Programmer's Reference to the DCOM Interface of DEWESoft 6.4

Version 2006-06 (for DEWESoft 6.4)

www.dewetron.com

Contents

List of Figures

List of Tables

1 Introduction

This reference should help developers to use DEWESoft's DCOM Interface. The interface as documented in this paper is available in DEWESoft 6.4 or newer.
As illustrated in Figure 1, the DCOM Interface of DEWESoft 6.4 and newer offers the possibility to design plug-ins for DEWESoft doing some special tasks on the one hand and to implement other applications controlling DEWESoft, which is called 'automation', on the other hand.
Plug-ins and applications for automating DEWESoft can be developed in different programming languages, e.g. Borland Delphi, MS Visual Basic, MS Visual C++/C# or NI LabVIEW (automation only). In this reference some examples are described using MS Visual Basic and Borland Delphi 7.

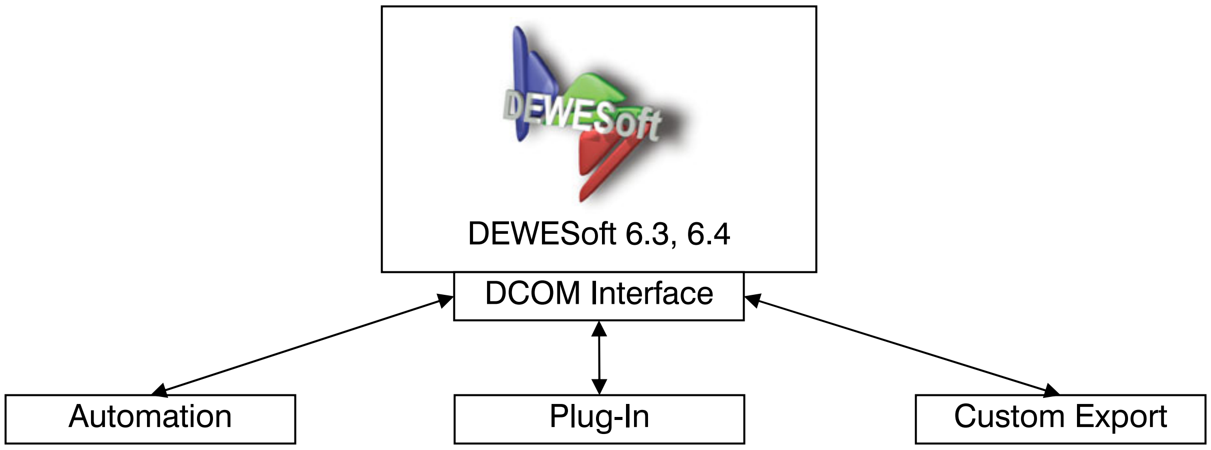
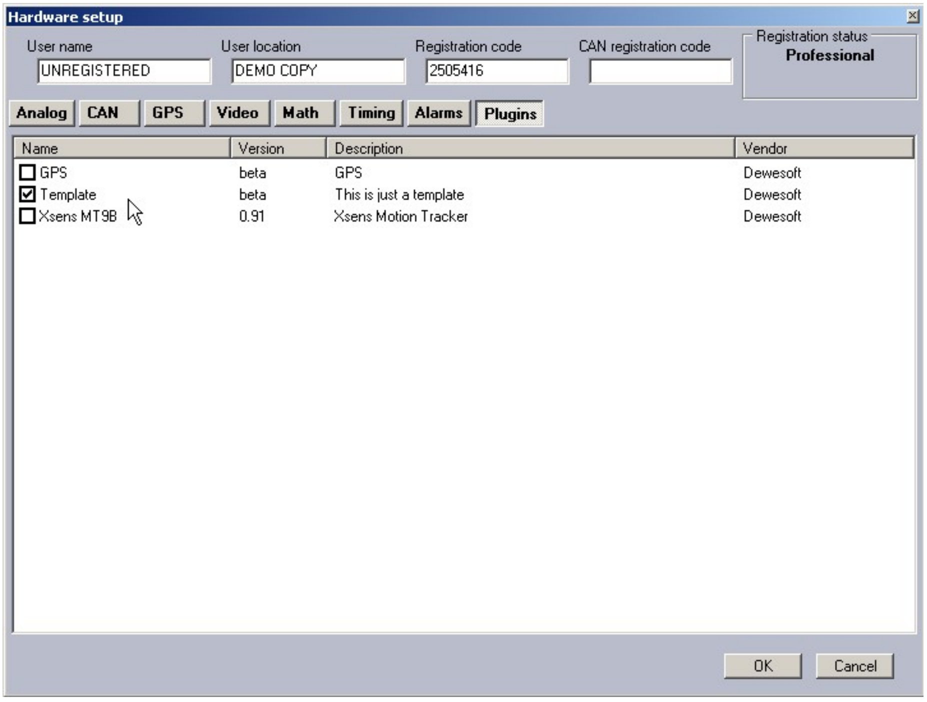


Figure 1 The DCOM Interface

2 Plug-ins for DEWESoft

2.1 General Information about DEWESoft Plug-ins

On the one hand, the motivation for creating plug-ins for DEWESoft can be to gain access to acquired data during measurement e.g. in order to make some special calculations on it or to output the data via RS-232, etc. On the other hand, the task of a plug-in could be to add further data channels to DEWESoft. This could be data of a special sensor (e.g. connected via RS-232) or data which is derived from special calculations on data currently measured by DEWESoft. In both ways, coming from DEWESoft or being added to DEWESoft, data can be either synchronous or asynchronous.
When a plug-in is registered to the windows registry before or on startup of DEWESoft, as described later, it will be listed in the "Plugins" tab of the hardware setup as shown in Figure 2. If a plug-in is checked here, it will appear in the "Plugins" tab of the measurement setup screen too.

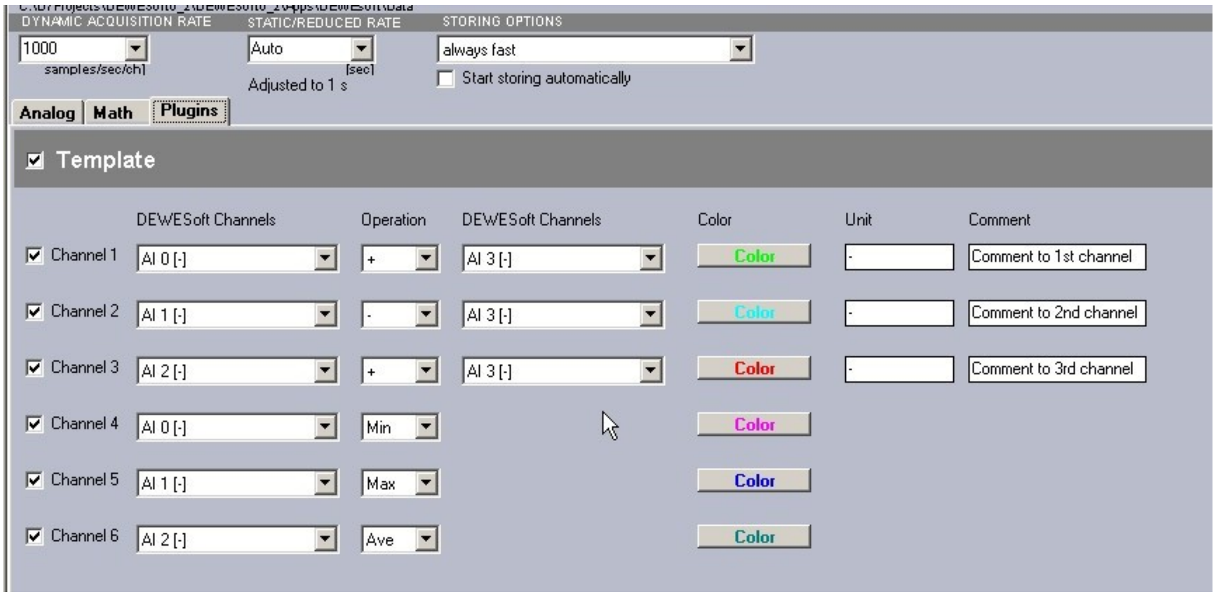


User name: UNREGISTERED
User location: DEMO COPY
Registration code: 2505416
CAN registration code:
Registration status: Professional

Name	Version	Description	Vendor
☐ GPS	beta	GPS	Dewesoft
☑ Template	beta	This is just a template	Dewesoft
☐ Xsens MT9B	0.91	Xsens Motion Tracker	Dewesoft

Figure 2 Plugins Tab of the Hardware Setup

A plug-in, which is an ActiveX library (ActiveX DLL), can consist of two main parts. The one part is the plug-in implementation itself, which is necessary in any case. The other part is a frame or a window which task is to allow the user to change some settings of the plug-in. Whether this part is a frame or a separate window depends on the development environment used for creating the plug-in. In case of a plug-in designed in Delphi, using a frame would be a suitable solution. This frame will appear directly embedded into the Plugins tab of the measurement setup-screen of DEWESoft as shown in Figure 3.


DYNAMIC ACQUISITION RATE: 1000 samples/sec/ch]
STATIC/REDUCED RATE: Auto [sec] Adjusted to 1 s
STORING OPTIONS: always fast
☐ Start storing automatically

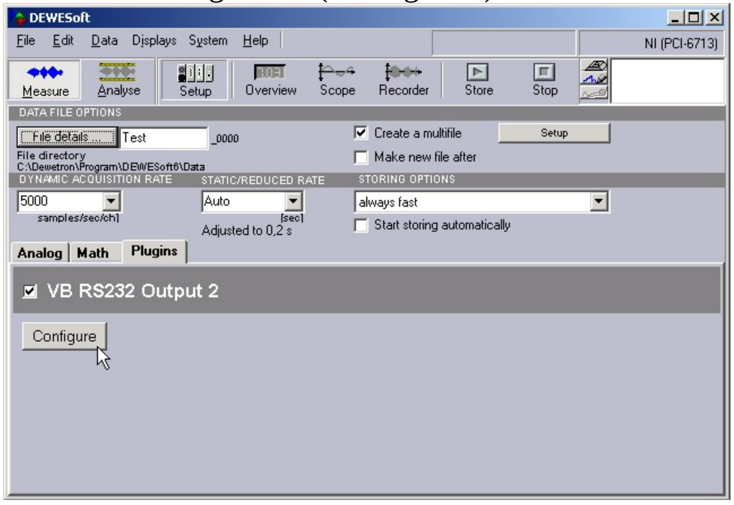
Analog | Math | Plugins

☑ Template

	DEWESoft Channels	Operation	DEWESoft Channels	Color	Unit	Comment
☑ Channel 1	AI 0 [-]	+	AI 3 [-]	Color	-	Comment to 1st channel
☑ Channel 2	AI 1 [-]	-	AI 3 [-]	Color	-	Comment to 2nd channel
☑ Channel 3	AI 2 [-]	+	AI 3 [-]	Color	-	Comment to 3rd channel
☑ Channel 4	AI 0 [-]	Min		Color		
☑ Channel 5	AI 1 [-]	Max		Color		
☑ Channel 6	AI 2 [-]	Ave		Color		

Figure 3 Plug-in Frame, directly embedded into the Setup Screen

In case of a plug-in created by means of a programming environment not supporting "frames", e.g. Visual Basic, it is not possible to create such a frame being embedded into DEWESoft. Therefore settings of a plug-in have to be made in a separate window. This window can be opened by clicking the "Configure"-button which will appear instead of the frame in the Plugins tab (see Figure 4).



DATA FILE OPTIONS
File details: Test _0000
File directory C:\Dewetron\Program\DEWESoft6\Data
☑ Create a multifile
☐ Make new file after
Setup
DYNAMIC ACQUISITION RATE: 5000 samples/sec/ch]
STATIC/REDUCED RATE: Auto [sec] Adjusted to 0,2 s
STORING OPTIONS: always fast
☐ Start storing automatically

Analog | Math | Plugins

☑ VB RS232 Output 2

Configure

Figure 4 "Configure" Button in Plugins Tab

2.2 Prerequisites for DEWESoft Plug-ins

The IPlugin2 interface is available in DEWESoft Version 6.2 and newer. With version 6.3 of DEWESoft the interface IPlugin3 is introduced, offering additional functionalities for plug-ins. IPlugin3 is derived from IPlugin2, so it has to be used together with the functions of IPlugin2.

It is not recommended to use the interface "IPlugin" for the development of any new plug-ins any longer. "IPlugin2" substitutes "IPlugin" and offers an extended functionality. "IPlugin" will still remain supported in DEWESoft in order to work in conjunction with any plug-ins developed by customers so far.

3 General Information about DEWESoft

3.1 The Buffer Structure

The data acquired by DEWESoft is stored according to the buffer structure illustrated in Figure 5. At first the data is stored to the so-called direct buffer. After having acquired a certain number of values, the minimum, maximum, average and RMS values are calculated and stored to intermediate buffer. After the collection of a certain amount of values within the intermediate buffer, once again the minimum, maximum, average and RMS values are calculated. Consequently these are stored to the so-called reduced buffer.

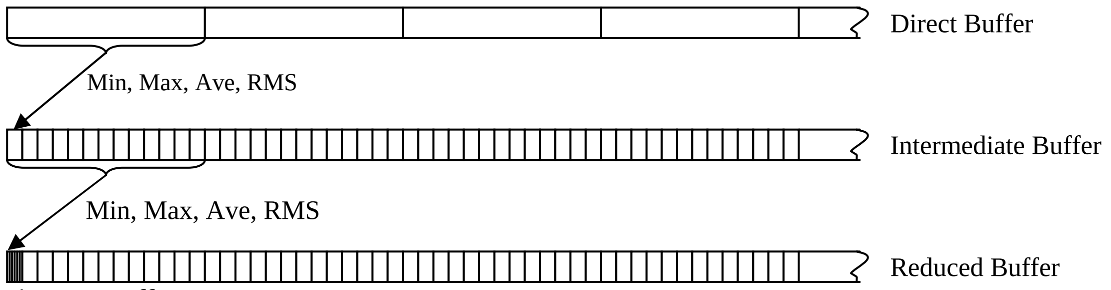


Figure 5 Buffer Structure

3.2 The Channel Structur

The channel structure of DEWESoft can be compared to Figure 6. The channels of the same kind are grouped together to a ChannelGroup. Such a group can be e.g. for analog channels, for digital channels, for CAN data, etc. These groupes are put together to the ChannelGroups which is an element of Data.
Another way for acessing a channel would be via the list of AllChannels or of UsedChannels.

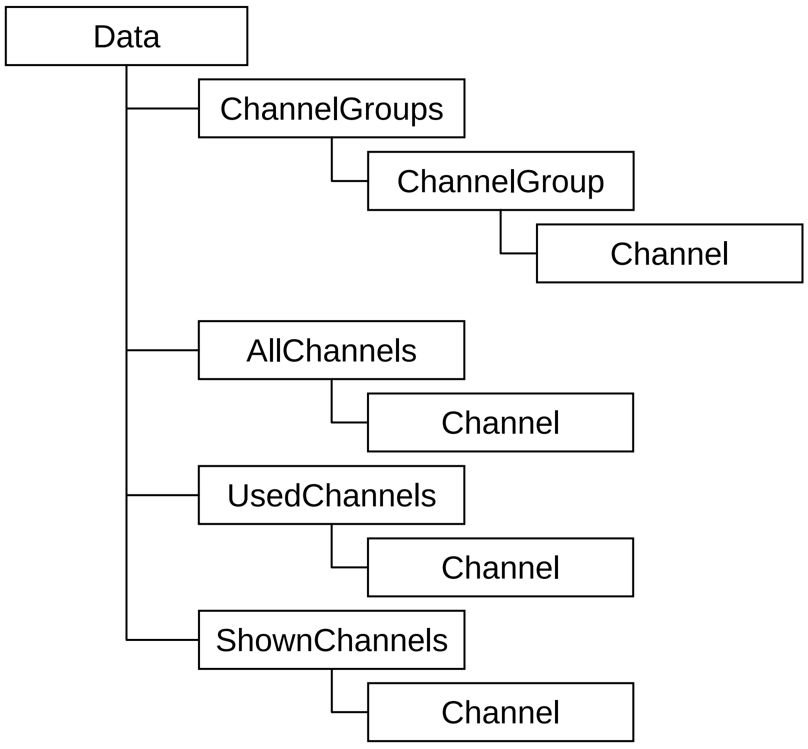
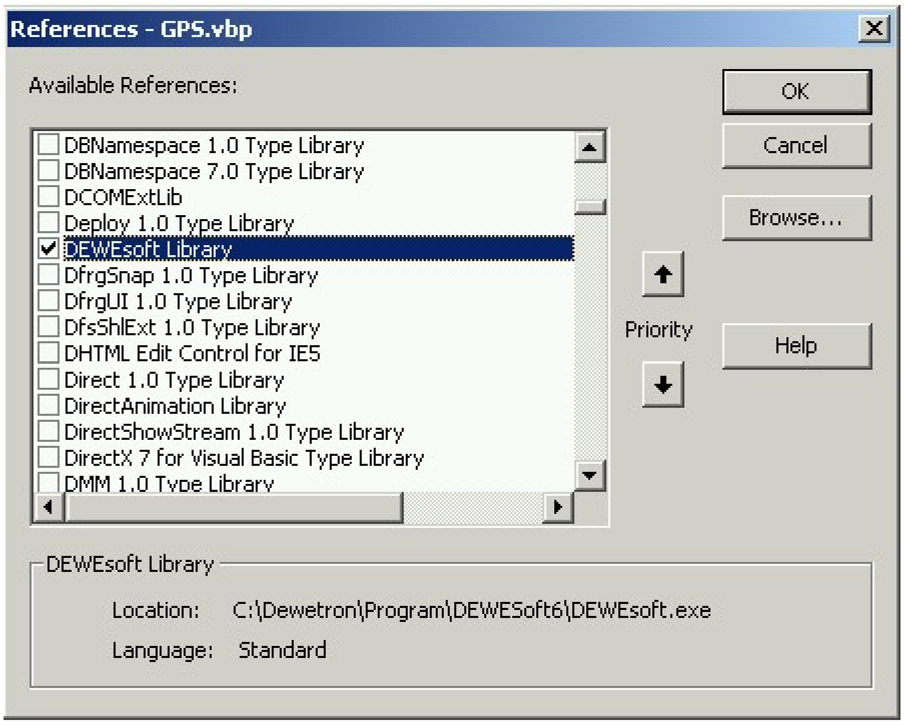


Figure 6 The Channel Structure

4 Creating Plug-Ins in Visual Basic 6

For developing plug-ins for DEWESoft, the DEWESoft library has to be referenced to the project within the development system. In Visual Basic this is done via ‘Project’ >> ‘References’ and checking the DEWESoft Library as shown in Figure 7.
Plug-ins created with an earlier version of the library will not work with newer versions of DEWESoft.



- [ ] DBNamespace 1.0 Type Library
- [ ] DBNamespace 7.0 Type Library
- [ ] DCOMExtLib
- [ ] Deploy 1.0 Type Library
- [✓] DEWEsoft Library
- [ ] DfrgSnap 1.0 Type Library
- [ ] DfrgUI 1.0 Type Library
- [ ] DfsShlExt 1.0 Type Library
- [ ] DHTML Edit Control for IE5
- [ ] Direct 1.0 Type Library
- [ ] DirectAnimation Library
- [ ] DirectShowStream 1.0 Type Library
- [ ] DirectX 7 for Visual Basic Type Library
- [ ] DMM 1.0 Type Library

DEWEsoft Library
Location: C:\Dewetron\Program\DEWESoft6\DEWEsoft.exe
Language: Standard

Figure 7 Reference to DEWESoft Library in a VB Project

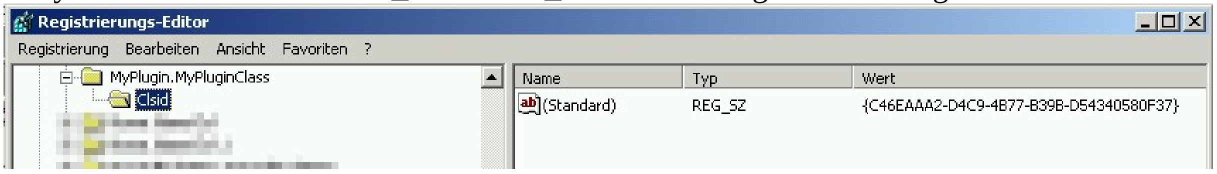
When programming new plug-ins, each plug-in has to implement all the functions of Plugin2.

