WinПОС

Пакет Обработки Сигналов

Руководство программиста

Редакция 3.2.11

БЛИЖ.409801.002-04 33

Оглавление

Об этой книге

Руководство программиста содержит подробное описание интерфейса программирования (API) WinПОС, примеры и рекомендации по оформлению программ. Отдельная часть руководства посвящена работе со встроенным редактором сценариев.

Книга рассчитана на пользователей WinПОС знакомых с азами программирования. Написание сценариев (Visual Basic Script или VBS) не требует от программиста высокой квалификации. Однако, для разработки подключаемых модулей (плагинов) может быть полезно знакомство с концепцией объектно-ориентированного программирования (ООП) и основами технологии OLE и ActiveX.

Структура книги

Часть 1 рассказывает о возможностях и области применения программного интерфейса (API) WinПОС. Описывается внутренняя структура приложения. *Часть 2* призвана помочь в выборе языка и среды программирования в зависимости от сложности решаемой задачи. В *части 3* подробно описаны интерфейсы объектов WinПОС, свойства и методы. *Часть 4* описывает интерфейсы подключаемых модулей (плагинов). *Часть 5* содержит описание вызова алгоритмов. В *части 6* описан встроенный редактор-отладчик сценариев WinПОС. В *Приложении* приводятся примеры и описание расположения на диске исходных текстов примеров сценариев, программ и подключаемых модулей (плагинов). В конце книги помещен алфавитный *Указатель методов*.

Принятые обозначения

Для облегчения восприятия текста в руководстве используются те же обозначения, что и в *Руководстве пользователя*.

Кроме того, при описании интерфейсов также используются графические обозначения:

 - свойство,

 - метод,

 - возвращаемое значение функции.

Часть 1. Введение

С помощью программного интерфейса (API) WinПОС вы можете создавать свои собственные алгоритмы обработки сигнала, автоматизировать процесс обработки входного сигнала от выбора входного файла до документирования результатов обработки.

Области применения:

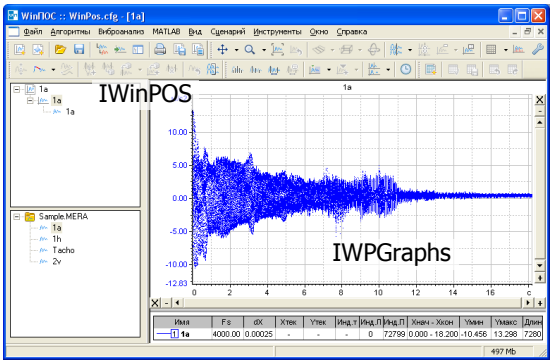
- циклическая обработка большого объема данных,
- вычисления по узкоспециализированным формулам,
- автоматический поиск характерных значений в результатах вычислений,
- программная генерация сигналов, к примеру, эталонных, с заданными свойствами,
- формирование шаблонных отчетов, таблиц,
- чтение и запись данных в нестандартных форматах и т.п.

Структура приложения WinПОС

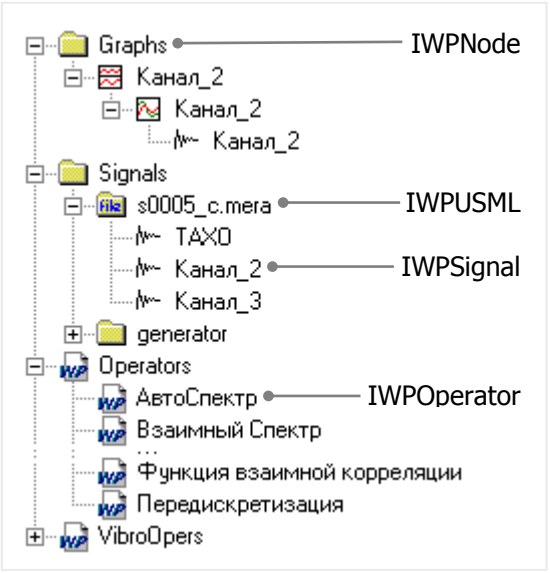
WinПОС является модульным приложением. В основе его программной модели лежит концепция объектно-ориентированного программирования (ООП). Условно объекты WinПОС можно разделить на группы, по их функциональной принадлежности:

- объекты, реализующие пользовательский интерфейс (меню, панели инструментов, окна настроек),
- объекты графической подсистемы (страницы, графики, линии),
- объекты, предоставляющие доступ к данным (сигналы, файлы данных),
- объекты, отвечающие за структурированное хранение данных (дерево объектов)
- объекты, реализующие математические алгоритмы (операторы).

Наиболее важные объекты доступны извне, что позволяет задействовать возможности пакета не только через пользовательский интерфейс, но и программно. Тем самым пользователи могут с помощью некоторого инструмента программирования автоматизировать часто встречающиеся задачи, которые нельзя было предвидеть в ходе разработки приложения. Такие объекты принято называть объектами ActiveX, а приложение – OLE-сервером. Объекты WinПОС предоставляют интерфейсы, вкратце описанные ниже.



IWinPOS – основной интерфейс приложения, всё взаимодействие с приложением так или иначе осуществляется через него, т.к. через этот интерфейс доступны не только графические элементы WinПОС (графическая подсистема доступна через вызов метода `GraphAPI()`), но и реализован доступ к дереву объектов.



Дерево объектов – это структурированное хранилище данных. Отдельные его поддеревья отображаются в окнах *дерева сигналов* (ветка “\Signals”), *дерева графиков* (ветка “\Graphs”), меню *Алгоритмы* (ветка “\Operators”) или панелях *Менеджера сигналов*.

Каждый объект (узел, «лист» дерева) доступен через интерфейс **IWPNode**. Метод `GetObject()` позволяет получить любой объект WinПОС из загруженных в его дерево. Посредством интерфейса **IWPNode** можно также осуществлять циклический перебор объектов.

К узлу дерева может быть привязан один из объектов:

- страница, график, линия (доступны через интерфейс **IWPGraphs**),
- файл МЕРА или УСМЛ (доступен через интерфейс **IWPUSML**),
- сигнал (**IWPSignal**),
- оператор (**IWPOperator**).

Часть 2. Варианты применения

WinПОС предоставляет пользователю интерфейсы, с помощью которых можно создавать свои подключаемые модули или приложения, работающие с данными и алгоритмами пакета, практически в любой современной среде программирования. Для примеров выбраны *Microsoft Visual Basic Script* и *Borland Delphi*. VBScript входит в состав поставки Microsoft Windows, не требует отдельного компилятора, а несложная среда редактирования сценариев включена в состав WinПОС. Delphi заслуженно пользуется славой самой удобной среды быстрой разработки приложений (RAD) и идеально подходит для создания небольших приложений частного применения.

Рассмотрим плюсы и минусы программирования сценариев или приложений этими средствами.

VBScript

- ⊕ не нужен компилятор или отдельная среда разработки,
- ⊕ требуются лишь минимальные навыки программирования,
- ⊖ невозможно написать приложение с собственными диалогами для настройки или формами.

Delphi

- ⊕ можно создавать диалоги и формы для настроек любой степени сложности, использовать многочисленные компоненты Delphi, создавать специфические отчеты,
- ⊕ легко писать сложные собственные алгоритмы для обработки данных,
- ⊖ требуется установленный пакет Borland Delphi и соответствующие навыки программирования.

Таким образом, VBScript больше подходит для написания небольших сценариев автоматизации работы WinПОС или несложных алгоритмов и слабо подходит для обработки значительных объемов данных, а Delphi лучше применять для написания собственных быстродействующих алгоритмов обработки и создания приложений, требующих дополнительных настроек или формирования специализированных отчетов.

Ниже рассматриваются способы совместного выполнения сценариев, приложений, модулей и среды WinПОС.

VBScript

Самый простой способ написать свой сценарий на VBScript – скопировать пример из *Справочной системы* или непосредственно с диска, вставить его в *Редактор сценариев* (меню *Сценарий*) и переделать его, добавив необходимую функциональность. Элементы управления *Редактором сценариев* описаны в *Части 5. Встроенный редактор сценариев*. Готовый сценарий можно запустить на выполнение несколькими способами:

- из редактора сценариев - *Выполнить программу* (F5),
- из главного окна WinПОС, пункт меню **Выполнить сценарий** или кнопкой быстрого запуска сценариев в *Панели инструментов*,
- из командной строки (“winpos.exe myscript.wps”).

Вот так на VBScript будет выглядеть классический пример:

```
sub main
      DebugPrint “Hello, world!”
end sub
```

Строка «Hello, world!» будет напечатана в окне отладочной печати Редактора сценариев (этот пример имеет смысл запускать только первым способом, т.к. иначе отладочная печать не будет задействована).

Delphi

С помощью RAD Delphi можно создавать не только приложения, которые будут обращаться к объектам и методам WinПОС, но и писать подключаемые модули, в виде динамических библиотек (DLL). Такие модули могут встраиваться в среду WinПОС. Например, легко можно добавить на панель инструментов WinПОС кнопки, вызывающие функции пользовательской библиотеки.

Приложения

Приложение (EXE-файл) может получить доступ к объектам их свойствам и методам WinПОС путем создания прокси-класса следующим образом:

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

Далее можно обращаться к методам WinПОС:

```
// открываем УСМЛ с помощью стандартного диалога WinПОС
FileName:= WinPos.USMLDialog();
```

Важно! С точки зрения приложения, WinПОС является выделенным (т.е. «out-of-process» или «outproc») сервером. А это означает, что вызовы методов объектов WinПОС будут происходить с неизбежными задержками, обусловленными небыстрым межпроцессным взаимодействием. Таким образом, отдельное приложение плохо подходит для создания собственных алгоритмов, циклически обращающихся к методам GetY сигналов или подобным. Первый пример (генератор синуса) наглядно показывает эту особенность. В то же время при разовых обращениях к сигналам или алгоритмам задержки практически неразличимы. Для задач, требующих постоянного взаимодействия с объектами WinПОС более подходят подключаемые модули, описанные ниже.

Подключаемые модули (плагины)

С помощью RAD Delphi можно создавать не только приложения, которые будут обращаться к объектам и методам WinПОС, но и писать подключаемые модули в виде динамических библиотек (DLL) – Plug-Ins(*англ.*), плагины. Такие модули могут встраиваться в среду WinПОС. Например, легко можно добавить на панель инструментов кнопки, вызывающие функции пользовательской библиотеки.

Для такого модуля WinПОС будет локальным сервером, и, таким образом, дополнительные временные издержки будут минимальными. Пользовательские алгоритмы по быстродействию практически не будут уступать встроенным.

Подключаемый модуль должен содержать COM-класс с дуальным интерфейсом, предоставляющим три метода: Connect(), Disconnect() и NotifyPlugin().

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

При запуске WinПОС вызывает метод Connect(), передавая указатель на себя, при завершении работы вызывается Disconnect(), а другие сообщения WinПОС передает, вызывая NotifyPlugin() с кодом и параметрами сообщения.

WinПОС загружает плагины по списку, сохраняемому в реестре. Добавление плагина в список и удаление из списка удобно совмещать с процедурами регистрации. Для этого требуется перекрыть DllRegisterServer и DllUnregisterServer.

Создание плагина шаг за шагом

1. Создайте новую библиотеку (DLL): **File→New→Other…→ ActiveX → ActiveX Library**.
2. Сохраните библиотеку: **File→Save**. В диалоге сохранения укажите имя библиотеки, например, «MyPlugin» и нажмите кнопку **Save**.
3. Создайте новый COM-объект: **File→New→Other…→ActivyX→COM Object**. Появится диалог *COM Object Wizard* (Рис. 2.1).

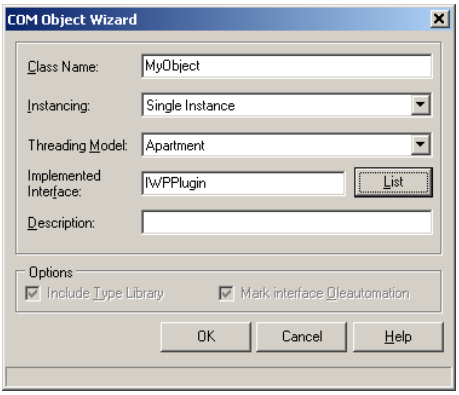


Рис 2.1 Диалог создания COM-объекта

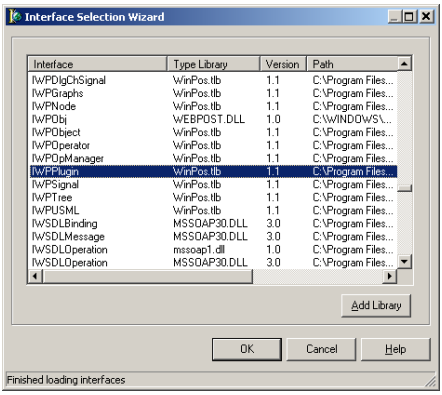


Рис 2.2 Диалог выбора интерфейса

- Укажите название класса в поле *Class Name*.
- Нажмите кнопку **List**. Откроется диалог *Interface Selection Wizard* (Рис. 2.2).
- Выберите интерфейс *IWPPlugin* (см. рис.) и нажмите **OK**. Откроется окно *Type Library*. Обратите внимание, что в левой части окна показаны новая библиотека и новый класс. Окно можно закрыть, для повторного вызова используйте **View→Type Library**.
- Нажмите **OK** для завершения создания COM-объекта.

4. В тексте исходного файла библиотеки (здесь это MyPlugin.dpr):
   - добавьте в секцию *uses* необходимые модули. Обычно это: SysUtils, Classes, Consts, Windows, ComServ, Registry;
   - перекройте функции DllRegisterServer и DllUnregisterServer, как показано ниже:

```
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;
```

Здесь второй параметр метода reg.writestring() – наименование нового COM-класса. Оно состоит из имен новой библиотеки и класса, разделенных точкой. Эти имена можно увидеть в левой части окна *Type Library*.

```
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;
```

- добавьте секцию exports:

```
exports dllgetclassobject Name 'DllGetClassObject',
  dllcanunloadnow Name 'DllCanUnloadNow',
  dllregisterserver name 'DllRegisterServer',
  dllunregisterserver Name 'DllUnregisterServer';
```

5. В модуле Unit1.pas:

   - добавьте в секцию *uses* раздела *implementation* модуль POSBase. В секции *uses* раздела *interface* должен быть включен Winpos_ole_TLB, а также Windows, ActiveX, Classes, ComObj, SysUtils, Forms и, при необходимости, другие;

- перекройте методы Connect(), Disconnect() и NotifyPlugin():

```
var ID_Run1 : Integer=0;      // Идентификатор (код) команды
var bar_ID : Integer;         // Дескриптор панели инструментов

function TMyObject.Connect(const app: IDispatch): Integer;
var hbmp:THandle;
begin
   WINPOS:=app as IWinPOS;
   ID_Run1 := WINPOS.RegisterCommand(); //Получение кода команды
   bar_ID:=WINPOS.CreateToolbar();   //Создание панели инструментов
                                     //Загрузка изображения кнопки
   hbmp:=LoadBitmap(HInstance,'TOOLBAR');
                                     //Создание кнопки
   WINPOS.CreatetoolbarButton( bar_ID, ID_Run1, hbmp,
                           'Новое действие'#10' Новое действие');
   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          //Проверка кода команды
    then
      begin                          //Создание формы
        Application.CreateForm(TForm1, Form1);
        Form1.Show;
      end
  except
  end;
  Result:= 0;
end;
```

В приведенном выше примере в методе Connect() создается новая панель инструментов и кнопка на ней. В методе NotifyPlugin() по нажатию кнопки создается форма, которой передается управление.

6. Создайте форму и нарисуйте изображение для кнопки панели инструментов – это bitmap 16x16 (можно использовать встроенный редактор ресурсов: **Tools→Image Editor**).

7. Скомпилируйте плагин. Зарегистрируйте его с помощью regsvr32. Закройте и снова запустите WinПОС. Должна появиться новая панель с копкой вызова подключаемого модуля.

