WinPOS

Signal Processing Package

Programmer's Guide

Edition 3.0

Table of contents

About this Guide

The Programmer's Guide includes a detailed description of the WinPOS application interface (API), examples and recommendations of the software writing. A separate part of this Guide covers the work with the built-in script editor.

This Guide is intended for WinPOS users familiar with the programming basics. Script writing (Visual Basic Script or VBS) does not demand high programming skills of a user. However, knowledge of the object-oriented programming (OOP) concept and the basics of OLE and ActiveX can be useful for development of plug-ins.

Structure of the Guide

Part 1 covers features and application areas of the WinPOS application interface (API). The application internal structure is described.

Part 2 shall help in selection of programming language and environment depending upon sophistication of a particular problem.

Part 3 contains a detailed description of WinPOS object interfaces, properties and methods.

Part 4 contains descriptions of the simplified algorithm calls.

Part 5 covers WinPOS embedded script editor and debugger.

Appendix contains samples and descriptions of source texts of sample scripts, routines and plug-ins.

Index of methods can be found at the end of the Guide.

Conventions

The following conventions are used in the present Guide for the reader's convenience.

< >	Angle brackets indicate function keys and their combinations, e.g., <Ctrl>
→	The symbol → is used to divide the menu levels. E.g., **File→Open…** means that the item **Open…** shall be selected in the **File** menu.
File	**Bold** denotes the names of menu items or dialog box elements that can be selected and enabled by mouse button click.
Script	*Italic* denotes the names of the Guide chapters, WinPOS windows.
`signal`	`Monospace` font denotes text or characters to be entered from the keyboard, function prototypes, parameter names, and examples.
ⓘ	Important information, caution or recommendation.

The following graphic conventions are used in the interface descriptions:

 - Property,

 - Method,

 - Return value.

Part 1. Introduction

WinPOS (versions **Professional** and **Expert)** lets creation of own signal processing algorithms, automation of the input signal processing from the input file selection to the processing output documenting.

The application areas of the WinPOS application programming interface (API) are:

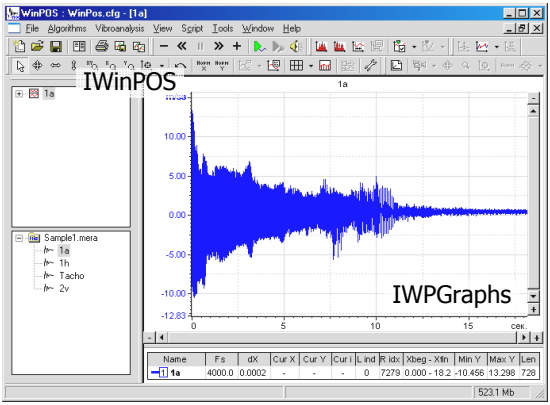
- Cyclic processing of huge data,
- Calculations by highly customized formulae,
- Automated searching of specific values in the results of calculations,
- Program generation of, e.g., reference signals with pre-set properties,
- Generation of template reports, tables,
- Data reading and writing in custom formats, etc.

WinPOS application structure

WinPOS is a modular application. WinPOS software model is based upon the object oriented programming (OOP) concept. The WinPOS object can be conditionally classified by the following functionality groups:

- User interface implementation objects (menu, toolbars, windows),
- Graphic subsystem objects (pages, graphs, lines),
- Data access objects (signals, data files),
- Structured data storage objects (WinPOS object tree),
- Mathematical algorithm implementation objects (operators).

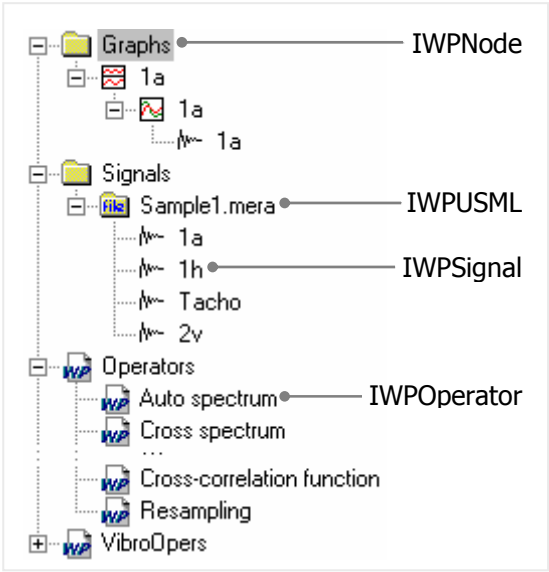
Since the most important objects are accessible from the outside, the WinPOS features can be employed by the software as well as by the user interface. Hence, the users are able to automate the solution of frequent problems not envisaged in course of the application development by some programming tool. Such objects are commonly called as ActiveX, and the application – as OLE server. The WinPOS objects represent the interfaces briefly described below.



IWinPOS is the main interface of the application. All interactions with the application are performed via this interface, since the access to the object tree is implemented via this interface in addition to the WinPOS graphic element access (graphic subsystem is accessible by `GraphAPI()` call).

WinPOS object tree is a structured data storage. Separate tree elements are represented in the windows *Signal Tree* ("\Signals"), *Graph Tree* ("\Graphs"), *Algorithms* ("\Operators") or on the panels *Signal Manager*.

Each object (node, «leaf» of the tree) can be accesses via the **IWPNode** interface. The method `GetObject()` lets to obtain any WinPOS object from those represented in the tree. Cyclic search of objects is also implemented by the **IWPNode** interface.



One of the WinPOS objects can be associated with the tree node:

- Page, graph, line (accessible via the **IWPGraphs** interface),
- USML or MERA format file (accessible via the **IWPUSML** interface),
- Signal (**IWPSignal**),
- Operator (**IWPOperator**).

Part 2. Application notes

WinPOS provides the interfaces enabling the user to create own plug-ins or applications operating with the data and algorithms of WinPOS almost in any modern programming environment. *Microsoft Visual Basic Script* and *Borland Delphi* are selected for examples. VBScript is included into the Microsoft Windows package, requires no separate compiler, and simple script editing environment is included into WinPOS. Delphi deserved the fame of the most convenient environment for rapid application development (RAD) and ideally suits for creation of small particular applications.

The pros and contras of script and application programming by the above mentioned tools are presented below.

VBScript

- ⊕ No compiler or separate development environment is necessary,
- ⊕ Programming basics only are required,
- ⊖ Application with own customized dialogs or forms cannot be created.

Delphi

- ⊕ Dialogs and forms for any customization can be created, numerous Delphi components can be used, specific reports can be generated,
- ⊕ Development of sophisticated own data processing algorithms is easy,
- ⊖ Installation of Borland Delphi and the respective programming skills are necessary.

Hence, VBScript is more suitable for small automation scripts of WinPOS operation or simple algorithms but is less convenient for huge data processing, and Delphi should be applied for writing of own fast processing algorithms and creation of applications which require additional customization or generation of specialized reports.

The methods of joint execution of scripts, applications, plug-ins and WinPOS are discussed below.

VBScript

The easiest way of creation of own script by VBScript is to copy a sample from *References* or from the disk, insert this sample to the *Script editor* (menu *Script*) and then modify this sample by addition of the necessary functionality. The *Script editor*

control elements are described in the *Part 5. Embedded script editor*. The resulting script can be executed in several ways:

- From the script editor – *Execute program* (F5),
- From the WinPOS main window, **Execute script** menu or by script hot button in *Toolbar*,
- From the command line ("winpos.exe myscript.wps").

A classical VBScript sample is shown below:

```
sub main
      DebugPrint "Hello, world!"
end sub
```

The line «Hello, world!» will be printed in the debugging print window of the Script editor (this sample should be enabled by the first method only, otherwise no debugging print is possible).

Delphi

RAD Delphi allows creation of the applications addressing the objects and methods of WinPOS, and also development of plug-ins as dynamic link libraries (DLL). Such plug-ins can be embedded into WinPOS. For example, the buttons calling the user library functions can be easily added to the WinPOS toolbar.

Applications

The application (EXE-file) is able to access the WinPOS objects, their properties and methods by creation of the proxy class as follows:

```
var WinPOS: TWinPOS;
...
WinPOS:= TWinPOS.Create(nil);
```

Further the WinPOS methods can be used:

```
// open USML by standard WinPOS dialog
FileName:= WinPos.USMLDialog();
```

Note! From the point of view of an application WinPOS is an out-of process server server. That is, the WinPOS object methods are called with inevitable delays caused by slow processors' interaction. Hence, a separate application poorly suits the creation of own algorithms which cyclically address the GetY signal methods or the

like. The first sample (sine generator) demonstrates this effect. At the same time, at particular addressing to the signals or algorithms, the delays are almost negligible. The plug-ins (see below) are more suitable for the tasks which demand constant interaction with the WinPOS objects.

Plug-Ins

RAD Delphi allows creation of the applications addressing the objects and methods of WinPOS, and also development of plug-ins as dynamic link libraries (DLL). Such plug-ins can be embedded into WinPOS. For example, the buttons calling the user library functions can be easily added to the WinPOS toolbar.

WinPOS will be a local server for such plug-in. Hence, the time delays will be insignificant. The user algorithms will be almost as effective as the embedded ones.

The plug-in must contain COM-class with dual interface providing three methods: Connect(), Disconnect() and NotifyPlugin(). At the start WinPOS calls the method Connect() by passing the pointer to itself, and Disconnect() is called at the operation end. Other messages are transferred by WinPOS by call of NotifyPlugin() with the message code and parameters.

```
function Connect(const app: IDispatch): Integer;
function Disconnect: Integer;
function NotifyPlugin(what: Integer; var param:
OleVariant): Integer;
```

At the starting WinPOS calls the method Connect() passing a pointer to itself, during a shutdown Disconnect() is called, and WinPOS passes other messages, calling NotifyPlugin () with the code and the parameters of the message.

WinPos uploads plug-ins on the list, saved in the system registry. An addition of the plug-in into the list and a removing from the list conveniently to combine with the registration procedures. For that it is necessary to overload DllRegisterServer and DllUnregisterServer.

Creating a plug-in step by step

1. Create a new library (DLL): **File→New→Other…→ActiveX→ActiveX Library**.
2. Save the library: **File→Save**. Enter the library name in the saving dialog, for example, "MyPlugin", then press the **Save** button.

3. Create a new COM object: **File→New→Other…→ActivyX→COM Object**. The dialog *COM Object Wizard* (Fig. 2.1) will be appeared.

