGRP        S          10i 0 inz(16)
d S_IXGRP        S          10i 0 inz(8)
d S_IRWXG        S          10i 0 inz(56)
d S_IROTH        S          10i 0 inz(4)
d S_IWOTH        S          10i 0 inz(2)
d S_IXOTH        S          10i 0 inz(1)
d S_IRWXO        S          10i 0 inz(7)

D access         pr          10i 0 extproc('access')
D filename       *          value options(*string)
d
D amode           10i 0 value

d open           pr          10i 0 extproc('open')
```

```

d filename          *   value options(*string)
d openflags         10i 0 value
d mode              10u 0 value options(*nopass)
d codepage          10u 0 value options(*nopass)

d close             pr   10i 0 extproc('close')
d filehandle        10i 0 value

d write             pr   10i 0 extproc('write')
d filehandle        10i 0 value
d datatowrite       *   value options(*string)
d nbytes            10u 0 value

d Qcmdexc           pr   extpgm('QCMDEXC')
d command           1024A const options(*varsize)
d length            15P 5 const

Dfd                 s    10i 0
Dpath                s    250  varying
DBuffer              s    2000 varying
DlnBuffer            s    10i 0
DpBuffer             s    *   inz(%addr(buffer))
DplnBuffer           s    *   inz(%addr(lnbuffer))
Dcmd                 s    500  varying
d flags              s    10i 0
d mode               s    10i 0

d tIconv            ds
d tIconvRc           10i 0
d tIconvCd           10i 0          dim(12)

d fromCode           ds
d fcCCSID            10i 0
d fcConvert           10i 0          inz(0)
d fcShiftState       10i 0          inz(0)
d fcInputLen         10i 0          inz(0)
d fcError            10i 0          inz(0)
d                    8   inz(*allx'00')

d toCode             ds
d tcCCSID            10i 0
d tcConvert           10i 0          inz(0)
d tcShiftState       10i 0          inz(0)
d tcInputLen         10i 0          inz(0)
d tcError            10i 0          inz(0)
d                    8   inz(*allx'00')

d iconvOpen          pr   like(tIconv) extproc('QtgIconvOpen')

```



```

d   fCode          *   options(*string) value
d   tCode          *   options(*string) value

d   iconvClose      pr               extproc('iconv_close')
d   iconv_t         like(tIconv) value

d   iconv           pr               10U 0 extproc('iconv')
d   iconv_t         like(tIconv) value
d   from            *
d   fromLen         *   value
d   to              *
d   toLen           *   value
D   cpt             s               10u 0 inz(0)
D   launchpath      s               250   varying
D   fileName        s               250   varying

/free
pbuffer=%addr(buffer) + 2;
fileName = %char(cpt)+'.app';
path = '/silverdev/applications/auto/'+fileName;
dow access(path:0) = 0;
  cpt = cpt + 1;
  fileName = %char(cpt)+'.app';
  path = '/silverdev/applications/auto/'+fileName;
enddo;
flags = O_WRONLY + O_CREAT + O_TRUNC+ O_CODEPAGE;
mode = S_IRUSR + S_IWUSR + S_IXUSR;
fd = open(path:flags:mode:819);
Buffer = '[001]'+X'0d25' + 'Description = MADESCRIPTION'+X'0d25' +
'Launch = call MALIB/MONPGM parm(''MONPARAMETRE'')' +X'0d25'+
'Caption = MONCAPTION' +X'0d25' + 'IconIdx = 1' +X'0d25' +
'MD5 = 26C0066C15293712903ECC1E20D4CEA1';
fcCCSID = 0;
tcCCSID = 1252;
tIconv = iconvOpen(%addr(toCode):%addr(fromCode));
lnBuffer = %len(buffer);
iconv(tIconv:pbuffer:plnBuffer:pbuffer:pLnbuffer);
iconvClose(tIconv);
callp write(fd:buffer:%len(buffer));
callp close(fd);
path = '/auto/001';
launchPath = '%SVDHOME%\launcher.exe';
cmd= 'STRPCCMD pccmd("'" + launchPath + '"silverdev:' +
'host=192.168.0.42&port=7003&application=/auto/'+fileName + '"')';
Qcmdexc( cmd: %len(cmd));
*inlr = *on;
/end-free

```

Note : In order to delete the .app file once it is used, add the code deleteApp='Y' in the .app file.

Launch a silverdev application from a html page

To launch a silverdev applicatoin from a html page, add a link in the html page formed as follows :

Example :