Регистрация и отмена регистрации подключаемых модулей производится стандартным инструментом Windows:

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

В директории DelphiCommon (см. параграф *Установка и размещение файлов примеров*) находятся файлы Winpos_ole_TLB.pas и POSBase.pas.

Winpos_ole_TLB.pas создается при импорте библиотеке типов и включает описания OLE-интерфейсов WinПОС (файл подключается автоматически при наследовании интерфейса IWPPlugin).

POSBase.pas содержит функции типа `RunXXXX()`, упрощающие доступ к алгоритмам WinПОС, и ряд констант (подключение этого файла может быть полезным). См. также главу 3 *Процедуры упрощенного вызова алгоритмов*.

Другие средства

Как упоминалось выше, помимо *VBScript* и *Delphi*, в качестве среды для разработки приложений или подключаемых модулей могут быть использованы и другие средства. Для работы с ними потребуется выполнить последовательность действий, направленную на генерацию соответствующего программного модуля по библиотеке типов (Type Library – TLB) WinПОС.

Так в *Borland C++ Builder*, как и в *Delphi*, последовательность будет такой:

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

а в *Microsoft Visual C++* такой:

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

Часть 3. Интерфейсы WinПОС

Большинством объектов WinПОС можно манипулировать при помощи интерфейсов, перечисленных ниже.

`IWinPOS` - основной интерфейс приложения

`IWPGraphs` - интерфейс графической подсистемы

`IWPSignal` - интерфейс сигнала

`IWPUSML` - интерфейс пакетных файлов (УСМЛ и МЕРА)

`IWPOperator` - интерфейс вызова математических алгоритмов

`IWPNode` - элемент дерева объектов WinПОС

`IWPUnits` - интерфейс единиц измерения

Следующие главы содержат описания методов этих интерфейсов в нотации ODL. Такая нотация, а ODL расшифровывается как Object Description Language (язык описания объектов), предпочтительна при описании OLE-интерфейсов. Ниже приведена таблица соответствия типов в разных языках.

ODL	Delphi	VBScript	описание
`BSTR`	`WideString`	`BSTR`	символьная строка
`double`	`Double`	`Variant`	вещественное число
`long`	`Integer`	`Variant`	целое (32 бита)
`short`	`Smallint`	`Variant`	короткое целое (16 бит)
`IDispatch*`	`IDispatch`	`IDispatch`	указатель на интерфейс, производный от IDispatch
`BOOL`	`Boolean`	`Variant`	Булева(логическая) переменная
`void`	`[procedure]`*	`[sub]`*	*функция, возвращающая void, т.е. процедура
`VARIANT`	`OleVariant`	`Variant`	переменная

IWinPOS

Основной интерфейс приложения. С его помощью осуществляется управление средой WinПОС, доступ к дереву объектов, чтение и запись данных, взаимодействие с подключаемыми модулями, вызываются методы документирования и процедуры отладки.

Свойства

SelectedGraph

```
BSTR SelectedGraph
```

Имя выбранного графика в дереве графиков. Только чтение.

SelectedSignal

```
BSTR SelectedSignal
```

Имя выбранного сигнала в дереве сигналов.

Методы

Открытие и сохранение файлов данных

LoadUSML

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

Загрузить файл МЕРА (или УСМЛ) в дерево сигналов.

	Интерфейс IWPUSML
`path`	Имя файла

SaveUSML

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

Сохранить папку дерева в файл формата МЕРА или УСМЛ.

`Name`	Полный путь папки в дереве сигналов.
`FileName`	Имя файла.

LoadSignal

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

Загрузить двоичный или текстовый файл данных и поместить результаты в дерево сигналов.

	Интерфейс IWPSignal
path	Имя файла
type	Тип данных файла:

тип данных	знач.	описание
FT_TextWiz	2	Текстовый(ASCII) файл, открывается с помощью мастера настройки
FT_UChar	3	Массив беззнаковых целых (1 байт)
FT_INT16	4	Массив целых со знаком (2 байта)
FT_WORD	5	Массив беззнаковых целых (2 байта)
FT_INT32	6	Массив целых со знаком (4 байта)
FT_Float	7	Массивы вещественных чисел (4 байта)
FT_Double	8	Массивы вещественных чисел (8 байт)
FT_XLS	9	Столбцы таблицы Microsoft Excel

SaveSignal

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

Сохранить сигнал как двоичный или текстовый файл данных.

Name	Полный путь сигнала в дереве
FileName	Имя файла
type	Тип данных файла. Список значений приведен в описании метода LoadSignal.

Доступ к объектам WinПОС

CreateSignal, CreateSignalXY, CreateSignal2

```
IDispatch* CreateSignal(long type)
IDispatch* CreateSignalXY(long xtype, long ytype)
IDispatch* CreateSignal2(long stype, long xtype,
long ytype, long ztype)
```

Создать новый сигнал. CreateSignalXY() создает сигнал, у которого ось X может быть неравномерной, см. IWPSignal.SetX().
CreateSignal2() включает в себя функциональность CreateSignal и CreateSignalXY, а также позволяет создать трехмерный сигнал. Для создания сигнала с равномерной осью по X (и/или Z, для 3х-мерного сигнала), значение соответствующего параметра (xtype/ztype) должно быть равно 0.

	Интерфейс IWPSignal
`type`	Тип данных сигнала. `xtype`, `ytype`, `ztype` – типы значений раздельно по осям X, Y и Z:

тип данных	знач.	описание
VT□I1	16	Целые, 1 байт
VT_UI1	17	Целые беззнаковые, 1 байт
VT_I2	2	Целые, 2 байта
VT_UI2	18	Целые беззнаковые, 2 байта
VT_I4	3	Целые беззнаковые, 4 байта
VT_R4	4	Вещественные, 4 байта
VT_R8	5	Вещественные, 8 байт

`stype` Тип сигнала.

тип сигнала	знач.	описание
STYPE_NORMAL_2D	0	Обычный (двухмерный) сигнал
STYPE_NORMAL_3D	1	Трехмерный сигнал
STYPE_PARAM_3D	2	Трехмерный параметрический сигнал

GetInterval

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

Возвращается сигнал, являющийся интервалом исходного сигнала. Метод используется при обработке диапазона исходного сигнала.

	Интерфейс IWPSignal
`src`	Исходный сигнал
`start`	Начало отрезка, 0 .. (src.size-1)
`count`	Кол-во значений, 0 .. (src.size-start)

GetOversampled

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

Возвращаемый виртуальный сигнал позволяет трактовать данные исходного сигнала так, как будто они получены с другой частотой дискретизации. Новые значения интерполируются линейно, алгоритм передискретизации не вызывается, фильтрация не используется. Вследствие этого не рекомендуется указывать новую частоту меньше исходной. С помощью этого метода вы можете обрабатывать сигналы с переменным шагом алгоритмами, требующими равномерную шкалу.

	Интерфейс IWPSignal
`src`	Исходный сигнал
`freq`	Новая частота

GraphAPI

```
IDispatch* GraphAPI()
```

Получить интерфейс графической подсистемы.

	Интерфейс IWPGraph

Link

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

Поместить объект в дерево. Не вызывает перерисовку. После завершения процедур изменения дерева рекомендуется вызывать IWinPOS.Refresh.

	Узел дерева, поддерживающий интерфейс IWPNode
`Path`	Путь в дереве
`Name`	Имя объекта
`Object`	Объект, который требуется поместить в дерево

Unlink, UnlinkStr

```
void Unlink(IDispatch* object)
void UnlinkStr(BSTR path)
```

Удалить объект из дерева. Не вызывает перерисовку дерева. После завершения процедур изменения дерева рекомендуется вызывать IWinPOS.Refresh.

`object`	Объект, который требуется удалить из дерева, или узел дерева, ссылающийся на этот объект
`path`	Путь удаляемого узла. Строки "/Signals" или "/Graphs" очищают поддеревья сигналов или графиков.

IsNodeExist

```
BOOL IsNodeExist(BSTR path)
```

Проверить наличие узла дерева.

	Существует ли узел, соответствующий данному пути
`path`	Строка, путь в дереве

Get

```
IDispatch* Get(BSTR path)
```

Найти узел дерева по пути.

	Узел дерева, поддерживающий интерфейс IWPNode
`path`	Строка, путь в дереве

GetObject

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

Найти объект WinПОС по пути в дереве.

	Указатель на интерфейс запрашиваемого объекта
`Path`	Строка, путь в дереве

GetNode

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

Получить позицию размещения (т. н. точку монтирования, узел) объекта в дереве.

	Интерфейс IWPNode
`Object`	Объект.

GetSelectedNode, GetSelectedObject

```
IDispatch* GetSelectedNode()
IDispatch* GetSelectedObject()
```

Получить выбранный узел или объект.

	Интерфейс узла (для GetSelectedNode) или сигнала, файла и т.д.

GetObjectType

```
long GetObjectType(IDispatch* Object)
```

Получить тип объекта.

Тип объекта

тип объекта	знач.	описание
OBJTYPE_EMPTY	0	Пустой объект
OBJTYPE_UNKNOWN	1	Объект неизвестного типа
OBJTYPE_NODE	2	Узел дерева (IWPNode)
OBJTYPE_SIGNAL	3	Сигнал (IWPSignal)
OBJTYPE_OPER	4	Оператор (IWPOperator)

`Object` Объект.

GetUnits

```
IDispatch* GetUnits()
```

Получить интерфейс единиц измерения

Интерфейс IWPUnits

Управление средой WinПОС

USMLDialog

```
BSTR USMLDialog()
```

Выбрать файл УСМЛ или МЕРА с помощью стандартного диалога.

Полное имя файла

Refresh

```
void Refresh()
```

Обновить окно дерева после вызова методов Link() или Unlink().

DoEvents

```
void DoEvents()
```

Обработать все события, накопившиеся за время работы продолжительной задачи. Процедура используется в процессе длительных вычислений для того, чтобы избежать эффекта "зависания" программы. DoEvents() приостанавливает выполнение сценария и позволяет среде обработать оконные сообщения.

execVBS

```
void execVBS(BSTR code)
```

Выполнить программу на VBScript. Используется в подключаемых модулях, например, если пользовательские настройки или алгоритмы обработки сохранены в виде сценария. ВНИМАНИЕ! Вызов из сценария VBScript приведет к падению.

`code`	Текст программы

AddTextInLog

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

Добавить строку текста в журнал.

`text`	Текстовая строка для журнала
`exttext`	Строка дополнительных параметров
`show`	true, если надо отобразить окно журнала, false, если нет

Взаимодействие с подключаемыми модулями

Методы этого раздела предоставляют доступ к управляющим элементам среды WinПОС (главному окну, панелям инструментов, меню) и предназначены для использования при создании подключаемых модулей.

MainWnd

```
long MainWnd()
```

Метод возвращает дескриптор (handle) главного окна, который может потребоваться подключаемым модулям, если в них определены собственные окна и диалоги.

⇗	Дескриптор (handle) главного окна WinПОС

RegisterCommand

```
long RegisterCommand()
```

Метод возвращает уникальное число – идентификатор команды.

⇗	Число, уникальный код команды, события

CreateToolbarN

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

Создать новую именованную панель инструментов.

	Указатель на панель инструментов
`name`	Наименование панели инструментов. Добавляется в меню «Вид»

CreatetoolbarButton

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

Добавить кнопку на панель инструментов.

	Ненулевое в случае успешного добавления кнопки, 0 – в противном случае
`bar`	Указатель на панель инструментов, полученный при вызове CreateToolbarN()
`command`	Присвоенная команда. Можно получить при помощи RegisterCommand()
`picture`	Дескриптор изображения для кнопки
`hint`	Текст подсказки

ShowToolbar

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

Отобразить или скрыть панель инструментов.

`bar`	Указатель на панель инструментов, полученный при вызове CreateToolbarN()
`visible`	1 – отображать, 0 – скрыть.

ToolbarSetButtonStyle

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

Изменить состояние кнопки панели инструментов.

`bar`	Указатель на панель инструментов, полученный при вызове CreateToolbarN()
`command`	Присвоенная команда. См. RegisterCommand ()
`nStyle`	Стиль кнопки. Используется: 0 – обычная кнопка, 4 – кнопка запрещена.

CreatemenuItem

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

Создать новый пункт меню.

	Ненулевое в случае успешного добавления строки меню, 0 – в противном случае
Command	Присвоенная команда. Можно получить при помощи RegisterCommand ()
reserved	Каждый байт этого числа (если не равен FF) представляет собой позицию в подменю данного уровня, куда будет добавлен новый пункт меню. Пример: 0xFFFF0301 - четвертое подменю главного меню, после второй позиции (начало отсчета - 0)
text	Строка меню
style	Вид элемента меню. Для обычного элемента меню устанавливается в 0. Значения:

вид элемента меню	знач.	вид элемента меню	знач.
MF_ENABLED	0h	MF_BITMAP	4h
MF_GRAYED	1h	MF_OWNERDRAW	100h
MF_DISABLED	2h	MF_POPUP	10h
MF_UNCHECKED	0h	MF_MENUBARBREAK	20h
MF_CHECKED	8h	MF_MENUBREAK	40h
MF_USECHECKBITMAPS	200h	MF_UNHILITE	0h
MF_STRING	0h	MF_HILITE	80h

picture	Дескриптор изображения для пункта меню. Используется только при установленном стиле MF_BITMAP.

RegisterImpExp

```
BOOL RegisterImpExp(IDispatch* imp, IDispatch* exp, LPCTSTR desc, LPCTSTR ext)
```

Зарегистрировать плагин импорта-экспорта. В окне открытия файла WinПОС добавится новый тип файла (параметр desc). См. описание интерфейсов импорта и экспорта.

	Признак успешности регистрации
Imp	Указатель на интерфейс импорта
Exp	Указатель на интерфейс экспорта
desc	Описание типа файлов
Ext	Расширение файлов в формате «*.расширение». Пример (Delphi): *RegisterImpExp(self, self, 'Файлы WAV', '*.wav');*

RegisterExtOper

```
BOOL RegisterExtOper(IDispatch* oper, long nsrc, long ndst, LPCTSTR name, LPCTSTR shortname, BOOL bauto)
```

Зарегистрировать новый оператор. Оператор будет добавлен в меню «Расширения». См. описание интерфейсов внешнего оператора IWPExtOper и, для сравнения, встроенного оператора IWPOperator.

	Признак успешности регистрации
`oper`	Указатель на интерфейс внешнего оператора
`nsrc`	Количество входных сигналов
`ndst`	Количество сигналов на выходе, результатов
`name`	Полное название оператора (отображается в меню)
`shortname`	Сокращенное название оператора
`bauto`	true – выбор исходных сигналов реализован в операторе, false – требуется стандартное окно настройки

Отображение состояния

ProgressStart

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

Создать индикатор прогресса.

`comment`	Строка текущего состояния. Например, «Поиск максимума»
`max`	Количество шагов индикатора

ProgressStep

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

Установить индикатор прогресса.

`pos`	Новая позиция индикатора: 0 .. max. Если pos=0, будет сделан один шаг индикатора.

ProgressFinish

```
void ProgressFinish()
```

Скрыть индикатор прогресса.

Документирование, печать результатов

SaveImage

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

Сохранить отображаемую страницу графиков в файл или буфер обмена.

[↵]	true – если операция прошла успешно, false – в противном случае
fname	Имя файла. Если передана пустая строка – изображение будет помещено в буфер обмена
comment	Строка подписи. Например, «Рис.1 Исходные уровни». Если задана пустая строка – подпись не печатается.

PrintPreview, Print

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

Распечатать отображаемую страницу графиков. PrintPreview выдает окно предварительного просмотра. Print – печатает на принтер, используя текущие настройки принтера и страницы.

comment	Строка подписи. Например, «Рис.1 Исходные уровни». Если задана пустая строка – подпись не печатается.

VBScript. Работа с двоичными файлами данных

Эти методы расширяют ограниченные возможности VBScript в части работы с двоичными файлами. Нецелесообразно обращаться к этим методам из Delphi, т.к. в Delphi можно вызывать функции для работы с файлами напрямую.

FileOpen

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

Открыть стандартный файловый диалог и выбрать имя файла.