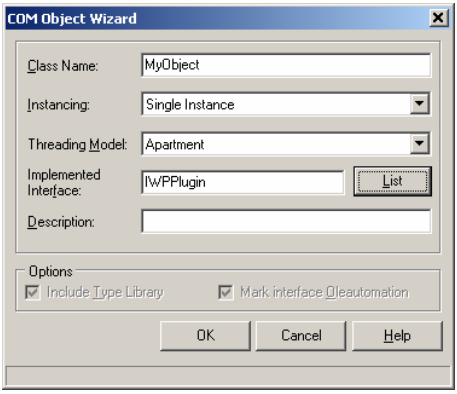


Fig 2.1 COM Object Wizard

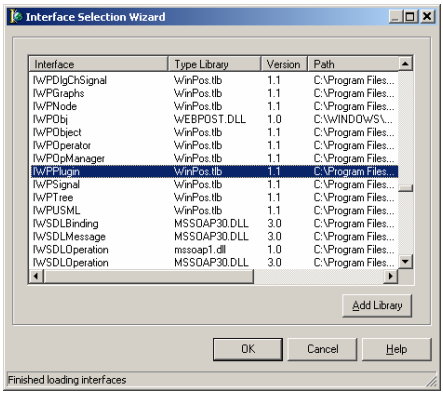


Fig 2.2 Interface Selection Wizard

- Enter the class name to the field *Class Name*.
- Press the button **List**. The dialog *Interface Selection Wizard* (Fig. 2.2) will open.
- Choose the interface *IWPPlugin* (see. fig.) and press **OK**. The window *Type Library* will open. Draw attention that the new library and the new class are displaying on the left window part. The window can be closed, and use **View→Type Library** to open it repeatedly.

4. Add required modules to the section *uses* of the initial library file (this is MyPlugin.dpr here). Usually used: SysUtils, Classes, Consts, Windows, ComServ, Registry.

5. Overload the function DllRegisterServer, as shown below.

Here the second parameter of the method *reg.writestring()* is the appellation of a new COM – object. It consists of new library names and the object witch are divided by a point. These names are displayed on the left part *Type Library* window.

```
function DllRegisterServer: hresult; stdcall;
var     reg: tregistry;
        buffer: array[0..255] of char;
begin
  reg := tregistry.Create;
  try
    reg.rootkey := hkey_local_machine;
    getmodulefilename(hinstance, buffer, 255);
    reg.openkey('\Software\MERA\Winpos\COMPlugins', True);
    reg.writestring(string(buffer), 'MyPlugin.MyObject');
  finally
    reg.Free;
   end;
  Result := comserv.dllregisterserver;
end;
```

6. Overload the function DllUnregisterServer, as shown below.

```
function DllUnregisterServer: hresult; stdcall;
var
  reg: tregistry;
var
  buffer: array[0..255] of char;
begin
  reg := tregistry.Create;
  try
    reg.rootkey := hkey_local_machine;
    getmodulefilename(hinstance, buffer, 255);
    if (reg.openkey('\Software\MERA\Winpos\COMPlugins',False)) then
      reg.DeleteValue(string(buffer));
  finally
    reg.Free;
  end;
  Result := comserv.dllunregisterserver;
end;
```

7. Overload methods Connect(), Disconnect() и NotifyPlugin() (see Unit1.pas).

In the example cited below a new toolbar and a button on it are created in the method Connect(). In the method NotifyPlugin() after pressing the button, a form to which the control is inherited are created.

Create a form and draw an image for the button of the toolbar – this is bitmap 19x19 (it is possible to use the built-in resource editor **Tools→Image Editor**).

```
var ID_Run1 : Integer=0;      // Command identifier
var bar_ID : Integer;         // Toolbar descriptor

function TMyObject.Connect(const app: IDispatch): Integer;
var hbmp:THandle;
begin
   WP:=app as IWinPOS;
   ID_Run1 := WP.RegisterCommand();  //Get the command Id
   bar_ID:=WP.CreateToolbar();       //Create toolbar
                                     //Load button image
   hbmp:=LoadBitmap(HInstance,'TOOLBAR');
                                     //Create button
   WP.CreatetoolbarButton( bar_ID, ID_Run1, hbmp,
                           'My action'#10'Do my action');
   Result:= 0;
end;

function TMyObject.Disconnect: Integer;
begin
   Result:= 0;
end;

function TMyObject.NotifyPlugin(what: Integer;
  var param: OleVariant): Integer;
var cmdln : AnsiString;
begin
  try
    if HiWord(what)=ID_Run1          //Check the command Id
    then
      begin                          //Form creation
        Application.CreateForm(TForm1, Form1);
        Form1.Show;
      end
  except
  end;
  Result:= 0;
end;
```

8. Compile the plug-in. Register it using regsvr32. Close and start WinPOS again. A new panel with the call button of the plug-in should appear.

A registration and a cancel of a registration of plug-ins are carried out by the standard Window tool.

```
regsvr32 myplugin.dll
regsvr32 /u myplugin.dll
```

The folder DelphiCommon (see. *Appendix. Samples*) contains the files Winpos_ole_TLB.pas and POSBase.pas.

Winpos_ole_TLB.pas is created automatically and includes the descriptions of OLE-interfaces WinPOS (the file is connected automatically at the inheritance of the interface IWPPlugin).

POSBase.pas contains the functions of the type `RunXXXX()` simplified an access to the algorithms WinPOS and a number of constants (a connect of this file can be effective). See also *Part 5. The call of algorithms*.

Other tools

As stated above, in addition to *VBScript* and *Delphi*, other means can be used for development of applications and plug-ins. The operation with such means requires performance of the sequence aimed at generation of the respective program module by the WinPOS Type Library (TLB).

In *Borland C++ Builder*, as in *Delphi*, the sequence is the following:

Project→Import Type Library…→[select winpos_ole]**→Create Unit**,

And in *Microsoft Visual C++*:

View→ClassWizard…→AddClass…→From a type library→[winpos.exe].

Part 3. WinPOS interfaces

The majority of WinPOS objects can be manipulated by the interfaces listed below.

IWinPOS	- Main application interface
IWPGraphs	- Graphic subsystem interface
IWPSignal	- Signal interface
IWPUSML	- Batch file (USML and MERA) interface
IWPOperator	- Mathematically based algorithm call interface
IWPNode	- WinPOS object tree element

The following chapters contain descriptions of the interface methods in ODL (Object Description Language) notation. This notation is preferable for the OLE interface description. The Table of type correspondence in different languages is given below.

ODL	Delphi	VBScript	Description
BSTR	String	BSTR	Symbol string
long	Integer	Variant	Integer (32 nits)
short	Smallint	Variant	Short integer (16 bits)
IDispatch*	IDispatch	IDispatch	Pointer to the IDispatch derived interface
VARIANT_BOOL	Boolean	Variant	Boolean (logic) variable
void	[procedure]*	[sub]*	*Void returning function, i.e., procedure
VARIANT	OleVariant	Variant	Variable

IWinPOS

This is the main application interface allowing control of the WinPOS environment, access to the object tree, data reading and writing, interaction with connected modules, calling of documenting methods and debugging routines.

Properties

SelectedGraph

```
BSTR SelectedGraph
```

The name of selected graph in the WoinPOS graph tree. This property is **read only.**

SelectedSignal

```
BSTR SelectedSignal
```

The name of selected signal in the WoinPOS signal tree. This property is **read only.**

Methods

Opening and saving of data files

LoadUSML

```
IDispatch* LoadUSML(BSTR path)
```

Load USML or MERA file and place it to the WinPOS signal tree.

	The object supporting IWPUSML interface
`path`	File name

SaveUSML

```
void SaveUSML(BSTR Name, BSTR FileName)
```

Store the WinPOS signal tree folder as USML or MERA file.

`Name`	Full folder name in the WinPOS signal tree
`FileName`	File name

LoadSignal

```
IDispatch* LoadSignal(BSTR path, long type)
```

Load a binary or text data file and place the results to the WinPOS signal tree.

	The object supporting IWPSignal interface
`path`	File name
`type`	Data file type. The list of constant values is provided in the Table below.

Data file type	Constant value	Description
`FT_TextWiz`	3	Text (ASCII) file opened by setup
`FT_UChar`	4	Unsigned integer array (1 byte)
`FT_INT16`	5	Signed integer array (2 bytes)
`FT_WORD`	6	Unsigned integer array (2 bytes)
`FT_INT32`	7	Signed integer array (4 bytes)
`FT_Float`	8	Real number arrays (4 bytes)
`FT_Double`	9	Real number arrays (8 bytes)
`FT_XLS`	10	Microsoft Excel table

SaveSignal

```
void SaveSignal(BSTR Name, BSTR FileName, long type)
```

Save the signal as binary or text data file.

`Name`	Full signal name in the WinPOS signal tree
`FileName`	File name
`type`	Data file type. The list of constant values is provided in the description of LoadSignal method.

Access to WinPOS objects

CreateSignal, CreateSignalXY

```
IDispatch* CreateSignal(long type)
IDispatch* CreateSignalXY(long xtype, long ytype)
```

Create a new signal. CreateSignalXY() creates a signal with possibly unequal X axis, the values can be set by the SetX() method. **NOTE!** If the signal is created by CreateSignal(), the SetX() method makes no sense!

	The object supporting IWPSignal interface
`type`	Data signal type. The list of constant values is provided in the table below. `xtype`, `ytype` – types of values for X and Y.

Data signal type	Constant value	Description
VT_I1	16	Integer, 1 byte
VT_UI1	17	Unsigned integer, 1 byte
VT_I2	2	Integer, 2 bytes
VT_UI2	18	Unsigned integer, 2 bytes
VT_I4	3	Unsigned integer, 4 bytes
VT_R4	4	Real, 4 bytes
VT_R8	5	Real, 8 bytes

GetInterval

```
IDispatch* GetInterval(IDispatch* src, long start,
long count)
```

Return value is the signal representing the source signal interval. The method is used for the source signal range processing.

	The object supporting IWPSignal interface
src	Source signal
start	Start of interval, 0 .. (src.size-1)
count	Number of values, 0 .. (src.size-start)

GetOversampled

```
IDispatch* GetOversampled(IDispatch* src, double
freq)
```

A returnable virtual signal allows interpreting the data of the original signal as if they were obtained with another sampling frequency. The new values are interpolated linearly, the algorithm of the oversampling is not called, and the filtration is not used. As a result is not recommended to select a new frequency less than the original.

	An object supporting the interface IWPSignal
src	An original signal
freq	A new frequency

GraphAPI

```
IDispatch* GraphAPI()
```

Obtain the graph subsystem interface.

	The object supporting IWPGraph interface

Link

```
IDispatch* Link(BSTR Path, BSTR Name, IDispatch* Object)
```

Place the object to the **WinPOS** tree.

	The object supporting IWPNode interface
`Path`	Path in **WinPOS** tree
`Name`	Object name
`Object`	Object to be placed to the tree

Unlink

```
void Unlink(IDispatch* Object)
```

Remove the object from the tree

`Object`	Object to be removed from the tree

GetObject

```
IDispatch* GetObject(BSTR path)
```

Find object in the **WinPOS** tree by name.

	Pointer to the requested object interface
`path`	String, path in the **WinPOS** tree

GetNode

```
IDispatch* GetNode(IDispatch* Object)
```

Obtain position (so called mounting point, node) of the object in the **WinPOS** tree.

	The object supporting IWPNode interface
`Object`	Object.

Control of WinPOS environment

USMLDialog

```
BSTR USMLDialog()
```

Select USML or MERA file by the standard WinPOS dialog box.

	Full file name

Refresh

```
void Refresh()
```

Refresh all WinPOS windows. Recommended to be used after calling the methods which modify the WinPOS status, such as Link().

DoEvents

```
void DoEvents()
```

Process all events accumulated during the long task operation. This procedure is used in the course of long calculations in order to avoid the program hang-up. DoEvents() suspends the script performance and allows the window message processing by **WinPOS**.