```
<a href=silverdev:host=192.168.0.80&port=7003&application=/examples/books/sddmbks&user=TOTO&pwd=1234>Texte</a>
```

Launch a silverdev application from a silverdev application.

To call a silverdev application within a silverdev application in the same process, juste use a call instruction.
To launch a silverdev application in a different process, use the function sdLaunch.

Prototype:

d sdLaunch	pr		
d Commande		255	Varying Value
d Caption		255	Varying Value
d MD5		255	Varying Value
d Ind		255	Varying Value

Description:

Enables another SilverDev application to be launched.

Parameters:

Command	System i (OS/400) command to be launched
Caption	Title of the application
MD5	MD5 code of the icon (if the user has this icon in his/her cache it will be used)
Ind	Index of the internal icon to be used if the icon

	corresponding to MD5 is not found.
--	------------------------------------

Launch an external program from a silverdev application.

To launch an external program, use the sdExecutePc function.

Prototype:

d	sdExecutePc	pr	
d	action	256a	varying value
d	file	256a	varying value
d	parameters	256a	varying value
d	Window	2 0	value

Description:

The sdExecutePc function enables execution of a file to be launched on the PC.

A executable or a file whose extension is associated with an executable (e.g.: .xls for excel) can be launched.

Parameters:

Action:

Action to be carried out on the file.

Common actions are:

open	Open the file specified by the second parameter. The file can be an executable, document or directory.
edit	Open the document with an associated editor.
print	Print the document.
find	Start a search in the directory specified by the second parameter.
explore	Open an explorer in the directory specified by the second parameter.

There may be other possible values for some files.
The possible values for a file are the items that you find in the register database under:
HKEY_CLASSES_ROOT\object_name\shell\verb
where object_name is the name of the object.
The subkey command contains the data indicated the action carried out.

File:

File to which the instruction applies.

Parameters:

Parameters for the executable launched.

Window:

Type of window opening.

Possible values:

constant	value	Meaning
W_HIDE	0	The window is invisible (useful for the print action)
W_NORMAL	1	The window is normal.
W_MIN	2	The window is minimised
W_MAX	3	The window is maximised.
W_NOA	4	The window is of normal size but is not activated.
W_MIN_NOA	7	The window is minimised and not activated.

Errors:

The program may not run correctly. In this case, check SilverDev's debugger. In the Trace part, a text is displayed indicating whether or not the program runs properly or any possible error messages.

Examples:

C	callp	sdExecutePc('open':'c:/test.txt':
C		':W_NORMAL)
C	callp	sdExecutePc('print':'c:/test.txt':
C		':W_HIDE)
C	callp	sdExecutePc('explore':'c:/winnt ':
C		':W_NORMAL)
C	callp	sdExecutePc('find':'c:/winnt ':
C		':W_NORMAL)
C	callp	sdExecutePc('open':'c:/test.txt':
C		':W_NORMAL)

Launch a web page from a silverdev application.

To launch a web page in a browser from a silverdev application, you can use the `sdExecutePc` function, with the url in the file parameter.

Example :

```
sdExecutePc('open': 'http://www.silverdev.com': '': w_normal);
```

Insert a 5250 application in a silverdev program.

You can insert a 5250 application in a silverdev form by using the CGreenScreen component.

The program running in the component runs in a different job than the silverdev application.

Here is an example of how to launch a 5250 program without havint a prompt in the 5250 session. (

c	callp	sdSetString(F1:myGS:
c		'host':sdGetServerIp())
c	callp	sdSetNum(F1:myGS:
c		'port':23)
c	callp	sdSetString(F1:myGS:
c		'user': 'ADUVAL')
c	callp	sdSetPwd(F1:myGS)
c	callp	sdSetString(F1:myGS:
c		'CurLib':pgmCurUser)
c	callp	sdSetString(F1:myGS:
c		'InitialPgm': 'SDDMGS0')
c	callp	sdConnect(F1:myGs:0

Insert a web page in a silverdev program

To insert a web page in a silverdev form, you can use the CWebBrowser component.

Use the sdGotoUrl function to change the url displayed in the component.

Example :