[↵]	Полное имя файла
isOpen	true - диалог открытия файла, false - сохранения
ext	Расширение файла по умолчанию

`fname`	Начальное имя файла
`flags`	Флаги для настройки внешнего вида и поведения диалога. Полный перечень можно найти в документации Microsoft (структура OPENFILENAME). Некоторые наиболее полезные флаги:

флаг	знач.	описание
`OFN_ALLOWMULTISELECT`	`200h`	Разрешает выбирать несколько файлов сразу.
`OFN_CREATEPROMPT`	`2000h`	Если пользователь указал файл, которого не существует, то диалог запросит разрешение создать новый файл с указанным именем.
`OFN_FILEMUSTEXI□T`	`1000h`	В поле *Имя файла* разрешается вводить только имена существующих файлов. Если указан этот флаг и введено некорректное имя файла - выдается предупреждение. Используется совместно с OFN_PATHMUSTEXIST.
`OFN_NOCHANGEDIR`	`8h`	Возвращает текущей папке первоначальное значение, если пользователь переходит из одной папки в другую при поиске файлов.
`OFN_NONETWORKBUTTON`	`20000h`	Убирает кнопку **Сеть** диалога
`OFN_NOREADONLYRETURN`	`8000h`	Поле *Только для чтения* не выбрано и возвращаемый файл не находится защищенной от копирования папке.
`OFN_OVERWRITEPROMPT`	`2h`	Диалог *Сохранить как* выдаст предупреждающее сообщение, если файл уже существует. Пользователь должен подтвердить перезапись файла.
`OFN_PATHMUSTEXIST`	`800h`	Пользователь может указать только существующий путь и имена файлов. При вводе неправильного имени файла или пути появится предупреждающее окно.

`filter`	Набор фильтров для диалога. Например, строка *"Binary files|*.bin|All files|*.*||"* предложит выбрать файл с расширением .bin или любой файл.

OpenFile

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

Открыть или создать новый файл.

	Файловый дескриптор, используется в дальнейших вызовах (см. параметр hFile)
`Path`	Путь файла
`flags`	Тип доступа. Значения:

флаг	знач.	описание
`READ_WRITE`	`100h`	Открыть по чтению и записи (GENERIC_READ | GENERIC_WRITE), 0 – только по чтению (GENERIC_READ).

SHARE_READ 1000h	Этот файл одновременно можно будет открыть только по чтению (FILE_SHARE_READ), 0 – по чтению и записи (FILE_SHARE_READ | FILE_SHARE_WRITE).

CloseFile

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

Закрыть файл.

hFile	Файловый дескриптор, полученный при вызове OpenFile()

SeekFile

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

Изменить позицию файлового указателя.

	Новая позиция файлового указателя
hFile	Файловый дескриптор, полученный при вызове OpenFile()
Pos	Позиция, в которую требуется переместить файловый указатель
flags	Точка отсчета при перемещении. Значения:

флаг	знач.	Описание
FILE_BEGIN	0	Начало файла
FILE_CURRENT	1	Текущая позиция в файле
FILE_END	2	Конец файла

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)
```

Прочитать из двоичного файла число в соотв. формате.

	Число, прочитанное из файла
hFile	Файловый дескриптор, полученный при вызове OpenFile()

WriteByte, WriteWord, WriteLong, WriteFloat, WriteDouble

```
BOOL WriteByte(long hFile, short Value)
BOOL WriteWord(long hFile, long value)
```

```
BOOL WriteLong(long hFile, long Value)
BOOL WriteFloat(long hFile, float value)
BOOL WriteDouble(long hFile, double Value)
```

Записать в двоичный файл число в соотв. формате.

	true – операция выполнена успешно, false – ошибка
hFile	Файловый дескриптор
Value	Число, которое должно быть записано в файл

VBScript. Отладка

DebugPrint, DebugPrintLn

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

Отладочная печать в окно вывода *Редактора сценариев*. Применяется только при запуске из *Редактора сценариев*. DebugPrintLn() в отличие от DebugPrint() переводит строку.

arg	Символьная строка. Пример: *DebugPrintLn "Max= "+FormatNumber(max,6,0,0,0)+";"*

IWPGraphs

Интерфейс графической подсистемы.

Для управления графиками сначала требуется получить данный интерфейс, вызвав метод GraphAPI интерфейса IWinPOS.

Последовательность действий при создании новой страницы для отображения в ней сигнала (например, результата работы скрипта или плагина):

```
// получаем доступ к графической подсистеме WinПОС
api := GraphAPI as IWPGraphs;

// создаем новую страницу для графиков
hPage := api.CreatePage;

// страница всегда создается с одним графиком, получаем его
hGraph := api.GetGraph(hPage, 0);

// график всегда имеет как минимум одну ось Y, получаем ее
hYAxis := api.GetYAxis(hGraph, 0);

// создаем  новую линию в графике
```

```
api.CreateLine(hGraph, hYAxis, signal.Instance);

// нормализуем график
api.NormalizeGraph(hGraph);
```

Методы

CreatePage, CreatePage2

```
long CreatePage()
long CreatePage2(long type)
```

Создать новую страницу. CreatePage2 позволяет создать страницу заданного типа. Будут учтены настройки по умолчанию.

	Указатель на новую страницу
`type`	Тип страницы

знач.	Описание
0	Обычная страница с графиками
1	Страница с трехмерным графиком
2	Страница с диаграммой Кембелла

DestroyPage

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

Удалить страницу.

`hPage`	Указатель на страницу

CreateGraph

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

Создать новый график. Будут учтены настройки по умолчанию.

	Указатель на новый график
`hPage`	Указатель на страницу, в которой требуется создать график

DestroyGraph

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

Удалить график.

`hGraph`	Указатель на график

CreateYAxis

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

Добавить новую ось ординат.

	Указатель на новую ось
hGraph	Указатель на график, куда будет добавлена ось

DestroyYAxis

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

Удалить ось. Единственную ось Y в графике удалить нельзя.

hAxis	Указатель на ось

CreateLine

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

Создать новую линию. Будут учтены настройки по умолчанию.

	Указатель на новую линию
hGraph	Указатель на график, в который будет добавлена линия
hAx	Указатель на Y-ось
hSig	Указатель на сигнал

DestroyLine

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

Удалить линию.

hLine	Указатель на линию

GetPageCount

```
long GetPageCount()
```

Число страниц графиков.

	Число страниц графиков

GetGraphCount

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

Число графиков на странице.

↵	Число графиков на странице
`hPage`	Указатель на страницу

GetYAxisCount

```
long GetYAxisCount(long hGraph)
```

Число осей Y графика.

↵	Число осей Y
`hGraph`	Указатель на график

GetLineCount

```
long GetLineCount(long hGraph)
```

Число линий в графике.

↵	Число линий в графике
`hGraph`	Указатель на график

GetPage

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

Получить страницу по порядковому номеру.

↵	Указатель на страницу
`nPage`	Порядковый номер страницы

GetGraph

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

Получить график по порядковому номеру.

↵	Указатель на график
`hPage`	Указатель на страницу
`nGraph`	Порядковый номер графика

GetYAxis

```
long GetYAxis(long hGraph, long nAxis)
```

Получить Y-ось по порядковому номеру.

	Указатель на ось
hGraph	Указатель на график
nAxis	Порядковый номер оси

GetLine

```
long GetLine(long hGraph, long nLine)
```

Получить линию по порядковому номеру.

	Указатель на линию
hGraph	Указатель на график
nLine	Порядковый номер линии. Если nLine=-1, будет получена текущая (активная) линия.

GetSignal

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

Получить ссылку на сигнал, который изображает линия hLine.

	Интерфейс IWPSignal
hLine	Указатель на линию

GetXCursorPos, SetXCursorPos

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

Получить или установить позицию курсора. Версия GetXCursorPos_V может использоваться в сценариях.

hGraph	Указатель на график
x	Позиция курсора
px	Адрес переменной для возврата позиции курсора
bSecond	Работать с позицией второй линии курсора (для двойного курсора)

ShowCursor, GetCursorType

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

Изменить / получить режим отображения курсора.

hPage	Указатель на страницу
mode/	Режим отображения курсора:

режим	знач.	описание
CT_NONE	0	Курсор отключен
CT_SINGLE_X	1	Обычный курсор
CT_DOUBLE_X	2	Двойной курсор
CT_SINGLE_Y	3	Обычный курсор по оси Y
CT_DOUBLE_Y	4	Двойной курсор по оси Y
CT_3D	5	Курсор трехмерного графика
CT_3DCD	6	Курсор порядков диаграммы Кэмпбелла
CT_VIBRO	7	Виброкурсор

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)
```

ⓘ В сценариях следует использовать функции **GetPageOpt2 / SetPageOpt2** с параметром “PGOPT2_RECT”

Получить, установить размеры страницы графиков.

hPage	Указатель на страницу
left	Координаты левого края страницы
top	Координаты верха страницы
right	Координаты правого края страницы
bottom	Координаты низа страницы

SetPageDim

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

Установить режим расположения графиков.

hPage	Указатель на страницу
mode	Режим расположения графиков:

режим	знач.	описание
PAGE_DM_VERT	0	Графики располагаются вертикально
PAGE_DM_HORZ	1	Графики располагаются горизонтально
PAGE_DM_TABLE	2	Графики располагаются в виде таблицы width*height

width	Число графиков на странице по горизонтали
height	Число графиков на странице по вертикали

GetXMinMax, SetXMinMax

```
void GetXMinMax(long hGraph, double* pmin, double* pmax)
void GetXMinMax_V(long hGraph, VARIANT* pmin, VARIANT* pmax)
void SetXMinMax(long hGraph, double min, double max)
```

Получить или установить границы по оси абсцисс. Таким образом, можно установить или получить видимый диапазон сигнала. Версия GetXMinMax_V может использоваться в сценариях.

hGraph	Указатель на график
min, pmin	Минимальное значение или указатель на него
max, pmax	Максимальное значение или указатель на него

GetYAxisMinMax, SetYAxisMinMax

```
void GetYAxisMinMax(long hAxis, double* pmin, double* pmax)
void GetYMinMax_V(long hAxis, VARIANT* pmin, VARIANT* pmax)
void SetYAxisMinMax(long hAxis, double min, double max)
```

Получить или установить границы выбранной оси ординат. Версия GetYMinMax_V может использоваться в сценариях.

hAxis	Указатель на Y-ось
min, pmin	Минимальное значение или указатель на него
max, pmax	Максимальное значение или указатель на него

GetZMinMax, SetZMinMax

```
void GetZMinMax(long hGraph, double* pmin, double* pmax)
void GetZMinMax_V(long hGraph, VARIANT* pmin, VARIANT* pmax)
void SetZMinMax(long hGraph, double min, double max)
```

Получить или установить границы по оси Z трехмерного графика. Таким образом, можно установить или получить видимый диапазон сигнала. Версия GetZMinMax_V может использоваться в сценариях.

hGraph	Указатель на график
min, pmin	Минимальное значение или указатель на него
max, pmax	Максимальное значение или указатель на него

GetCurYRange

```
void GetCurYRange(long hLine, double* pymin, double* pymax, double* pxmin, double* pxmax)
void GetCurYRange_V(long hLine, VARIANT* pymin, VARIANT* pymax, VARIANT* pxmin, VARIANT* pxmax)
```

Получить минимум и максимум по оси Y на отображаемом отрезке и значения оси X, соответствующие экстремумам. Версия GetCurYRange_V может использоваться в сценариях.

hLine	Указатель на линию
pymin, pymax	Указатель на текущие минимальное и максимальное значения по оси Y
pxmin, pxmax	Указатель на значения оси X, отвечающие текущим экстремумам

GetLineInfo

```
BSTR GetLineInfo(long hGraph, long hLine, long ntype, BOOL* pbShow)
BSTR GetLineInfo_V(long hGraph, long hLine, long ntype, VARIANT* pbShow)
```

Получить информацию из легенды. Версия GetLineInfo_V может использоваться в сценариях.

	Строка ячейки легенды
hGraph	Указатель на график
hLine	Указатель на линию
ntype	Номер столбца легенды
pbShow	Возвращаемое значение: отображается или нет столбец

NormalizeGraph

```
void NormalizeGraph(long hGraph)
```

Выполнить нормализацию графика, т.е. изменить диапазоны по осям X и Y так, чтобы показать сигналы полностью.

`hGraph`	Указатель на график

Invalidate

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

Обновить поле отрисовки графика.

`hGraph`	Указатель на график

SetClearBufferFlag

```
void SetClearBufferFlag(long hGraph, long hLine)
```

Принудительно вызвать пересчет и перерисовку выбранной линии.

`hGraph`	Указатель на график
`hLine`	Указатель на линию

ActiveGraphPage

```
long ActiveGraphPage()
```

Получить указатель на активную страницу графиков.

	Указатель на активную страницу графиков

ActiveGraph

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

Получить указатель на активный график.

	Указатель на активный график
`hPage`	Указатель на страницу графиков

Folder2Graphs, Folder2GraphsRecursive

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

Разместить все сигналы указанной папки или пакетного файла на новой странице. Второй вариант метода обходит вложенные папки.

`Node`	Интерфейс узла дерева (IWPNode)

Locate

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

Найти элемент графика в дереве по указателю.

	Интерфейс узла дерева (IWPNode)
hGrItem	Указатель на страницу графиков, график или линию

SetPageOpt

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

ⓘ Устаревшая функция! Следует использовать функцию **SetPageOpt2**.

Установить параметры выбранной страницы.

hPage	Указатель на страницу
opt	Битовое поле: установить (1) или снять (0) бит состояния:

битовая маска	знач.	описание
PGOPT_SHOWNAME	1	Отображение названия страницы
PGOPT_SINGLEX	2	Флаг одной оси X на страницу
PGOPT_SINGLEY	4	Флаг одной оси Y на страницу
PGOPT_SINCCURS	8	Синхронизация курсоров

mask	Маска. Показывает, какие биты поля opt следует изменить. См. таблицу выше.

SetGraphOpt

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

ⓘ Устаревшая функция! Следует использовать функцию **SetGraphOpt2**.

Установить параметры выбранного графика.

hGraph	Указатель на график
opt	Битовое поле: установить (1) или снять (0) бит состояния:

битовая маска	знач.	описание
GROPT_SHOWNAME	1h	Флаг отрисовки названия
GROPT_YINDENT	2h	10% отступ для линий сверху и снизу
GROPT_SUBGRID	4h	Флаг отрисовки пунктирных линий на сетке
GROPT_GRIDLABS	□h	Значения линий в узлах сетки
GROPT_LINEN□MS	10h	Показывать номера линий
GROPT_AUTONORM	20h	Автоматически нормализововать график при добавлении новых линий
GROPT_POLAR	40h	Полярные координаты
GROPT_AXCOLUMN	80h	Флаг размещения осей Y друг над другом

GROPT_AXROW	100h	Флаг размещения осей Y друг за другом
GROPT_SHOWLEGEND	200h	Показывать легенду

`mask` — Маска. Показывает, какие биты поля opt следует изменить. См. таблицу выше.