AddTextInLog

```
void AddTextInLog(BSTR text,BSTR
exttext,VARIANT_BOOL show)
```

Add text line to the log.

`text`	Text line for the log
`exttext`	Additional parameter line
`show`	True, if the log window is to be represented; false, if not

Interoperation with plug-ins

The methods described in the present section provide the access to the control elements of WinPOS environment (main window, toolbars, menu) and shall be used for the plug-ins creation.

MainWnd

```
long MainWnd()
```

This method returns the WinPOS main window handle which may be necessary for the connected modules, if such modules have their own windows and dialogs.

	WinPOS main window handle

RegisterCommand

```
long RegisterCommand()
```

This method returns a unique number which can be used as a command identifier.

↵	Number, a unique code of command or event.

CreateToolbarN

```
long CreateToolbarN(BSTR name)
```

Create new toolbar.

↵	Toolbar pointer
name	Toolbar name. Is added to the menu “View”

CreatetoolbarButton

```
long CreatetoolbarButton(long bar, long command,
long picture, BSTR hint)
```

Add button-«tool» to the toolbar.

↵	Non-zero if the button is successfully added, otherwise - 0
bar	Toolbar pointer which can be obtained by CreateToolbar()
command	Assigned command which can be obtained by RegisterCommand ()
picture	Button picture handle
hint	Prompt text

ToolbarSetButtonStyle

```
void ToolbarSetButtonStyle(long bar, long command,
long nStyle)
```

Change the state of the toolbar button

bar	Toolbar. See CreateToolbar()
command	Assigned command. См. RegisterCommand ()
nStyle	Button style. Is used: 0 – normal button, 4 – button is disabled.

ShowToolbar

```
void ShowToolbar(long bar, long visible)
```

Show or hide the toolbar.

bar	Toolbar pointer which can be obtained by CreateToolbar()

visible	1 – show, 0 - hide

CreatemenuItem

```
long CreatemenuItem(long Command, long reserved,
BSTR text, long style, long picture)
```

Create a new menu item.

↵	Non-zero if the menu item is successfully added, otherwise - 0
Command	Assigned command which can be obtained by RegisterCommand ()
reserved	Each byte of this number (if not equal to FF) represents a submenu item of the present level, where to a new menu item shall be added. For example: 0xFFFF0301 - the 4th submenu of the main menu, after the 2nd position (count start - 0)
text	Menu item
style	Menu item style. Set to 0 for a typical menu item. The style constant values are provided in the Table below.
picture	Menu item picture handle. Used only if MF_BITMAP style is installed.

Menu item style	Value
MF_ENABLED	0h
MF_GRAYED	1h
MF_DISABLED	2h
MF_UNCHECKED	0h
MF_CHECKED	8h
MF_USECHECKBITMAPS	200h
MF_STRING	0h

Menu item style	Value
MF_BITMAP	4h
MF_OWNERDRAW	100h
MF_POPUP	10h
MF_MENUBARBREAK	20h
MF_MENUBREAK	40h
MF_UNHILITE	0h
MF_HILITE	80h

RegisterImpExp

```
boolean RegisterImpExp(LPDISPATCH imp, LPDISPATCH
exp, LPCTSTR desc, LPCTSTR ext)
```

Register an import - export plug-in. In the window of the file WinPOS open a new file type (parameter desc) will be added. See the description of import - export interfaces.

↵	true – if the operation was successful, false – otherwise
imp	A pointer to the import interface
exp	A pointer to the export interface

desc	A description of the files type
ext	A file extension in format '*. ext'. Example (Delphi): *RegisterImpExp(self, self, 'WAV files', '*.wav');*

State displaying

ProgressStart

```
void ProgressStart(BSTR comment, long max)
```

Create a progress indicator.

comment	Line of a current state. For example, "Finding the maximum"
max	Maximum number of indicator steps

ProgressStep

```
void ProgressStep(long pos)
```

Set a progress indicator.

pos	New position indicator: 0 .. max. If pos = 0, one step of the indicator will be taken.

ProgressFinish

```
void ProgressFinish()
```

Hide a progress indicator.

Documenting and printing the results

SaveImage

```
boolean SaveImage(BSTR fname, BSTR comment)
```

Save the displayed graph page in file or buffer.

	True if successful, otherwise false
fname	File name. If the transferred string is empty – the image is placed to the exchange buffer
comment	Comment string. For example, «Fig.1 Source Levels». If an empty string is set no comment is printed.

PrintPreview, Print

```
void PrintPreview(BSTR comment)
void Print(BSTR comment)
```

Print the displayed graph page. PrintPreview shows a print preview window. Print - sends the file to printer using the current printer and page settings.

comment	Comment string. For example, «Fig.1 Source Levels». If an empty string is set no comment is printed.

VBScript. Operation with binary data files

These methods extend the limited features of VBScript concerning operation with binary files. These methods should not be used when working with Delphi, since Delphi allows calling of the direct file handling functions.

FileOpen

```
BSTR FileOpen(long isOpen, BSTR ext, BSTR fname,
long flags, BSTR filter)
```

Open a standard file dialog and select the file name.

	Full file name
isOpen	true - file open dialog, false - file saving dialog
ext	Default file extension
fname	Initial file name
flags	Flags for setting the outlook and behavior of the dialog. Some useful flags are summarized in the Table below; other flags are found in the file POSBase.pas and the references (Description of OPENFILENAME).
filter	Dialog filter set. For example: *"USML files|*.usm|All files|*.*||"* – select .usm files or all files.

Flag	Value	Description
OFN_ALLOWMULTISELECT	200h	Enable selection of several files
OFN_CREATEPROMPT	2000h	If the user specifies a non-existing file, the dialog box prompts creation of a new file with the entered name.
OFN_FILEMUSTEXIST	1000h	The field *File name* enables entering valid file names only. If the flag is entered with incorrect file name, a warning is issued. Used jointly with OFN_PATHMUSTEXIST.

OFN_NOCHANGEDIR	8h	Return the source value to the current folder if the user changes folders when searching files.
OFN_NONETWORKBUTTON	20000h	Remove the dialog button **Network**
OFN_NOREADONLYRETURN	8000h	The filed *Read only* is not selected; the returned file is not in the copy-protected folder.
OFN_OVERWRITEPROMPT	2h	The dialog *Save as* gives a warning message is the file already exists. The user should confirm the file overwriting.
OFN_PATHMUSTEXIST	800h	The user is able to enter the valid path and file names only. If an incorrect file name or path is entered, a warning window appears.

OpenFile

```
long OpenFile(BSTR Path, long flags)
```

Open or create a new file.

	File handle to be used by further calls (see the hFile parameter)
Path	File name (with path)
flags	Access type. The values are provided in the Table below.

Flag	Value	Description
READ_WRITE	100h	Open the file for read and write (GENERIC_READ | GENERIC_WRITE), 0 – read only (GENERIC_READ).
SHARE_READ	1000h	This file can be simultaneously opened as read only (FILE_SHARE_READ), 0 – for read and write (FILE_SHARE_READ | FILE_SHARE_WRITE).

CloseFile

```
void CloseFile(long hFile)
```

Close file.

hFile	File handle

SeekFile

```
long SeekFile(long hFile, long Pos, long flags)
```

Change position of the file pointer.

	New position of file pointer
hFile	File handle
Pos	Desirable position of the file handle

`flags`	Flag of the file pointer movement. The values are provided in the Table below.

Flag	Value	Description
`FILE_BEGIN`	`0`	The file beginning is zero
`FILE_CURRENT`	`1`	Zero – current position of file pointer
`FILE_END`	`2`	Zero – end of file

ReadByte, ReadWord, ReadLong, ReadFloat, ReadDouble

```
VARIANT ReadByte(long hFile)
VARIANT ReadWord(long hFile)
VARIANT ReadLong(long hFile)
VARIANT ReadFloat(long hFile)
VARIANT ReadDouble(long hFile)
```

Read a value in the respective format from the binary file.

	The value read from the file
`hFile`	File handle

WriteByte, WriteWord, WriteLong, WriteFloat, WriteDouble

```
void WriteByte(long hFile, short Value)
void WriteWord(long hFile, long value)
void WriteLong(long hFile, long Value)
void WriteFloat(long hFile, float value)
void WriteDouble(long hFile, double Value)
```

Write to the binary file using a respective format.

`hFile`	File handle
`Value`	The value to be written to the file

VBScript. Debugging

DebugPrint, DebugPrintLn

```
void DebugPrint(VARIANT arg)
void DebugPrintLn(VARIANT arg)
```

Debugging printing to the *Script editor* output window. Applied only at the operation with *Script editor*. DebugPrintLn() feeds the line only as distinct from DebugPrint().

`Arg`	Symbol line. For example: *DebugPrintLn "Max= "+FormatNumber(max,6,0,0,0)+";"*

IWPGraphs

Graph subsystem interface.

This graph control interface is obtained by calling the GraphAPI method of the IWinPOS interface.

Operation sequence at creation of a new page to represent a signal (e.g., operation output of script or plugin):

```
// obtain access to the WinPOS graph subsystem
api := GraphAPI as IWPGraphs;

// create new page for graphs
hPage := api.CreatePage;

// the page is always created with one graph, obtain the
graph
hGraph := api.GetGraph(hPage, 0);

// the graph always has at least one axis Y, obtain the axis
hYAxis := api.GetYAxis(hGraph, 0);

// create a new line in the graph
api.CreateLine(hGraph, hYAxis, signal.Instance);

// normalize the graph
api.NormalizeGraph(hGraph);
```

Methods

CreatePage

```
long CreatePage()
```

Create a new page. The default settings are observed.

↵	New page pointer

DestroyPage

```
void DestroyPage(long hPage)
```

Delete page.

hPage	Page pointer

CreateGraph

```
long CreateGraph(long hPage)
```

Create a new graph. The default settings are observed.

↵	New graph pointer
hPage	Pointer of the page where the graph is to be created

DestroyGraph

```
void DestroyGraph(long hGraph)
```

Delete graph.

hGraph	Graph pointer

CreateYAxis

```
long CreateYAxis(long hGraph)
```

Add a new ordinate axis.

↵	A pointer to new axis
hGraph	A pointer graph, to which will be added an axis

DestroyYAxis

```
void DestroyYAxis(long hAxis)
```

Remove the axis.

hAxis	A pointer to an axis

CreateLine

```
long CreateLine(long hGr, long hAx, long hSig)
```

Create a new line. The default settings are observed.

↵	New line pointer
hGr	Pointer of the graph where to the line is to be added
hAx	Y-axis pointer
hSig	Signal pointer

DestroyLine

```
void DestroyLine(long hLine)
```

Delete line.

hLine	Line pointer

GetPageCount

```
long GetPageCount()
```

Number of graph pages.

↵	Number of graph pages

GetGraphCount

```
long GetGraphCount(long hPage)
```

Number of graphs in a page.

↵	Number of graphs in a page
hPage	Page pointer

GetYAxisCount

```
long GetYAxisCount(long hGr)
```

Number of Y-axis of the graph.