4.1 GUID and Registration

In order to apply a plug-in to DEWESoft, it is necessary to register its GUID (Globally Unique IDentifier) to windows. This can be done using ‘regsvr32.exe’. Within the command window (cmd.exe) type in as follows:

```
regsvr32 c:\myfolder\myplugin.dll
```

(where `‘c:\myfolder’` has to be substituted by the corresponding path and `‘myplugin’` by the name or the ActiveX-library). Having done this, the GUID is inserted into the registry and can be found using the registry editor (‘regedit.exe’). The entry will be named like the name-property of the class as defined in the development environment. This entry will be found in “HKEY_CLASSES_ROOT” looking similar to Figure 8.

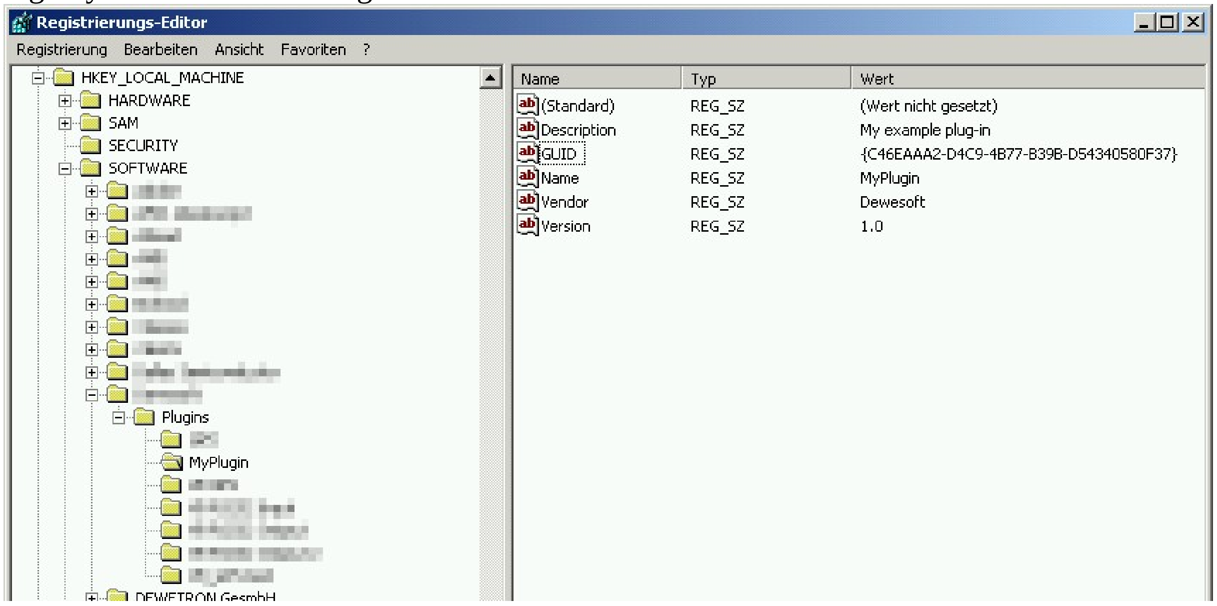


Name	Typ	Wert
(Standard)	REG_SZ	{C46EAAA2-D4C9-4B77-B39B-D54340580F37}

Figure 8 Class ID registered using regsvr32

In order to inform DEWESoft about the existence of a plug-in, an additional entry needs to be created within the registry. In the section “HKEY_LOCAL_MACHINE\SOFTWARE\Dewesoft\Plugins” a key, e.g. “MyPlugin” must be created. Within this key, the string-values “Description”, “GUID”, “Name”, “Vendor” and

“Version” can be set. Setting “GUID” to the corresponding value in “HKEY_CLASSES_ROOT” (e.g. at “HKEY_CLASSES_ROOT\ MyPlugin.Class1\Clsid”) is necessary. Setting the value “Name” is recommended in order to be able to identify the plug-in within the hardware configuration dialog of DEWESoft. The entries within the windows registry can be similar to Figure 9.



Name	Typ	Wert
(Standard)	REG_SZ	(Wert nicht gesetzt)
Description	REG_SZ	My example plug-in
GUID	REG_SZ	{C46EAAA2-D4C9-4B77-B39B-D54340580F37}
Name	REG_SZ	MyPlugin
Vendor	REG_SZ	Dewesoft
Version	REG_SZ	1.0

Figure 9 DEWESoft plug-in entries

4.2 MS Visual Basic Example Code

```
Implements IPlugin2
Dim FApp As DEWEsoft.App
Dim Ch1 As DEWEsoft.IChannel
Dim Ch2 As DEWEsoft.IChannel
```

```
Private Sub Class_Terminate()

      Set FApp = Nothing
      Set Ch1 = Nothing
      Set Ch2 = Nothing

End Sub
```

```
Private Sub IPlugin2_ClearChannelsInstance()

      'Executed before all channels are destroyed in DEWESoft.
      'All instances have to be cleared.
      '(set to “nothing” or “nil” in other languages)

      Set Ch1 = Nothing
      Set Ch2 = Nothing

End Sub
```

```
Private Sub IPlugin2_Configure()

      'Executed when the ‘Configure’-button is clicked by the user
      frmMyConfigurationDialog.frmMyConfigurationDialog_Init

End Sub
```

```
Private Sub IPlugin2_HideFrame()
      'Executed, when the setup screen is left.

End Sub
```

```
Private Property Get IPlugin2_Id() As String
      'The GUID of the plug-in should be returned here.

End Property
```

```
Private Sub IPlugin2_Initiate(ByVal DeweApp As DEWEsoft.IApp)
      'When DEWESoft is started and the plug-in
      'is checked in the hardware configuration.

      Set FApp = DeweApp

End Sub
```

```
Private Sub IPlugin2_LoadSetup(ByVal Data As Variant)
      'When a channel setup is loaded.
      '(corresponds to File >> Load Setup)

      MountChannels

End Sub
```

```
Private Sub IPlugin2_NewSetup()
      'When a new channel setup is created.
      '(corresponds to File >> New Setup)

      MountChannels

End Sub
```

```
Private Sub IPlugin2_OnGetData()
      'When new data is added to a channel.
      'This happens every 40 ms.
      'It is not necessary to add any new data.

Dim TimeStamp As Double

      TimeStamp = FApp.MasterClock.GetCurrentTime
      Call Ch1.AddAsyncSingleSample(Rnd, TimeStamp)
      Call Ch2.AddAsyncSingleSample(Rnd, TimeStamp)

End Sub
```

```
Private Sub IPlugin2_OnOleMsg(ByVal Msg As Long, ByVal Param As Long)

End Sub
```

```
Private Sub IPlugin2_OnStartAcq()
      'When data acquisition starts

End Sub
```

```
Private Sub IPlugin2_OnStartStoring()
       'When the storing of data is started

End Sub
```

```
Private Sub IPlugin2_OnStopAcq()
       'When data acquisition is stopped

End Sub
```

```
Private Sub IPlugin2_OnStopStoring()
       'When the storing of data is stopped

End Sub
```

```
Private Sub IPlugin2_OnTrigger(ByVal Time As Double)
       'When a trigger occurs.
       'Returns the corresponding measurement time in seconds.

End Sub
```

```
Private Sub IPlugin2_SaveSetup(Data As Variant)
       'When a channel setup is saved
       '(corresponds to File >> Save Setup)

End Sub
```

```
Private Function IPlugin2_ShowFrame(ByVal Parent As Long) As Boolean
       'When entering the setup screen.
       IPlugin2_ShowFrame = False

End Function
```

```
Private Sub IPlugin2_UpdateFrame()
       'When the frame is updated.

End Sub
```

```
Private Property Let IPlugin2_Used(ByVal RHS As Boolean)

       FUsed = RHS

End Property
```

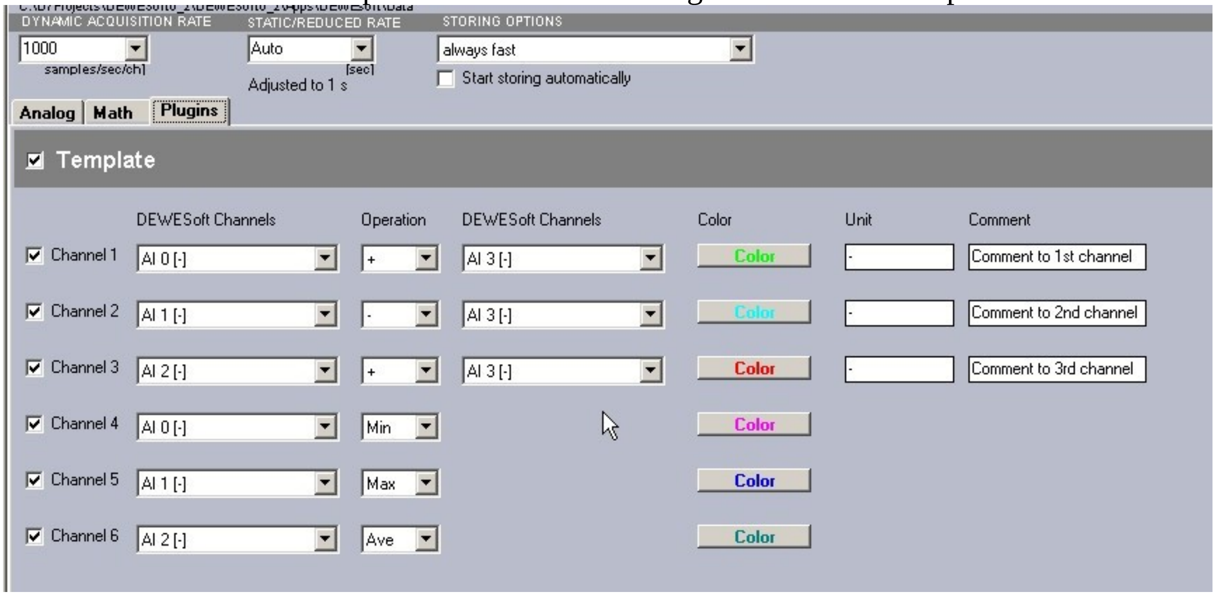
```
Private Property Get IPlugin2_Used() As Boolean
       'Corresponds to the check box in the
       'hardware configuration in the plugin tab.
       IPlugin2_Used = FUsed

End Property
```

```
Private Sub MountChannels()
       'Sub for mounting the channels.
Dim PluginGroup As DEWEsoft.IPluginGroup
Set PluginGroup = FApp.Data.Groups(8)
```

```
      Err.Clear
      On Error Resume Next
      ChLat.Used = True
      ChLon.Used = True

      If Err.Number <> 0 Then
            Err.Clear
            On Error GoTo errorhandler
            Set Ch1 = PluginGroup.MountChannel(5, True, -1)
            Ch1.Name = "First channel"
            Ch1.Used = True

            Set Ch2 = PluginGroup.MountChannel(5, True, -1)
            ChLon.Name = "Second channel"
            ChLon.Used = True
      End If

      Set PluginGroup = Nothing
      Exit Sub

errorhandler:
      MsgBox ("Error on MountChannels")

End Sub
```

5 Creating Plug-ins in Borland Delphi 7

Delphi is the most suitable programming environment for creating plug-ins. The main advantage is the ability to create so-called frames. These frames are similar to forms but do not have window-like characteristics like a title bar or a surrounding border, when embedded in DEWESoft.
The frame of a plug-in created in Delphi will appear seamlessly integrated in the "Plugins" tab of the measurement setup screen of DEWESoft e.g. like shown in the picture below.


DYNAMIC ACQUISITION RATE: 1000 samples/sec/ch
STATIC/REDUCED RATE: Auto [sec] Adjusted to 1 s
STORING OPTIONS: always fast
☐ Start storing automatically

Analog | Math | Plugins

☑ Template

	DEWESoft Channels	Operation	DEWESoft Channels	Color	Unit	Comment
☑ Channel 1	AI 0 [-]	+	AI 3 [-]	Color	-	Comment to 1st channel
☑ Channel 2	AI 1 [-]	-	AI 3 [-]	Color	-	Comment to 2nd channel
☑ Channel 3	AI 2 [-]	+	AI 3 [-]	Color	-	Comment to 3rd channel
☑ Channel 4	AI 0 [-]	Min		Color		
☑ Channel 5	AI 1 [-]	Max		Color		
☑ Channel 6	AI 2 [-]	Ave		Color		

Figure 10 Plug-in Frame, directly embedded into the Setup Screen

The task of the frame of a plug-in is to allow the user to alter some necessary settings required to make the plug-in work.

5.1 The basic steps

The basic steps for creating plug-ins using Delphi are as follows:
Create a new ActiveX Library (New >> Other... >> ActiveX >> ActiveX Library).

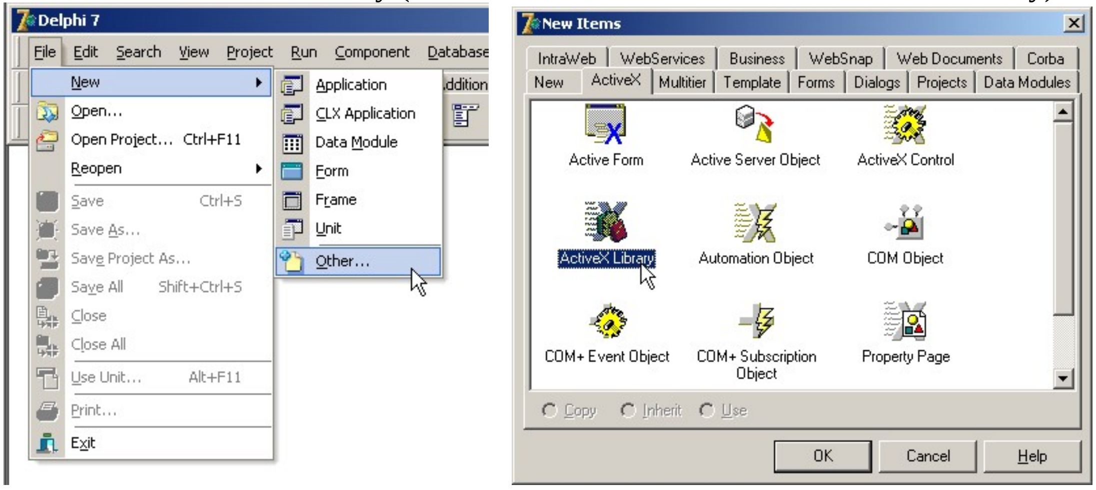
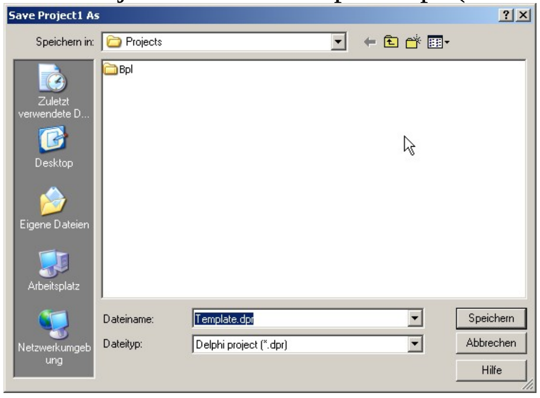


Figure 11 New ActiveX Library

Save Project as... >> Template.dpr (choose a name suitable for your application).



Dateiname: Template.dpr
Dateityp: Delphi project (*.dpr)

Figure 12 Save Project as...

Add a new type library (New >> Other... >> ActiveX >> Type Library).

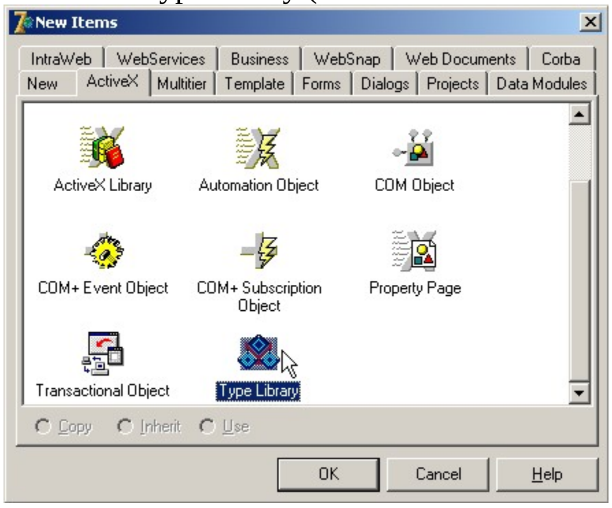
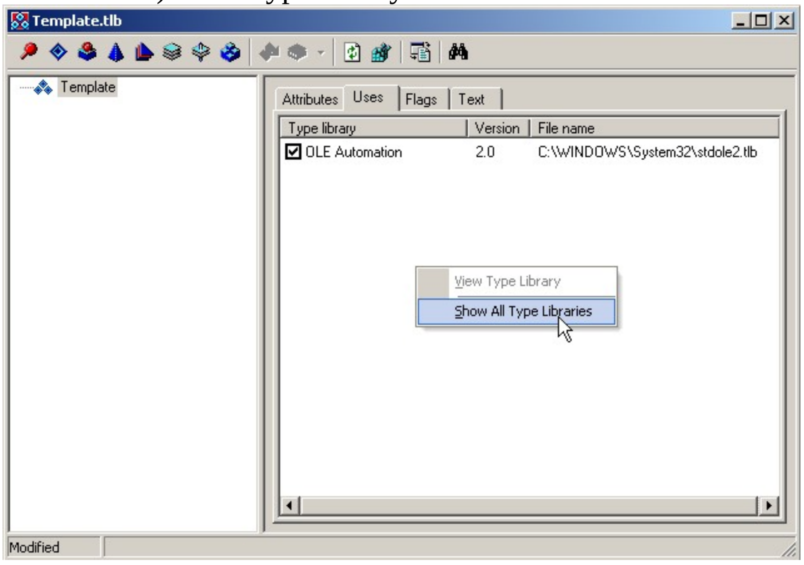


Figure 13 New Type Library

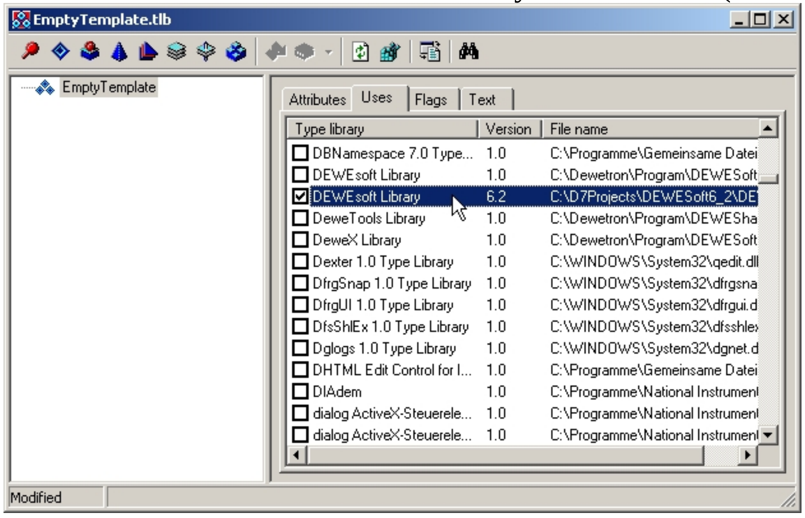
Within the type library window switch to the “Uses” tab. Only the "OLE Automation" type library is listed and checked. Select "Show all type libraries" from the context menu (right mouse click) of the type library list.