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)
```

ⓘ Устаревшие функции! Следует использовать функции **GetAxisOpt2 / SetAxisOpt2**.

Получить / установить параметры выбранной оси.

`hGraph` — Указатель на график (нужен для оси абсцисс)

`hAxis` — Указатель на ось (для оси абсцисс: 0)

`opt` — Битовое поле: установить (1) или снять (0) бит состояния:

битовая маска	знач.	описание
AXOPT_LOG	1h	Логарифмический масштаб
AXOPT_FZERO	2h	Добавлять нули в конце числа ("1.500" вместо "1.5")
AXOPT_TIME	4h	Добавить шкалу времени в формате «чч.мм.сс.мск»
AXOPT_COLOR*	8h	Установить вручную цвет оцифровки оси
AXOPT_RANGE*	10h	Установить полный диапазон оси
AXOPT_NAME*	20h	Установить имя или размерность оси
AXOPT_FORMAT*	40h	Установить формат оцифровки

* - используются только в поле mask

`mask` — Маска. Показывает, какие биты поля opt следует изменить. Также показывает, надо ли изменить имя или формат оцифровки оси. См. таблицу выше.

`minR,maxR` — Отображаемый диапазон оси (с флагом AXOPT_RANGE – полный диапазон, т.е. пределы масштабирования)

`szname` — Имя оси (обычно берется размерность)

`szftempl` — Формат оцифровки (описан в *Руководстве Пользователя*, ч.5, *Создание графиков*, *Настройка графиков*)

`color` — Цвет в формате RGB (белый = FFFFFFh, черный = 0h)

GetLineOpt, SetLineOpt

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

ⓘ Устаревшие функции! Следует использовать функции **GetLineOpt2 / SetLineOpt2**.

Получить / установить параметры выбранной линии.

`hLine`	Указатель на линию
`opt`	Битовое поле: установить (1) или снять (0) бит состояния:

битовая маска	знач.	описание
LNOPT_LINE2BASE	1h	Добавлять вертикальные линии от значения до 0
LNOPT_ONLYPOINTS	2h	Флаг соединения точек линиями
LNOPT_VISIBLE	4h	Флаг отображения/скрыт я линии
LNOPT_HIST	8h	В виде гистограммы
LNOPT_HISTTRANSP	40h	"Прозрачная" гистограмма
LNOPT_PARAM	80h	В виде Y(idx), с реальными значениями на шкале оси X
LNOPT_INTERP	300h	Порядок интерполяции (два бита!)
LNOPT_COLOR*	10h	Изменить цвет линии. См. поле color
LNOPT_WIDTH*	20h	Изменить толщину линии. См. поле width

* - используются только в поле mask

`mask`	Маска. Показывает, какие биты поля opt следует изменить. Также показывает, надо ли изменить толщину или цвет линии. См. таблицу выше.
`width`	Толщина линии
`color`	Цвет в формате RGB (белый = FFFFFFh, черный = 0h)

GetPageOpt2, SetPageOpt2

```
void GetPageOpt2(long hPage, long opt, VARIANT* pval)
void SetPageOpt2(long hPage, long opt, VARIANT val)
```

Установить параметры выбранной страницы.

`hPage`	Указатель на страницу
`opt`	Выбранная опция:

константа	знач.	тип	описание
PGOPT2_NAME	1	BSTR	Название страницы
PGOPT2_SHOWNAME	2	bool	Отображение названия страницы
PGOPT2_RECT	3	long[4]	Положение и размер страницы графиков
PGOPT2_MAXIMIZED	4	bool	Флаг максимизации страницы
PGOPT2_SINGLEX	5	bool	Флаг одной оси X на страницу

PGOPT2_SINGLEY	6	bool	Флаг одной оси Y на страницу
PGOPT2_SINCCURS	7	bool	Синхронизация курсоров

val,pval Значение

GetGraphOpt2, SetGraphOpt2

```
void GetGraphOpt2(long hGraph, long opt, VARIANT* pval)
void SetGraphOpt2(long hGraph, long opt, VARIANT val)
```

Установить параметры выбранного графика.

hGraph Указатель на график

opt Выбранная опция:

константа	знач.	тип	описан е
GROPT2_NAME	1	BSTR	Название графика
GROPT2_SHOWNAME	2	bool	Отображение названия графика
GROPT2_BORDER	3	long[4]	Размер границ графика
GROPT2_SHOWLEGEND	4	bool	Флаг отображения легенды
GROPT2_YINDENT	5	bool	10% отступ для линии сверху и снизу
GROPT2_SUBGRID	6	bool	Отображение пунктирных линий на сетке
GROPT2_GRIDLABS	7	bool	Значения линий в узлах сетки
GROPT2_LINENUMS	8	bool	Показывать номера линий
GROPT2_AUTONORM	9	bool	Автоматически нормализовать график при добавлении новых линий
GROPT2_AXCOLUMN	10	bool	Флаг размещения осей Y друг над другом
GROPT2_AXROW	11	bool	Флаг размещения осей Y друг за другом
GROPT2_POLAR	12	bool	Полярные координаты

val,pval Значение

GetLineOpt2, SetLineOpt2

```
void GetLineOpt2(long hLine, long opt, VARIANT* pval)
void SetLineOpt2(long hLine, long opt, VARIANT val)
```

Установить параметры выбранной линии.

hLine Указатель на линию

opt Выбранная опция:

константа	знач.	тип	описание
LNOPT2_NAME	1	BSTR	Название линии
LNOPT2_VISIBLE	2	bool	Флаг отображение линии
LNOPT2_COLOR	3	long	Цвет линии (в формате RGB)
LNOPT2_WIDTH	4	long	Толщина линии
LNOPT2_LINETYPE	5	long	Тип линии (сплошная, пунктирная и т.д.)
LNOPT2_POINTTYPE	6	long	Тип точек на линии

LNOPT2_LINE2BASE	7	bool	Добавлять вертик. линии от значения до 0
LNOPT2_ONLYPOINTS	8	bool	Флаг соединения точек линиями
LNOPT2_HIST	9	bool	Отображать в виде гистограммы
LNOPT2_HISTTRANSP	10	bool	"Прозрачная" гистограмма
LNOPT2_PARAM	11	bool	Отображение в виде Y(idx), с реальными значениями на шкале оси X
LNOPT2_INTERP	12	long	Порядок интерполяции
LNOPT2_SYSTEM	13	bool	"Системная" линия (не отобр. в легенде)

val,pval Значение

GetAxisOpt2, SetAxisOpt2

```
void GetAxisOpt2(long hGraph, long hAxis, long opt, VARIANT* pval)
void SetAxisOpt2(long hGraph, long hAxis, long opt, VARIANT val)
```

Установить параметры выбранной оси.

hGraph Указатель на график

hAxis Указатель на ось (для оси абсцисс: 0)

opt Выбранная опция:

константа	знач.	тип	описание
AXOPT2_NAME	1	BSTR	Название оси
AXOPT2_UNITSNAME	2	BSTR	Название единиц измерения
AXOPT2_FORMAT	3	BSTR	Формат оцифровки
AXOPT2_FZERO	4	bool	Добавлять нули в конце числа ("1.500")
AXOPT2_LOG	5	bool	Логарифмический масштаб
AXOPT2_TIME	6	bool	Добавить шкалу времени в формате «чч:мм:сс:мск»
AXOPT2_DATE	7	bool	Добавить шкалу времени в формате «ДД.ММ.ГГГГ»

val,pval Значение

GetLegendOpt2, SetLegendOpt2

```
void GetLegendOpt2(long hGraph, long opt, VARIANT* pval)
void SetLegendOpt2(long hGraph, long opt, VARIANT val)
```

Установить параметры легенды на выбранном графике.

hGraph Указатель график

opt Выбранная опция:

константа	знач.	тип	описание
LGNOPT2_VISIBLE	1	bool	Флаг отображения легенды
LGNOPT2_MODE	2	long	Режим отображения легенды

LGNOPT2_ALIGN	3	long	Расположение легенды
LGNOPT2_SIZE	4	long	Размер поля легенды, для текущего положения

val,pval	Значение

AddLabel, AddLabel3D

```
void AddLabel(long hLine, long mode, double x, double offsX, double offsY, BSTR text)
void AddLabel3D(long hLine, long mode, double x, double z, double offsX, double offsZ, BSTR text)
```

Добавить выноску.

hLine	Указатель на линию
mode	Тип выноски:

тип	знач.	описание
LAB_SINGLE	0	На одну линию
LAB_MULTI	1	На все линии
LAB_TEXT	2	Текстовая выноска

x, z	Значение на оси x/z, к которому привязана выноска
offsX, offsY, offsZ	Положение выноски в поле графика, выраженное в процентах относительно размеров графика
text	Текст выноски, если mode = LAB_TEXT

GetLabelCount

```
long GetLabelCount(long hLine)
```

Получить количество выносок на указанной линии.

	Количество выносок
hLine	Указатель на линию

GetLabelPos

```
void GetLabelPos(long hLine, long iLab, double* x, double* z)
void GetLabelPos_V(long hLine, long iLab, VARIANT* x, VARIANT* z)
```

Получить значение на оси x/z, к которому привязана выноска. Версия GetLabelPos_V может использоваться в сценариях.

hLine	Указатель на линию
iLab	Номер выноски
x, z	Значение на оси x/z

AddComment

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

Добавить комментарий.

`hGraph`	Указатель на график
`text`	Текст комментария
`x, y`	Положение левого верхнего угла комментария, выраженное в процентах относительно размеров графика
`dx, dy`	Размеры комментария, выраженные в процентах относительно размеров графика

SaveSession, LoadSession

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

Сохранить текущий сеанс работы и загрузить ранее сохраненный сеанс.

	Результат операции (TRUE – успешно)
`path`	Путь на диске к файлам сессии.

IWPSignal

Интерфейс сигнала.

Свойства

size

```
long size
```

Количество значений (измерений) сигнала.

DeltaX

```
double DeltaX
```

Шаг по оси X для сигнала с равномерной осью абсцисс. Для сигнала с неравномерной осью DeltaX=0.

StartX

```
double StartX
```

Начальное значение по оси X для сигнала с равномерной осью абсцисс. Для сигнала с неравномерной осью StartX содержит значение абсциссы первого элемента сигнала.

SName

```
BSTR SName
```

Имя сигнала.

NameY

```
BSTR NameY
```

Единицы измерения значений сигнала, строка.

NameX

```
BSTR NameX
```

Единицы измерения оси абсцисс, строка.

Comment

```
BSTR Comment
```

Комментарий, дополнительная, расширенная текстовая информация по данному сигналу.

Characteristic

```
long Characteristic
```

Характеристика сигнала, влияет на вид графика. Возможные значения:

характеристика	знач.	описание
SC_NORMAL	0	Обычный сигнал
SC_SPECTR	1	Спектр
SC_LOGSPEC	2	Логарифмический спектр
SC_LOGX	4	Сигнал с логарифмической осью абсцисс
SC_AMP	8	Амплитуда
SC_FASE	16	Фаза
SC_PARAM	32	Параметрический сигнал

MinY, MaxY

```
double MinY
double MaxY
```

Минимальное и максимальное значения сигнала. Если MaxY < MinY, то это означает, что минимальное и максимальное значения сигнала еще не рассчитаны.

MinX, MaxX

```
double MinX
double MaxX
```

Минимальное и максимальное значения оси абсцисс сигнала. Если сигнал содержит изменения параметра во времени, то это начало и окончание регистрации параметра. Доступны только по чтению, изменить MinX и MaxX можно, изменив StartX и DeltaX или, при неравномерном шаге, с помощью метода SetX().

k0, k1

```
double k0
double k1
```

Коэффициенты калибровочной характеристики, заданной как линейная функция: $y = k_1 \cdot (x – k_0)$.

Методы

Instance

```
long Instance()
```

Возвращает указатель на объект, предоставляющий этот интерфейс. Используется для передачи объекта в качестве параметра.

	Указатель на объект

IndexOf

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

Возвращается индекс (порядковый номер) значения, соответствующего заданному времени, а если точного значения для этого времени нет – индекс ближайшего значения.

	Номер элемента сигнала из диапазона 0..(size-1)
x	Заданное значение времени (оси абсцисс)

GetY, GetX

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

Возвращается значение элемента сигнала по оси ординат или абсцисс.

↵	Значение сигнала по оси ординат или абсцисс
index	Номер элемента сигнала из диапазона 0..(size-1)

GetYX

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

Возвращается значение сигнала, соответствующего заданному времени, а если точного значения для этого времени нет – интерполированное значение. Pow определяет метод интерполяции.

↵	Значение сигнала
x	Заданное значение времени (оси абсцисс)
pow	Вид интерполяции: 0 – отсутствует (берется последнее по времени значение), 1 – линейная интерполяция, 2 – квадратный полином, 3 – интерполяция кубическими локальными сплайнами.

SetY, SetX

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

Устанавливается значение элемента сигнала по оси ординат или абсцисс. SetX не имеет смысла для сигналов с равномерной осью X, для таких сигналов следует задавать свойства StartX и DeltaX.

index	Номер элемента сигнала из диапазона 0..(size-1)
value	Новое значение

GetOriginalY, SetOriginalY

```
double GetOriginalY(long index)
void SetOriginalY(long index, double value)
```

Аналогично методам GetY и SetY дают доступ к значению сигнала, однако оперируют значениями в исходных кодах, не применяя ГХ.

index	Номер элемента сигнала из диапазона 0..(size-1)
↵, value	Значение

GetType

```
long GetType()
```

Узнать тип данных сигнала.

	Значение типа данных сигнала. См. таблицу в описании IWinPOS.CreateSignal

GetSProperty, SetSProperty

```
BSTR GetSProperty(BSTR name)
void SetSProperty(BSTR name, BSTR value)
```

Доступ к текстовым свойствам сигнала.

,value	Значение свойства, строка.
name	Название свойства.

GetStartTime, SetStartTime

```
double GetStartTime()
void SetStartTime(double time)
```

Доступ к астрономическому времени начала регистрации сигнала.

,time	Восьмибайтовое вещественное представление времени (VariantTime). Целая часть – сутки с 30.12.1899, дробная – часть суток. Например, 2.5 соответствует полудню 1 января 1900 года.

GetParent

```
IDispatch* GetParent()
```

Получить интерфейс файла регистрации.

	Интерфейс IWPUSML

GetArray, SetArray

```
      BOOL GetArray(long index, long* pnCount, VARIANT*
varYValues, VARIANT* varXValues, BOOL bUseScale)
      BOOL SetArray(long index, long* pnCount, VARIANT*
varYValues, VARIANT* varXValues, BOOL bUseScale)

      BOOL GetArray_V(long index, VARIANT * pnCount,
VARIANT* varYValues, VARIANT* varXValues, BOOL
bUseScale)
      BOOL SetArray_V(long index, VARIANT * pnCount,
VARIANT* varYValues, VARIANT* varXValues, BOOL
bUseScale)
```

Процедуры буферизованного доступа к значениям сигнала. Позволяют значительно ускорить обработку сигналов в плагинах за счет уменьшения числа вызовов через dispatch-интерфейс. Функции GetArray_V и SetArray_V могут вызываться из сценариев.

	Результат запроса массива.
`index`	Начальное смещение в сигнале.
`pnCount`	Требуемое количество значений. В этой же переменной возвращается реально скопированное количество.
`varYValues`	Указатель на вариантный массив значений по оси Y.
`varXValues`	Указатель на вариантный массив значений по оси X. Может быть равен 0, если сигнал с постоянной частотой дискретизации.
`bUseScale`	Применять градуировочную характеристику (true) или работать с «сырыми» данными (false).

Clone

```
IDispatch* Clone(long begin, long count)
```

Получить копию сигнала в указанном интервале.

	Интерфейс IWPSignal
`begin`	Индекс первого значения.
`count`	Количество значений.

GetSType, SetSType

```
unsigned __int64 GetSType(long nType, long nAx)
void SetSType(long nType, long nAx, unsigned __int64 nVal)
```

Получить или установить единицы измерения для указанной оси.

	Зависит от указанного типа значения
`nType`	Тип значения: знач. — описание 0 — Тип характеристики 1 — Величина 2 — Единицы измерения
`nAx`	Константа оси (0 – X, 1 – Y, 2 – Z).
`nVal`	Константа единиц измерения.

ConvertTime

```
BOOL ConvertTime(double srcTime, double* pdstTime, long type)
BOOL ConvertTime_V(double srcTime, VARIANT* pdstTime, long type)
```

Пересчитать значение времени рекордера во время СЕВ и наоборот. Функция ConvertTime_V может вызываться из сценариев. Сигнал должен содержать канал СЕВ.

	Результат операции (TRUE – успешно)
`srcTime`	Исходное значение времени.
`pdstTime`	Пересчитанное значение времени
`type`	Тип пересчета:

знач.	описание
1	Время рекордера во время СЕВ
2	Время СЕВ во время рекордера

IsSEVExists

```
BOOL IsSEVExists()
```

Проверить наличие канала СЕВ в сигнале.

	TRUE, если в сигнале указан канал СЕВ

GetSEVSignal