↵	Number of Y-axis of the graph
hGr	Graph pointer

GetLineCount

```
long GetLineCount(long hGr)
```

Number of lines in a graph.

↵	Number of lines in a graph
hGr	Graph pointer

GetPage

```
long GetPage(long nPage)
```

Get a page by the number.

↵	Page pointer
nPage	Page number

GetGraph

```
long GetGraph(long hPage, long nGraph)
```

Get a graph by the number.

↵	Graph pointer
hPage	Page pointer
nGraph	Graph number

GetYAxis

```
long GetYAxis(long hGr, long nAxis)
```

Get Y-axis by number.

↵	Axis pointer
hGr	Graph pointer
nAxis	Axis number

GetLine

```
long GetLine(long hGr, long nLine)
```

Get a line by number.

↵	Line pointer
hGr	Graph pointer
nLine	Line number

GetSignal

```
IDispatch* GetSignal(long hLine)
```

Get a reference to the signal represented by the hLine line.

	The object supporting IWPSignal interface
hLine	Line pointer

GetXCursorPos, SetXCursorPos

```
void GetXCursorPos(long hGraph, double* px, BOOL bSecond)
void SetXCursorPos(long hGraph, double x, BOOL bSecond)
```

Get or set a cursor position.

hGraph	A pointer to the praph
x	A cursor position
px	A variable address for a return of the cursor position
bSecond	To work with the position of the cursor second line (only for differented cursor)

ShowCursor

```
void ShowCursor(long hPage, long mode)
```

Get or set a cursor position.

hPage	A pointer to the page
mode	A mode of a cursor displaying. See table

Flag	Value	Description
TM_NONE	0	Cursor off
TM_CURSOR	1	Cursor over all lines
TM_DBLCURS	8	Double cursor
TM_SLCURS	9	Cursor of the current line

GetPageRect, SetPageRect

```
void GetPageRect(long hPage, long* left, long* top, long* right, long* bottom)
void SetPageRect(long hPage, long left, long top, long right, long bottom)
```

Obtain and set the graph page layout dimensions.

`hPage`	Page pointer
`left`	Page left coordinates
`top`	Page top coordinates
`rigth`	Page right coordinates
`bottom`	Page bottom coordinates

SetPageDim

```
void SetPageDim(long hPage, long mode, long width, long height)
```

Set the graph positioning mode.

`hPage`	Page pointer
`mode`	Type of graph positioning. Possible positioning versions are provided in the Table below.
`width`	Number of graphs in a page by width
`height`	Number of graphs in a page by height

Flag	Value	Description
`PAGE_DM_VERT`	0	Graphs positioned vertically
`PAGE_DM_HORZ`	1	Graphs positioned horizontally
`PAGE_DM_TABLE`	2	Graphs positioned as width*height table

GetXMinMax, SetXMinMax

```
void GetXMinMax(long hGR, double* pmin, double* pmax)
void SetXMinMax(long hGr, double min, double max)
```

Get, set margins on the abscissa axis. Thus, it is possible to set or to get the visible range of the signal.

`hGr`	Graph pointer
`min`	Minimum value or the pointer on it
`max`	Maximum value or the pointer on it

GetYAxisMinMax, SetYAxisMinMax

```
void GetYAxisMinMax(long hAxis, double* pmin, double* pmax)
```

```
void SetYAxisMinMax(long hAxis, double min, double max)
```

Get, set margins of the selected Y-axis.

hAxis	Y- axis pointer
min	Minimum value or the pointer on it
max	Maximum value or the pointer on it

NormalizeGraph

```
void NormalizeGraph(long hGr)
```

Normalize the graph.

hGr	Graph pointer

Invalidate

```
void invalidate(long hGraph)
```

Refresh the graph plotting field.

hGraph	Graph pointer

ActiveGraphPage

```
long ActiveGraphPage()
```

Get the active graph page pointer.

	Active graph page pointer

ActiveGraph

```
long ActiveGraph(long hPage)
```

Get the active graph pointer.

	Active graph pointer
hPage	Graph page pointer

Folder2Graphs, Folder2GraphsRecursive

```
void Folder2Graphs(IDispatch* Node)
void Folder2GraphsRecursive(IDispatch* Node)
```

Place all signals of this folder or batch file on a new page. The second option avoids the embedded folders.

Node	The object supporting IWPNode interface

Locate

```
IDispatch* Locate(long hGrItem)
```

Find a graph element in the tree over the sign

	An object that supports the interface IWPNode
hGrItem	A pointer to a page of graphs, a graph or a line

SetPageOpt

```
void SetPageOpt(long hPage, long opt, long mask)
```

Set parameters of the selected page.

hPage	Pointer on the page
opt	Bit field showing a particular bit to be set or removed. See table below.
mask	A mask. Shows which bits of the field opt should change. See table below.

Flag	Value	Description
PGOPT_SHOWNAME	1	The displaying of the page name
PGOPT_SINGLEX	2	Flag one X-axis to the page
PGOPT_SINGLEY	4	Flag one Y-axis to the page
PGOPT_SINCCURS	8	Synchronizing cursors

SetGraphOpt

```
void SetGraphOpt(long hGraph, long opt, long mask)
```

Set the parameters of the selected graph.

hGraph	Pointer on the graph
opt	Bit field showing a particular bit to be set or removed. See table below.
mask	The mask showing the opt field bits to be changed. See table below.

Flag	Value	Description
GROPT_SHOWNAME	1h	Flag of the name drawing
GROPT_YINDENT	2h	10% indentation for the lines above and below
GROPT_SUBGRID	4h	Flag of the drawing of dotted lines on the grid
GROPT_GRIDLABS	8h	The lines values in the grid
GROPT_LINENUMS	10h	Show numbers of lines

GROPT_AUTONORM	20h	Automatically to normalize graph during an addition new lines
GROPT_POLAR	40h	Polar coordinate
GROPT_AXCOLUMN	80h	Flag of the placement Y axis one after another
GROPT_AXROW	100h	Flag of the placement Y axis one after another

GetAxisOpt, SetAxisOpt

```
void GetAxisOpt(long hGraph, long hAxis, long* opt, double *minR, double *maxR, BSTR *szname, BSTR *szftempl, long *color)
void SetAxisOpt(long hGraph, long hAxis, long opt, long mask, double minR, double maxR, BSTR szname, BSTR szftempl, long color)
```

Get / set the parameters of the selected graph.

hGraph	Pointer on the graph (needed for the X-axis)
hAxis	Pointer to the axis (for the X-axis: 0)
opt	Bit field showing a particular bit to be set or removed. See table below.
mask	The mask showing the opt field bits to be changed. It also shows whether or not to change the name or tick label number format. See table below.
minR,maxR	Displaying axis range (with the flag AXOPT_RANGE - full range, i.e. margins of a zoom)
szname	Axis name (usually taken the dimension)
szftempl	Tick label number format (described in the *User's guide*, part 5, *Graphs creating*, *Graphs settings*)
color	Colour of the format RGB (white = FFFFFFh, black = 0h)

Flag	Value	Description
AXOPT_LOG	1h	Logarithmic scale
AXOPT_FZERO	2h	Add zeros to the end of the number ("1.500" instead of "1.5")
AXOPT_TIME	4h	Add a time scale in the format "hh:mm:ss.msc"
AXOPT_COLOR*	8h	Set manually the color of tick label numbers
AXOPT_RANGE*	10h	Set a full range of axis
AXOPT_NAME*	20h	Set the name or the dimension of the axis
AXOPT_FORMAT*	40h	Set tick label number format

* - to be used in the mask field only

SetLineOpt

```
void SetLineOpt(long hLine, long opt, long mask, long width, long color)
```

Set parameters of the selected line.

`hLine`	Line pointer
`opt`	Bit field showing a particular bit to be set or removed. See the Table below.
`mask`	The mask showing the opt field bits to be changed. Also the mask shows if the line width or color has to be changed. See the Table below.
`width`	Line width
`color`	RGB color (white = FFFFFFh, black = 0h)

Flag	Value	Description
`LNOPT_LINE2BASE`	`1h`	Add vertical lines from the value to 0
`LNOPT_ONLYPOINTS`	`2h`	Flag of points joining by lines
`LNOPT_VISIBLE`	`4h`	Flag of a displaying / hiding of lines
`LNOPT_HIST`	`8h`	As histogram
`LNOPT_HISTTRANSP`	`40h`	"Transparent" histogram
`LNOPT_PARAM`	`80h`	In the form of Y (idx), with the actual values on the scale of the X-axis
`LNOPT_INTERP`	`300h`	The order of the interpolation (2 bytes)
`LNOPT_COLOR`*	`10h`	Change the color line. See field *color*
`LNOPT_WIDTH`*	`20h`	Change the line thickness. See field *width*

* - to be used in the mask field only

AddLabel

```
void AddLabel(long hLine, long mode, double x, double offsX, double offsY, BSTR text)
```

Add a label

`hLine`	Pointer to the line
`mode`	Label type. See table below
`x`	The value of time to which the label is tied
`offsX`, `offsY`	The label position in the graph field, expressed as a percentage relative to the graph size
`text`	The label text, if mode = LAB_TEXT

Flag	Value	Description
`LAB_SINGLE`	`0`	On one line
`LAB_MULTI`	`1`	On all lines
`LAB_TEXT`	`2`	Text label

AddComment

```
void AddComment(long hGr, BSTR text, double x, double y, double dx, double dy)
```

Add comment

hGr	Pointer to the graph
text	The text of the comment
x, y	The position of the upper left corner of the comment, expressed as a percentage relative to the graph size
dx, dy	Comment sizes, expressed as a percentage relative to the graph size

SaveSession, LoadSession

```
BOOL SaveSession(BSTR path)
BOOL LoadSession(BSTR path)
```

Save a current session of a work and load an earlier saved session.

	The result of the operation (TRUE - success)
path	The path to the session files on the disk

IWPSignal

Signal interface.

Properties

size

```
long size
```

Number of signal values (measurements).

DeltaX

```
double DeltaX
```

Step on axis X for a signal with a uniform X-axis. DeltaX=0 for a signal with non-uniform axis.

StartX

```
double StartX
```

Start X axis value for the signal with uniform abscissa axis. StartX contains the abscissa axis first element value of the signal with non-uniform signal.

SName

```
BSTR SName
```

Signal name.

NameY

```
BSTR NameY
```

Measurement unites of signal values, line.

NameX

```
BSTR NameX
```

Measurement unites of the abscissa axis, line.

Comment

```
BSTR Comment
```

Comment, additional extended text information on a given signal.

Characteristic

```
long Characteristic
```

Signal characteristics which impacts the graph type. The possible values are provided in the Table.

Characteristics	Value	Description
SC_NORMAL	0	Normal signal
SC_SPECTR	1	Spectrum
SC_LOGSPEC	2	Logarithmic spectrum
SC_LOGX	4	Logarithmic abscissa axis signal
SC_AMP	8	Amplitude
SC_FASE	16	Phase
SC_PARAM	32	Parametrical signal

MinY, MaxY

```
double MinY
double MaxY
```

Minimum and maximum signal values. If MaxY< MinY, the minimum and maximum signal values are not yet calculated.