Type library	Version	File name
☑ OLE Automation	2.0	C:\WINDOWS\System32\stdole2.tlb

Figure 14 Type Library, Uses Tab

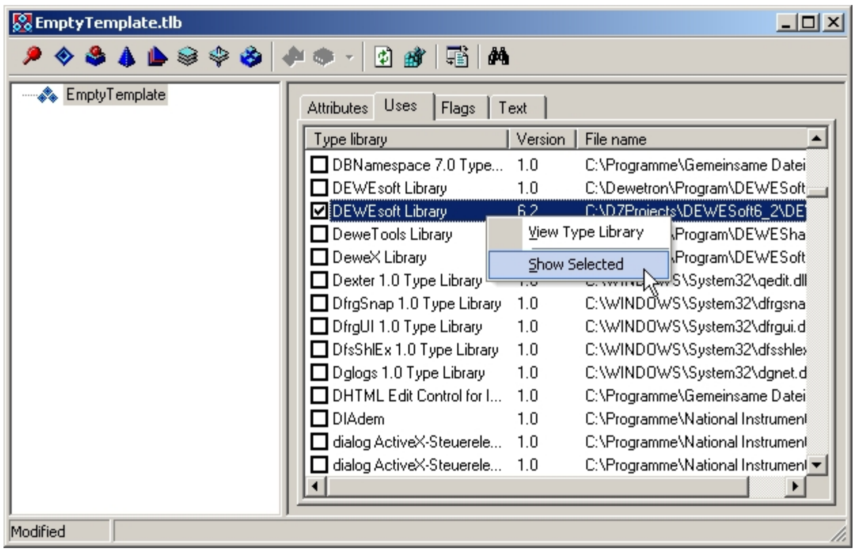
Scroll to the latest version of the "DEWEsoft library" and check it. Since DEWESoft version 6.2b43 the DEWESoft library version 6.2 is available. Plug-ins created with earlier versions of the library (1.0) will not work with newer versions of DEWESoft. Since DEWESoft version 6.3.x/6.4x the DEWESoft library version 630.6 (276.6 in Hex notation) is available.



Type library	Version	File name
☐ DBNamespace 7.0 Type...	1.0	C:\Programme\Gemeinsame Datei
☐ DEWEsoft Library	1.0	C:\Dewetron\Program\DEWESoft
☑ DEWEsoft Library	6.2	C:\D7Projects\DEWESoft6_2\DE
☐ DeweTools Library	1.0	C:\Dewetron\Program\DEWESha
☐ DeweX Library	1.0	C:\Dewetron\Program\DEWESoft
☐ Dexter 1.0 Type Library	1.0	C:\WINDOWS\System32\qedit.dll
☐ DfrgSnap 1.0 Type Library	1.0	C:\WINDOWS\System32\dfrgsna
☐ DfrgUI 1.0 Type Library	1.0	C:\WINDOWS\System32\dfrgui.d
☐ DfsShlEx 1.0 Type Library	1.0	C:\WINDOWS\System32\dfsshlex
☐ Dglogs 1.0 Type Library	1.0	C:\WINDOWS\System32\dgnet.d
☐ DHTML Edit Control for I...	1.0	C:\Programme\Gemeinsame Datei
☐ DIAdem	1.0	C:\Programme\National Instrument
☐ dialog ActiveX-Steuerele...	1.0	C:\Programme\National Instrument
☐ dialog ActiveX-Steuerele...	1.0	C:\Programme\National Instrument

Figure 15 Type Library, Select DEWEsoft Library

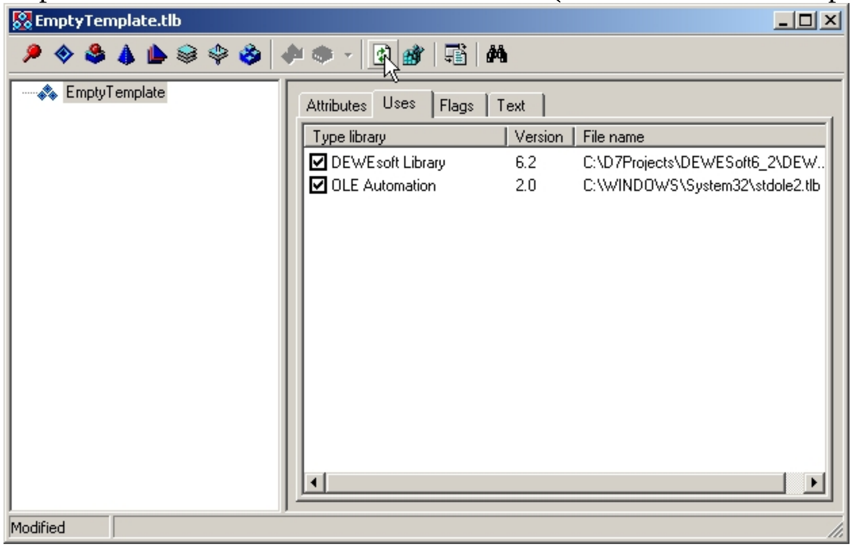
Click "Show Selected" from the context menu.



Type library	Version	File name
☐ DBNamespace 7.0 Type...	1.0	C:\Programme\Gemeinsame Datei
☐ DEWEsoft Library	1.0	C:\Dewetron\Program\DEWESoft
☑ DEWEsoft Library	6.2	C:\D7Projects\DEWESoft6_2\DE
☐ DeweTools Library		Program\DEWESha
☐ DeweX Library		Program\DEWESoft
☐ Dexter 1.0 Type Library		S\System32\qedit.dll
☐ DfrgSnap 1.0 Type Library	1.0	C:\WINDOWS\System32\dfrgsna
☐ DfrgUI 1.0 Type Library	1.0	C:\WINDOWS\System32\dfrgui.d
☐ DfsShlEx 1.0 Type Library	1.0	C:\WINDOWS\System32\dfsshlex
☐ Dglogs 1.0 Type Library	1.0	C:\WINDOWS\System32\dgnet.d
☐ DHTML Edit Control for I...	1.0	C:\Programme\Gemeinsame Datei
☐ DIAdem	1.0	C:\Programme\National Instrument
☐ dialog ActiveX-Steuerele...	1.0	C:\Programme\National Instrument
☐ dialog ActiveX-Steuerele...	1.0	C:\Programme\National Instrument

Figure 16 Type Library, Show Selected

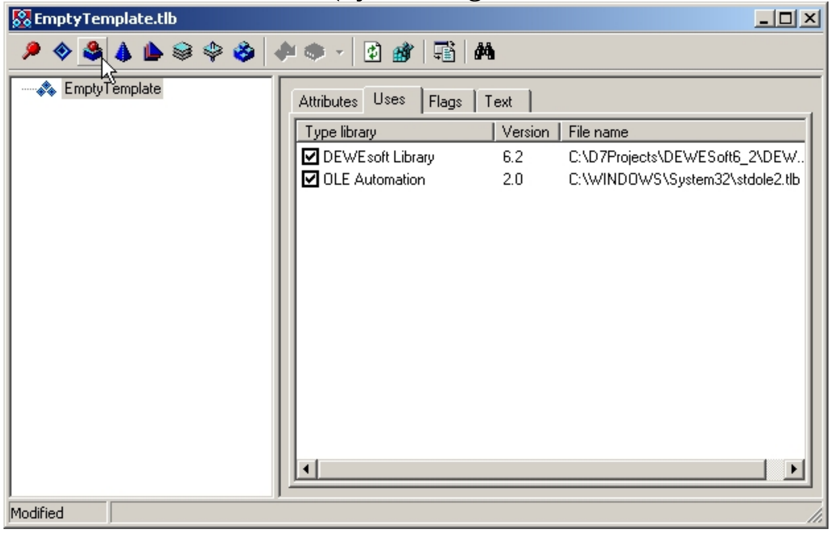
Now you should see just the two necessarily selected type libraries. Click the “Refresh Implementation” button from the tool bar. (See mouse cursor in picture below.)



Type library	Version	File name
☑ DEWEsoft Library	6.2	C:\D7Projects\DEWESoft6_2\DEW..
☑ OLE Automation	2.0	C:\WINDOWS\System32\stdole2.tlb

Figure 17 Type Library, “Refresh Implementation”

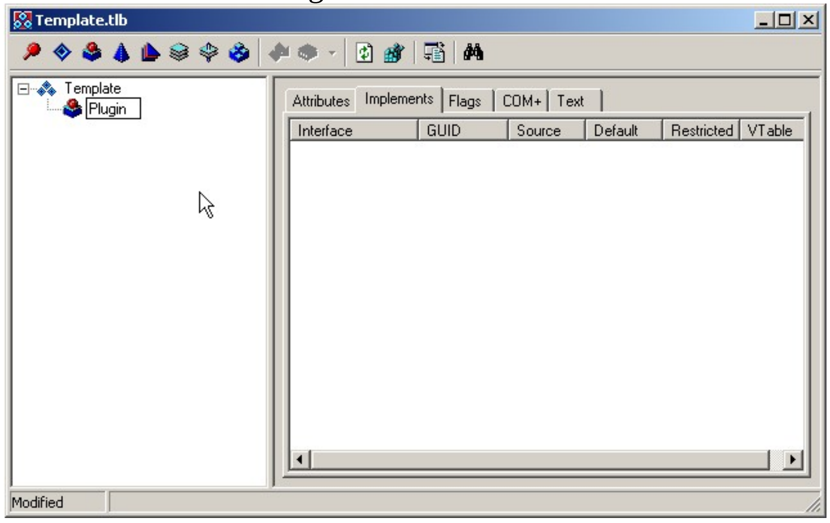
Create a “New CoClass” (by clicking the third icon of the tool bar)…



Type library	Version	File name
☑ DEWEsoft Library	6.2	C:\D7Projects\DEWESoft6_2\DEW..
☑ OLE Automation	2.0	C:\WINDOWS\System32\stdole2.tlb

Figure 18 Type Library, New CoClass

…and rename it to “Plugin”.



Interface	GUID	Source	Default	Restricted	VTable

Figure 19 Type Library, Rename CoClass

Select “Insert Interface” from the context menu of the “Implements” tab…

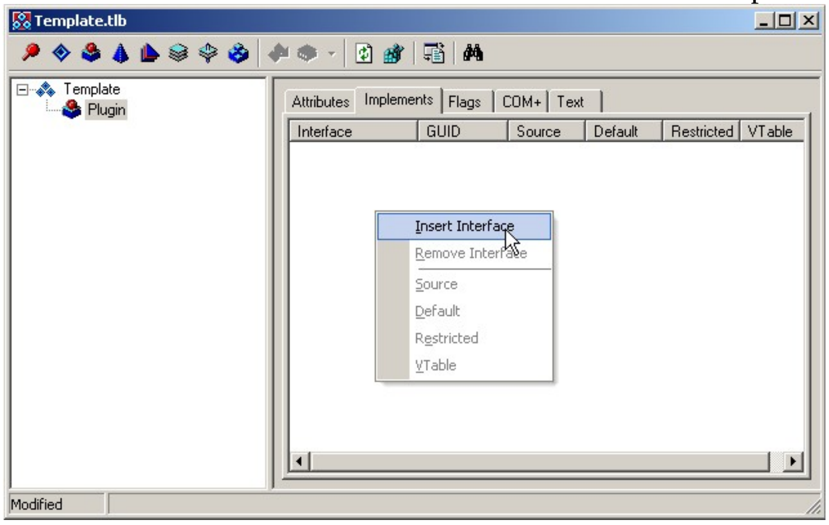


Figure 20 Type Library, Insert Interface

…and choose “IPlugin2” from the “Insert Interface” dialog. (You may choose both, “IPlugin2” and “IPlugin3” with latest versions of DEWESoft.)

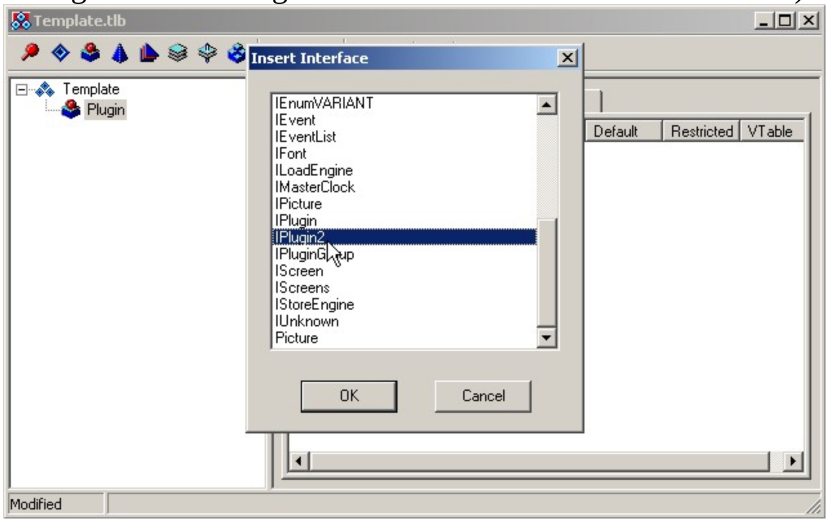
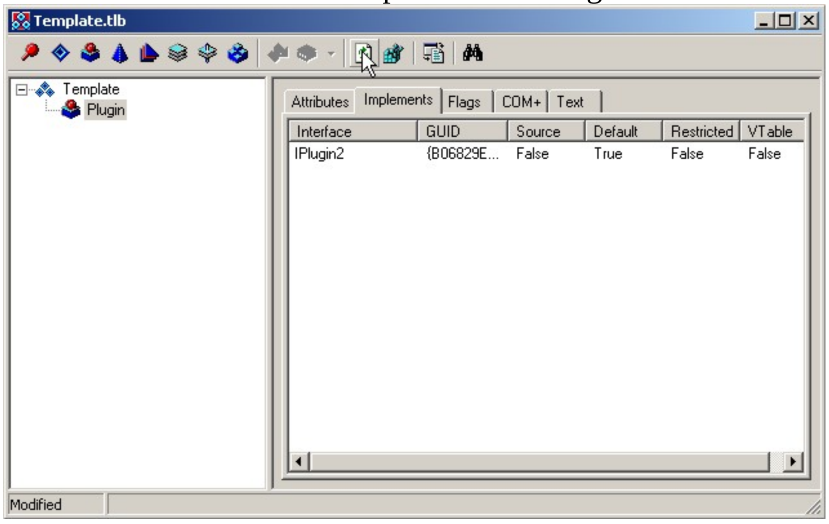


Figure 21 TypeLibrary, Insert Interface "IPlugin2"

Afterwards click “Refresh Implementation” again.



Interface	GUID	Source	Default	Restricted	VTable
IPlugin2	{B06829E...	False	True	False	False

Figure 22 Type Library, "Refresh Implementation"

Now save the type library as e.g. “Template.tlb” (choose a name which is suitable for your application).

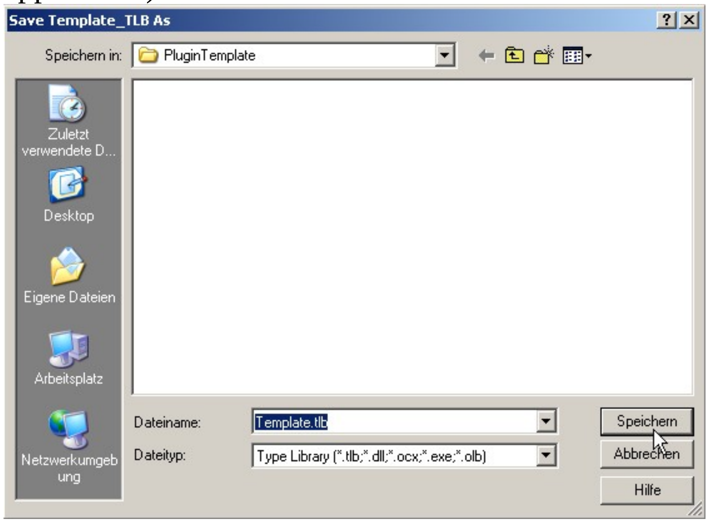
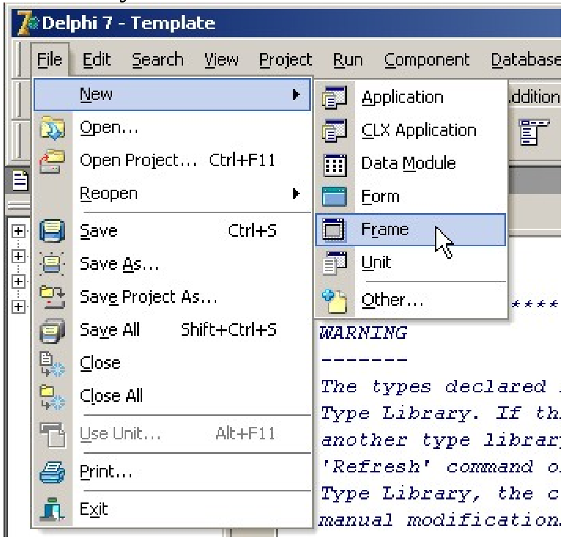


Figure 23 Save Type Library as...

Next step is to add a “frame” by clicking “File >> New >> Frame”… (For some applications it might be useful to choose a Form instead of a Frame. If a Form is used, its property BorderStyle has to be set to bsNone via the Object Inspector.)



```
WARNING
-------
The types declared
Type Library. If th
another type librar
'Refresh' command o
Type Library, the c
manual modification
```

Figure 24 New Frame

…and to save the frame as e.g. “TemplateFrame.pas” (choose a name suitable for your application).

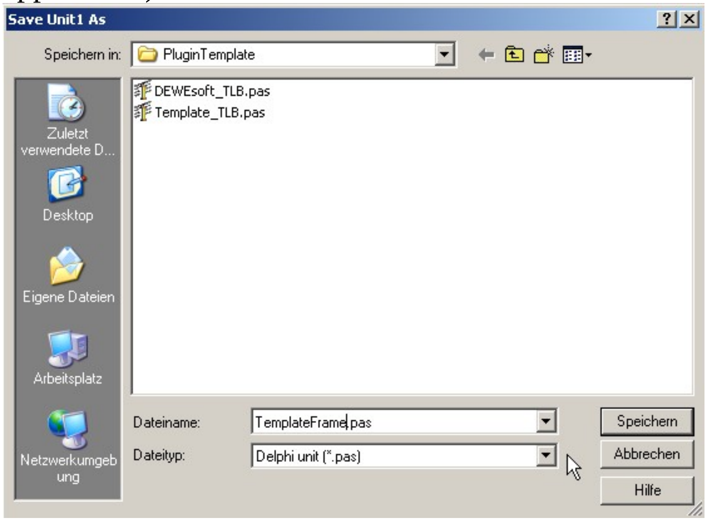


Figure 25 Save Frame as...

Now you may rename the frame to e.g. “SetupFrame” by altering the Name property in the Object Inspector.