```
IDispatch* GetSEVSignal()
```

Получить канал СЕВ из сигнала.

	Интерфейс IWPSignal

SetSEVSignal

```
void SetSEVSignal(BSTR sev)
```

Привязать канал СЕВ к сигналу.

`sev`	Название канала СЕВ

GetStartX_Orig, GetDetlaX_Orig

```
double GetStartX_Orig()
double GetDeltaX_Orig()
```

Получить время старта и шаг по X сигнала, без учета канала СЕВ.

	Время старта, шаг по X

3D-сигналы

С трехмерным сигналом можно работать, как с обычным двумерным сигналом, однако, есть и несколько специализированных методов.

Методы

GetSizeX, GetSizeZ

```
int GetSizeX(int nReserved)
int GetSizeZ()
```

Получить размеры трехмерного сигнала.

	Размерность сигнала по оси X или Z

ResizeXZ

```
void ResizeXZ(int xsize, int zsize, BOOL bReservd)
```

Установить размеры трехмерного сигнала.

xsize, zszie	Размерность сигнала по оси X и Z
bReservd	Зарезервированный параметр, должен быть false.

GetStartZ, SetStartZ, GetDeltaZ, SetDeltaZ

```
double GetStartZ(void)
void SetStartZ(double startZ)
double GetDeltaZ(void)
void SetDeltaZ(double deltaZ)
```

Получить/установить начальное значение и шаг по оси Z.

GetZ, SetZ

```
double GetZ(int index)
void SetZ(int index, double d)
```

Получить/установить значение по оси Z. При равномерном шаге по Z, установить значение можно только через SetStartZ(), SetDeltaZ().

,d	Значение по оси Z
index	Номер элемента сигнала из диапазона 0..(GetSizeZ()-1)

GetRangeZ

```
void GetRangeZ(double* minZ, double* maxZ)
void GetRangeZ_V(VARIANT* minZ, VARIANT* maxZ)
```

Получить диапазон значений по оси Z. Второй вариант – для использования в сценариях.

minZ, maxZ	Минимальное и максимальное значения по оси Z, возвращаемые значения.

GetY_iZ, SetY_iZ

```
double GetY_iZ(int indexX, int indexZ, BOOL viaTransformer)
void SetY_iZ(int indexX, int indexZ, double d, BOOL viaTransformer)
```

Доступ к значению трехмерного сигнала по индексам.

, d	Значение сигнала
indexX, indexZ	Индексы по осям X и Z
viaTransformer	Использовать ГХ: 1 – использовать, 0 – нет.

GetYXZ

```
double GetYXZ(double x, double z, int pow)
```

Возвращается значение сигнала, соответствующего заданным значениям по осям X и Z. Если точного значения нет – интерполированное значение. Pow определяет метод интерполяции.

	Значение сигнала
x, z	Значения по осям X и Z
pow	Вид интерполяции. См. GetYX().

GetIntervalSignalX, GetIntervalSignalZ

```
IDispatch* GetIntervalSignalX(int indexX, int ibeg, int iend)
```

```
IDispatch*  GetIntervalSignalZ(int  indexZ,  int
ibeg, int iend)
```

Создать интервальный сигнал по срезу (проекции) исходного 3D-сигнала.

	Интерфейс сигнала (IWPSignal)
`indexX, indexZ`	Фиксированный индекс по оси X или Z, для которого берется срез сигнала
`ibeg, iend`	Диапазон индексов, ограничивающий размеры среза

IWPUSML

Интерфейс пакетных файлов (УСМЛ и МЕРА).

Свойства

FileName

```
BSTR FileName
```

Полное имя файла.

ParamCount

```
long ParamCount
```

Количество параметров пакетного (УСМЛ или МЕРА) файла.

Name, Test, Date, Time

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

Название изделия, испытания, дата и время регистрации.

Методы

Instance

```
long Instance()
```

Возвращает указатель на объект, предоставляющий этот интерфейс. Используется для передачи объекта в качестве параметра.

	Указатель на объект

Parameter

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

Возвращает из пакетного файла сигнал по порядковому номеру.

	Интерфейс сигнала (IWPSignal)
index	Порядковый номер сигнала из диапазона 0..(ParamCount-1)

FileSave

```
void FileSave()
```

Сохранить изменения в файле.

AddParameter

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

Добавить сигнал в файл УСМЛ или МЕРА.

signal	Интерфейс сигнала (IWPSignal)

DeleteParameter

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

Удалить параметр с указанным порядковым номером.

index	Порядковый номер сигнала из диапазона 0..(ParamCount-1)

IWPOperator

Интерфейс вызова математических алгоритмов. В принятой терминологии *оператор* – это математический *алгоритм* в совокупности с *параметрами* его выполнения.

Свойства

Name

```
BSTR Name
```

Сокращенное название алгоритма.

Fullname

```
BSTR Fullname
```

Полное название алгоритма.

nSrc, nDst

```
long nSrc
long nDst
```

Число входных и выходных параметров. Например, для амплитудного спектра nSrc=1, nDst=1, а для функции взаимной корреляции nSrc=2, nDst=1.

Методы

Instance

```
long Instance()
```

Возвращает указатель на объект, предоставляющий этот интерфейс. Используется для передачи объекта в качестве параметра.

	Указатель на объект

Exec

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

Выполнить алгоритм. Если фактическое число входных (выходных) сигналов алгоритма меньше двух, то неиспользуемые параметры игнорируются.

	Код ошибки. В случае успеха равен нулю.
`src`	Первый входной сигнал
`src2`	Второй входной сигнал
`dst`	Первый выходной сигнал
`dst2`	Второй выходной сигнал

Error

```
long Error()
```

Получить код последней ошибки.

	Код ошибки. В случае отсутствия ошибок равен нулю.

MsgError

```
BSTR MsgError()
```

Получить текстовое описание последней ошибки.

	Текстовое описание последней ошибки.

getProperySet

```
BSTR getProperySet()
```

Получить список опций алгоритма.

	Строка имен опций алгоритма, перечисленных через запятую.

setProperty

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

Установить значение выбранного свойства алгоритма.

`name`	Имя свойства
`value`	Новое значение свойства

getProperty

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

Прочитать значение выбранного свойства алгоритма.

	Значение свойства
`name`	Имя свойства

loadProperties

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

Установить значение набора свойств алгоритма.

`values`	Новые значения набора свойств. Строка формата « имя_свойства1 = значение_свойства1 , имя_свойства2 = значение_свойства2 , … » . Пример: *" kindFunc = 3 , numPoints = 1024 , nBlocks = 1 "*. Значения пропущенных свойств не присваиваются (сохраняются значения по умолчанию).

getPropertyValues

```
BSTR getPropertyValues()
```

Прочитать значение всех свойств алгоритма.

Строка формата « имя_свойства1 = значение_свойства1, имя_свойства2 = значение_свойства2, … » .

SetupDlg

```
long SetupDlg()
```

Вызвать диалог настройки опций алгоритма и выбора исходных сигналов.

Результат выполнения диалога:

результат	знач.	описание
IDOK	1	Алгоритм запущен на выполнение
IDCANCEL	2	Отмена операции
IDERROR	-1	Ошибка открытия диалога

IWPNode

Узел дерева объектов WinПОС, «точка монтирования» объекта.

Свойства

Name

```
BSTR Name
```

Имя узла, объекта.

ChildCount

```
long ChildCount
```

Количество дочерних элементов данного узла. Например, для узла пакетного файла это число сигналов.

Методы

Instance

```
long Instance()
```

Возвращает указатель на объект, предоставляющий этот интерфейс. Используется для передачи объекта в качестве параметра.

	Указатель на объект

Root

```
IDispatch* Root()
```

Указатель на корневой узел дерева.

	Интерфейс узла дерева – IWPNode.

AbsolutePath, RelativePath

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

Полный или относительный путь узла.

	Строка, путь узла.
baseNode	Узел, от которого вычисляется относительный путь.

Reference

```
IDispatch* Reference()
```

Объект, на который ссылается данный узел.

	Объект, на который ссылается данный узел.

IsDirectory

```
long IsDirectory()
```

Проверка, является ли данный узел папкой или пакетным файлом.

	1 - если папка, 0 - в противном случае.

GetReferenceType

```
long GetReferenceType()
```

Тип объекта, на который ссылается данный узел.

	Тип объекта:

результат	знач.	описание
OT_FOLDER	0	Простая папка
OT_PFILE	1	Файл формата УСМЛ или МЕРА
OT_SIGNAL	2	Сигнал

Link

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

Поместить объект в список дочерних узлов данного узла.

	Объект, поддерживающий интерфейс IWPNode
object	Объект, который требуется поместить в дерево
name	Имя объекта
flag	Если узел с таким именем уже существует, то при flag=1 будет изменено имя нового узла («Имя» перейдет в «Имя#1»), при flag=0 старый объект будет замещен.

Unlink

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

Удалить дочерний узел с заданным именем.

objname	Строка, имя узла для удаления

IsChild

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

Проверить, является ли узел дочерним для данного узла.

	1 – является, в противном случае – 0.
testNode	Интерфейс IWPNode

GetNode, GetObject

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

Получить дочерний узел по имени / получить объект дочернего узла.

	Интерфейс узла / объекта
path	Путь к дочернему узлу

At

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

Получить дочерний узел по порядковому номеру.

↱	Интерфейс IWPNode
index	Порядковый номер, 0 .. (ChildCount-1)

IWPUnits

Интерфейс единиц измерения.

Методы

GetFuncCount

```
ULONG GetFuncCount()
```

Получить количество типов характеристик.

↱	Количество типов характеристик

GetFunc

```
ULONG GetFunc(ULONG n)
```

Получить идентификатор характеристики по порядковому номеру.

↱	Идентификатор характеристики
n	Номер характеристики

GetFuncName

```
BSTR GetFuncName(ULONG nFn)
```

Получить название характеристики по идентификатору.

↱	Название характеристики
nFn	Идентификатор характеристики

GetTypeCount

```
ULONG GetTypeCount(ULONG nFn, ULONG nAx)
```

Получить количество величин.

↱	Количество величин
nFn	Идентификатор характеристики
nAx	Константа оси

GetType

```
ULONG GetType(ULONG n, ULONG nFn, ULONG nAx)
```

Получить идентификатор величины по порядковому номеру.

↵	Идентификатор величины
n	Номер величины
nFn	Идентификатор характеристики
nAx	Константа оси

GetTypeName

```
BSTR GetTypeName(ULONG nTp)
```

Получить название величины по идентификатору.

↵	Название величины
nTp	Идентификатор величины

GetUnitsCount

```
ULONG GetUnitsCount(ULONG nTp)
```

Получить количество единиц измерения.

↵	Количество единиц измерения
nTp	Идентификатор величины

GetUnits

```
ULONGLONG GetUnits(ULONG n, ULONG nTp)
```

Получить идентификатор единиц измерения по порядковому номеру.

↵	Идентификатор единиц измерения
n	Номер единиц измерения
nTp	Идентификатор величины

GetUnitsByMark

```
ULONGLONG GetUnitsByMark(BSTR mark)
```

Получить идентификатор единиц измерения по обозначению.

↵	Идентификатор единиц измерения
mark	Обозначение единиц измерения

GetBaseUnits

```
ULONGLONG GetBaseUnits(ULONG nTp)
```

Получить идентификатор стандартных единиц измерения для указанной величины.

↗	Идентификатор единиц измерения
nTp	Идентификатор величины

GetUnitsName

```
BSTR GetUnitsName(ULONGLONG nUn)
```

Получить название единиц измерения по идентификатору.

↗	Название единиц измерения
nUn	Идентификатор единиц измерения

GetUnitsMark

```
BSTR GetUnitsMark(ULONGLONG nUn)
```

Получить обозначение единиц измерения по идентификатору.

↗	Обозначение единиц измерения
nUn	Идентификатор единиц измерения

ConvertValue

```
double ConvertValue(ULONGLONG nUnSrc, ULONGLONG nUnDst, double value)
```

Пересчитать значение из одних единиц измерения в другие. Если пересчет невозможен, возвращается исходное значение.

↗	Пересчитанное значение
nUnSrc	Идентификатор исходных единиц измерения
nUnDst	Идентификатор результирующих единиц измерения
value	Значение

GetCoefs, GetCoefs_V

```
void GetCoefs(ULONGLONG nUnSrc, ULONGLONG nUnDst, double* pK1, double* pK0)
void GetCoefs_V(ULONGLONG nUnSrc, ULONGLONG nUnDst, VARIANT* pK1, VARIANT* pK0)
```

Получить коэффициенты для пересчета из одних единиц измерения в другие. Функция GetCoefs_V может вызываться из сценариев. Если пересчет невозможен, возвращается значения коэффициентов K1=1, K0=0.

`nUnSrc`	Идентификатор исходных единиц измерения
`nUnDst`	Идентификатор результирующих единиц измерения
`pK1, pK0`	Коэффициенты

Часть 4. Интерфейсы плагинов

Любой подключаемый модуль должен реализовывать интерфейс IWPPlugin. См. главу *«Создание плагина шаг за шагом»* в части 2. Остальные интерфейсы позволяют значительно расширять возможности WinПОС:

`IWPPlugin`	– основной интерфейс любого подключаемого модуля,
`IWPImport`	– интерфейс импорта данных,
`IWPExport`	– интерфейс экспорта данных,
`IWPExtOper`	– интерфейс внешнего оператора.

Интерфейсы плагинов являются дуальными. Возвращаемое значение (HRESULT) – целое число: 0 – вызов прошел успешно (S_OK), в противном случае – код ошибки.

IWPPlugin

Основной интерфейс подключаемого модуля, принимающий команды и сообщения WinПОС.

Методы

Connect

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

WinПОС вызывает этот метод во время загрузки, передавая указатель на главный интерфейс приложения.

`app`	Указатель на главный интерфейс приложения – IWinPOS
`Value`	Возвращаемое значение. Не используется.

Disconnect

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

Отключение плагина.

`Value`	Возвращаемое значение. Не используется.

NotifyPlugin

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

Извещение о событиях WinПОС.

what	Код события. Например, для нажатия кнопки панели инструментов: старшее слово, HiWord(what) – код команды (см. RegisterCommand), младшее – LoWord(what) = 2.
param	Дополнительные данные, зависящие от типа сообщения
value	Возвращаемое значение. Не используется.

IWPImport

Список поддерживаемых WinПОС форматов файлов можно расширить. Для чтения файлов реализуйте интерфейс IWPImport и в методе Connect() вызовите функцию IWinPOS.RegisterImpExp(), передав в первом параметре указатель на этот интерфейс.

Методы

Open

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

Вызывается при нажатии кнопки **Открыть** в окне выбора файла. В этом методе можно считать все сигналы файла и создать список, к которому будет обращаться WinПОС через GetSignal().

path	Имя выбранного файла.
count	Число сигналов, содержащихся в файле.
ErrorCode	Код ошибки, 0 – если файл прочитан корректно.

Close

```
HRESULT Close()
```

Закрыть файл. Вызывается по окончании чтения сигналов. Здесь можно очистить список открытых сигналов.

GetSignal

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

WinПОС вызывает этот метод, размещая сигналы в дереве сигналов

`n`	Порядковый номер сигнала в файле.
`value`	Указатель на интерфейс IWPSignal очередного сигнала.

GetPreviewText

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

Запрос информации о файле до его открытия.

`path`	Имя выбранного файла.
`value`	Строка, размещаемая в нижней части окна открытия файла. Может содержать сведения о числе сигналов и особенностях записи.

IWPExport

Список поддерживаемых WinПОС форматов файлов можно расширить. Для сохранения файлов реализуйте интерфейс IWPExport и в методе Connect() вызовите функцию IWinPOS.RegisterImpExp(), передав во втором параметре указатель на этот интерфейс.

Методы

AddSignal