MinX, MaxX

```
double MinX
double MaxX
```

Minimum and maximum values of the signal abscissa axis. If the signal parameters change in time, MinX and MaxX are the start and end of the parameter registration, respectively. MinX and MaxX are read only accessible and van be modified by changing StartX and DeltaX or, at unequal interval, by SetX().

k0, k1

```
double k0
double k1
```

The coefficients of a calibration character, given as a linear function: $y = k_1 \cdot (x - k_0)$.

Methods

Instance

```
long Instance()
```

Return the object pointer which provides the present interface. Instance is used to transfer the object as parameter.

	Object pointer

IndexOf

```
long IndexOf(double x)
```

The value index (ordinal number) is returned corresponding to the given time, and if the exact value for this time is not - index of the nearest value.

	Number of a signal element from the range 0..(size-1)
x	The given value of time (abscissa axes)

GetY, GetX

```
double GetY(long index)
double GetX(long index)
```

Return the signal element value by ordinate or abscissa axis.

	Signal value by ordinate or abscissa axis

index	Signal element number of the range 0..(size-1)

GetYX

```
double GetYX(double x, int pow)
```

The signal value is returned corresponding to the given time and if the exact value for this time is not – the interpolated value.

Pow determines the method of an interpolation.

	Given value
x	Given the time value времени (abscissa axes)
pow	Type of an interpolation: 0 – absent (is taken the last value over time), 1 - linear interpolation, 2 - square polynomial, 3 - interpolation by cubic local splines

SetY, SetX

```
void SetX(long index, double value)
void SetY(long index, double value)
```

Set the signal element number by ordinate or abscissa axis. SetX makes no sense for the signals with uniform X axis, and for such signals StartX and DeltaX properties shall be set.

index	Signal element number of the range 0..(size-1)
value	New value

IWPUSML

Batch file interface (USML and MERA).

Properties

FileName

```
BSTR FileName
```

Full file name.

ParamCount

```
long ParamCount
```

Number of parameters of the batch file (USML or MERA).

Name, Test, Date

```
BSTR Name
BSTR Test
BSTR Date
```

Name of product, test and test data in “dd.mm.yy” format.

Methods

Instance

```
long Instance()
```

Return the object pointer which provides the present interface. Instance is used to transfer the object as parameter.

	Object pointer

Parameter

```
IDispatch* Parameter(long index)
```

Return the signal from the batch file by number.

	The object supporting IWPSignal interface
index	Number of signal of the range 0..(ParamCount-1)

FileSave

```
void FileSave()
```

Save file.

AddParameter

```
void AddParameter(IDispatch* signal)
```

Add signal to USML or MERA file.

signal	The object supporting IWPSignal interface

DeleteParameter

```
void DeleteParameter(long index)
```

Delete parameter with the specified number.

index	Number of signal of the range 0..(ParamCount-1)

IWPOperator

Calling interface of mathematical algorithms. According to the used terminology, *Operator* is a mathematical *Algorithm* jointly with *Parameters* of performance of this algorithm.

Properties

Name

```
BSTR Name
```

Short algorithm name.

Fullname

```
BSTR Fullname
```

Full algorithm name.

nSrc, nDst

```
long nSrc
long nDst
```

Number of input and output parameters. For example, for the amplitude spectrum nSrc=1, nDst=1; and for the mutual correlation function nSrc=2, nDst=1.

Methods

Instance

```
long Instance()
```

Return the object pointer which provides the present interface. Instance is used to transfer the object as parameter.

	Object pointer

Exec

```
long Exec(VARIANT src, VARIANT src2, VARIANT dst,
VARIANT dst2)
```

Execute the algorithm. If the actual number of the input (output) signals of the algorithm is less than two, the unused parameters are ignored.

	Error code. Zero, if successful.

src	First input signal
src2	Second input signal
dst	First output signal
dst2	Second output signal

Error

```
long Error()
```

Get the last error code.

↵	Error code. Zero, if no error.

MsgError

```
BSTR MsgError()
```

Get the last error message.

↵	Text description of the last error.

getProperySet

```
BSTR getProperySet()
```

Get the algorithm option list.

↵	Line of the algorithm option names, enlisted by comas.

setProperty

```
void setProperty(BSTR name, VARIANT value)
```

Set the selected algorithm property value.

Name	Property name
Value	New property value

getProperty

```
VARIANT getProperty(BSTR name)
```

Read the selected algorithm property value.

↵	Property value
Name	Property name

loadProperties

```
void loadProperties(BSTR values)
```

Load the algorithm property set values.

values	New values of the properties set. Format line « name_property1 = value_property1 , name_property2 = value_property2 , … » . For example: *" kindFunc = 3 , numPoints = 1024 , nBlocks = 1 "*. The values of skipped properties are not assigned (default values are stored).

getPropertyValues

```
BSTR getPropertyValues()
```

Read values of all algorithm properties.

	Format line « name_property1 = value_property1 , name_property2 = value_property2 , … » .

SetupDlg

```
long SetupDlg()
```

Call the dialog of the algorithm option setup and the source signals' selection.

	Dialog execution output. The Table below contains the returned values.

Result	Value	Description
IDOK	1	Algorithm execution started
IDCANCEL	2	Operation canceling
IDERROR	-1	Error open dialog

IWPNode

WinPOS object tree node, the object «mount point».

Properties

Name

```
BSTR Name
```

Name of node, object.

ChildCount

```
long ChildCount
```

Number of child elements of a given node. For example, the number of signals for the batch file node.

Methods

Instance

```
long Instance()
```

Return the object pointer which provides the present interface. Instance is used to transfer the object as parameter.

	Object pointer

Root

```
IDispatch* Root()
```

Pointer to the **WinPOS** object tree root node.

	The object supporting IWPNode interface

AbsolutePath, RelativePath

```
BSTR AbsolutePath()
BSTR RelativePath(IDispatch* baseNode)
```

Absolute or relative node path.

	Line, node path.
`baseNode`	The node by which the relative path is calculated

Reference

```
IDispatch* Reference()
```

The reference object of a given node.

	The reference object of a given node.

IsDirectory

```
long IsDirectory()
```

Check if a given node is a folder or a batch file.

	1 – folder, 0 - other.

GetReferenceType

```
long GetReferenceType()
```

Type of object the given node is referring to.

	Type of object. The possible types are given in the Table below.

Type	Value	Description
OT_FOLDER	0	Ordinary folder
OT_PFILE	1	USML or MERA file
OT_SIGNAL	2	Signal

Link

```
IDispatch* Link(IDispatch* Object, BSTR name, long flag)
```

Place the object to the child node list of the given node.

	The object supporting IWPNode interface
Object	The object to be placed to the tree
name	Object name
flag	If the node with such name already exists, at flag=1 new node name is modified («Name» is changed to «Name#1»), at flag=0 the old object is replaced.

Unlink

```
void Unlink(BSTR objname)
```

Delete child node with the set name of the given node.

objname	Line, node name to be deleted

IsChild

```
long IsChild(IDispatch* testNode)
```

Check if the node is a child of a given node.

	Child - 1, otherwise – 0.

`testNode`	The object supporting IWPNode interface

GetNode

```
IDispatch* GetNode(BSTR path)
```

Get the child node by name.

	The object supporting IWPNode interface
`path`	Path to the child node

At

```
IDispatch* At(long index)
```

Get the child node by number.

	The object supporting IWPNode interface
`index`	Number, 0 .. (ChildCount-1)

Part 4. Interfaces of plug-ins

Any connected module has to incarnate the interface IWPPlugin. See chapter *"Creating a plug-in step by step"* part 2. The interfaces IWPImport and IWPExport serve for the access to data files of irregular format.

`IWPPlugin`	- the main interface of any plug-in
`IWPImport`	- the interface of the data import
`IWPExport`	- the interface of the data export

Interfaces of plug-ins are dual. Returnable value (HRESULT) – an integer: 0 – a call successfully realized (S_OK), otherwise – an error code.

IWPPlugin

The main interface of a plug-in, taking commands and messages of WinPOS.

Methods

Connect

```
HRESULT Connect(IDispatch* app, long* Value)
```

WinPOS calls this method during an upload, passing a pointer to the main application interface.

`app`	A pointer to the main application interface – IWinPOS
`Value`	A returnable value. Is not used.

Disconnect

```
HRESULT Disconnect(long* Value)
```

To disconnect a plug-in.

`Value`	A returnable value. Is not used.

NotifyPlugin

```
HRESULT NotifyPlugin(long what, VARIANT* param, long * Value)
```

A notification of WinPOS events.

what	An event code. For example, for a pressing of a button on the toolbar: superior word, HiWord(what) – the command code (see RegisterCommand), junior – LoWord(what) = 2.
param	Additional data, depending on a message type
Value	A returnable value. Is not used.

IWPImport

A list of file formats maintained by the WinPos can be expanded. For the files reading realize the interface IWPImport and call the function RegisterImpExp() in the method Connect(), passed the pointer to this interface in the first parameter.

Methods

Open

```
HRESULT Open(BSTR path, long * Count, HRESULT * ErrorCode)
```

It is called at a pressing of the button **Open** in the window of a file select. In this method it is possible to count all file signals and to create a list to which WinPOS will appeal through GetSignal().

path	A name of the selected file
Count	A number of signals which are contained in the file.
ErrorCode	A code error, 0 – if the file has been read correctly.

Close

```
HRESULT Close()
```

To close a file. It is called upon termination of a reading of signals. Here it is possible to clear a list of opened signals

GetSignal

```
HRESULT GetSignal(long n, IDispatch ** Value)
```

WinPOS calls this method placing signals in the tree of signals.

n	A serial number of the signal in the file.
Value	An interface pointer IWPSignal of a next signal.

GetPreviewText

```
HRESULT GetPreviewText(BSTR path, BSTR * Value)
```

A notification of WinPOS events.

path	A name of the selected file.
Value	A string, placed in the bottom of the window of the file opening. It may contain information about a quantity of signals and recording features.

IWPExport

A list of file formats maintained by the WinPos can be expanded. For the files saving realize the interface IWPExport and call the function RegisterImpExp() in the method Connect(), passed the pointer to this interface in the second parameter.

Methods

AddSignal

```
HRESULT AddSignal(IDispatch* sig)
```

By means of this method WinPOS transmits to the export plug-in the signals selected for a saving.

sig	An interface pointer IWPSignal of a next signal.

Save

```
HRESULT Save(BSTR path, HRESULT* ErrorCode)
```

To disconnect of a plug-in

path	A name of a selected file
ErrorCode	A code error, 0 – if the file has been saved correctly.