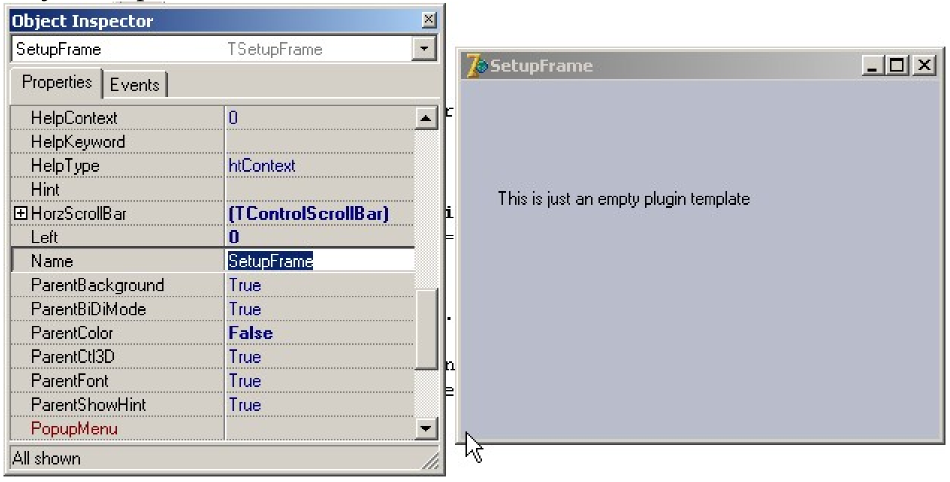
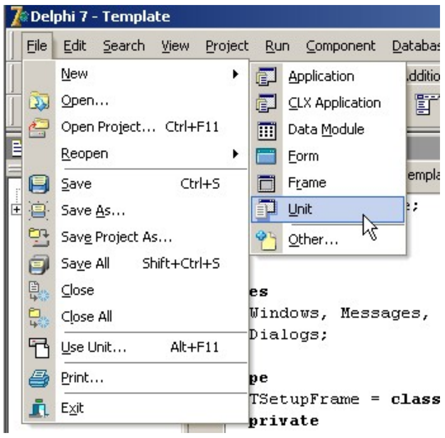


Figure 26 Rename Frame

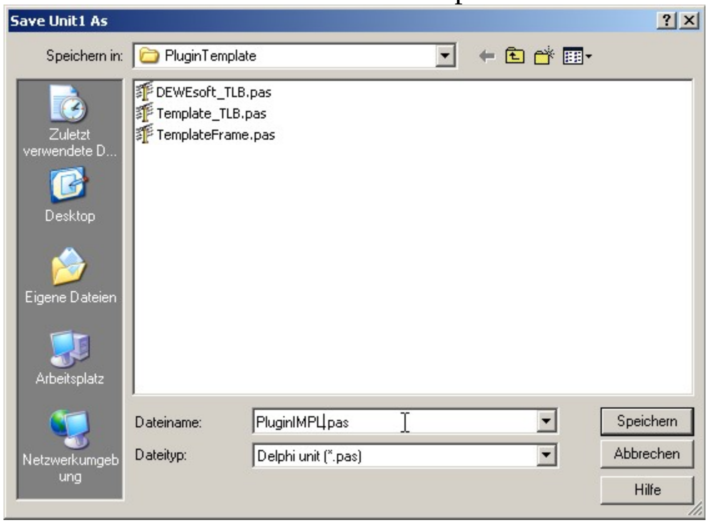
Now click “File >> New >> Unit” to add a new unit to the project.



```
es
Windows, Messages,
Dialogs;
pe
TSetupFrame = class
private
```

Figure 27 New Unit

Save this unit which is the main implementation of the plug-in, as e.g. "PluginIMPL"

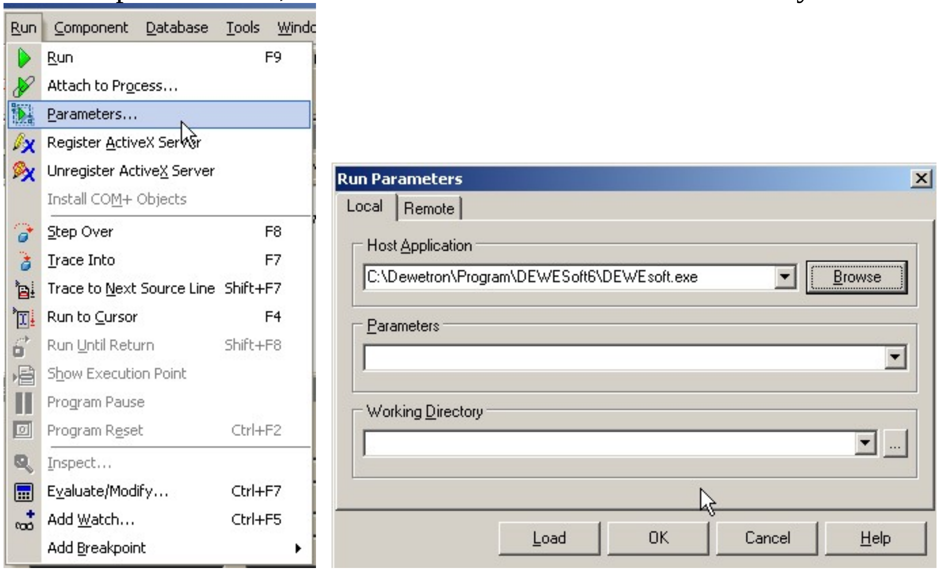


DEWEsoft_TLB.pas
Template_TLB.pas
TemplateFrame.pas

Dateiname: PluginIMPL.pas
Dateityp: Delphi unit (*.pas)

Figure 28 Save Unit as...

In order to be able to debug the application during development, specify DEWEsoft.exe as the "Host Application". Click Run >>Parameters… and click the browse button within the "Run Parameters" dialog to select the path to DEWESoft. (In most cases this will be "C:\Dewetron\Program\DEWESoft6\DEWEsoft.exe".) Every time the debugging of the plug-in in Delphi is started, DEWESoft will be started automatically.



Host Application: C:\Dewetron\Program\DEWESoft6\DEWEsoft.exe

Figure 29 Setting the Host Application

After having completed these steps described above, some standard code has to be added. Each of the functions which can be called by DEWESoft has to be implemented in the plug-in.

5.2 EmptyTemplate Delphi 7 Source Code

The following listing contains the fundamental source code lines which will be necessary in any plug-in developed in Delphi 7.

5.2.1 Source Code of the Project EmptyTemplate.dpr

```
library EmptyTemplate;

uses
  ComServ,
  EmptyTemplate_TLB in 'EmptyTemplate_TLB.pas',
  EmptyTemplateIMPL in 'EmptyTemplateIMPL.pas',
  EmptyTemplateForm in 'EmptyTemplateForm.pas' {SetupForm},
  DEWEsoft_TLB in 'DEWEsoft_TLB.pas';

exports
  DllGetClassObject,
  DllCanUnloadNow,
  DllRegisterServer,
  DllUnregisterServer;

{$R *.TLB}

{$R *.RES}

begin
end.
```

5.2.2 Source code of EmptyTemplateIMPL.pas:

```
unit EmptyTemplateIMPL;

interface

uses ComObj, Dewesoft_TLB, Forms, EmptyTemplate_TLB, Controls, SysUtils;


type
  TPlugin = class(TAutoObject, IPlugin2, IPlugin3)
  private
    FForm: TForm;
    FApp: IApp;
    FUsed: Boolean;
  protected
    //IPlugin2 -->
    function Get_Id: WideString; safecall;
    function Get_Used: WordBool; safecall;
    function ShowFrame(Parent: Integer): WordBool; safecall;
    procedure ClearChannelsInstance; safecall;
    procedure Configure; safecall;
    procedure HideFrame; safecall;
    procedure Initiate(const DeweApp: IApp); safecall;
    procedure LoadSetup(Data: OleVariant); safecall;
    procedure NewSetup; safecall;
    procedure OnGetData; safecall;
    procedure OnStartAcq; safecall;
    procedure OnStartStoring; safecall;
    procedure OnStopAcq; safecall;
    procedure OnStopStoring; safecall;
    procedure OnTrigger(Time: Double); safecall;
    procedure SaveSetup(var Data: OleVariant); safecall;
    procedure Set_Used(Value: WordBool); safecall;
    procedure UpdateFrame; safecall;
    procedure OnOleMsg(Msg: Integer; Param: Integer); safecall;
    //<-- IPlugin2

    //IPlugin3 -->
    procedure OnStartSetup; safecall;
    procedure OnStopSetup; safecall;
    procedure OnBeforeStartAcq(var AllowStart: WordBool); safecall;
    procedure OnAfterStartAcq; safecall;
    procedure OnBeforeStopAcq(var AllowStop: WordBool); safecall;
    procedure OnAfterStopAcq; safecall;
    procedure OnRepaintFrame; safecall;
    procedure OnTriggerStop(Time: Double; TrigDuration: Double); safecall;
    procedure OnAfterCalcMath; safecall;
    function GetDWTypeLibVersion: Integer; safecall;
    procedure OnGetSetupData; safecall;
    procedure OnResizeFrame(Width: Integer; Height: Integer); safecall;
    procedure ProvidesClock(var Value: WordBool); safecall;
    procedure OnGetClock(var ClockLow: Integer; var ClockHigh: Integer); safecall;
    procedure SetCANPort(Port: Integer); safecall;
    procedure OnAlarm(CondIndex: Integer; Status: WordBool); safecall;
    procedure OnBigListLoad(const TextSetup: WideString); safecall;
    procedure OnExit; safecall;
    //<-- IPlugin3

  public
    procedure Initialize; override;
    destructor Destroy; override;
  end;

  TPluginObjectFactory = class(TAutoObjectFactory)
  public
    procedure UpdateRegistry(Register: Boolean); override;
  end;

implementation

uses ComServ, Registry, Windows, EmptyTemplateForm, StdCtrls, Variants;

{ TPluginObjectFactory }
procedure TPluginObjectFactory.UpdateRegistry(Register: Boolean);
var
  Reg: TRegistry;
begin
  inherited UpdateRegistry(Register);
  if Register then
  begin
    Reg := TRegistry.Create(KEY_ALL_ACCESS);
    Reg.RootKey := HKEY_LOCAL_MACHINE;
    Reg.OpenKey('Software\Dewesoft\Plugins\EmptyTemplate', True);
    Reg.WriteString('GUID', GUIDToString(CLASS_Plugin));
    Reg.WriteString('Name', 'EmptyTemplate');
    Reg.WriteString('Version', 'beta');
    Reg.WriteString('Description', 'Just an empty template...');
    Reg.WriteString('Vendor', 'Dewesoft');
    Reg.WriteInteger('TLB', DEWEsoftMinorVersion);
    Reg.Free;
  end else
  begin
    Reg := TRegistry.Create(KEY_ALL_ACCESS);
    Reg.RootKey := HKEY_LOCAL_MACHINE;
    Reg.DeleteKey('Software\Dewesoft\Plugins\EmptyTemplate');
    Reg.Free;
  end;
end;

function TPlugin.Get_Id: WideString;
begin
  Result := GUIDToString(CLASS_Plugin);
end;

function TPlugin.Get_Used: WordBool;
begin
  Result := FUsed;
end;

function TPlugin.ShowFrame(Parent: Integer): WordBool;
begin
  FForm := TSetupForm.Create(nil, Parent);
  FForm.ParentWindow := HWND(Parent);
  FForm.Align := alClient;
  TSetupForm(FForm).Plugin := Self;
  Result := True;
end;

procedure TPlugin.ClearChannelsInstance;
```

```
begin

end;

procedure TPlugin.Configure;
begin

end;

procedure TPlugin.HideFrame;
begin
  if FForm <> nil then
  begin
    FForm.Visible := False;
    FForm.ParentWindow := 0;
  end;
  FreeAndNil(FForm);
end;

procedure TPlugin.Initiate(const DeweApp: IApp);
begin
  FApp := DeweApp;
end;

procedure TPlugin.LoadSetup(Data: OleVariant);
begin
//  Mount Channels here, if necessary!
end;

procedure TPlugin.NewSetup;
begin
//  Mount Channels here, if necessary!
end;

procedure TPlugin.OnGetData;
begin

end;

procedure TPlugin.OnStartAcq;
begin

end;

procedure TPlugin.OnStartStoring;
begin

end;

procedure TPlugin.OnStopAcq;
begin

end;

procedure TPlugin.OnStopStoring;
begin

end;

procedure TPlugin.OnTrigger(Time: Double);
begin

end;

procedure TPlugin.SaveSetup(var Data: OleVariant);
begin

end;

procedure TPlugin.Set_Used(Value: WordBool);
```

```
begin
  FUsed:= Value;
end;

procedure TPlugin.UpdateFrame;
begin

end;

destructor TPlugin.Destroy;
begin
  inherited;
end;

procedure TPlugin.OnOleMsg(Msg, Param: Integer);
begin

end;

function TPlugin.GetDWTypeLibVersion: Integer;
begin
  Result:= DEWEsoftMinorVersion;
end;

procedure TPlugin.OnAfterCalcMath;
begin

end;

procedure TPlugin.OnAfterStartAcq;
begin

end;

procedure TPlugin.OnAfterStopAcq;
begin

end;

procedure TPlugin.OnBeforeStartAcq(var AllowStart: WordBool);
begin

end;

procedure TPlugin.OnBeforeStopAcq(var AllowStop: WordBool);
begin

end;

procedure TPlugin.OnGetClock(var ClockLow, ClockHigh: Integer);
begin

end;

procedure TPlugin.OnGetSetupData;
begin

end;

procedure TPlugin.OnRepaintFrame;
begin

end;

procedure TPlugin.OnResizeFrame(Width, Height: Integer);
begin
  if Assigned(FForm) then
  begin
    FForm.Width := Width;
    FForm.Height := Height;
  end;
end;

procedure TPlugin.OnStartSetup;
begin

end;

procedure TPlugin.OnStopSetup;
begin

end;

procedure TPlugin.OnTriggerStop(Time, TrigDuration: Double);
begin

end;

procedure TPlugin.ProvidesClock(var Value: WordBool);
begin

end;

procedure TPlugin.SetCANPort(Port: Integer);
begin

end;

procedure TPlugin.OnAlarm(CondIndex: Integer; Status: WordBool);
begin

end;

procedure TPlugin.OnBigListLoad(const TextSetup: WideString);
begin

end;

procedure TPlugin.OnExit;
begin

end;

procedure TPlugin.Initialize;
begin
  inherited;
end;

initialization
  TPluginObjectFactory.Create(ComServer, TPlugin, Class_Plugin,
    ciMultiInstance, tmApartment);
end.
```

5.2.3 Source code of EmptyTemplateForm.pas:

```
unit EmptyTemplateForm;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, EmptyTemplateIMPL;

type
  TSetupForm = class(TForm)
    Label1: TLabel;
  private
```

```
    FPlugin: TPlugin;
    FParentWindow: THandle;
  protected
    procedure CreateParams(var Params: TCreateParams); override;
    procedure CreateWnd; override;
  public
    property Plugin: TPlugin read FPlugin write FPlugin;
    constructor Create(AOwner: TComponent; ParentWin: THandle);
  end;

implementation

{$R *.dfm}
procedure TSetupForm.CreateParams(var Params: TCreateParams);
begin
  inherited;
  if (Parent = nil) and IsLibrary and
     not (csDestroying in ComponentState) then
     Params.WndParent := FParentWindow;
end;

constructor TSetupForm.Create(AOwner: TComponent; ParentWin: THandle);
begin
  FParentWindow := ParentWin;
  inherited Create(AOwner);

end;

procedure TSetupForm.CreateWnd;
begin
  inherited;
end;

end.
```

5.3 Registering a Delphi Plug-in

Plug-ins have to be registered to the Windows-Registry in order to be recognized by DEWESoft on startup. Therefore the parameters specified in the procedure `“TPluginObjectFactory.UpdateRegistry”` of “EmptyTemplateIMPL.pas” are necessary.

Note: Registering plug-ins designed in Delphi having the registration-information properly included in the code will be registered automatically by DEWESoft version 6.3x and newer on start-up and it is necessary that plug-in reside in the subdirectory “Addons” within the DEWESoft installation folder.

In conjunction with DEWESoft version 6.2x, the registration has to be done manually as follows. It can be done within Delphi by clicking “Run >> Register ActiveX Server”. Another way to register the Plug-in would be by using regsvr32 from the command line window. To do so, `regsvr32 EmptyPlugin.dll` has to be called at the path where the plug-in resides. In order to remove the plug-in parameters from the registry, the following line can be used: `regsvr32 /u EmptyPlugin.dll`. Another way to un-register the plug-in is to click “Run >> Unregister ActiveX Server” within Delphi.

6 CustomExport

6.1 General Information on CustomExport

CustomExport offers the ability to create user-defined file exports for DEWESoft. Such a CustomExport will be added to the list of exports within DEWESoft.

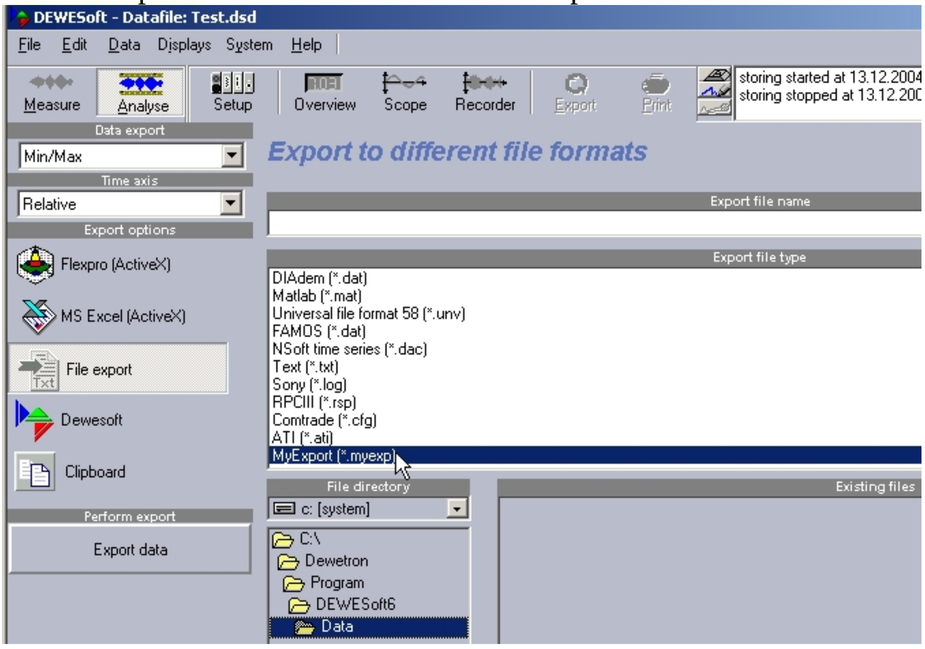


Figure 30 File Export – Export file types

A CustomExport is an ActiveX library having the extension ".exp". If a CustomExport library resides in the subdirectory "Addons" of the DEWESoft installation forlder, it will be recognized on start-up.

6.2 Creating a CustomExport in Delphi 7

For creating a new CustomExport in Delphi 7 the following steps have to be done:
First, create a new ActiveX Library Project by clicking "File >> New >> Other…" (See Figure 31)

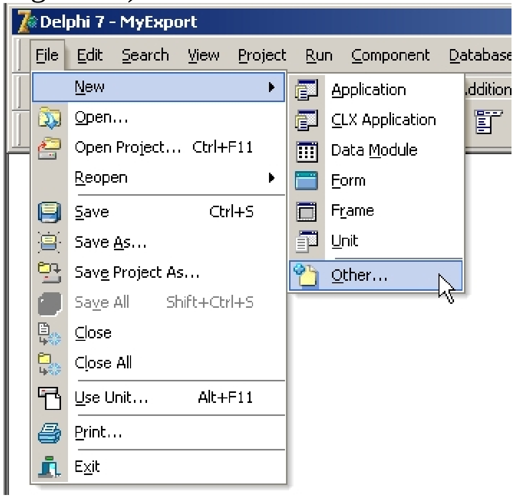


Figure 31 File >> New >> Other...

…and select ActiveX Library as shown in Figure 32.

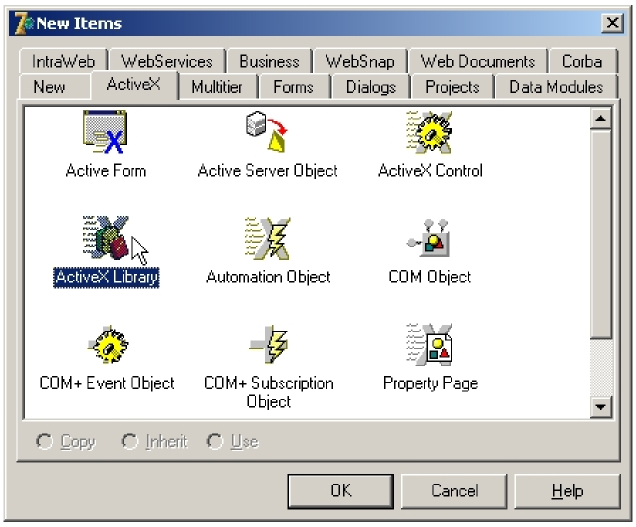


Figure 32 New ActiveX Library

Now save the ActiveX Library Project as e.g. MyExport.dpr (Figure 33)

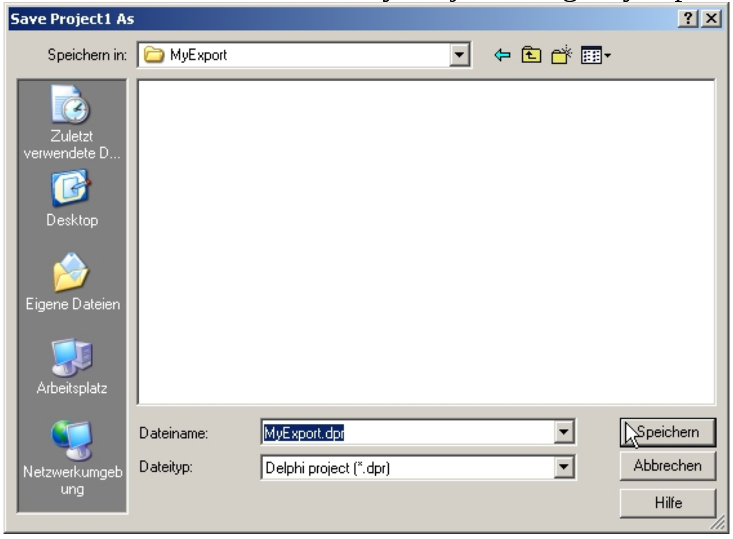


Figure 33 Save Project As...