```
HRESULT AddSignal(IDispatch* signal)
```

С помощью этого метода WinПОС передает в плагин экспорта сигналы, выбранные для сохранения.

`signal`	Указатель на интерфейс IWPSignal очередного сигнала.

Save

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

Сохранение файла. Следует вызывать после добавления всех сигналов с помощью последовательных вызовов метода AddSignal().

`path`	Имя, под которым должен быть сохранен файл.
`ErrorCode`	Код ошибки, 0 – если файл сохранен корректно.

IWPExtOper

Для того чтобы подключить свой собственный алгоритм обработки, реализуйте в плагине интерфейс IWPExtOper. В методе Connect() вызовите функцию IWinPOS.RegisterExtOper(), передав указатель на этот интерфейс.

Методы

GetPropStr, SetPropStr

```
HRESULT GetPropStr(BSTR* pstr)
HRESULT SetPropStr(BSTR str)
```

Методы используются для передачи настроек алгоритма в виде строки. Вызываются средой при сохранении и загрузке настроек.

pstr, str	Строка / указатель на строку настроек, например, вида «param1=0.5, param2=0.75».

Exec

```
HRESULT Exec(IDispatch* psrc1, IDispatch* psrc2, IDispatch** pdst1, IDispatch** pdst2)
```

Метод вызывается при нажатии кнопки **Выполнить** диалога настройки или при иной форме обращения к алгоритму.

psrc1, psrc2	Интерфейсы входных сигналов.
pdst1, pdst2	Интерфейсы результатов – возвращаемые значения.

GetError

```
HRESULT GetError(long* pnerrcode, BSTR* perrstr)
```

С помощью этого метода внешний оператор может вернуть WinПОС код ошибки и строку, описывающую эту ошибку.

pnerrcode	Указатель для возврата кода ошибки.
perrstr	Указатель для возврата строки-пояснения к ошибке.

OnSetup

```
HRESULT OnSetup(long hWndParent, long* phWnd)
```

Вызывается при создании окна настройки алгоритма, например, при вызове внешнего оператора из меню. В этом методе можно создать

свою форму для редактирования параметров алгоритма и вернуть её дескриптор.

`hWndParent`	Дескриптор (HWND) родительского окна настройки.
`phWnd`	Дескриптор окна настройки, которое будет встроено в стандартное окно, – возвращаемое значение.

OnApply

```
HRESULT OnApply()
```

Метод OnApply вызывается при нажатии кнопки **Выполнить** или **Применить** в диалоге настройки алгоритма. В этом методе можно считать значения из полей редактирования формы, созданной в OnSetup.

OnClose