Part 5. The call of algorithms

The algorithms are available through the tree of WinPOS objects. That is, the algorithms can be accessed by name, choosing from the objects tree. The sequence of calls is such: to get an operator, to load the necessary settings, to carry out the operator. So the call of an autospectrum with current settings looks:

```
var oper : IWPOperator;
…
oper:= WINPOS.GetObject('/Operators/ Auto spectrum'') as
IWPOperator;
oper.Exec (signal, signal, refvar(dst), refvar(dst2));
```

Download the settings of the algorithm can either in turn by the method setProperty(), or simultaneously, by the method loadProperties(), see above the description of the interface IWPOperator. Values of the omitted parameters are not assigned (values by default are saved). So the same call of the autospectrum with specifying settings looks:

```
var oper : IWPOperator;
…
oper:= WINPOS.GetObject('/Operators/Auto spectrum') as
IWPOperator;
oper.loadProperties(' kindFunc = 3 , numPoints = 1024 , typeWindow
= 1 ');
oper.Exec(signal, signal, refvar(dst), refvar(dst2));
```

Procedures of a simplified call of algorithms

A call most used algorithms is automated. In the file **POSBase.pas** for VBScript – in the **WinPOS.wps**) the procedures, designed in the style of calls " POS command mode", simplifying the call of algorithms, are realized. The example above can be rewritten as:

```
RunFFT(signal, dst, dst2, Opt, Err);
```

The names of the procedures are listed below together with a description of the algorithms settings. Designation `Src, Src2, Dst, Dst2` are variables, pointing to objects with interface IWPSignal, `Err` – the error code (0, if no errors), `Opt` – the line of settings, where the parameters with the values listed through a comma: « name_property1 = value1 , name_property2 = value 2 , … » . The example: *" kindFunc = 3 , numPoints = 1024 , nBlocks = 1 ".*

Algorithms on basis of the Fast Fourier Transformation (FFT).

Algorithms, implementing FFT, have some common settings:

`type`	Type of function. See the Table below
`kindFunc`	Depending on the value of the field TYPE, may contain values from different sets of constants More details see the table below
`method`	Calculation method: 0 – FFT, 1 - DFT
`numPoints`	Number of points of FFT calculations: 32…1048576
`nBlocks`	Number of averaging blocks: 1…(signal length/numPoints)
`ofsNextBlock`	Block shift in respect to each other: 1, numPoints/4, numPoints/2, numPoints*3/4, numPoints
`typeWindow`	Window function type (see the Table below)
`typeMagnitude`	Type of values (see the Table below)
`isMO`	Centering: 1 – enabled, 0 – disabled
`isFill0`	Supplement by zeros: 1 – supplement, 0 – no
`fMaxVal`	Maximum values: 1 – Maximum, 0 – averaged
`fLog`	Llogarithm: 1 – the result in dB, 0 – no
`log_kind`	0 – 20*logX, 1 – 10*logX
`log_fOpZn`	Use the reference value: 1 – use, 0 - no
`log_OpZn`	Reference value
`fPrSpec`	To implement the transformation of the spectrum
`prs_kind`	A kind of a transformation: 0 – 1, 1 – $1/\omega$, 2 – $1/\omega^2$, 3 – $2\sqrt{2}/\omega^2$, 4 – $1*\omega$, 5 – $1*\omega^2$
`prs_loFreq`	Lower frequency
`prs_s2n`	A relation signal / noise
`prs_fCorr`	To use the function-corrector
`prs_typeCorr`	A function type: 0 – custom (given in prs_strCorr), 1 – function A, 2 – function B, 3 – function C
`prs_strCorr`	The function-corrector (Line of the type "x1 y1 x2 y2 x3 y3...")
`f3D`	Flag of the three-dimensional presentation of the results: 1 - result - a three-dimensional spectrum, 0 - no
`fSwapXZ`	Time along the axis X: 1 - along the axis X - the time, along the axis Z - frequency, 0 - along the axis X - frequency, along the axis Z - time (for 3D)

Values of the field `type`

Constant	Value	Description
AUTOSPECTR	0	Auto spectrum
CROSS	20	Cross spectrum
COHEREN	30	Coherence function
TRANS	40	Transfer function

COMPLEX	50	Complex spectrum

Values of the field `typeWindow`

Constant	Value	Description
SINGLEWIN	1	Rectangular function
TRIANGLEWIN	2	Triangle function
HANNINGWIN	3	Hanning function
BLACKMANWIN	4	Blackman function
FLATTOP	5	Flat-Top

Values of the field `typeMagnitude`

Constant	Value	Description
MEAD	1	Effective
PEAK	2	Amplitude values
MAXPEAK	3	Maximum amplitude values

Auto spectrum

Shortcut:

```
procedure RunFFT(const Src : OleVariant; var Dst, Dst2, Opt,
Err : OleVariant)
```

Settings:

`type`	0 (AUTOSPECTR)

`kindFunc` may take the following values.

Constant	Value	Description
SPM	1	Power density spectrum
SM	2	Power spectrum
SPP	3	Energy density spectrum
SMAG	4	Amplitude spectrum
SRI	5	Complex spectrum as real and imaginary parts
SMF	6	Complex spectrum as module and phase

Octave spectrum

Shortcut: no

Setting:

`type`	0 (AUTOSPECTR)
`fFlt`	Calculation method of the spectrum: 1 – band-pass filters, 0 – FFT
`fQual`	1 – use filters of a high accuracy, 0 – simple

`kindFunc` may take the following values:

Constant	Value	Description
Oktav1	10	Octave spectrum
Oktav3	11	third-octave spectrum
Oktav12	12	1/12- octave spectrum
Oktav24	13	1/24- octave spectrum

Cross spectrum

Shortcut:

```
procedure RunCrossFFT(const Src, Src2 : OleVariant; var Dst,
Dst2, Opt, Err : OleVariant)
```

Settings:

`type` 20 (CROSS)

`kindFunc` may take the following values.

Constant	Value	Description
CrSPM	21	Power density spectrum
CrRI	22	Cross spectrum as real and imaginary parts
CrMF	23	Cross spectrum as module and phase

Complex spectrum

Shortcut:

```
procedure RunComplexFFT(const Real, Imag : OleVariant; var
Dst, Dst2, Opt, Err : OleVariant)
```

Settings:

`type` 50 (COMPLEX), `kindFunc` is ignored

Coherence function. Non-coherence function

Shortcut:

```
procedure RunCoher(const Src, Src2 : OleVariant; var Dst, Opt,
Err : OleVariant)
```

Settings:

`type` 30 (COHEREN)

`kindFunc` may take the following values.

Constant	Value	Description
COHERF	31	Coherence function
COP	32	Coherent output power

S_N	33	SNR
NOTCOP	34	Non-coherent output power
NOTCHR	35	Non-coherence function

Transfer function

Shortcut:

```
procedure RunFuncTransfer(const Src, Src2 : OleVariant; var
Dst, Dst2, Opt, Err : OleVariant)
```

Settings:

type	40 (TRANS)

`kindFunc` may take the following values.

Constant	Value	Description
H1	41	H1 transfer function
H2	42	H2 transfer function

Spectrum transformation

Shortcut: no

Settings:

kind	transformation type: 0 – 1, 1 – $1/\omega$, 1 – $1/\omega$, 2 – $1/\omega^2$, 3 – $2\sqrt{2}/\omega^2$, 4 – $1*\omega$, 5 – $1*\omega^2$
loFreq	Lower frequency
signal2noise	ratio signal / noise
useCorrector	Use a function-corrector
strCorrector	Function-corrector (string type "x1 y1 x2 y2 x3 y3...")
typeCorr	Function type: 0 – user (in strCorrector), 1 – function A, 2 – function B, 3 – function C

Filtering algorithms

Infinite impulse response filtering (IIR)

Shortcut:

```
procedure RunIIRFiltering(const Src : OleVariant; var Dst,
Opt, Err : OleVariant)
```

Settings:

iType	Approximation type (see the Table below)
iKind	Filter type (see the Table below)
nOrder	Number of 2nd order sections (order): 1…20
nRipple	Ripple (%) in the passband: 1…5
fsr	Cutoff frequency (for LPF, HPF)
fn	Low cutoff frequency (for BPF)
fv	Top cutoff frequency (for BPF)
fs	Sampling rate
HO	Filter coefficient

Values of the field `iKind`

Constant	Value	Description
LowPass	1	Low-Pass Filter (LPF)
BandPass	2	Band-Pass Filter (BPF)
HighPass	3	High-Pass Filter (HPF)

Values of the field `iType`

Constant	Value	Description
Butterworth	1	Butterworth filter
Chebyshev	2	Chebyshev filter
Elliptic	3	Elliptic filter

Finite impulse response filtering (FIR)

Shortcut:

```
procedure RunFIRFiltering(const Src : OleVariant; var Dst,
Opt, Err : OleVariant)
```

Settings:

iType	Approximation type (with Fourier series). Ignored
iKind	Filter type (see the Table below)

iTypeWin	Window type (see the Table below)
nOrder	Number of coefficients (order), odd number: 1…1001
fsr	Cutoff frequency (for LPF, HPF)
fn	Low cutoff frequency (for BPF, BEF)
fv	Top cutoff frequency (for BPF, BEF)
fs	Sampling rate

Values of the field iKind

Constant	Value	Description
LowPass	1	Low-Pass Filter (LPF)
BandPass	2	Band-Pass Filter (BPF)
HighPass	3	High-Pass Filter (HPF)
BandStop	4	Band-Eliminate Filter (BEF)

Values of the field iTypeWin

Constant	Value	Description
HANN	2	Hann window
HAMMINGWIN	3	Hamming window

Median Filtering

Shortcut: no

Settings:

Type	Filter type: 0 - discrete, 1 - analog
nPoints	Number of points
Level	Threshold (only for analog filter)
LevelLow	The lower level (for the discrete filter)
LevelHi	The upper level (for the discrete filter)
bAuto	Automatic detection of levels (for discr.).

Operations on signals

Differentiation

Shortcut:

```
procedure RunDiff(const Src : OleVariant; var Dst, method, Err
: OleVariant)
```

Settings: `method` `method` may take the following values.

Constant	Value	Description
`THREE_POINTS`	3	3 point method
`FIVE_POINTS`	5	5 point method

Integration

Shortcut:

```
procedure RunIntegral(const Src : OleVariant; var Dst, method,
numpointsAverg, typeRezult, Err : OleVariant)
```

Settings:

`method`	The method of an integration. See table below.
`typeRezult`	Centering, 1 - enabled, 0 - off
`numpointsAverg`	Number of points averaged (only for RC)
`flagDelPerProcess`	The suppression of the transition process: 1 - enabled, 0 - off. (Only for vibro)
`npointsPerProcess`	The length of the transition process (only for vibro)
`fsr`	The cutoff frequency of filtration (only for vibro)

Constant	Value	Description
`AILER_INT`	1	Euler method
`HANNING_INT`	2	Hanning method
`RC_INT`	3	RC-chain method
`VIBRO_INT`	4	Vibrointegration

Normalization

Shortcut: no

Setting:

`HiFront`	Upper confine
`LoFront`	Lower confine
`EnaShift`	The shift of signal values relative to 0: 1 - allow (changing statistical characteristics: MO, dispersion, etc., are changed) 0 - disable

Centering

Shortcut: no

Settings: no

Arithmetic operation

Shortcut: no

Settings:

`kind`	Type of the operation. Can take the values shown in the table.
`const`	Constant (for operation with one signal)

Constant	Value	Description
CONST_PLUS	0	addition of the constant *const*
CONST_MINUS	1	subtraction of the constant *const*
CONST_MULTI	2	multiplication by the constant *const*
CONST_DIV	3	division into the constant *const*
BUF_PLUS	4	addition of the two signal values
BUF_MINUS	5	subtracting the value of second signal from the value of the first
BUF_MULTI	6	multiplication of the values of two signals
BUF_DIV	7	dividing the value of first signal on the value of the second signal

Taking the logarithm

Shortcut: no

Settings:

`kind`	20logX (0) или 10logX (1)
`useOpZn`	Use the reference value (1), otherwise (0) - max
`OpZn`	Reference value

Resampling

Shortcut:

```
procedure RunResampling(const Src : OleVariant; var Dst :
OleVariant; Freq, Method, FltType : OleVariant; var Err : OleVariant)
```

Settings:

`freq`	New sampling frequency
`kind`	Types of interpolation. See table below.
`type`	Type of filtering. See table below.
`srcdt`	Save the initial data type

Values `kind`

Constant	Value	Description
NOINT	0	No interpolation
LINEINT	1	Linear interpolation
PARABINT	2	Interpolation of a second order polynomial
SPLINE3INT	3	Cubic local splines

Values `type`

Constant	Value	Description
NOFLT	0	No filtering
IIRFLT	1	Recursive filtering
FIRFLT	2	Nonrecursive filtering

Hilbert transformation

Shortcut: no

Settings:

`nPoints`	The number of points on which the FFT is calculated: 32 ... 1048576
`nBlocks`	Number of averaging servings: 1 ... (length of the signal / nPoints)
`isMO`	Centering: 1 - On, 0 - off

Envelope

Shortcut: no

Settings:

`kind`	Method: 0 - peak-detector, 1 - Hilbert transform
`coef`	Coefficient (K) for the method of peak-detector

If the method of the Hilbert transform is selected, apply the settings of the Hilbert transform are also applied for this algorithm (see above).