Add a new Unit to the project by clicking “File >> New >> Unit” (Figure 34)

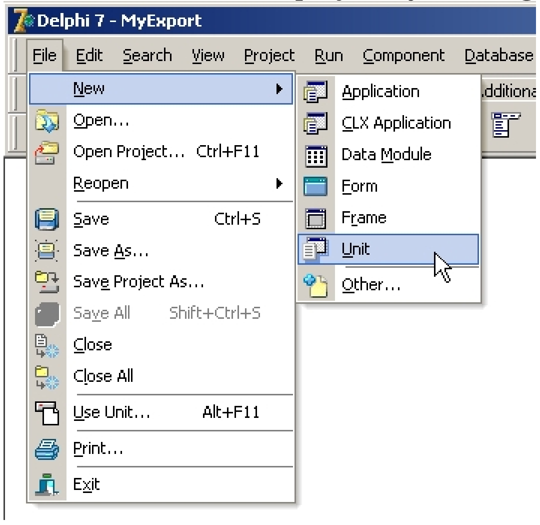
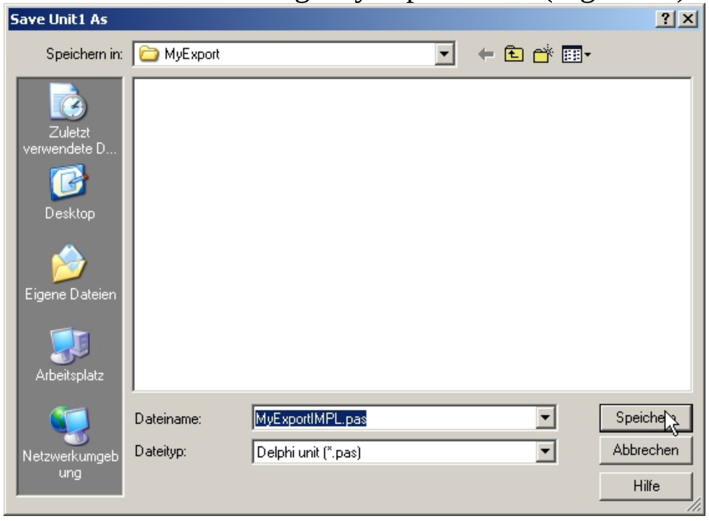


Figure 34 File >> New >> Unit

And save the unit as e.g. MyExportIMPL (Figure 35).



Save Unit1 As

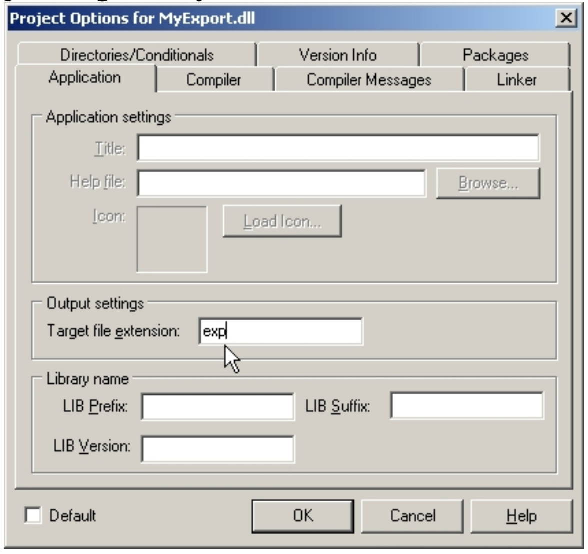
Speichern in: MyExport

Dateiname: MyExportIMPL.pas

Dateityp: Delphi unit (*.pas)

Figure 35 Save Unit As...

The file extension of the ActiveX Library has to be ".exp". This has to be set in the "Project Options" dialog (Figure 36) which is opened by clicking "Project >> Options…" or by pressing the keys <Shift> + <Ctrl> + <F 11>.



Project Options for MyExport.dll

Application settings

Title:

Help file:

Icon:

Output settings

Target file extension: exp

Library name

LIB Prefix:

LIB Suffix:

LIB Version:

Default

Figure 36 Target File Extension

Finally the DEWESoft Library has to be imported by clicking "Project >> Import Type Library…" (Figure 37).

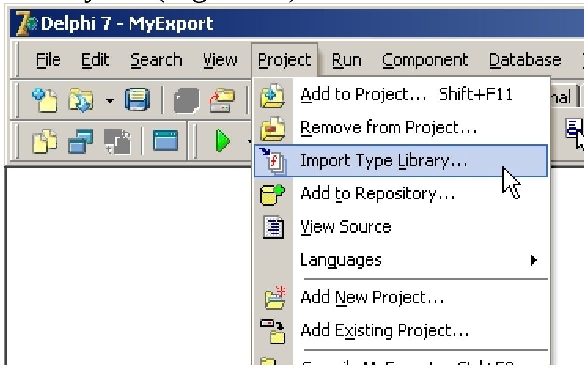
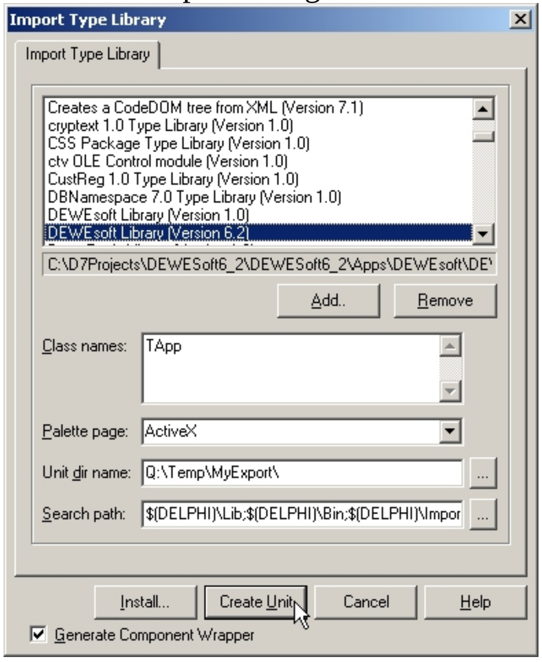


Figure 37 Import Type Library

From the list in the "Import Type Library" dialog choose DEWESoft Library. Change the "Unit dir name" to the path where your project resides and click "Create Unit". Be carefull to choose the write version of the DEWESoft Library. The version 6.2 will work DEWESoft Version 6.2b43 or newer. The older version of the library (1.0) will only work with older versions of DEWESoft. If DEWESoft 6.3 is installed on the computer, the version number will be something like “276.6”. The part left to the dot is the “DEWESoftMajorVersion” in Hex notation. So, this example would be of DEWESoft version 630 in decimal notation. The part right to the dot is the “DEWESoftMinorVersion” which is the version of DEWESoft’s type library. This version number is shown in the “Plugin” section of the DEWESoft’s hardware setup at the right bottom.



Import Type Library

Creates a CodeDOM tree from XML (Version 7.1)
cryptext 1.0 Type Library (Version 1.0)
CSS Package Type Library (Version 1.0)
ctv OLE Control module (Version 1.0)
CustReg 1.0 Type Library (Version 1.0)
DBNamespace 7.0 Type Library (Version 1.0)
DEWEsoft Library (Version 1.0)
DEWEsoft Library (Version 6.2)

C:\D7Projects\DEWESoft6_2\DEWESoft6_2\Apps\DEWEsoft\DE

Class names: TApp

Palette page: ActiveX

Unit dir name: Q:\Temp\MyExport\

Search path: $(DELPHI)\Lib;$(DELPHI)\Bin;$(DELPHI)\Impor

Figure 38 Import Type Library – Create Unit

Since version 6.4 of DEWESoft, an additional interface for embedding a configuration frame to DEWESoft is available. If this functionality is used, a Frame or Form has to be added to the project.

6.3 ICustomExport, ICustomExport2, IExportFrame

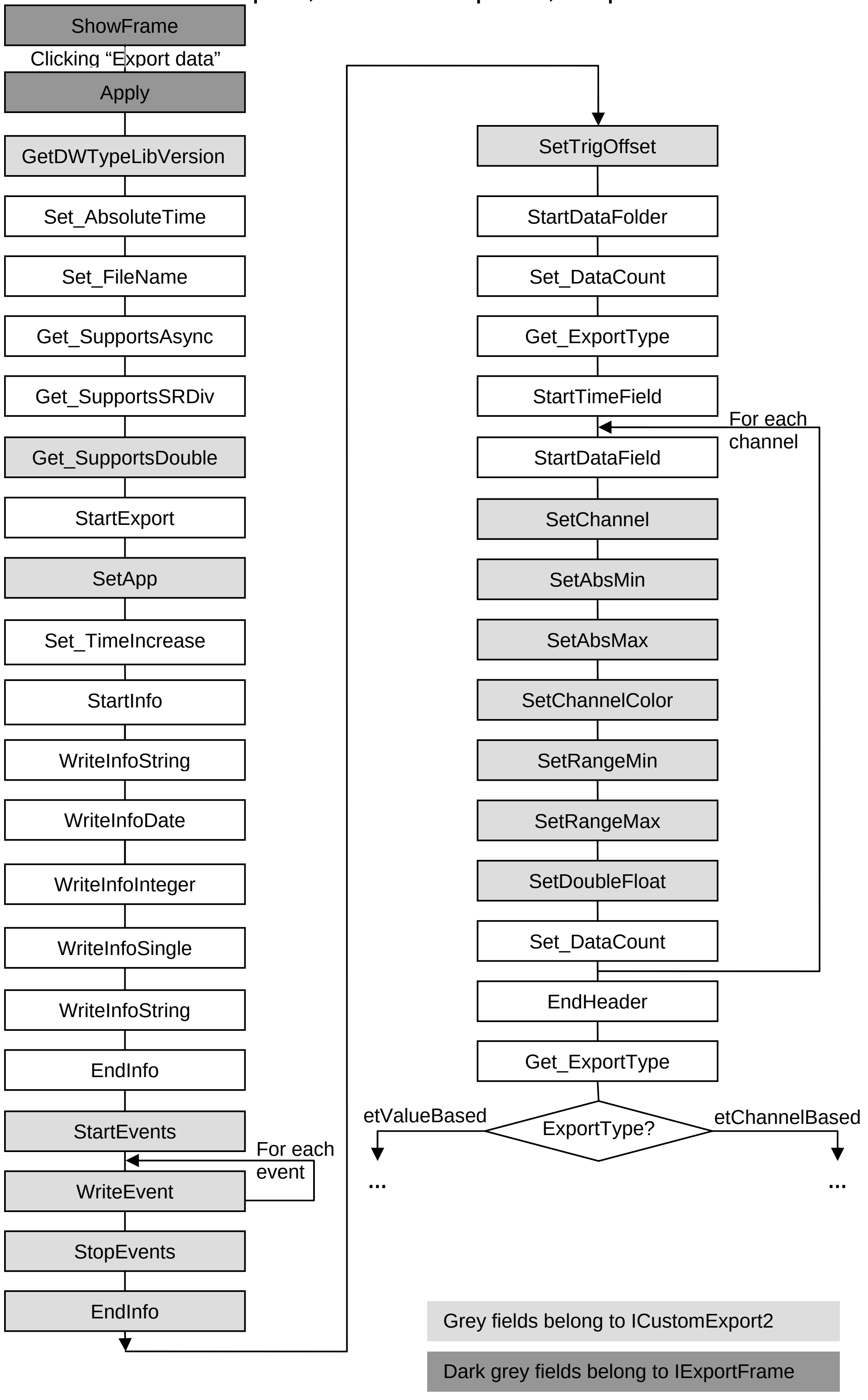


Figure 39 CustomExport Flowchart (1)

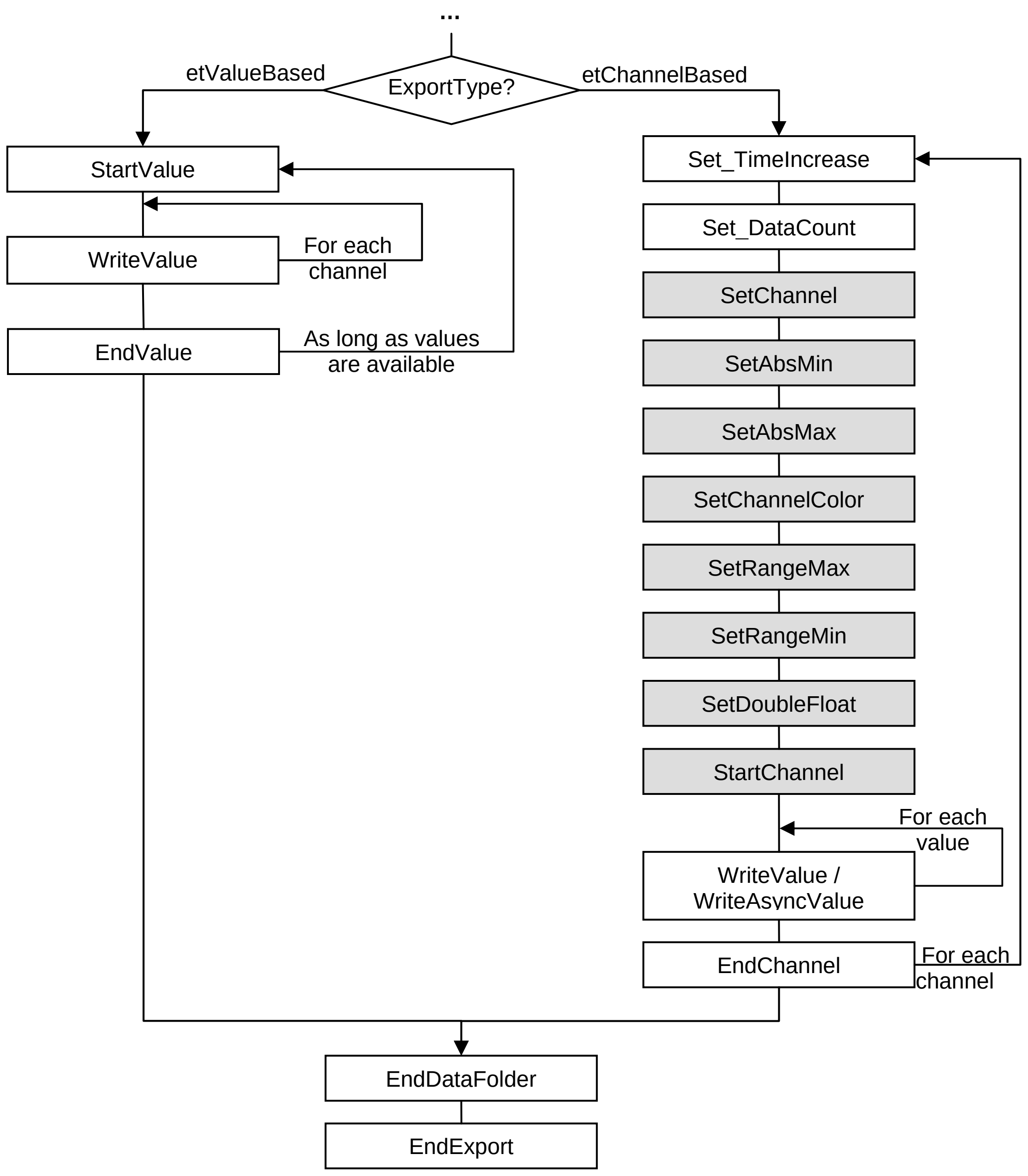


Figure 40 CustomExport Flowchart (2)

6.4 ICustomExport

6.4.1 Methods of ICustomExport

function EndChannel: HResult;
EndChannel is called at the end of the export of each channel when the ExportType is set to etChannelBased.

function EndDataFolder: HResult;
EndDataFolder is called at the end of each data folder. A data folder is a section of time where data is stored; in other words the time between a start and a stop trigger.

function EndExport: HResult;
EndExport is called at the very end of the export. E.g. DLLs used during export can be released here.

function EndHeader: HResult;
EndHeader is called at the end of writing the header information of an export and before the export of the measurement data is started.

function EndInfo: HResult;
EndInfo is called after writing the general information to the export file (functions WriteInfoDate, WriteInfoInteger, WriteInfoSingle, and WriteInfoString). In most cases EndInfo will remain empty.

function EndValue: HResult;
EndValue is called, when the ExportType is etValueBased, after one value of each channel is exported.

function StartAbsValue(DateTime: TDateTime): HResult;
StartAbsValue is called when the ExportType is set to etValueBased and the “Time axis” is set to “Absolute” before one value of each channel is exported.
Parameters:
DateTime is the time stamp in absolute time for one set values.

function StartChannel(const FieldName: WideString; const FieldUnit: WideString; Async: WordBool): HResult;
StartChannel is called when ExportType is set to channel-based every time the export of a new channel is started.
Parameters:
FieldName is the name and the comment of the channel. E.g. this could be “AI 0 - MyComment” for an analog channel.
FieldUnit is the Unit of the channel. E.g. this could be “m/s^2” or just “-“, if no unit was specified for the channel.
Async specifies whether the channel contains asynchronous data or not.

function StartDataField(const FieldName: WideString; const FieldUnit: WideString; ExportRate: Integer): HResult;
StartDataField is called for each channel for exporting the name, the unit and the rate for each channel.
Parameters:
FieldName is the name and the comment of the channel. E.g. this could be “AI 0 - MyComment” for an analog channel.
FieldUnit is the unit of a channel. E.g. this could be “m/s^2” or just “-“, if no unit was specified for the channel.
ExportRate equals to the acquisition rate of a channel.

function StartDataFolder(const FolderName: WideString; AbsTime: TDateTime): HResult;
StartDataFolder is called when the export of channel-specific data starts. A data folder is a section of time where data is stored; in other words the time between a start and a stop trigger.
Parameters:
FolderName is the name of the DataFolder. E.g. this can be “Data1”.
AbsTime is the absolute start time of the folder.

function StartExport(const App: IApp): HResult;
At StartExport DLLs for exporting should be initialized if necessary (depending on the type of export) and the file to which data will be exported should be opened here.
Parameters:
App is a reference to DEWESoft for accessing its functions and properties.

function StartInfo(const Info: WideString): HResult;
At StartInfo the writing of general information of the exported data begins. The functions WriteInfoDate, WriteInfoInteger, WriteInfoSingle, and WriteInfoString will follow.
Parameters:
Info can contain any general header information.

function StartTimeField(const FieldName: WideString; const FieldUnit: WideString): HResult;
StartTimeField is called for exporting the name of the time channel and its unit.
Parameters:
FieldName is the name of the time channel. In most cases this will be just “Time”.
FieldUnit is the time unit. E.g. this could be “s” for seconds.

function StartValue(TimeValue: Double): HResult;
StartValue is called when the ExportType is set to etValueBased and the “Time axis” is set to “Relative” before one value of each channel is exported.
Parameters:
TimeValue is the time value for the following set of values to export.

function WriteAsyncValue(TimeStamp: Double; Value: Single): HResult;
WriteAsyncValues is called for each asynchronous value when the ExportType is channel-based.
TimeStamp is the asynchronous time value corresponding to the value to export.
Value is data value to export.

function WriteInfoDate(const Description: WideString; Value: TDateTime): HResult;
WriteInfoDate writes date information to the header of the export file.
Parameters:
Description is a string describing the value. E.g. this string could be something like “Start time”.
Value is the appropriate time and date.

function WriteInfoInteger(const Description: WideString; Param: Integer): HResult;
WriteInfoInteger writes information of the type Integer to the header of the export file.
Parameters:
Description is a string describing the value. E.g. this string could be “Sample rate”.
Param is the appropriate Integer value.

function WriteInfoSingle(const Description: WideString; Value: Single): HResult;
WriteInfoSingle writes information of the type Single to the header of the export file.
Parameters:
Description is a string describing the value.
Value is the appropriate Single value.

function WriteInfoString(const Description: WideString; const Value: WideString): HResult;
WriteInfoString writes information of the type string to the header of the export file.
Parameters:
Description is a string describing the value.
Value is the appropriate string.

function WriteValue(Value: Single): HResult;
WriteValue is called for each value to export, when the ExportType is value-based or when synchronous values are exported in channel-based mode. (For asynchronous values in channel-based mode, WriteAsyncValue is called.)
Parameters:
Value is the data value to export.

6.4.2 Properties of ICustomExport

property AbsoluteTime: WordBool;
AbsoluteTime specifies whether the export is done in absolute time or not. Cf. Figure 41. This property is set by DEWESoft.