```
HRESULT OnClose()
```

Метод OnClose вызывается при закрытии окна настройки алгоритма. Подходяший момент, чтобы уничтожить форму, созданную в OnSetup.

Часть 5. Вызов алгоритмов

Алгоритмы доступны через дерево объектов WinПОС. То есть к алгоритмам можно обращаться по имени, выбирая из дерева объектов. Последовательность вызовов такая: получить оператор, загрузить необходимые настройки, выполнить оператор. Например, так выглядит вызов автоспектра с текущими настройками:

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

Загрузить настройки алгоритма можно либо поочередно, методом **setProperty**, либо одновременно, методом **loadProperties**, см. выше описание интерфейса IWPOperator. Значения пропущенных параметров не присваиваются (сохраняются значения по умолчанию). Так выглядит тот же вызов автоспектра с указанием настроек:

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

Процедуры упрощенного вызова алгоритмов

Вызов наиболее употребимых алгоритмов автоматизирован. В файле **POSBase.pas** (для VBScript – в файле **WinPOS.wps**) реализованы процедуры, оформленные в стиле вызовов «Командного режима ПОС», упрощающие вызов алгоритмов. Пример, приведенный выше, можно переписать так:

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

Названия процедур приведены ниже вместе с описанием настроек алгоритмов. Обозначения: `Src, Src2, Dst, Dst2` – это переменные, указывающие на объекты с интерфейсом IWPSignal, `Err` – код ошибки (0, если ошибок нет), `Opt` – строка настроек, где параметры со значениями перечислены через запятую: « имя_свойства1 = значение1 , имя_свойства2 = значение2 , … » . Пример: *" kindFunc = 3 , numPoints = 1024 , nBlocks = 1 "*.

Алгоритмы на основе быстрого преобразования Фурье (БПФ)

Алгоритмы, выполняющие БПФ, имеют ряд общих настроек:

`type` Тип функции:

тип функции	знач.	описание
AUTOSPECTR	0	автоспектр
CROSS	20	взаимный спектр
COHEREN	30	функции когерентности
TRANS	40	передаточных функций
COMPLEX	50	комплексный спектр

`kindFunc` В зависимости от значения поля type, может содержать значения из разных наборов констант. Подробнее см. ниже.

`method` Метод расчета: 0 – БПФ, 1 - ДПФ

`numPoints` Число точек, по которым вычисляется БПФ: 32…1048576

`nBlocks` Число порций усреднения: 1…(длина сигнала/numPoints)

`ofsNextBlock` Смещение порций относительно друг друга: 1, numPoints/4, numPoints/2, numPoints*3/4, numPoints

`typeWindow` Тип весовой функции:

тип весовой функции	знач.	описание
SINGLEWIN	1	прямоугольная функция
TRIANGLEWIN	2	треугольная функция
HANNINGWIN	3	функция Хэннинга
BLACKMANWIN	4	функция Блэкмана
FLATTOP	5	Flat-Top

`typeMagnitude` Тип значений:

тип значений	знач.	описание
MEAD	1	эффективные
PEAK	2	амплитудные значения

`isMO` Центрирование: 1 – включено, 0 – выключено

`isFill0` Дополнение нулями: 1 – дополнять, 0 – нет

`fMaxVal` Максимальные значения: 1 – максимальные, 0 – усредненные

`fLog` Логарифмирование: 1 – результат в дБ, 0 – нет

`log_kind` 0 – 20*logX, 1 – 10*logX

`log_fOpZn` Использовать опорное значение: 1 – использовать, 0 - нет

`log_OpZn` Опорное значение

`fPrSpec` Выполнить преобразование спектра

`prs_kind` Вид преобразования: 0 – 1, 1 – $1/\omega$, 2 – $1/\omega^2$, 3 – $2\sqrt{2}/\omega^2$, 4 – $1*\omega$, 5 – $1*\omega^2$

`prs_loFreq` Нижняя частота

`prs_s2n` Отношение сигнал/шум

`prs_fCorr` Использовать функцию-корректор

`prs_typeCorr` Тип функции: 0 – пользовательская (задана в prs_strCorr), 1 – функция A, 2 – функция B, 3 – функция C

prs_strCorr	Функция-корректор (строка вида "x1 y1 x2 y2 x3 y3...")
f3D	Флаг трехмерного представления результатов: 1 – результат – трехмерный спектр, 0 - нет
fSwapXZ	Время по оси X: 1– по оси X – время, по оси Z – частота; 0 – по оси X – частота, по оси Z – время (для 3D)

АвтоСпектр

Быстрый вызов:

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

Настройки:

type	0 (AUTOSPECTR)
kindFunc	Вид функции:

вид функции	знач.	описание
SPM	1	спектр плотности мощности
SM	2	спектр мощности
SPP	3	спектр плотности энергии
SMAG	4	спектр амплитудный
SRI	5	комплексный спектр в виде реальной и мнимой части
SMF	6	комплексный спектр в виде модуля и фазы

Октавный спектр

Быстрый вызов: нет

Настройки:

type	0 (AUTOSPECTR)
fFlt	Метод расчета спектра: 1 – полосовые фильтры, 0 – БПФ
fQual	1 – использовать фильтры высокой точности, 0 – простые
kindFunc	Вид функции:

вид функции	знач.	описание
Oktav1	10	октавный спектр
Oktav3	11	третьоктавный спектр
Oktav12	12	1/12-октавный спектр
Oktav24	13	1/24-октавный спектр

Комплексный спектр

Быстрый вызов:

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

Настройки:

type	50 (COMPLEX), kindFunc игнорируется

Взаимный спектр

Быстрый вызов:

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

Настройки:

type	20 (CROSS)
kindFunc	Вид функции:

вид функции	знач.	описание
CrSPM	21	спектр плотности мощности
CrRI	22	взаимный спектр в виде действ. и мнимой части
CrMF	23	взаимный спектр в виде модуля и фазы

Функция когерентности

Быстрый вызов:

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

Настройки:

type	30 (COHEREN)
kindFunc	Вид функции:

вид функции	знач.	описание
COHERF	31	функция когерентности
COP	32	когерентная выходная мощность
S_N	33	отношение сигнала к шуму
NOTCOP	34	некогерентная выходная мощность
NOTCHR	35	функция некогерентности

Передаточная функция

Быстрый вызов:

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

Настройки:

type	40 (TRANS)
kindFunc	Вид функции:

вид функции	знач.	описание
H1	41	передаточная функция H1
H2	42	передаточная функция H2

Преобразование спектра

Быстрый вызов: нет

Настройки:

kind	Вид преобразования: 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	Нижняя частота
signal2noise	Отношение сигнал/шум

useCorrector	Использовать функцию-корректор
strCorrector	Функция-корректор (строка вида "x1 y1 x2 y2 x3 y3...")
typeCorr	Тип функции: 0 – пользовательская (в strCorrector), 1 – функция A, 2 – функция B, 3 – функция C

Алгоритмы фильтрации

Рекурсивная фильтрация

Быстрый вызов:

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

Настройки:

iType	Тип аппроксимации:

тип аппроксимации	знач.	описание
Butterworth	1	Фильтр Баттерворта
Chebyshev	2	Фильтр Чебышева
Elliptic	3	эллиптический фильтр

iKind	Вид фильтра:

вид фильтра	знач.	описание
LowPass	1	ФНЧ
BandPass	2	полосовой фильтр
HighPass	3	ФВЧ

nOrder	Количество двухполюсников (порядок): 1…20
nRipple	Неравномерность (процент) в полосе пропускания: 1…5
fsr	Частота среза (для ФНЧ, ФВЧ)
fn	Частота среза нижняя (для полосового)
fv	Частота среза верхняя (для полосового)
fs	Частота опроса
HO	Коэффициент фильтра

Нерекурсивная фильтрация

Быстрый вызов:

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

Настройки:

iType	Тип аппроксимации (с использованием рядов Фурье). Игнорируется
iKind	Вид фильтра:

вид фильтра	знач.	описание
LowPass	1	ФНЧ
BandPass	2	полосовой фильтр
HighPass	3	ФВЧ
BandStop	4	режекторный

iTypeWin	Тип весового окна:

тип весового окна	знач.	описание
HANN	2	окно Хэнна
HAMMINGWIN	3	окно Хэмминга

nOrder	Количество коэффициентов (порядок), нечетное число: 1…1001
fsr	Частота среза (для ФНЧ, ФВЧ)
fn	Частота среза нижняя (для полосового и режекторного)
fv	Частота среза верхняя (для полосового и режекторного)
fs	Частота опроса

Медианная фильтрация

Быстрый вызов: нет

Настройки:

Type	Тип фильтра: 0 – дискретный, 1 – аналоговый
nPoints	Количество точек
Level	Порог (только для аналогового фильтра)
LevelLow	Нижний уровень (для дискретного фильтра)
LevelHi	Верхний уровень (для дискретного фильтра)
bAuto	Автоматическое определение уровней (для дискр.)

Действия над сигналами

Интегрирование (Первообразная)

Быстрый вызов:

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

Настройки:

method	Метод интегрирования:

метод интегрирования	знач.	описание
AILER_INT	1	метод Эйлера
HANNING_INT	2	метод Хэмминга
RC_INT	3	метод RC-цепочек
VIBRO_INT	4	виброинтегрирование

typeRezult	Центрирование, 1 – включено, 0 – выключено
numpointsAverg	Число точек осреднения (только для RC)
flagDelPerProcess	Подавление переходного процесса: 1– вкл., 0 – выкл. (только для вибро)
npointsPerProcess	Длина переходного процесса (только для вибро)
fsr	Частота среза фильтрации (только для вибро)

Дифференцирование

Быстрый вызов:

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

Настройки:

`method` Метод дифференцирования:

метод дифференц.	знач.	описание
THREE_POINTS	3	трехточечный метод
FIVE_POINTS	5	пятиточечный метод

Нормирование

Быстрый вызов: нет

Настройки:

`HiFront`	Верхняя граница
`LoFront`	Нижняя граница
`EnaShift`	Сдвиг значений сигнала относительно 0: 1 – разрешить (меняются статистические характеристики: МО, дисперсия и т.п.), 0 - запретить

Центрирование

Быстрый вызов: нет *Настройки:* нет

Арифметические операции

Быстрый вызов: нет

Настройки:

`kind` Вид операции:

вид операции	знач.	описание
CONST_PLUS	0	прибавление константы const
CONST_MINUS	1	вычитание константы const
CONST_MULTI	2	умножение на константу const
CONST_DIV	3	деление на константу const
BUF_PLUS	4	сложение значений двух сигналов
BUF_MINUS	5	вычитание значений второго сигнала из значений первого
BUF_MULTI	6	перемножение значений двух сигналов
BUF_DIV	7	деление значений первого сигнала на значения второго

`const` Константа (для операций с одним сигналом)

Логарифмирование

Быстрый вызов: нет

Настройки:

kind	20logX (0) или 10logX (1)
useOpZn	Использовать опорное значение (1), иначе (0) - максимум
OpZn	Опорное значение

Передискретизация

Быстрый вызов:

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

Настройки:

freq	Новая частота дискретизации
kind	Виды интерполяции:

вид интерполяции	знач.	описание
NOINT	0	Интерполяция отсутствует
LINEINT	1	Линейная интерполяция
PARABINT	2	Интерполяция полиномом второго порядка
SPLINE3INT	3	Кубические локальные сплайны

type — Тип фильтрации:

тип фильтрации	знач.	описание
NOFLT	0	Фильтрация отсутствует
IIRFLT	1	Рекурсивная фильтрация
FIRFLT	2	Нерекурсивная фильтрация

srcdt	Сохранять исходный тип данных

Преобразование Гильберта

Быстрый вызов: нет

Настройки:

nPoints	Число точек, по которым вычисляется БПФ: 32…1048576
nBlocks	Число порций усреднения: 1…(длина сигнала/nPoints)
isMO	Центрирование: 1 – включено, 0 – выключено

Огибающая

Быстрый вызов: нет

Настройки:

kind	Метод: 0 - пик-детектор, 1 – преобразование Гильберта
coef	Коэффициент (K) для метода пик-детектора

Если выбран метод преобразования Гильберта, к этому алгоритму также применяются настройки преобразования Гильберта (см. выше).

Исследование сигналов

Вероятностные характеристики

Элементы результирующего сигнала (Dst) содержат значения вероятностных характеристик исходного сигнала.

значение	смещение	описание
IDX_MO	0	Математическое ожидание
IDX_D	1	Дисперсия
IDX_SIG	2	Среднеквадр.отклон.
IDX_A3	3	Асиметрия
IDX_A4	4	Эксцесс
IDX_MAG	5	Амплитуда

Таким образом, чтобы получить, например, дисперсию сигнала, следует вызвать метод Dst.GetY(1) после выполнения алгоритма.

Быстрый вызов: нет *Настройки:* нет

Плотность вероятности

Быстрый вызов:

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

Настройки:

npoints	Число точек вычисляемой характеристики
type	Метод расчета и представления плотности распределения вероятности:

метод расчета	знач.	описание
PARZEN	1	ПРВ, метод ядерных оценок
HIST	2	ПРВ, расчет в виде гистограммы
PARZNORM	8	вероятность попадания, метод ядерных оценок
HISTNORM	4	вероятность попадания, расчет в виде гистограммы

Функция автокорреляции

Быстрый вызов:

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

Настройки:

npoints	Количество точек для построения корреляционной функции
eps	Возвращается оценка статистической погрешности

Функция взаимной корреляции

Быстрый вызов:

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

Настройки:

npoints	Количество точек для построения корреляционной функции
eps	Возвращается оценка статистической погрешности

Параметрический график

Быстрый вызов: нет

Настройки:

type	0 – параметрический график, 1 – полярный, 2 – параметрический для сигналов с одинаковой дискретизацией (берутся значения с одинаковыми индексами)

Анализ динамических процессов, вибраций

Эта группа алгоритмов выделена в отдельное меню «Виброанализ». Для них не определены функции быстрого вызова, однако любой из алгоритмов можно найти с помощью метода GetObject и выполнить, вызвав метод Exec. Алгоритмы виброанализа расположены в папке «VibroOpers» дерева объектов. Так для вызова алгоритма расчета АФЧХ надо вызвать GetObject со строкой «/VibroOpers/расчет АФЧХ».

Последовательная обработка (тренды)

Настройки:

typeRez	Тип функции:

тип функции	знач.	описание
RezRMS	0	СКЗ
RezMAG	1	Амплитуда
Rez2MAG	2	Размах
RezMO	3	среднее
RezCRS	4	Пик-фактор

nPoints	Размер блока (число точек)
bEquivMag	Эквивалентная амплитуда (1 – расчет амплитуды через СКЗ, 0 – нет)

СКЗ в полосе

Настройки:

fn	Нижняя частота
fv	Верхняя частота (значение -1 соответствует максимальному значению)
kindFunc	Вид характеристики. Здесь: kindFunc=2 (спектр мощности)
nPoints	Кол-во точек для расчета БПФ
tTransf	Интегрирование:

знач.	описание
0	Виброускорение (интегрирование отсутствует)

1	Виброскорость (однократное интегрирование)
2	Виброперемещение (двукратное интегрирование)

Тахо

Настройки:

typeAxesX	Ось X: 0 – секунды, 1 – Гц, 2 – обороты
deltaF	Ширина полосы фильтра, Гц
LevelLo, LevelHi	Уровни для расчета сигнала оборотов. См. flagAbsOtn
level_dF	Предельный уровень изменения частоты на участке («Макс шаг.») Используется при установленном flagFiltrTaho.
dPNTwide	Ширина импульса для фильтрации исходного тахо-сигнала. Используется при установленном flagFiltrTahoSrc.
coefTaho	Множитель, передаточный коэффициент для оборотов.
flagFront	Тип фронта: 1 – отрицательный, 0 – положительный.
flagAbsOtn	Уровни: 1 – относительные(%), 0 – абсолютные.
flagFiltrTaho	Сглаживание оборотов: 1 – включена, 0 – выключена
flagFiltrTahoSrc	Фильтрация исходного тахо-сигнала.
typeBlkSett	Способ задания размера исходного блока при расчете тахо:

знач.	описание
0	Фиксированный блок
1	Фиксированное количество периодов
2	Таблица периодов для разных частотных диапазонов

nPeriods	Кол-во периодов на одном блоке (freqTab не используется)
freqTab	Таблица кол-ва оборотов для разных частот (в качестве разделителя чисел используется «;»).
bTahoFFT	Использовать метод БПФ
nPointsFFT	Размер блока для метода БПФ (блок дополняется нулями до этого размера).

расчет АФЧХ

Включает настройки Тахо (см. выше). *Собственные настройки:*

typeRez — Тип функции:

тип весовой функции	знач.	описание
RezMag	21	Амплитуда гармоники
RezFase	22	Фаза
RezMagFase	23	Амплитуда/фаза
RezTAXO	24	Тахо
RezSMMAG	25	Амплитуда от частоты
RezSF	28	Фаза от частоты
RezGRMS	29	Grms

typeRezultMag — Тип амплитудных значений:

тип весовой функции	знач.	описание
RezRMS	0	СКЗ
RezMAG	1	Амплитуда (А)
Rez2MAG	2	Размах (2*А)

rGarm	Номер (порядок) гармоники
stepF	Шаг по частоте, Гц

flagStepF	Шаг по частоте: 0 – среднее значение, 1 – максимальное
freqLo	Диапазон частот: нижняя частота
freqHi	Диапазон частот: верхняя частота (-1 для максимальной)
typeWin	Тип весовой функции:

тип весовой функции	знач.	описание
SINGLEWIN	1	прямоугольная функция
TRIANGLEWIN	2	треугольная функция
HANNINGWIN	3	функция Хэннинга
BLACKMANWIN	5	функция Блэкмана

typeTransf — Интегрирование:

знач.	описание
0	Виброускорение (интегрирование отсутствует)
1	Виброскорость (однократное интегрирование)
2	Виброперемещение (двукратное интегрирование)

faseOfs	Фазовая поправка, °
flagSortRez	Сортировка: 1 – включена, 0 – выключена
flagMonotonFase	Монотонная фаза (не ограничена диапазоном 0°..360°)
flagMethodOb	Расчет по оборотам
bXLog	Логарфимическая ось X
bYLog	Логарфимическая ось Y
freqLoSp, freqHiSp	Grms: диапазон по частоте для спектра
freqLoSpR, freqHiSpR	Grms: диапазон по частоте в относительных единицах от частоты тахо (0.8..1.2 соответствует -20%..+20%)

Диаграмма Кэмпбелла

Включает настройки Тахо (см. выше) с префиксом «t_» (пример: «t_LevelLo»). *Собственные настройки:*

typeRez	Тип функции: 24 – Тахо, любое другое число – диаграмма.
typeRezultMag	Тип амплитудных значений:

тип весовой функции	знач.	описание
RezRMS	0	СКЗ
RezMAG	1	Амплитуда (А)
Rez2MAG	2	Размах (2*А)

typeAxesX	Тип оси X: 1 – Гц, 2 – об./мин
stepF	Шаг по оборотам, Гц
flagStepF	Шаг по частоте: 0 – среднее значение, 1 – максимальное
freqLo	Диапазон оборотов: нижняя частота
freqHi	Диапазон оборотов: верхняя частота (-1 для максимальной)
nPoints	Число точек, по которым вычисляется БПФ: 32…262144
blkOffset	Смещение следующей порции длины nPoints
kindFuncFFT	Вид характеристики:

вид характеристики	знач.	описание
SPM	1	спектр плотности мощности
SM	2	спектр мощности
SPP	3	спектр плотности энергии
SMAG	4	спектр амплитудный

typeWin	Тип весовой функции:

тип весовой функции	знач.	описание
SINGLEWIN	1	прямоугольная функция
TRIANGLEWIN	2	треугольная функция
HANNINGWIN	3	функция Хэннинга
BLACKMANWIN	5	функция Блэкмана
FLATTOP	7	Flat Top

typeTransf	Интегрирование:

знач.	описание
0	Виброускорение (интегрирование отсутствует)
1	Виброскорость (однократное интегрирование)
2	Виброперемещение (двукратное интегрирование)

flagSortAmp	Сортировка по амплитуде: 1 – включена, 0 – выключена
deadband	Мертвая зона для поиска максимумов
freqLoSp, freqHiSp	Диапазон для диаграммы в виде столбиков и кружочков

Порядковый анализ 3D

Включает настройки Диаграммы Кэмпбелла и Тахо (см. выше). *Собственные настройки:*

typeAxesX	Тип оси Z: 0 – секунды, 1 – Гц, 2 – об./мин
typeXAxis	Тип оси X: 0 – порядок, 1 – Гц
ordLo	Диапазон порядков, минимум
ordHi	Диапазон порядков, максимум (-1 для автоопределения)
stepN	Шаг порядков
fAFC	Расчет через АЧХ: 1 – вкл., 0 – выкл.
fLog	Логарифмирование: 1 – результат в дБ, 0 – нет
log_kind	0 – 20*logX, 1 – 10*logX
log_fOpZn	Использовать опорное значение: 1 – использовать, 0 - нет
log_OpZn	Опорное значение

Ударный спектр

Настройки:

bShockAuto	Выделение удара (если bShockAuto = 0, сразу считается спектр по выбранному диапазону)
level	Порог срабатывания (Соотношение удар/шум)
length	Полная длительность удара, с
length2	Отступ влево от пика, с
freqLo, freqHi	Диапазон частот, Гц
stepF	Шаг по частоте (если bLog = 0)
coeff	Коэффициент демпфирования
nPoints	Количество точек на декаду (если bLog = 1)
bLevelAmp	Поиск удара по превышению заданного уровня, в %
bLen2Auto	Автоматический выбор отступа влево (если bLen2Auto ==

	TRUE, length2 не используется)
bLog	Логарифмический шаг: 1 – вкл., 0 – выкл.
bCoeffQ	Вместо коэф. демпфирования, в диалоге настраивается добротность
bFlt	Фильтрация: 1 – вкл., 0 – выкл.
bResampl	Передискретизация: 1 – вкл., 0 – выкл.
FltLo, FltHi	Частоты среза ФНЧ/ФВЧ, Гц
freqNew	Новая частота для передискретизации, Гц

Следящий фильтр

Включает настройки Тахо (см. выше) с префиксом «t_» (пример: «t_LevelLo»). *Собственные настройки:*

typeRez	Тип функции: 24 – Тахо, любое другое число – след.фильтр.
iFlt	Алгоритм фильтрации: 0 – рекурсивный, 1 – нерекурсив.
iType	Тип аппроксимации
iKind	Тип фильтра
iTypeWin	Тип весовой функции (нерекурсивный фильтр): 2 – Ханн, 3 - Хэмминг
nOrder	Порядок фильтра
nRipple	Неравномерность в полосе пропускания
nGarm	Номер гармоники
iMethod	Способ расчета полосы: 0 – от частоты тахо, 1 – от частоты выбранной гармоники
freqLo, freqHi	Частоты среза, в относительных единицах (0…1)
freqMin, freqMax	Мин./макс. значения ширины полосы, Гц

Часть 6. Встроенный редактор сценариев

Для написания собственных быстродействующих алгоритмов обработки, для обработки значительных объемов данных, и создания приложений, опирающихся на WinПОС, но требующих дополнительных настроек или способных формировать специализированные отчеты, лучше всего подходит Borland Delphi. Также можно использовать Borland C++ Builder, Microsoft Visual C++, Visual Basic или FoxPro.

Однако для написания небольших сценариев автоматизации работы WinПОС или несложных алгоритмов больше подходит Visual Basic Script. VBScript входит в состав поставки Microsoft Windows, не требует отдельного компилятора, а удобная среда редактирования и отладки сценариев включена в состав WinПОС.

Редактор сценариев (Рис. 5.1) открывается через меню **Сценарий→Редактор сценариев…**

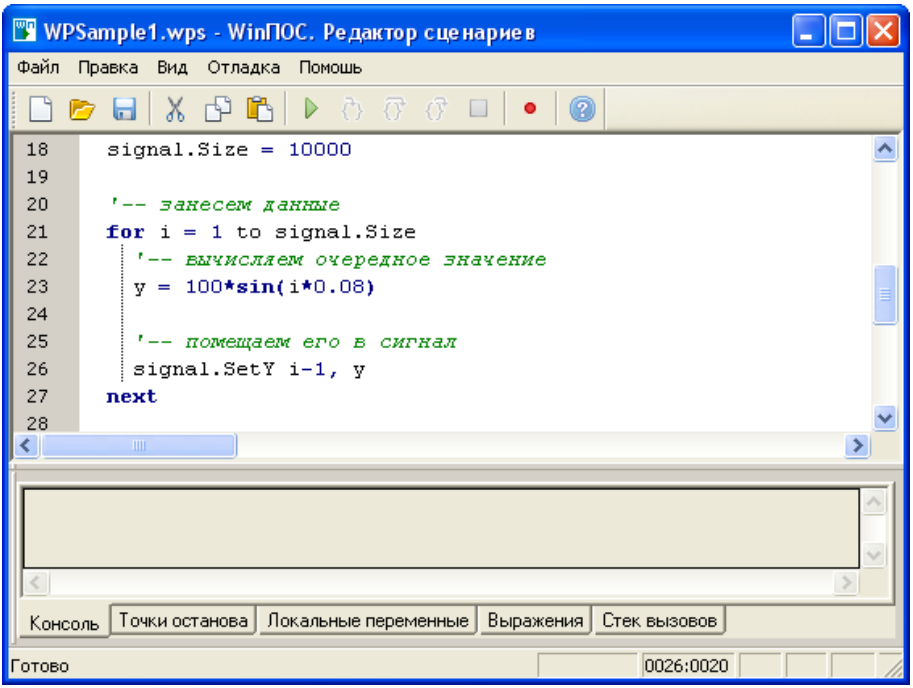


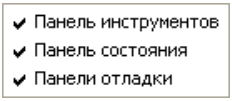
Рис. 5.1. Окно подготовки сценариев обработки

Редактор сценариев – это текстовый редактор с синтаксической подсветкой и стандартным набором инструментов, к которым можно обратиться с помощью меню, панели инструментов и горячих клавиш. Редактор также предоставляет все необходимые инструменты для выполнения сценариев под отладкой: точки останова и выполнение по шагам, просмотр локальных переменных и стека вызовов, вычисление выражений.

Отличительные особенности встроенного редактора сценариев:

- кнопки управления отладкой на панели инструментов,
- синтаксическая подсветка,
- контроль парности скобок и визуализация отступов,
- линейка нумерации строк с полосой точек останова (расположена слева),
- панели отладки (под областью редактирования).

С помощью меню **Вид** можно отключать и подключать визуальные элементы редактора.



Синтаксическая подсветка позволяет сократить количество ошибок при наборе текста сценария и облегчает восприятие текста:

синим жирным шрифтом выделяются зарезервированные слова VBScript,

синим – символы,

курсивом – строковые константы,

зеленым курсивом – комментарии, а

идентификаторы – обычным черным шрифтом.

Парные скобки при наборе подсвечиваются зеленым цветом:

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

Символ табуляции отмечается вертикальной полосой, что позволяет более наглядно изображать вложенность циклов и условий.

Режим редактирования

Команды режима редактирования сценария:

Панель инстр.	Меню	Сочетание клавиш	Описание
	Файл→Новый сценарий	Ctrl+Shift+N	Очистить окно для нового сценария
	Файл→Открыть сценарий…	Ctrl+Shift+O	Открыть файл для редактирования
	Файл→Сохранить сценарий	Ctrl+Shift+S	Сохранить редактируемый файл
	Файл→Сохранить сценарий как…		Сохранить сценарий под новым именем
	Файл→Закрыть редактор	Alt+F4	Закрыть окно редактора
	Правка→Отменить изменения	Ctrl+Z	Отменить последние действия
	Правка→Вырезать	Ctrl+X	Вырезать выделенный фрагмент в буфер
	Правка→Копировать	Ctrl+C	Копировать выделенный фрагмент в буфер
	Правка→Вставить	Shift+V	Вставить в позицию курсора текст из буфера
	Правка→Выделить все	Shift+A	Выделить весь текст
	Правка→Найти…		Поиск заданной строки
	Правка→Заменить…		Поиск и замена строки
	Правка→Перейти к строке…		Переход к строку с заданным номером
	Помощь→Содержание…	F1	Справка по объектам WinПОС.

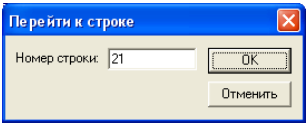


Рис. 5.2. Перейти к строке

Диалог *Перейти к строке* (Рис. 5.2, меню **Правка→Перейти к строке…**) помогает при навигации в пространном сценарии. Номер строки можно увидеть как на линейке слева, так и в панели состояния.

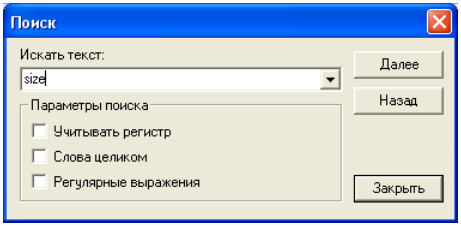


Рис. 5.3. Диалог поиска

Диалог *Поиск* (Рис. 5.3, меню **Правка→Найти…**) позволяет найти все вхождения заданной в поле *Искать текст* строки, с учетом регистра (**Учитывать регистр**) и положения в окружающем тексте (**Слова целиком** и **Регулярные выражения**). Кнопки **Далее** и **Назад** задают направления поиска относи-

тельно текущей позиции курсора.

Диалог *Поиск и замена* (Рис. 5.4, меню **Правка→Заменить…**) повторяет диалог *Поиск*, но позволяет заменить найденную строку на указанную в поле *Заменить на*. Кнопка **Заменить** выполняет однократную замену, **Заменить все** – автоматически заменяет все найденные участки текста. Флажок **Заменить в выбранном блоке** позволяет ограничить область изменения текста.

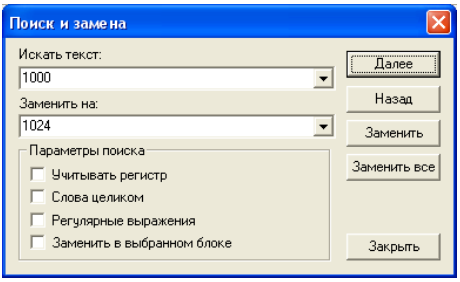


Рис. 5.4. Диалог поиска и замены

Режим отладки

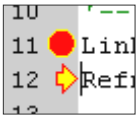
Команды режима выполнения и отладки сценария:

Панель инстр.	Меню	Сочетание клавиш	Описание
	Отладка→Начать/Продолжить исполнение	Ctrl+F10	Переход в режим выполнения сценария
	Отладка→Вкл. /Выкл. точку останова	F9	Установка брекпоинта
	Отладка→Шаг внутрь	F11	Выполнение по шагам со заходом в процедуры
	Отладка→Шаг вперед	F10	Выполнение по шагам
	Отладка→Шаг наружу	Ctrl+F11	Выход из процедуры
	Отладка→Остановить отладку	Alt+F10	Прекращение выполнения сценария
	Отладка→Останов на старте		Режим остановки после запуска

ⓘ В режиме отладки становится недоступным редактирование текста сценария.

Переход в режим отладки происходит по кнопке . Если не были установлены точки останова (●, брекпоинты), и не была включена опция остановки после запуска (**Отладка→Останов на старте**), сценарий будет выполнен полностью.

Для установки точки останова на выбранной строке нужно нажать кнопку ● или <F9>. Брекпоинт будет поставлен на текущей строке, в поле линейки появится соответствующий



значок – красная точка.

Для продолжения выполнения сценария можно снова нажать кнопку . Можно также продолжить выполнение сценария по шагам, с помощью кнопок , и . Позиция строки, которая будет выполняться следующей, отмечается желтой стрелкой в поле линейки.

Отладочные панели

Отладочные панели дают полную картину состояния выполняемого сценария в каждый момент времени (позволяют следить за ходом выполнения сценария, изменением содержимого переменных и т.п.).

Консоль

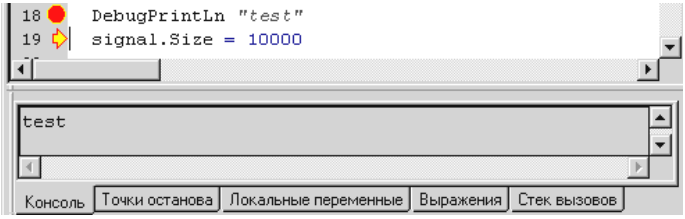


Рис. 5.5. Консоль

На *Консоль* (рис. 5.5) направляется отладочная печать (функции DebugPrint() и DebugPrintLn()).

Точки останова

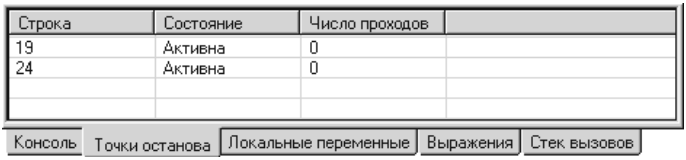


Рис. 5.6. Закладка точек останова

Закладка *точек останова* (рис. 5.6) показывает список брекпоинтов с номерами строк, состоянием и счетчиком количества проходов через строку. Состояние точки (*Активна* или *Блокирована* – изображается серой) можно изменить через контекстное меню панели, удалить точку можно не только по <F9>, но и через контекстное меню.

Локальные переменные

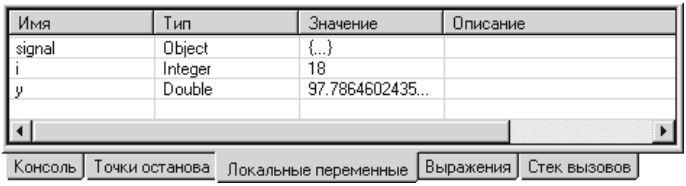


Рис. 5.7. Закладка локальных переменных

Значения и типы *локальных переменных* можно просмотреть на одноименной

закладке (рис. 5.7).

Двойной щелчок мышки на строке переменной открывает диалог 5.8, в котором можно не только увидеть значение переменной, но и изменить его (поле *Значение*, кнопка **Обновить**).

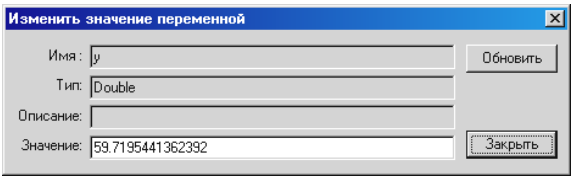


Рис. 5.8. Диалог просмотра и изменения переменной

Выражения

Закладка *выражения* (рис. 5.9) дает возможность вычисления любых выражений, записанных в синтаксисе VBScript.

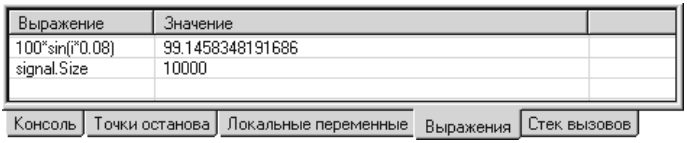


Рис. 5.9. Закладка вычисления выражений

Новое выражение можно добавить, удалить и изменить с помощью контекстного меню, откуда вызывается диалог редактирования выражений (рис. 5.10).

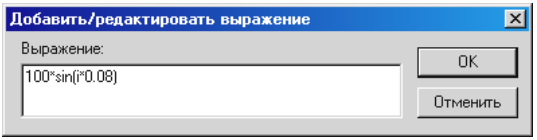


Рис. 5.10. Диалог добавления выражений

Выделив строку сценария, ее также можно скопировать на закладку *Выражения* через контекстное меню редактора.

Стек вызовов

Последняя закладка, *Стек вызовов* (рис. 5.11), помогает ориентироваться при отладке сценария, состоящего из большого количества процедур, и просто незаменима при написании кода с рекурсивными вызовами. Верхней в стеке является текущая процедура.

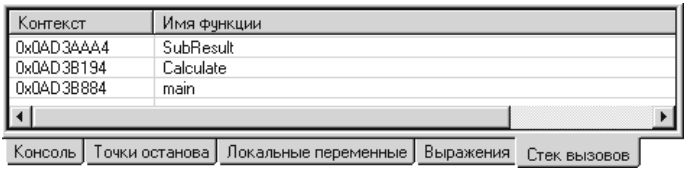


Рис. 5.11. Стек вызовов

Приложение.

Примеры

При установке WinПОС в каталоге C:\Mera Files\WinPOS создает подкаталог Samples, в котором можно найти примеры реализации сценариев на VBScript и примеры создания приложений и подключаемых модулей на Delphi.

`VBS`	– примеры сценариев на Visual Basic Script
`Delphi`	– примеры приложений на Delphi
`DelphiPlugIn`	– пример подключаемого модуля на Delphi
`DelphiCommon`	– служебные модули (Delphi)

В каталогах VBS и Delphi находится по 6 примеров программ.

№	Описание	Используемые возможности
1	Генератор сигнала. Создается сигнал и заполняется значениями, вычисляемыми с помощью функции.	✓ создание сигнала ✓ добавление сигнала в дерево WinПОС ✓ доступ к значениям сигнала
2	Отображение сигнала в графике. Создается и заполняется сигнал, готовый сигнал помещается в график.	✓ создание сигнала с неравномерной осью X ✓ доступ к графической подсистеме WinПОС ✓ отображение сигнала на графике
3	Загрузка произвольного УСМЛ файла, обработка сигнала (АвтоСпектр).	✓ загрузка пакетного файла ✓ вызов процедуры RunFFT()
4	Вызов диалога выбора файла, загрузка двоичного файла данных, обработка сигнала (вызов алгоритма WinПОС по имени, задание опций), печать.	✓ работа с диалогом открытия файла ✓ загрузка двоичного файла ✓ выполнение оператора через Exec() ✓ вызов метода PrintPreview()
5	Загрузка УСМЛ файла, вычисление АвтоСпектра. Доп. обработка результата: по спектру находим три значения частоты, в которых величины амплитуд максимальны, и печатаем эти значения	✓ дополнительная обработка результатов выполнения алгоритмов WinПОС ✓ создание страницы с двумя графиками (исходный сигнал и результат)
6	Загрузка УСМЛ файла, обработка сигнала: передискретизация, фильтрация, автоспектр.	✓ последовательная выполнение цепочки алгоритмов ✓ отображение нескольких

параметров в одном графике

Для удобства восприятия примеры на Delphi скомпонованы в отдельные, включаемые директивой «$I», файлы, каждый из которых содержит только тело процедуры. Для компиляции этих файлов в качестве отдельных модулей-«юнитов» необходимо дополнить их стандартным обрамлением: unit, interface, implementation,…

В каталоге DelphiPlugIn расположен пример подключаемого модуля (plug-in), добавляющего на панель инструментов WinПОС кнопку, по которой запускается один из примеров из каталога Delphi.

В каталоге DelphiCommon расположен автоматически созданный по библиотеке типов (TLB) файл с описанием интерфейсов WinПОС (WinPOS_ole_TLB.pas) и файл PosBase.pas, содержащий процедуры упрощенного доступа к алгоритмам WinПОС и описания констант.

Кроме того, большинство плагинов, размещенных в подпапке PlugIns рабочего каталога пакета (как правило, это «C:\Program Files\WinPOS»), устанавливаются вместе с исходными файлами, которые также можно использовать как примеры.

Ниже приведен пример программы, которая создает тестовый сигнал, выполняет расчет спектра и отображает результат в графическом виде.