Investigation of signals

Probabilistic characteristics

Elements of the resulting signal (Dst) contain the values of the probability characteristics of the original signal.

Constant	Displacement	Description
IDX_MO	0	average of distribution
IDX_D	1	dispersion
IDX_SIG	2	mean-square deflection.
IDX_A3	3	Asymmetry
IDX_A4	4	Kurtosis
IDX_MAG	5	amplitude

Thus, to obtain, for example, the dispersion of the signal, should call the method Dst.GetY (1) after the execution of the algorithm.

Shortcut: no *Settings:* no

Probability density

Shortcut:

```
procedure RunPRV(const Src : OleVariant; var Dst, npoints,
type, Err : OleVariant)
```

Settings:

`npoints`	Number of calculation points
`type`	Method of calculation and representation of PDF (see the Table below).

Constant	Value	Description
`PARZEN`	1	PDF, core estimation method
`HIST`	2	PDF, histogram calculation
`PARZNORM`	8	Probability, core estimation method
`HISTNORM`	4	Probability, histogram calculation

Auto correlation

Shortcut:

```
procedure RunAutoCorel(const Src : OleVariant; var Dst,
npoints, eps, Err : OleVariant)
```

Settings:

`npoints`	Number of points for correlation function plotting
`type`	Return the statistic error value

Cross correlation

Shortcut:

```
procedure RunCrossCorel(const Src, Src2 : OleVariant; var Dst,
npoints, eps, Err : OleVariant)
```

Settings:

`npoints`	Number of points for correlation function plotting
`type`	Return the statistic error value

Parametric graph

Shortcut: no

Settings:

`type`	0 - parametric graph 1 - polar, 2 - parameter for the signals at the same sampling (values are taken with the same indexes)

Part 6. Embedded script editor

Borland Delphi is the best suitable tool for writing own effective processing algorithms, processing of huge data, creation of applications based on WinPOS but requiring additional customization or able to generate specialized reports. Borland C++ Builder, Microsoft Visual C++, Visual Basic or FoxPro can also be used.

However, Visual Basic Script is the most suitable tool for writing of small scripts for WinPOS operation or simple algorithms. VBScript is included into the Microsoft Windows package, requires no separate compiler, and WinPOS includes a convenient editing and debugging environment for scripts.

Script editor (Fig. 6.1) is opened by the menu **Script→ Script editor…**

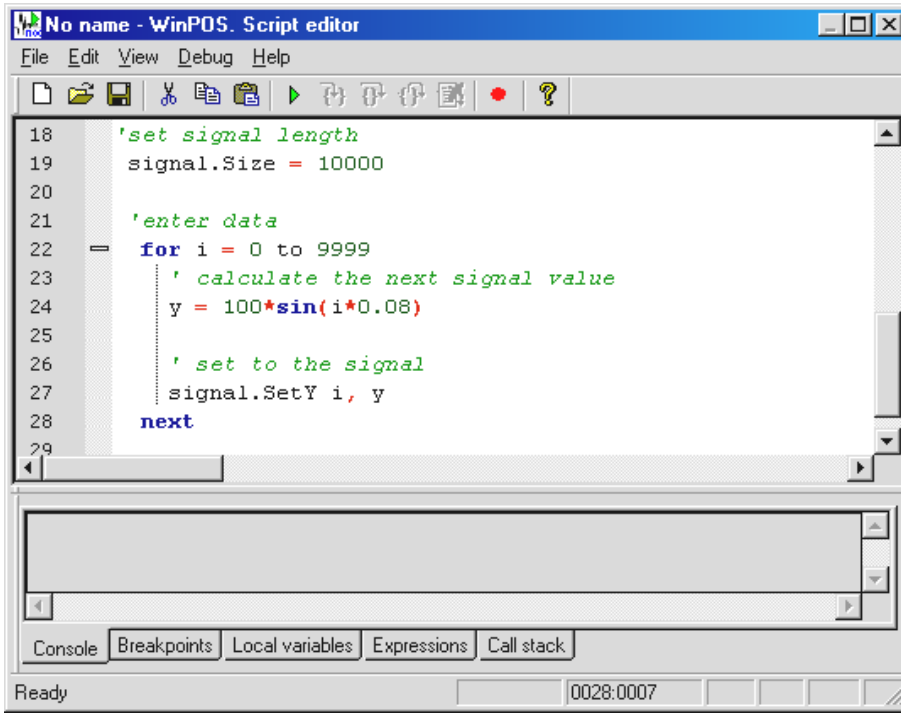
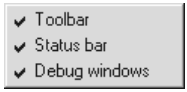


Fig. 6.1. Script editor window

The script editor is a text editor with syntax highlighting and a standard toolset accessible through menu, toolbars and hotkeys. The editor also provides all necessary tools for the debugging script execution: breakpoints and step execution, viewing of local variables and call stack, calculation of expressions.

The embedded script editor is distinguished by the following features:

- Debugging control buttons on the toolbar,
- Syntax highlighting,
- Brace matching control and indent control,
- Line numbering with breakpoints (on the left),
- Debugging panels (below the editing area).



The menu **View** serves for enabling and disabling of the editor visual elements.

The syntax highlighting allows reduction of errors at the script text typing and helps the text perception:

Blue bold shows VBScript reserved words,

Blue – symbols,

Italic – line constants,

Green – comments, and

Identifiers are shown by conventional black font.

The pair brackets are shown by green background:

```
y = 100*sin(i*0.08)
```

The Tab symbol is marked by a vertical strip which allows better visibility of included cycles and conditions.

Editing mode

The script editing mode commands are:

Toolbar	Menu	Keyboard shortcut	Description
	File→ New script	Ctrl+Shift+N	Clear window for a new script
	File → Open script …	Ctrl+Shift+O	Open file to edit
	File → Save script	Ctrl+Shift+S	Save edited file
	File →Save script as…		Save script with a new name
	File →Quit	Alt+F4	Close the editor window
	Edit→Undo	Ctrl+Z	Undo the last action
	Edit →Cut	Ctrl+X	Cut the selected fragment to the buffer
	Edit →Copy	Ctrl+C	Copy the selected fragment to the buffer
	Edit →Paste	Shift+V	Paste the buffer text at the cursor position
	Edit →Select all	Shift+A	Select all text
	Edit →Find…		Find the line
	Edit →Replace…		Find and replace the line
	Edit →Go to line…		Go to the line with the set number
	Help→Index…	F1	WinPOS object help.

The dialog *Go to line* (Fig. 6.2, menu **Edit→Go to line…**) helps the navigation in a lengthy script. The line number is shown both on the left margin and on the status bar (cursor position, the line is the first digit in the pair "llll:cccc").

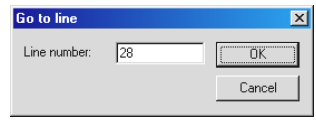


Fig. 6.2. Go to line

The dialog *Find* (Fig. 6.3, menu **Edit→Find…**) allows identification of all occurrences of the line set in the field *Find text*, considering the case (**Case sensitive**) and the position in the surrounding text (**Whole words** and **Regular expressions**). The buttons **Next** and **Back** set the searching directions in respect to the current cursor positions.

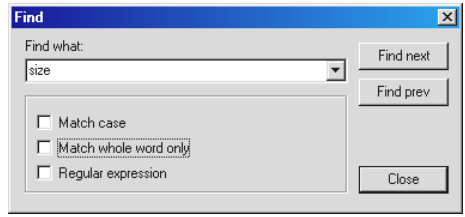


Fig. 6.3. Find dialog

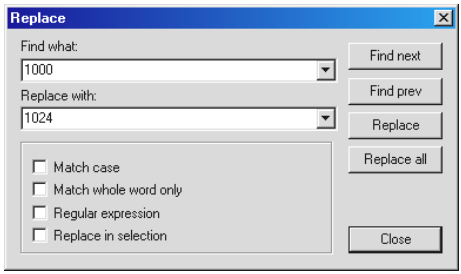


Fig. 6.4. Find and replace dialog

The dialog *Find and replace* (Fig. 6.4, menu **Edit→replace…**) repeats the dialog *Finds* with additional option of replacement of the found line by the line specified in the field *Replace*. The button **Replace** makes one replacement, **Replace all** – automatically replaces all identified text. The flag **Replace in selection** allows limiting of the text modification area.

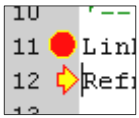
Debugging mode

The commands of the script execution and debugging are:

Toolbar	Menu	Keyboard shortcut	Description
	Debugging→Start/Continue execution	Ctrl+F10	Switch to the script execution mode
	Debugging →Enable/Disable stop point	F9	Set breakpoint
	Debugging →Step In	F11	Step execution of procedures
	Debugging →Step Over	F10	Step execution
	Debugging →Step Out	Ctrl+F11	Quit procedure
	Debugging →Stop debugging	Alt+F10	Stop script execution
	Debugging →Restart		Restart mode

ⓘ In the debugging mode the script text cannot be edited.

The transfer to the debugging mode is made by the button . If the breakpoints (, breakpoints) are not set, and the restart option is not enabled (**Debugging→Restart**), the script will be completely executed.



To set the breakpoint on the selected line press the button or <F9>. Breakpoint appears on the current line, and a read point will be made on the margin.

In order to continue the script execution the button can be pressed again. The scrtpt execution can be continued by steps by the buttons , and . The position of the line being the next for execution is marked by yellow arrow in the margin.

Debugging panels

The debugging panels provide a full overview of the executed script status at each instance (allow tracking of the script execution, variable content modifications, etc.).

Console

The debugging printing (functions DebugPrint() and DebugPrintLn()) is directed to the *Console* (Fig. 6.5).