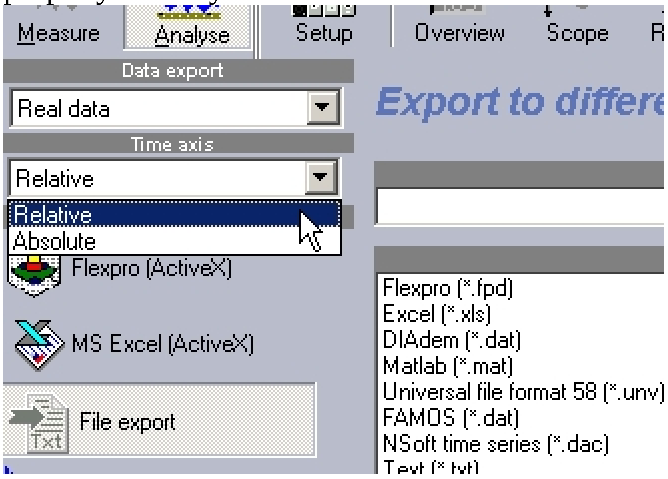


Figure 41 Time Axis for Export

property DataCount: Largeuint;
DataCount specifies the number of samples. This property is set by DEWESoft. For channel-based export the DataCount differs for each channel. For value-based export the DataCount is equal for each channel. This property can be necessary for some exports needing to know that count at the beginning.

property ExportType: ExportTypes; (read-only)
ExportType specifies whether the export is value-based or channel-based. These two types correspond to the constants etValueBased (=0) or etChannelBased (=1). Value-based means, that the first value of each channel is exported before the second value of each channel and so on. Channel-based means that each values of the first channel are exported before each values of the second channel and so on.

property Extension: WideString; (read-only)
Extension specifies the file extension denoting the file type, e.g. ".txt" for a text file.

property FileName: WideString;
FileName is the name of the export file entered by the user before performing the export. This property is set by DEWESoft.

property SupportsAsync: WordBool; (read-only)
SupportsAsync specifies whether the custom export supports asynchronous data or not. This property is relevant only for channel-based export. If SupportsAsync is true, asynchronous channels will be exported as such. If SupportsAsync is false, asynchronous channels will be exported as synchronous channels and gaps between values will be interpolated.

property SupportsSRDiv: WordBool; (read-only)
SupportsSRDiv defines whether the export supports a sample rate divider or not. This property is relevant only for channel-based export. If SupportsSRDiv is false, additional samples will be inserted.

property TimeIncrease: Double;
TimeIncrease defines the interval between two subsequent values of synchronous data.

6.5 ICustomExport2

6.5.1 Methods of ICustomExport2

function GetDWTypeLibVersion(out Value: Integer): HResult;
At GetDWTypeLibVersion the DEWESoft type library version should be returned.
Parameters:
Value is the DEWESoft type library version number. It is defined as DEWESoftMinorVersion in DEWESoft's type library.

function Get_SupportsDouble(out Value: WordBool): HResult;
At Get_SupportsDouble it should be returned whether the custom export supports the export of measurement values of the type Double.
Parameters:
Value defines whether Double values are supported or not.

function SetAbsMax(AbsMax: Single): HResult;
At SetAbsMax the absolute maximum of the measurement is made available to the export. E.g. this can be used for scaling purpose.
Parameters:
AbsMax is the absolute maximum measured value.

function SetAbsMin(AbsMin: Single): HResult;
At SetAbsMin the absolute minimum of the measurement is made available to the export. E.g. this can be used for scaling purpose.
Parameters:
AbsMin is the minimal measured value.

function SetApp(const App: IApp): HResult;
At SetApp a reference to the IApp is provided.

function SetChannel(const Ch: IChannel): HResult;
At SetChannel a reference to current channel is provided.

function SetChannelColor(Color: Integer): HResult;
SetChannelColor gives the color of the channel to the export.
Parameters:
Color is the integer value of the color of a channel.

function SetDoubleFloat(DoubleFloat: WordBool): HResult;
SetDoubleFloat give back to the export whether the values which will be exported are of the type Double of not. If not, its type is Single.
Parameters:
DoubleFloat is True if the values to export are of the type Double.

function SetRangeMax(Value: Single): HResult;
SetRangeMax makes the maximum range setting available to the export.
Parameters:
Value is the maximum limit of the measurement range.

function SetRangeMin(Value: Single): HResult;
SetRangeMin makes the minimum range setting available to the export.
Parameters:
Value is the minimum limit o fthe measurement range.

function SetTrigOffset(TrigTime: Double): HResult;
SetTrigOffset gives back the pre-time of a trigger event. The time given back is not overlapping with previous trigger events.
Parameters:
TrigTime is the pre-time of the trigger event.

function StartEvents: HResult;
StartEvents is called before the events are exported.

function StopEvents: HResult;
StopEvents is called after the events are exported.

function Get_SupportsDouble(out Value: WordBool): HResult;
Get_SupportsDouble defines whether the custom export supports the export of values of the type Double.
Parameters:
Value has to be set to True is the export supports Double values.

function WriteAsyncDoubleValue(TimeStamp: Double; Value: Double): HResult;
WriteAsyncDoubleValue is similar to WriteAsyncValue of ICustomExport.
WriteAsyncDoubleValues is called for each asynchronous value when the ExportType is channel-based.
TimeStamp is the asynchronous time value corresponding to the value to export.
Value is data value to export.

function WriteDoubleValue(Value: Double): HResult;
WriteDoubleValue is similar to WriteValue of ICustomExport. WriteValue is called for each value to export, when the ExportType is value-based or when synchronous values are exported in channel-based mode. (For asynchronous values in channel-based mode, WriteAsyncDoubleValue is called.)
Parameters:

Value is the data value to export.

function WriteEvent(EventType: Integer; const EventTypeString: WideString; Time: Double; const Comment: WideString): HResult;
WriteEvent is called for each event of the exported data.
Parameters:
EventType defines the type of event. This can be etText (=$00000015), etKeyboard (=$00000014);
EventTypeString is a string defining the event type.
Time is the time when the event occurred.
Comment is a possible remark associated with the event.

6.6 IExportFrame

DEWESoft 6.4 and newer offers the possibility of embedding an ExportFrame into its export screen. This frame will appear next to the list of available file exports, cf. Figure 42.

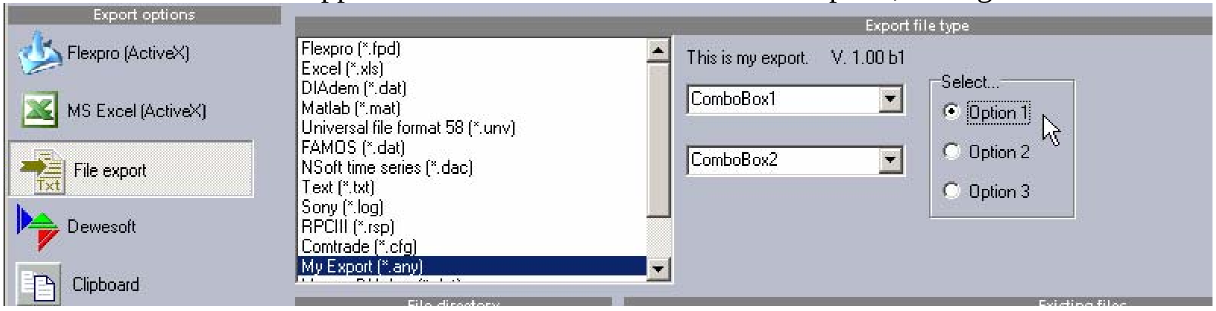


Figure 42 Export Frame in DEWESoft's Export Screen

6.6.1 Methods of IExportFrame

procedure ShowFrame(Handle: Integer; out FrameHeight: Integer); safecall;
ShowFrame is called by DEWESoft when the setup frame of the custom export is shown.
Parameters:
Handle is the handle which should be assigned to the ParentWindow property of the frame.
FrameHeight returns the height of the configuration frame to DEWESoft.

procedure HideFrame; safecall;
HideFrame is called when the custom export is left and its onfiguration frame should be hidden.

procedure Apply; safecall;
Apply is called by DEWESoft when the “Export data” button is clicked. At this moment changes of the exports configuration within the configuration frame can be applied.

6.7 Source Code of the CustomExport

Now, the code of the CustomExport project should look like the following:

```
library MyExport;

uses
  ComServ,
  MyExportFormUnit in 'MyExportFormUnit.pas' {MyExportForm},
  MyExportIMPL in 'MyExportIMPL.pas';

{$E .exp}

exports
```

```
  DllGetClassObject,
  DllCanUnloadNow,
  DllRegisterServer,
  DllUnregisterServer;

{$R *.RES}

begin
end.
```

6.7.1 Source Code of a CustomExport Unit

The following source code should be added to the unit MyExportIMPL.pas. Specific code depending on the export has to be added to the functions.

```
unit MyExportIMPL;

{$WARN SYMBOL_PLATFORM OFF}
interface
uses
  Windows, ActiveX, Classes, ComObj, Dewesoft_TLB, IniFiles, Dialogs,
  StrUtils, Math, SysUtils, Controls, Forms, ClipBrd, Variants;

const
  Version = '1.00 b1';

type

TExport = class(TComObject, ICustomExport, ICustomExport2, IExportFrame)
private
    FForm: TForm;
    FApp: IApp;

    FFileName: TFileName;
    TimeIncrease: Double;
    AbsoluteTime: WordBool;
    DataCount: Largeuint;
protected
    //-->ICustomExport
    function Get_FileName(out Value: WideString): HResult; stdcall;
    function Set_FileName(const Value: WideString): HResult; stdcall;
    function StartExport(const App: IApp): HResult; stdcall;
    function EndExport: HResult; stdcall;
    function Get_ExportType(out Value: ExportTypes): HResult; stdcall;
    function Get_AbsoluteTime(out Value: WordBool): HResult; stdcall;
    function Set_AbsoluteTime(Value: WordBool): HResult; stdcall;
    function StartDataFolder(const FolderName: WideString; AbsTime: TDateTime):
HResult; stdcall;
    function EndDataFolder: HResult; stdcall;
    function StartInfo(const Info: WideString): HResult; stdcall;
    function WriteInfoString(const Description: WideString; const Value:
WideString): HResult; stdcall;
    function WriteInfoInteger(const Description: WideString; Param: Integer):
HResult; stdcall;
    function WriteInfoSingle(const Description: WideString; Value: Single):
HResult; stdcall;
    function WriteInfoDate(const Description: WideString; DateTime: TDateTime):
HResult; stdcall;
    function EndInfo: HResult; stdcall;
    function StartTimeField(const FieldName: WideString; const FieldUnit:
WideString): HResult; stdcall;
    function StartDataField(const FieldName: WideString; const FieldUnit:
WideString; ExportRate: Integer): HResult; stdcall;
    function EndHeader: HResult; stdcall;
    function StartValue(TimeValue: Double): HResult; stdcall;
    function StartAbsValue(DateTime: TDateTime): HResult; stdcall;
    function WriteValue(Value: Single): HResult; stdcall;
    function WriteAsyncValue(TimeStamp: Double; Value: Single): HResult; stdcall;
    function StartChannel(const FieldN: WideString; const FieldUnit: WideString;
Async: WordBool): HResult; stdcall;
    function Get_Extension(out Value: WideString): HResult; stdcall;
    function Get_TimeIncrease(out Value: Double): HResult; stdcall;
    function Set_TimeIncrease(Value: Double): HResult; stdcall;
    function EndChannel: HResult; stdcall;
    function EndValue: HResult; stdcall;
    function Get_SupportsAsync(out Value: WordBool): HResult; stdcall;
    function Get_SupportsSRDiv(out Value: WordBool): HResult; stdcall;
    function Get_DataCount(out Value: Largeuint): HResult; stdcall;
    function Set_DataCount(Value: Largeuint): HResult; stdcall;
    //<--

    //-->ICustomExport2
    function SetAbsMin(AbsMin: Single): HResult; stdcall;
    function SetAbsMax(AbsMax: Single): HResult; stdcall;
    function SetTrigOffset(TrigTime: Double): HResult; stdcall;
    function SetChannelColor(Color: Integer): HResult; stdcall;
    function SetRangeMax(Value: Single): HResult; stdcall;
    function SetRangeMin(Value: Single): HResult; stdcall;
    function WriteDoubleValue(Value: Double): HResult; stdcall;
    function WriteAsyncDoubleValue(TimeStamp: Double; Value: Double): HResult;
stdcall;
    function GetDWTypeLibVersion(out Value: Integer): HResult; stdcall;
    function Get_SupportsDouble(out Value: WordBool): HResult; stdcall;
    function SetDoubleFloat(DoubleFloat: WordBool): HResult; stdcall;
    function StartEvents: HResult; stdcall;
    function StopEvents: HResult; stdcall;
    function WriteEvent(EventType: Integer; const EventTypeString: WideString;
Time: Double; const Comment: WideString): HResult; stdcall;
    function SetChannel(const Channel: IChannel): HResult; stdcall;
    function SetApp(const Channel: IApp): HResult; stdcall;
    //<--

    //-->IExportFrame
    procedure ShowFrame(Handle: Integer; out FrameHeight: Integer); safecall;
    procedure HideFrame; safecall;
    procedure Apply; safecall;
    //<--

public
    procedure Initialize; override;
    destructor Destroy; override;
end;

TExportObjectFactory = class(TComObjectFactory)
public
    procedure UpdateRegistry(Register: Boolean); override;
end;

const
  Class_Export: TGUID = '{8BA63341-3DEE-4BFB-90AA-DAA66DC874E2}';
  //New GUID can be generated by <Ctrl> + <Shift> + <G>
implementation
  uses ComServ, Registry, MyExportFormUnit;

procedure TExportObjectFactory.UpdateRegistry(Register: Boolean);
var
Reg: TRegistry;
begin
  inherited UpdateRegistry(Register);
  if Register then
  begin
    Reg := TRegistry.Create(KEY_ALL_ACCESS);
    Reg.RootKey := HKEY_LOCAL_MACHINE;
    Reg.OpenKey('Software\Dewesoft\Exports\ExportTemplate', True);
    Reg.WriteString('GUID', GUIDToString(Class_Export));
    Reg.WriteString('Name', 'My Export (*.any)');
    Reg.WriteString('Extension', 'any');
    Reg.WriteString('Version', Version);
    Reg.WriteInteger('TLB', DEWEsoftMinorVersion);
    Reg.WriteString('Vendor', 'Dewesoft');
    Reg.Free;
  end else
  begin
    Reg := TRegistry.Create(KEY_ALL_ACCESS);
    Reg.RootKey := HKEY_LOCAL_MACHINE;
    Reg.DeleteKey('Software\Dewesoft\Exports\ExportTemplate');
    Reg.Free;
  end;
end;

{ TExport }
function TExport.EndChannel: HResult;
begin
  Result := S_OK;
end;

function TExport.EndDataFolder: HResult;
begin
  Result := S_OK;
end;

function TExport.EndExport: HResult;
begin
  FApp := nil;
  Result := S_OK;
end;

function TExport.EndHeader: HResult;
begin
  Result := S_OK;
end;

function TExport.EndInfo: HResult;
begin
  Result := S_OK;
end;

function TExport.EndValue: HResult;
begin
  Result := S_OK;
end;

function TExport.Get_AbsoluteTime(out Value: WordBool): HResult;
begin
  Value:= AbsoluteTime;
  Result := S_OK;
end;

function TExport.Get_DataCount(out Value: Largeuint): HResult;
begin
  Value:= DataCount;
  Result := S_OK;
end;

function TExport.Get_ExportType(out Value: ExportTypes): HResult;
begin
  Value := etChannelBased; //or: Value:= etValueBased;
  Result := S_OK;
end;

function TExport.Get_Extension(out Value: WideString): HResult;
begin
  Value := '.any';
  Result := S_OK;
end;
```

6.7.2 Source Code of a CustomExport Frame/Form

```
unit MyExportFormUnit;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

type
  TMyExportForm = class(TForm)
    lblTitle: TLabel;
    lblVersion: TLabel;
  private
    { Private declarations }
  public
    procedure ShowFrame(Handle: Integer; out FrameHeight: Integer);
    procedure HideFrame;
    procedure Apply;
  end;

var
  MyExportForm: TMyExportForm;

implementation

{$R *.dfm}

{ TMyExportForm }

procedure TMyExportForm.Apply;
begin

end;

procedure TMyExportForm.HideFrame;
begin
  Visible := False;
  ParentWindow := 0;
end;

procedure TMyExportForm.ShowFrame(Handle: Integer;
  out FrameHeight: Integer);
begin
  ParentWindow := Handle;
  Visible := True;
  Left := 0;
  Top := 0;
  FrameHeight := Height;
end;

end.
```

7 Description of Methods and Properties

7.1 IAISetupScreen

7.1.1 Methods of IAISetupScreen

procedure ShowChannelSetup(ChNo: Integer);
ShowChannelSetup opens a channel setup dialog window similar to pressing the Setup-button of a channel.
Parameters:
ChNo defines the number of the channel of which the setup dialog should be shown.

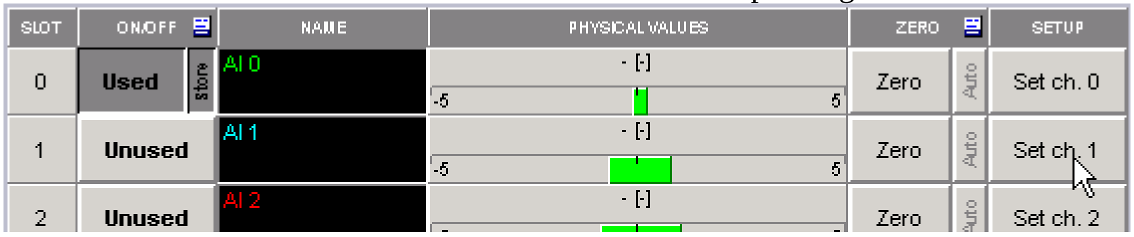


Figure 43 ShowChannelSetup

procedure SetColumnVisible(ColNo: Integer; Visible: WordBool);
SetColumnVisible allows showing or hiding the different columns of the channel setup. (This procedure works for analog channels only.)
Parameers:
ColNo is the number of the column.
Visible defines whether to show or to hide the column specified by ColNo.

7.2 IAlarms

7.2.1 Properties of IAlarms

property ActiveCount: Integer; (read-only)
ActiveCount is the number of alarm conditions which are currently active.

property ActiveItem[I: Integer]: IAlarmCond; (read-only)
ActiveItem[I] is the active alarm condition at the index I. I is in the range of 0…ActiveCount–1. ActiveItem list is consistent only during measurement.

property Count: Integer; (read-only)
Count is the number of alarm conditions which are set.

property Item[I: Integer]: IAlarmCond; (read-only)
Item[I] is the alarm condition at index I. I is in the range of 0…Count–1.

7.3 IAlarmCond

7.3.1 Properties of IAlarmCond

property Index: Integer; (read-only)
Returns the Index of the alarm condition.

property Name: WideString; (read-only)
Name is the name which is assigned to an alarm condition.

property Status: WordBool; (read-only)
Status is true if an alarm ondition is met and the alarm is currently still active.

property StopOption: Integer;
StopOption defines how and whether an alarm condition is stopped. This can be one of the following:
0…never
1…manual
2…on stop condition
3…on time

property StopTime: Single;
StopTime is the time after which an alarm is stopped if the StopOption is set to 3 (on time). The unit of StopTime is seconds.

property StopTrigger: ITrig; (read-only)
StopTrigger is the combination of trigger conditions deactivating an alarm if StopOption is set to 2.

property Trigger: ITrig; (read-only)
Trigger is the combination of trigger conditions activating an alarm. (cf. 7.36 ITrig)

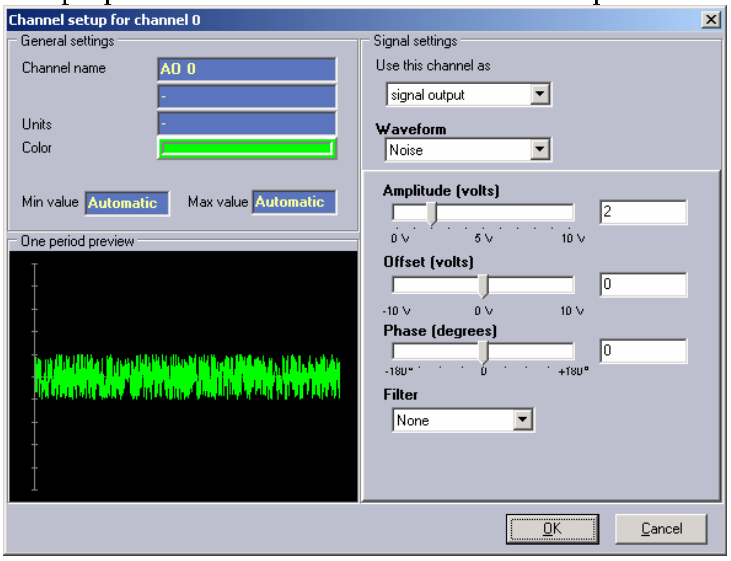
7.3.2 Methods of IAlarmCond

procedure EndAlarm;
EndAlarm can be called in order to stop an alarm having StopOption set to 1 (manual).