```
// Пример 1.
// Создание сигнала, вызов алгоритма, отрисовка в графиках.
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, Opt, Err : OleVariant;
    api : IWPGraphs;
    i, hPage, hGraph, hGraphFFT, hAxis, hAxisFFT : Integer;

begin
  Application.Initialize;

  with WINPOS do
  begin

  // 1) создаем сигнал со значениями типа Double
  signal:= CreateSignal(VT_R8) as IWPSignal;
```

```
  if Assigned(signal) then   // если сигнал создан
    begin

      // помещаем сигнал в дерево
      Link('/Signals/generator', 'sinus', signal as IDispatch);
      Refresh();

      signal.size:= 10000; // зададим длину сигнала

      for i:= 0 to 9999 do // занесем данные
      begin
        y:= (100)*sin(i*0.08); // вычисляем очередное значение
        signal.SetY(i, y); // помещаем его в сигнал
      end;

      // 2) применим к сигналу оператор "Автоспектр"
      RunFFT(signal, dst1, dst2, Opt, Err);

      // положим результат в дерево WinПОС
      Link('/Signals/Result', 'spectrum', dst1);

      // 3) отобразим исходный и результирующий сигналы
      // получаем доступ к графической подсистеме WinПОС
      api:= GraphAPI as IWPGraphs;

      // создаем новую страницу для графиков
      hPage:= api.CreatePage;

      // страница всегда создается с областью для рисования
      hGraph:= api.GetGraph(hPage,0);

      // создаем дополнительный график для спектра
      hGraphFFT:= api.CreateGraph(hPage);

      // получаем ось Y
      hAxis:= api.GetYAxis(hGraph,0);

      // создаем  новую линию в графике
      api.CreateLine(hGraph, hAxis, signal.Instance);

      // получаем ось Y во втором графике
      hAxisFFT:= api.GetYAxis(hGraphFFT,0);

      // создаем новую линию в графике спектра
      api.CreateLine(hGraphFFT, hAxisFFT, dst1.Instance);

      // нормализуем графики
      api.NormalizeGraph(hGraph);
      api.NormalizeGraph(hGraphFFT);

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

Следующий пример - программа для формирования нестандартного варианта экспресс-отчета. В цикле обрабатывается файл УСМЛ или МЕРА, параметры которого раскладываются по страницам 3x3 графика, устанавливается новый масштаб по оси Y, позволяющий оценить исходные уровни параметров заданного файла.

```
// Пример 2.
// Программа для формирования экспресс-отчета
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 определяется и инициализируется в модуле POSBase.pas
  with WINPOS do
  begin
    // открываем УСМЛ с помощью стандартного диалога WinПОС
    FileName:= USMLDialog();

    if fileName<>'' then // если файл был выбран
    begin
      // получаем доступ к графической подсистеме Winpos
      api:= GraphAPI as IWPGraphs;

      usml:= LoadUsml(fileName) as IWPUSML; //загружаем УСМЛ

      // так мы сразу попадем в создание новой страницы (см. ниже)
      nGr:= nHor*nVer;
      nPg:= 0;
      for i:=0 to usml.ParamCount-1 do
      begin
        // теперь можно взять сигнал по порядковому номеру в файле
        signal:= usml.Parameter(i) as IWPSignal;
        if (nGr = nHor*nVer) then
        begin
```

```
          // создаем новую страницу для графиков
          hPage:= api.CreatePage;

          // устанавливаем вид 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);
        // получаем ось Y
        hAxis:= api.GetYAxis(hGraph,0);

        // создаем  новую линию в графике
        api.CreateLine(hGraph, hAxis, signal.Instance);

        // нормализуем график
        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.
```

Вызов виброалгоритмов

Для выполнения большинства виброалгоритмов из скрипта или плагина, требуется предварительно выполнить расчет тахо по исходному тахо-сигналу. Ссылка на рассчитанный сигнал тахо сохраняется в операторе, если надо выполнить расчет по нескольким сигналам с тем-же тахо, отдельно пересчитывать его каждый раз не требуется.

Если уже имеется сигнал частоты тахо (рассчитанный или записанный с аппаратуры), его можно передать в алгоритм с помощью функции **setProperty**, дополнительно выполнять расчет тахо не требуется. В этом случае вторым параметром функции **Exec** необходимо передать пустой объект, иначе алгоритм воспримет его как ссылку на исходный тахо-сигнал и расчет будет выполнен неправильно.

Пример расчета АЧХ из скрипта:

```
sub main
  set oper = WinPOS.GetObject("/VibroOpers/расчет АФЧХ")
  set src = WinPOS.GetObject("/Signals/test.MERA/test_data")
  set src_taho = WinPOS.GetObject("/Signals/test.MERA/taho")

  ' Сначала необходимо посчитать тахо,
  ' ссылка на результат сохраняется в операторе
  oper.LoadProperties "typeRez = 24" ' тахо
  oper.Exec src_taho, src_taho, dst_taho, dst2

  ' Потом поменять тип на требуемый
  ' (21 - ампл, 22 - фаза, 23 ампл/фаза) и повторить расчет
  oper.LoadProperties "typeRez = 21" ' Амплитуда гармоники
  oper.Exec src, src_taho, dst_a1, dst2

  WinPOS.Link "/Signals/res", dst_taho.SName, dst_taho
  WinPOS.Link "/Signals/res", dst_a1.SName, dst_a1
  WinPOS.Refresh
end sub
```

Пример расчета АЧХ из скрипта с использованием готового сигнала частоты:

```
sub main
  set oper = WinPOS.GetObject("/VibroOpers/расчет АФЧХ")
  set src = WinPOS.GetObject("/Signals/test.MERA/test_data")
  set taho = WinPOS.GetObject("/Signals/test.MERA/taho_f")

  ' Передаем сигнал тахо в алгоритм
  oper.setProperty "pRezTaho", taho
  ' Создаем пустой объект, чтобы передать его в качестве
  ' второго параметра в функцию Exec
  set src_empty = WinPOS.GetObject("")

  oper.LoadProperties "typeRez = 21" ' Амплитуда гармоники
  oper.Exec src, src_empty, dst_a1, dst2

  WinPOS.Link "/Signals/res", dst_a1.SName, dst_a1
  WinPOS.Refresh
end sub
```

Указатель методов