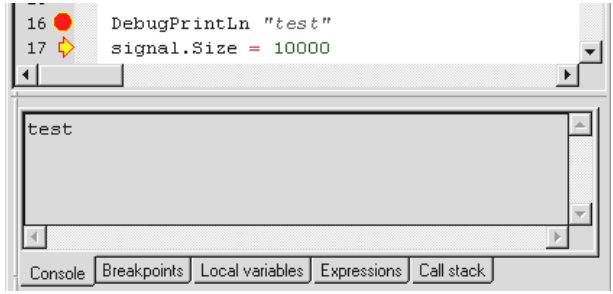


Fig. 6.5. Console

Breakpoints

Setting of *breakpoints* (Fig. 6.6) shows the list of breakpoints with line numbers, status, and the counter of line passing. The breakpoint status (*active* or *blocked* – is marked by grey) can be modified by the context menu, the breakpoint can be removed by <F9> and by the context menu.

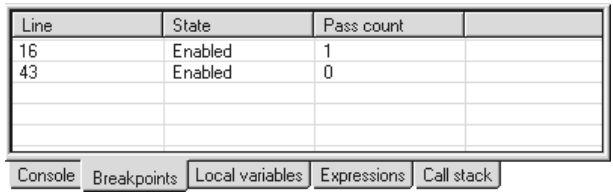


Fig. 6.6. Breakpoints tab

Local variables

The values and types of *local variables* can be viewed at the bookmark of the same name (Fig. 6.7).

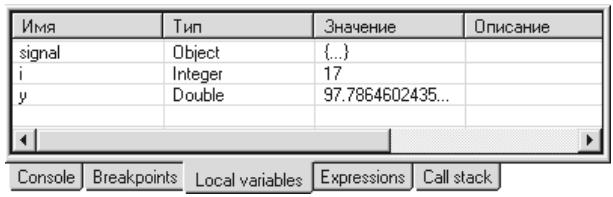


Fig. 6.7. Local variables tab

Double mouse click at the variable line opens the dialog 6.8 where the variable value can be viewed and modified (the field *Value,* the button **Refresh**).

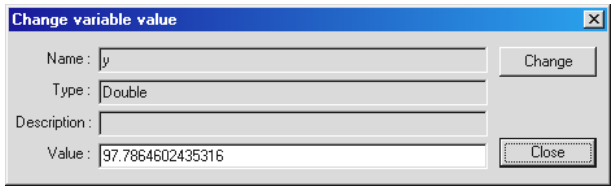


Рис. 6.8. Change variable value dialog

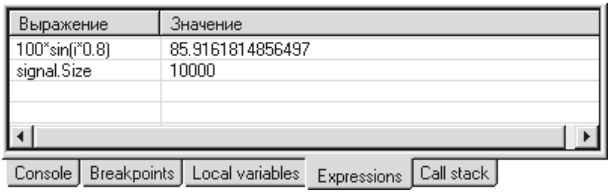


Fig. 6.9. Setting expression calculations tab

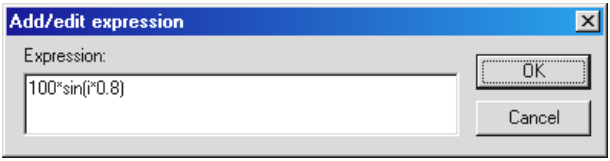


Fig. 6.10. Expression addition dialog

Expressions

Bookmark of *expression* (Fig. 6.9) enables calculation of any expression written by VBScript syntax.

New expression can be added, deleted or modified by context menu containing the expression editing dialog (Fig. 6.10).

The selected script line can be copied to the bookmark *Expressions* via the editor context menu.

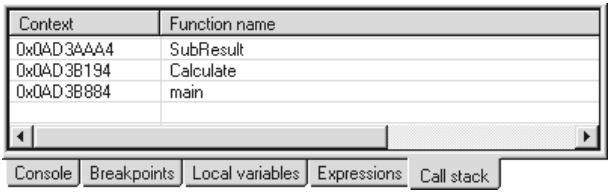


Fig. 6.11. Call stack tab

Call stack

The last bookmark, *Call stack* (Fig. 6.11), is an aid for the script debugging. This bookmark including a large number of procedures is irreplaceable for writing a code with recursive calls. The current procedure stands at the top of the stack.

Appendix. Samples

When installed **WinPOS** creates the subdirectory Samples in its operation catalog containing the samples of script implementation by VBScript and the samples of creation of applications and plug-ins by Delphi.

`Samples \`	
`VBS`	- Sample scripts (Visual Basic Script)
`Delphi`	- Sample scripts (Delphi)
`DelphiPlugIn`	- Sample of plug-in (Delphi)
`DelphiCommon`	- Service modules (Delphi)

The catalogs VBS and Delphi contain six program samples each.

Sample number	Description	Employed options
1	Signal generator. The signal is created and filled by the values calculated by the function.	✓ Signal creation ✓ Adding signal to the WinPOS tree ✓ Access to signal values
2	Graph representation of the signal. The signal is created and filled, and the generated signal is placed to the graph.	✓ Create signal with unequal X axis ✓ Access the WinPOS graph subsystem ✓ Signal plotting
3	Loading of arbitrary USML file, signal processing (AutoSpectrum).	✓ Load batch file ✓ Call RunFFT() procedure
4	Calling of the file selection dialog, load of binary data file, signal processing (call of WinPOS algorithm by name, setting of options), printing.	✓ Operate the file opening dialog ✓ Load binary file ✓ Execute the operator by Exec() ✓ Call PrintPreview() method
5	Loading of USML file, calculation of AutoSpectrum. Additional processing of the result: find three frequencies with the maximum amplitude values in the spectrum and print these values	✓ Additional processing of the WinPOS algorithm execution results ✓ Create page with two plots (source signal and resulting signal)
6	Loading of USML file, signal processing: resampling,	✓ Sequential execution of the algorithm chain

	filtering, autospectrum.	✓ Representation of several parameters by one graph

For the sake of the user's convenience, Delphi samples are represented by separate files enabled by the directive «$I». Each file contains the procedure body only. For compiling of the files as separate units, these files have to be completed by a standard outline: unit, interface, implementation…

The catalog DelphiPlugIn contains a sample of plug-in which adds the button enabling one of Delphi samples to the WinPOS toolbar.

The catalog DelphiCommon contains TLB file with description of **WinPOS** interfaces (WinPOS_ole_TLB.pas) and the file PosBase.pas with simplified access procedures for **WinPOS** algorithms and constant values descriptions.

A program sample of test signal generation program, spectrum calculation and graph representation is given below.

```
// Sample 1.
// Signal generation, algorithm call, graph plotting.
program Sample1;

uses
  Forms,
  Winpos_ole_TLB in '../DelphiCommon/Winpos_ole_TLB.pas',
  PosBase in '../DelphiCommon/PosBase.pas';

var signal: IWPSignal;
    y : Double;
    dst1, dst2, OptFFT, Err : OleVariant;
    api : IWPGraphs;
    i, hPage, hGraph, hGraphFFT, hAxis, hAxisFFT : Integer;

begin
  Application.Initialize;

  with WINPOS do
  begin

  // 1) create signal
  signal:= CreateSignal() as IWPSignal;
```

```
  if Assigned(signal) then   // if the signal is created
    begin

      // place signal to the tree
      Link('/Signals/generator', 'sinus', signal as IDispatch);
      Refresh();

      signal.size:= 10000; // set signal length

      for i:= 0 to 9999 do // enter data
      begin
        y:= (100)*sin(i*0.08); // calculate the next signal
        signal.SetY(i, y); // set to the signal
      end;

      // 2) apply the "Autospectrum" operator to the signal
      RunFFT(signal, dst1, dst2, OptFFT, Err);

      // place the result to the WinPOS tree
      Link('/Signals/Result', 'spectrum', dst1);

      // 3) represent the source and result signals
      // obtain access to the WinPOS graph subsystem
      api:= GraphAPI as IWPGraphs;

      // create new page for graphs
      hPage:= api.CreatePage;

      // the page is always created with plotting area
      hGraph:= api.GetGraph(hPage,0);

      // create additional graph for the spectrum
      hGraphFFT:= api.CreateGraph(hPage);

      // obtain Y axis
      hAxis:= api.GetYAxis(hGraph,0);

      // create a new line in the graph
      api.CreateLine(hGraph, hAxis, signal.Instance);

      // obtain Y axis in the second graph
      hAxisFFT:= api.GetYAxis(hGraphFFT,0);

      // create a new line in the spectrum graph
      api.CreateLine(hGraphFFT, hAxisFFT, dst1.Instance);

      // normalize graphs
      api.NormalizeGraph(hGraph);
      api.NormalizeGraph(hGraphFFT);

      Refresh;
    end;
  end; // with
end.
```

The following sample is a program generating a non-standard variant of express report. In the cycle USML or MERA file which parameters are displayed on the pages of 3x3 graph is processed, new scale is established on Y axis allowing estimation of the initial parameter levels of the set file.

```
// Sample 2.
// Express report generation program
program Express;

uses
  Forms,
  Winpos_ole_TLB in '../DelphiCommon/Winpos_ole_TLB.pas',
  PosBase in '../DelphiCommon/PosBase.pas';

var FileName  : string;
    signal    : IWPSignal;
    usml      : IWPUSML;
    api       : IWPGraphs;
    hPage, hGraph, hAxis, hLine : Integer;
    i, j, nGr, nPg : Integer;
    range, min, max : Double;

const nVer : Integer = 3;
const nHor : Integer = 3;

begin
  Application.Initialize;

  // WINPOS is defined and initialized in the POSBase.pas module
  with WINPOS do
  begin
    // open USML by standard WinPOS dialog
    FileName:= USMLDialog();

    if fileName<>'' then // if file is selected
    begin
      // obtain access to Winpos graph subsystem
      api:= GraphAPI as IWPGraphs;

      usml:= LoadUsml(fileName) as IWPUSML; //load USML

      // we get to new page creation (see below)
      nGr:= nHor*nVer;
      nPg:= 0;
      for i:=0 to usml.ParamCount-1 do
      begin
        // now we can take a signal by its number in the file
        signal:= usml.Parameter(i) as IWPSignal;
        if (nGr = nHor*nVer) then
        begin
```

```
          // create a new page for graphs
          hPage:= api.CreatePage;

          // set 3x3
          api.SetPageDim(hPage, PAGE_DM_TABLE, nVer, nHor);

          for j:=2 to nHor*nVer do
            api.CreateGraph(hPage);

          Inc(nPg);
          nGr:= 0;
        end;

        hGraph:= api.GetGraph(hPage, nGr);
        // obtain Y axis
        hAxis:= api.GetYAxis(hGraph,0);

        // create a new line in graph
        api.CreateLine(hGraph, hAxis, signal.Instance);

        // normalize graph
        api.NormalizeGraph(hGraph);

        range:= signal.MaxY - signal.MinY;
        max:= signal.MaxY + range*10;
        min:= signal.MinY - range*10;
        api.SetYAxisMinMax(hAxis, min, max);

        Inc(nGr);
      end;
    end;

    Refresh;
  end;
end.
```

Index of methods