7.4 IAOChannel

7.4.1 Properties of IAOChannel

The properties of AOChannel refer to the setup of an AO channel. Cf. right side of Figure 44.



Channel setup for channel 0

General settings
Channel name: AO 0
-
Units: -
Color: 🟩
Min value: Automatic
Max value: Automatic

One period preview

Signal settings
Use this channel as: signal output
Waveform: Noise
Amplitude (volts): 2
0 V 5 V 10 V
Offset (volts): 0
-10 V 0 V 10 V
Phase (degrees): 0
-180° 0 +180°
Filter: None

Figure 44 Analog Output Channel Setup

property Ampl: Single;
Ampl specifies the amplitude value of the output signal.

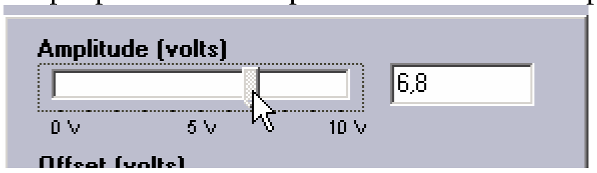


Figure 45 Analog Output Amplitude

property FilterFreq1: Single;
FilterFreq1 is the first frequency value of a low pass or band pass filter (only applicable to noise output signals), cf. Figure 46 and Figure 47.

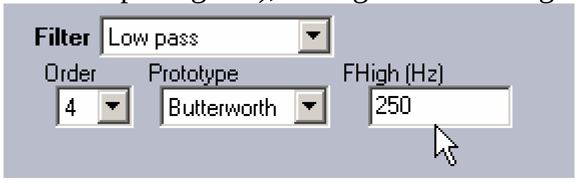


Figure 46 AOChannel FilterFreq1 (for low pass filter)

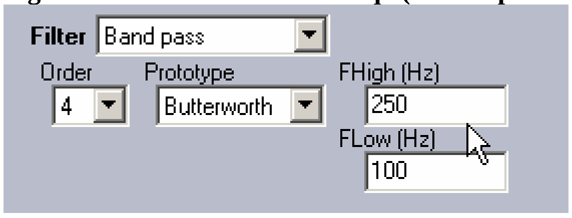


Figure 47 AOChannel FilterFreq1 (for band pass filter)

property FilterFreq2: Single;
FilterFreq2 is the second frequency of a filter (band pass) (only applicable to noise output signals), cfFigure 48.

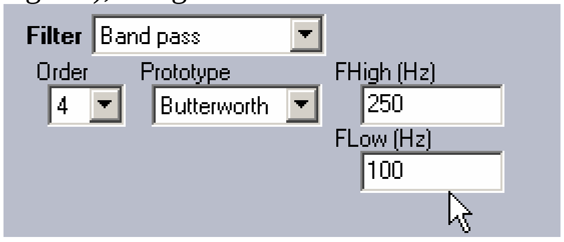


Figure 48 AOChannel FilterFreq2 (for band pass filter)

property FilterType: Integer;
FilterType is the type of a filter which can be e.g. one of the following: None, Pink, Low Pass, Band pass. Cf. Figure 49.

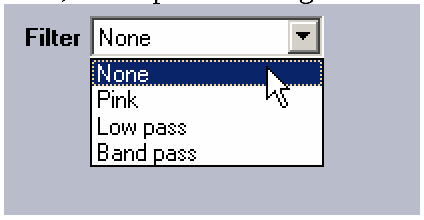


Figure 49 AOChannel FilterType

property FilterOrder: Integer;
FilterOrder denotes the order of a low pass or band pass filter (only applicable to noise output signals), cf. Figure 50.

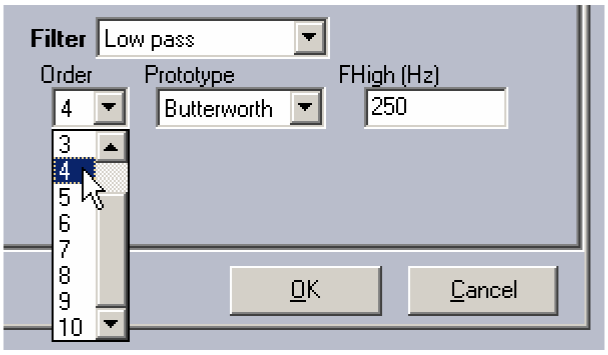


Figure 50 AOChannel FilterOrder

property FilterProtoType: Integer;
FilterProtoType refers to the filter's prototype which can be e.g. Butterworth, Chebyshev or Bessel, cf. Figure 51. This setting is applicable to low pass and band pass filters of noise output signals.

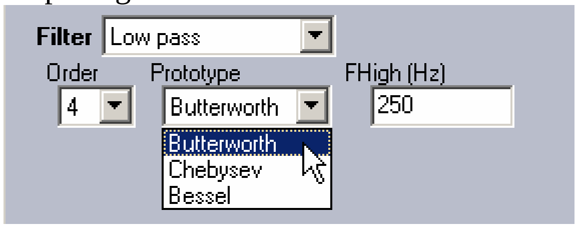


Figure 51 AOChannel FilterProtoType

property Offset: Single;
Offset specifies the offset value of the output signal.

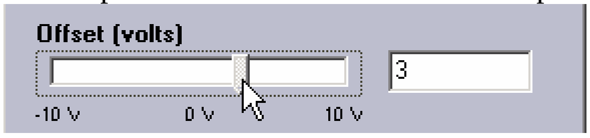


Figure 52 Analog Output Offset

property Phase: Single;
Phase specifies the phase angle of the output signal.

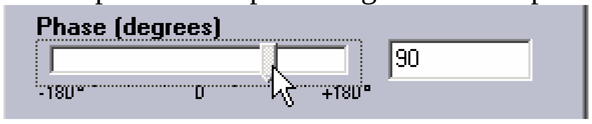


Figure 53 Analog Output Phase

property Range: Integer;
Range defines the voltage range of the output channel.
0…+/– 10V
1…+/– 1V

property WaveForm: AOWaveForm;
WaveForm denotes the type of waveform, which can be one of the following:
0…Sine
1…Triangular
2…Rectangular
3…Saw
4…Noise
5…Arbitrary

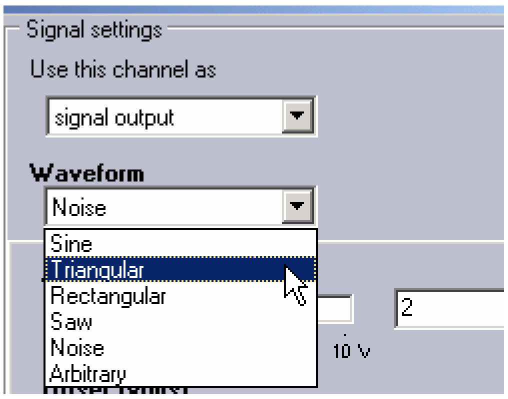


Figure 54 Analog Output Waveform

7.5 IAOGroup

7.5.1 Properties of IAOGroup

property AmplChangeFactor: Single;
AmplChangeFactor is the rate at which amplitude changes are done. Its unit is [V/s].

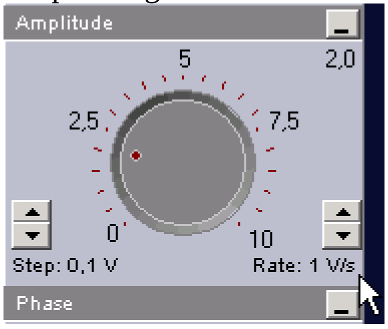


Figure 55 AmplChangeFactor

property AOChannels: IChannelList; (read-only)
AOChannels provides a list of analog output channels. It is of the type IChannelList.

property ControlsClock: WordBool; (read-only)
ControlsClock is true, if the analog output-board is the master clock (Dependant on the hardware setup).

property DCChangeFactor: Single;
DCChangeFactor is the rate at which DC value changes are done. Its unit is [V/s].

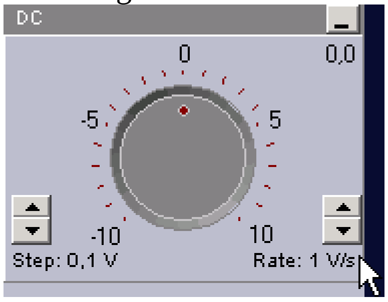


Figure 56 DCChangeFactor

property DeltaFreq: Single;
DeltaFreq is the frequency difference between subsequent steps of “Step sweep” signal.

property Freq: Single;
Freq is the frequency of a signal.

property FreqChangeFactor: Single;
FreqChangeFactor is the rate at which frequency changes are done. Its unit is [Hz/s].

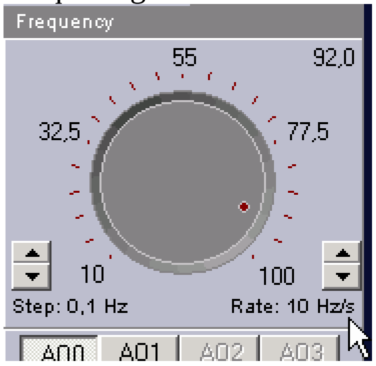


Figure 57 FreqChangeFactor

property LogSweep: WordBool;
LogSweep defines whether the frequency change of a sweep is logarithmic or not.

property OperationMode: AOOperationMode;
OperationMode defines the mode of the function generator, which can be one of the following:
0…Fixed Frequency
1…Sweep
2…Step sweep
3…Burst
4…Chirp

property PhaseChangeFactor: Single;
PhaseChangeFactor is the rate at which phase changes are done. Its unit is [°/s].

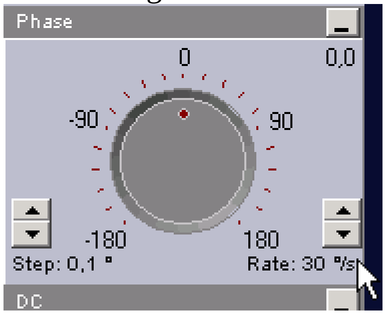


Figure 58 PhaseChangeFactor

property SampleRate: Integer;
SampleRate is the sample rate for analog output.

property ShowInfoChannels: WordBool;
ShowInfoChannels defines whether informative channels should be available during measurement. These are “Ampl”, “Phase”, “Offset” and “Freq”, providing information about the current waveform settings.

property StartFreq: Single;
StartFreq defines the start frequency of a signal like “Sweep”, “Step sweep” or “Chirp”.

property StartTime: Single;
StartTime defines the startup time of a waveform.

property StopFreq: Single;
StopFreq is the end frequency of a signal like “Sweep”, “Step sweep” or “Chirp”.

property StopTime: Single;
StopFreq defines the fall time of a waveform.

property SweepMode: AOSweepMode;
SweepMode defines the mode of a sweep waveform, which decides whether a sweep signal will be output repeatedly or only once. This can be one of the following:
0…Loop
1…Single

7.6 IApp

7.6.1 Methods of IApp

function AOGetManualAvail: WordBool;
Whether the manual start/stop feature of analog output is available in the current setup. Cf. Figure 59.

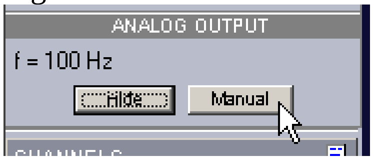


Figure 59 Analog Output Manual Start/Stop Button

procedure AOSetManual;
To manually start or stop the AO equal to clicking the “Manual” button on DEWESoft, cf. Figure 59.

procedure ChangeDaqType(DaqType: Integer);
ChangeDaqType changes the device type for analog acquisition. Cf. Figure 60.
0…No A/D
1…Dewetron DAQ
2…Dewetron DSA
3…Spectrum
4…National Instruments
5…National Instruments MX
6…National Instruments DSA
7…Data Translation
8…Microstar DAP

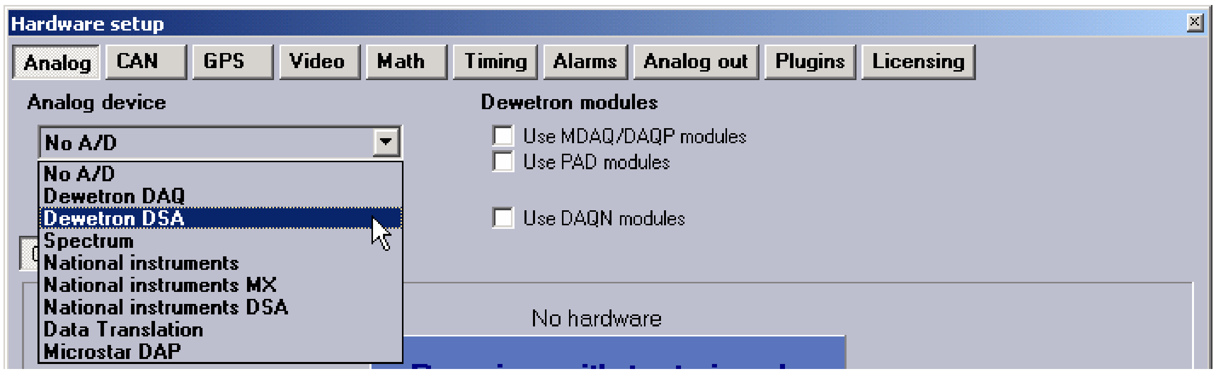


Figure 60 Changing the DAQ Type in DEWESoft's Hardware Setup

procedure ChangeComPort(ComPort: Integer);
ChangeComPort changes the com port which is used for communication to the modules. Cf. Figure 61.
Parameters:
ComPort is the number of the com port which should be selected. 0…com 1, 1… com 2, etc.

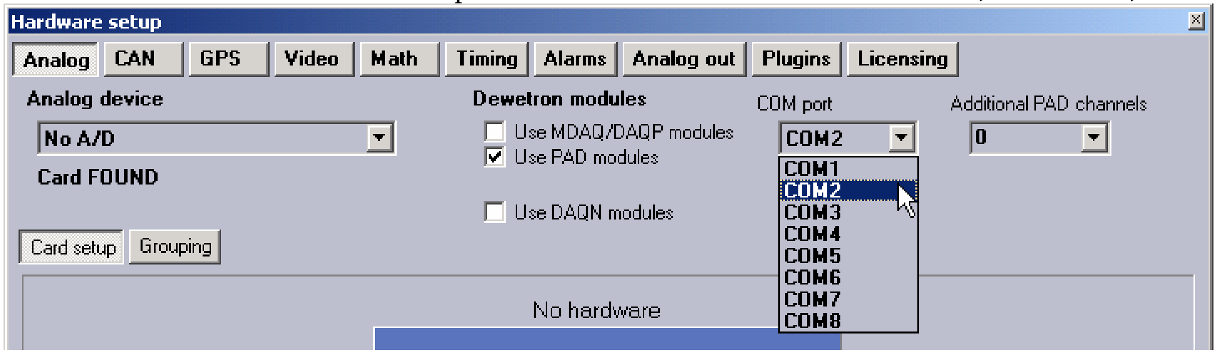


Figure 61 Com Port for the Modules

procedure ExportData(ExportType: Integer; const FileName: WideString);
ExportData corresponds to the file export feature of DEWESoft (see Figure 62). It exports the currently selected DEWESoft data file to the specified file type to the target path.
Parameters:
ExportType specifies the export file type which can be one of the following:
0…Flexpro (*.fpd)
1…Excel (.xls)
2…DIAdem (*.dat)
3…Matlab (*.mat)
4…Universal File Format 58 (*.unv)
5…FAMOS (*.dat)
6…NSoft time series (*.dac)
7…Text (‘.txt)
8…Sony (*.log)
9…RPCIII (*.rsp)
10…Comptade (*.cfg)
FileName specifies the name of the target-file including its path.

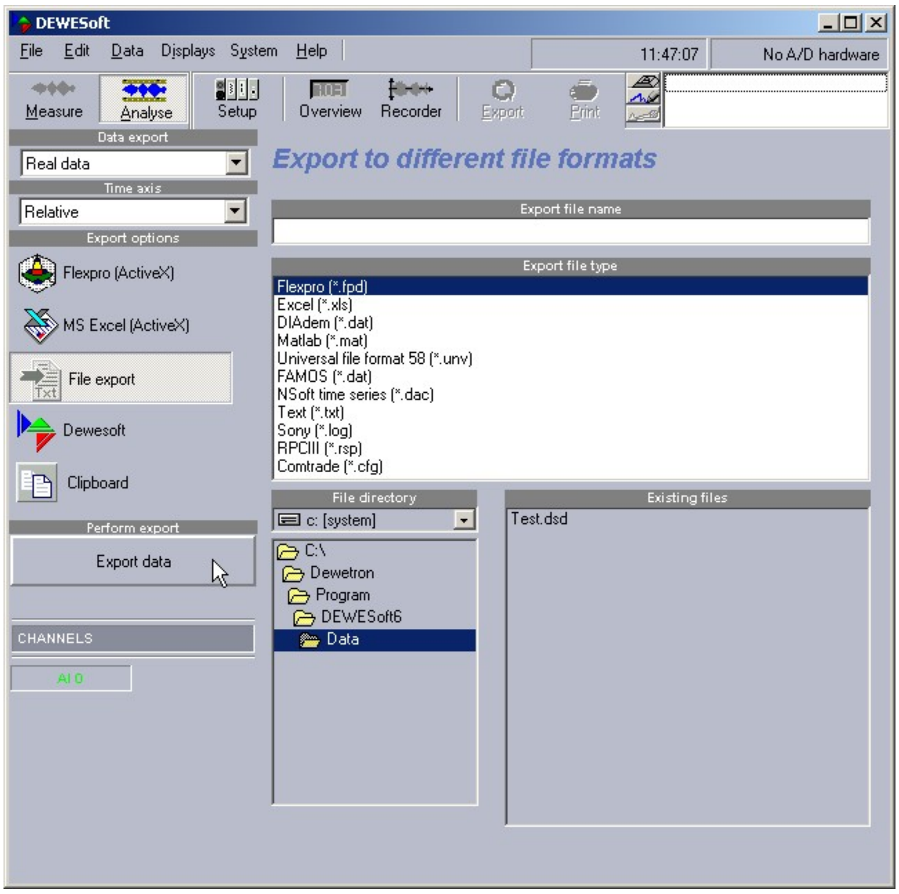


Figure 62 File Export in DEWESoft

procedure GetInterfaceVersion(var Major: Integer; var Minor: Integer; var Revision: Integer);
GetInterfaceVersion can be used for reading the version of DEWESoft's DCOM interface.
Parameters:
Major is the major version number.
Minor is the minor version number.
Revision is the revision number.

procedure HardwareSetup(Plugins: WordBool);
HardwareSetup starts the dialog window of the hardware setup like clicking "System >> Harware setup".
Parameters:
Plugins defines whether to show the Plugins tab in the hardware setup or not. If Plugins is set to True, the windows registry will be searched for installed plug-ins, the plug-ins will be re-initialized and the Plugins tab will be shown. If Plugins is set to False, the Plugins tab will not be shown.

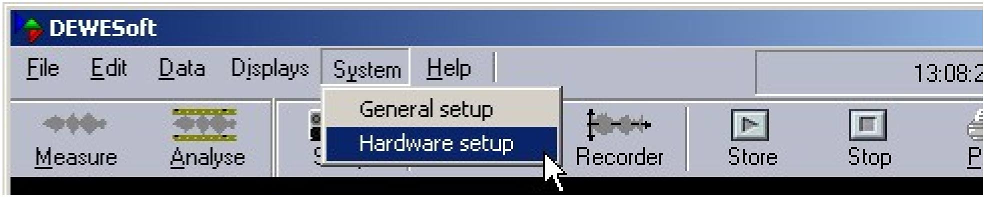


Figure 63 Hardware Setup

procedure Init;
Init starts the initialization of DEWESoft.

procedure LoadDBC(PortNo: Integer; const FileName: WideString);
LoadDBC loads the dbc-file specified by FileName and applies it the CAN-port specified by PortNo.
Parameters:
PortNo is the port-number to which the dbc-file is applied (first port is 0).
FileName is the name of the dbc-file including its path. If the dbc-file resides in the DEWESoft's "Setup"-directory, the file-name without the whole path would be sufficient.

procedure LoadDisplaySetup(const FileName: WideString);
LoadDisplaySetup allows loading a display setup as it can be done by the menu item "File >> Load Display Setup", cf. Figure 64.
Parameters:
FileName is the file name including the path of the DEWESoft setup file which should be applied.