```
c      callp      sdGotoUrl(F1: 'webBrowser1':  
c      'www.silverdev.com')
```

Note : It is possible to execute javascript code in the browser from a silverdev program, and to read a variable value in the browser.

Chapitre 28. sdsrvjson service program

Introduction :

The sdsrvjson service program is delivered with silverdev.
To use it, refer to bound directory sdsrvjson.
Prototypes of functions are in member silverdev/h/sdsrvjson

Functions

The jsonparse function allows to parse json document and store result in a hierarchical structure.

Chapitre 29. WebBrowser

Chapitre 30. Web services

Example of a call of a web service :

```

S * /BLOCK RPGSPCIH
, *----- RPGSPCIH : H specifications (Heading)
h bnddir ('SDSRVDSTR' : 'SDSRVLST' : 'SDSRVNODES') 1

S * /BLOCK RPGSPCIF
, *-----, RPGSPCIF : Files declarations (F Spec.)

S * /BLOCK RPGSPCID
, *----- RPGSPCID : Data descriptions (D Spec.)
, *-----
/copy h, sdsrvjson
/copy h, sdsrvDStr
/copy h, sdsrvnodes 2
/copy h, res_parse

```

Part 1 bounds the program with bound directories.

Each bound directory has a service program of the same name.

SDSRVLST is a service program that allows to handle a liste of dynamically allocated structures. These lists are like arrays that increase automatically.

SDSRVSTR is a service program that allows to handle dynamically allocated strings. This allows to avoid the size limit of alphanumeric variables.

SDSRVNODES is a service program that allows to load a xml node from a text and that allows to read all subnodes and attributes eaysily.

Part 2 includes definitions of procedures and data types declared in service programs previously seen.

Form has a CWebService component :



We declare following variables :

```
D xml          ds          likeDs (DsDynStr)
D rootNode     ds          likeDs (DsNode)  inz
D subNode      ds          likeDs (DsNode)
D              based(ptrNode)

D nodeName     s           500    varying
D nodeText     s           500    varying
```

DsDynStr type is declared in sdsrvstr

DsNode type is declared in sdsrvnodes

```

* -----*
D Parameters      ds      based(pevtinf)
D Win              5u 0
D Evt              48
/free

// Passage de parametres au composant webservice
sdSlClear(Fl : 'webService1' : 'Params') ;
sdSlAdd(Fl : 'webService1' : 'Params' : %trim(para1)) ;
sdSlAdd(Fl : 'webService1' : 'Params' : %trim(para2)) ;
sdSlAdd(Fl : 'webService1' : 'Params' : %trim(para3)) ;
sdSetString(Fl : 'webService1' : 'url' : url);
sdSetBool(Fl : 'webService1' : 'asynchronous' : *off);

// Execution du webservice
sdExecute(Fl : 'webService1');
xml = sdGetDynStr(Fl : 'webService1' : 'httpResponse.data');
code = sdGetInt(Fl : 'webService1' : 'httpResponse.code');
if code = 200;
  loadNodesFromText(rootNode:resultParse:xml);
  freeDynStr(xml);
  nodeName = getDynStr(rootNode.name);
  if nodeName = 'response' ;
    for i= 1 to rootNode.nodes.count;
      ptrNode = getItem(rootNode.nodes:i);
      nodeName = getDynStr(subNode.name);
      if nodeName = 'tokenDec';
        nodeText = getDynStr(subNode.text);
        jeton=nodeText;
      endif;
    endfor;
  endif;
  clearNode(rootNode);
else;
  sdShowMessage('web service failed' + X'0d25' +
    ' code : ' + %char(code) + X'0d25' +
    ' data : ' + getDynStr(xml));
  anomalie='O';
endif;
/end-free

```

Part 1 prepares webService component by adding parameters to the request.

Part 2 calls the web service.

Part 3 gets data returned by web service in a dsdynstr variable type.

Part 4 loads a node (structure of type DsNode) from a variable of type DsDynStr.

Part 5 frees the dynamically allocated memory for the xml variable.

Part 6 reads rootnode node

Part 7 frees memory allocated for variable rootNode