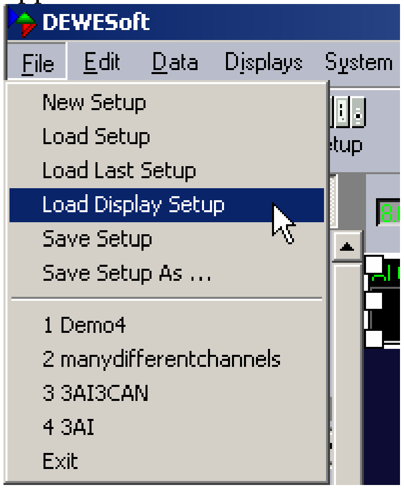


Figure 64 Load Display Setup Menu Item

procedure LoadFile(const FileName: WideString);
LoadFile loads a data file specified by FileName. This function corresponds to clicking "Data >> Load Data File" (see Figure 65).
Parameters:
FileName is the file name including its path.

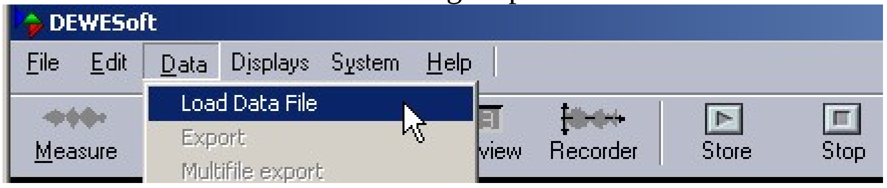


Figure 65 Load Data File

procedure LoadModuleSetup(const FileName: WideString);
LoadModuleSetup loads the setup of modules only (no screens, no triggers, …) from a DEWESoft setup file (.dss).
Parameters:
FileName is the file name of the setup file including its path.

procedure LoadSetup(const FileName: WideString);
LoadSetup loads a setup file specified by FileName. This function corresponds to clicking "File >> Load Setup" (see Figure 66)
Parameters:
FileName is the file name of the setup file including its path.

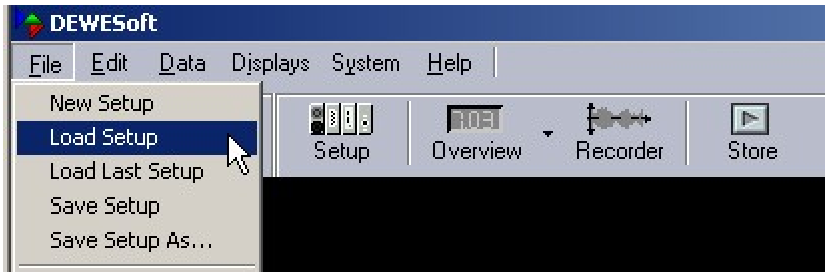


Figure 66 Load Setup

procedure MainWndMessage(Msg: Integer; WParam: Integer; Wait: WordBool);
MainWindowMessage can be used in conjuction with plug-ins. MainWindowMessage is the only allowed message to be called from any thread within a plug-in. If it is necessary to send a message to DEWESoft from an additional thread of a plug-in, MainWndMessage has to be used. “Msg” and “Param” will be available at “IPlugin2.OnOleMsg” which will follow. On this, the message has to be sent to DEWESoft from the main thread of the plug-in.
Parameters:
Msg and WParam are Integer values specifying the message in order to be able to send the appropriate message when OnOleMsg is called within the plug-in.
Wait specifies whether the thread from which MainWndMessage was called will wait or continue execution.

procedure ManualStart;
ManualStart causes a manual start trigger, when DEWESoft is in measure mode and a trigger condition is set (see Figure 67) and armed. This function corresponds to clicking the trigger button as shown in Figure 68.

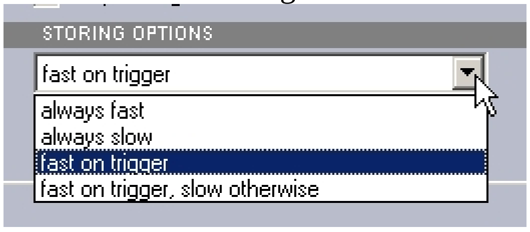


Figure 67 Storing Options

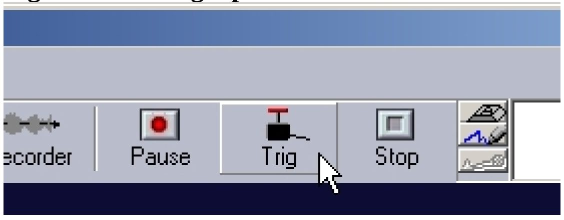


Figure 68 Manual Start Trigger

procedure ManualStop;
ManualStop causes a manual stop trigger, when DEWESoft is in measure mode and a trigger condition is set and currently active. This function corresponds to clicking the trigger button when a trigger is currently active, as shown in Figure 69.

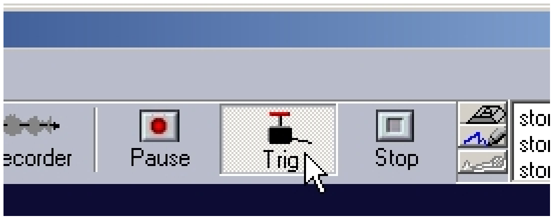


Figure 69 Manual Stop Trigger

procedure Measure;
Measure makes DEWESoft to switch to measure mode, corresponding to clicking the "Measure" button to do so (see Figure 70).

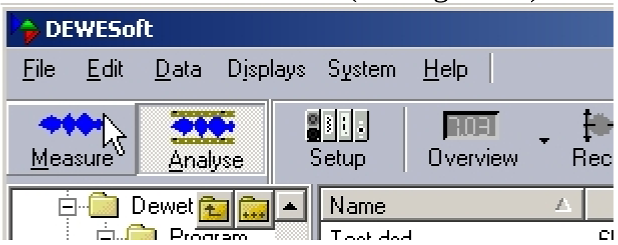


Figure 70 Measure

procedure MenuClick(Item: MenuItems);
MenuClick causes the menu item defined by "Item" to be clicked. E.g. Item would set to 2 would cause the Load setup file dialog to be opened equal to clicking the menu item from the File menu as shown in Figure 71.
Parameters:
Item can be one of the following constants defined as MenuItems:
ItemSaveSetup = 0
ItemSaveSetupAs = 1
ItemLoadSetup = 2

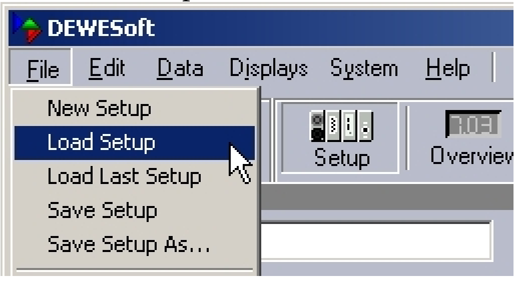


Figure 71 MenuItem Load Setup

procedure NewSetup;
NewSetup equals to clicking "New Setup" from the File menu (see Figure 72), causing a new setup to be created.

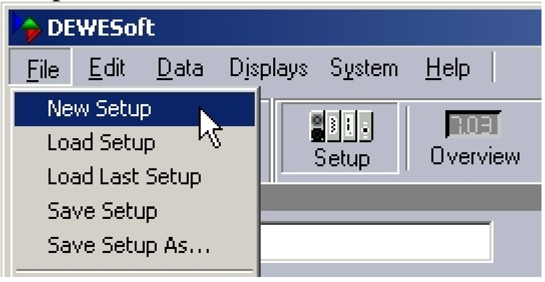


Figure 72 New Setup

procedure PauseStoring;
PauseStoring pauses the storing of data equal to pressing the Pause-button (cf. Figure 73) which is available when storing is in progress.

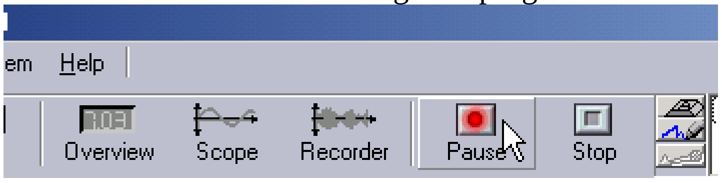


Figure 73 Pause Storing

procedure PrintScreen(ShowDialog: WordBool);
PrintScreen corresponds to the "Print" button in DEWESoft (cf.Figure 74). This procedure can only be used when the application is in Analyse mode (when the Print button would be shown). By default this procedure prints the screen on the default printer.
Parameters:
ShowDialog defines whether the printing dialog is shown or not.

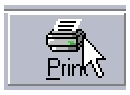


Figure 74 Print Button

procedure ResumeStoring;
ResumeStoring resumes the storing of data after storing has been paused before. This function is equal to pressing the Resume-button which is available when storing is paused. (Cf. Figure 75)

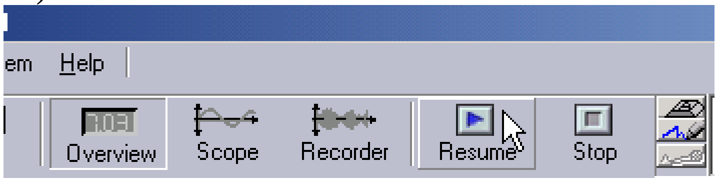


Figure 75 Resume Storing

procedure SaveSetup(const FileName: WideString);
SaveSetup saves the current setup to the file specified by FileName. Compare to clicking "File >> Save Setup" (see Figure 76).
Parameters:
FileName is the file name of the setup file to store to including its path.

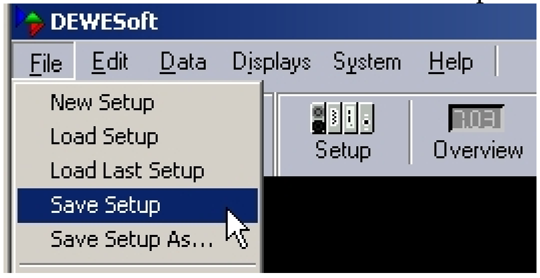


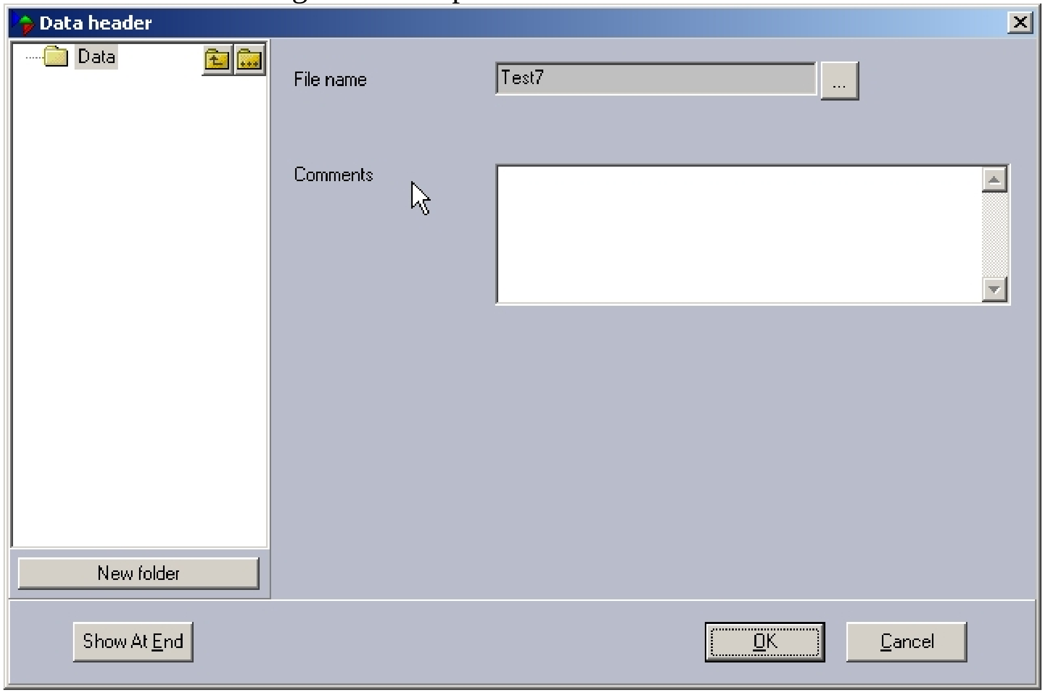
Figure 76 SaveSetup

function SendCommand(const Cmd: WideString; Timeout: Integer): WideString;
SendCommand sends a command string to a hardware module via the (internal) module bus and returns its answer.
Parameters:
Cmd is the command string to send to the module.
Timeout specifies how long to wait for an answer of the module. Its unit is ms.

procedure SetFullScreen(Full: WordBool);
SetFullScreen switches between full screen display mode and standard display mode, similar to pressing <CTRL> + <F> to do so.
Parameters:
Full defines the mode to switch to. If Full is True, the window switches to full screen. If Full is False, the window switches to normal display mode.

procedure SetHeaderData(const Caption: WideString; const Header: WideString);
SetHeaderData allows changing the data of the header of a measurement file.
Parameters:

Caption denotes which field of the header to change. E.g. the caption of the standard field would be “Comments” (see mouse cursor in Figure 77).
Header is the text being set to the specified field.



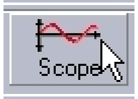
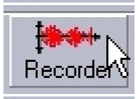
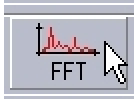
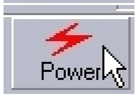
File name: Test7

Comments

Figure 77 Data Header

procedure SetInstrument(Id: Integer);
SetInstrument sets DEWESoft to a certain screen.
Parameters:
Id specifies the screen to set. Id can be one of the following values corresponding to the buttons on the right:

Table 1 Screen Buttons

0…“Overview”,	
1… “Scope”,	
2… “Recorder”,	
3…”FFT”,	
4…”Video”,	
5…”GPS”,	
6…”Vert. rec.”,	
7…”Power”.	

procedure SetMainDataDir(const DataDir: WideString);
SetMainDataDir allows setting the main data directory used by DEWESoft similar to setting it via the “General setup” manually, cf. Figure 78.
Parameters:
DataDir is path of the directory which should be set.

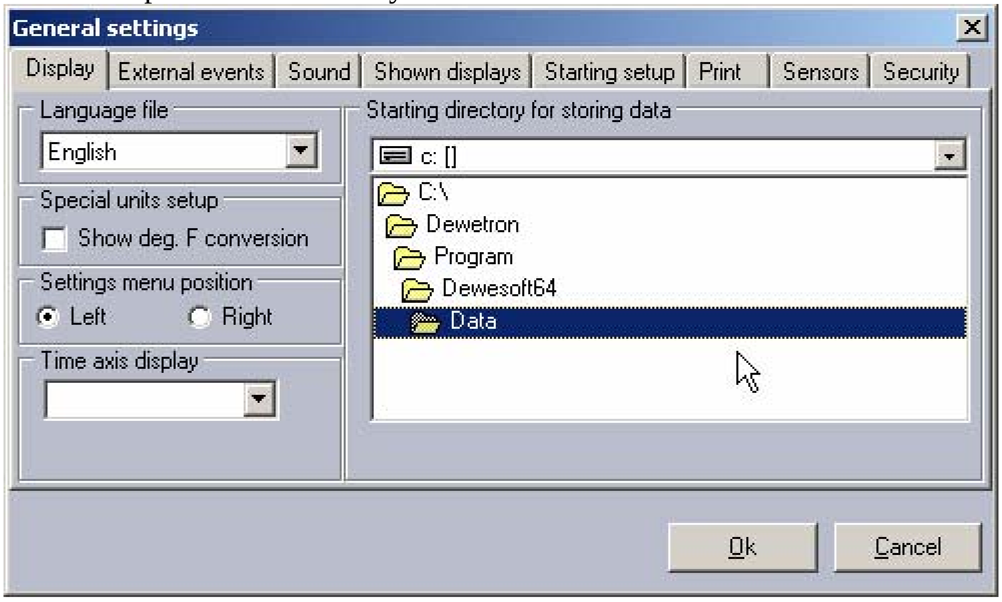


Figure 78 Setting the Main Data Directory

function SetScopeParams(PreTime: TDateTime; PostTime: TDateTime; const Channel: IChannel; Level: Single): Integer;
SetScopeParams sets the trigger properties of the scope.
Parameters:
PreTime is the pre-trigger time,
PostTime is the post-trigger time,
Channel specifies the channel used for triggering,
Level specifies the trigger level.
The data type TDateTime is a Double value which is interpreted as follows:
The integral part of a Delphi TDateTime value is the number of days that have passed since 12/30/1899. The fractional part of the TDateTime value is fraction of a 24 hour day that has elapsed.
Following are some examples of TDateTime values and their corresponding dates and times:

0	12/30/1899 12:00 am
2.75	1/1/1900 6:00 pm
-1.25	12/29/1899 6:00 am
35065	1/1/1996 12:00 am

procedure SetScopeUsed(Value: WordBool);
SetScopeUsed activates or deactivates the scope trigger.
Parameters:
Value denotes whether the scope trigger is used or not. True will activate the scope trigger and False will deactivate it.

procedure SetScreenIndex(Index: Integer);
SetScreenIndex sets the screen index of the active screen, if more than one screen of one kind exists. Additional screens can be added by clicking “Displays >> Add display” as shown in Figure 79.
Parameters:

Index is the index of the screen to show. The screens are indexed starting at 0 up to number of screens -1. Figure 80 shows 3 Overview screens to select. In this case the screen named "Overview" would corresponds to index = 0, "Overview 2" to index = 1 and "Overview 3" to index =2.

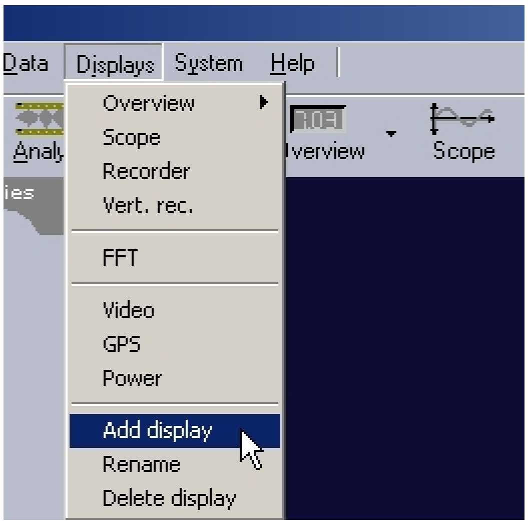


Figure 79 Add display

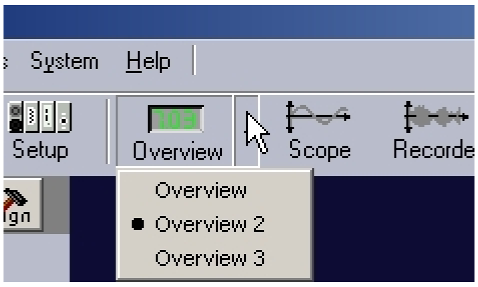


Figure 80 Screens of a kind

procedure SetStoreMode(Mode: Integer);
SetStoreMode allows setting DEWESoft's storing options, cf. Figure 81. (The storing mode can be read via the property "StoreMode".
Parameters:
Mode is the list index of the storing option, which can be one of the following:
0… "always fast"
1… "always slow"
2… "fast on trigger"
3… "fast on trigger, slow otherwise"

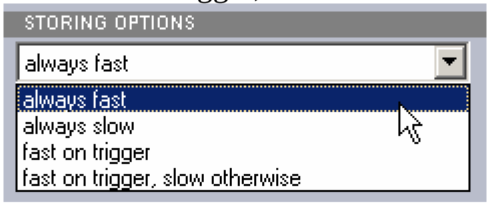


Figure 81 DEWESoft's Storing Options

procedure SetupScreen;
SetupScreen switches the application to the setup screen like clicking the Setup button (Figure 82)

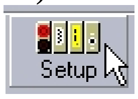


Figure 82 Setup Button

procedure ShowSensorEditor;
ShowSensorEditor opens the sensor editor window.

procedure StartModuleScan;
StartModuleScan allows to manually start the scanning of the modules after having stoped it by StopModuleScan.

procedure StartStoring(const FileName: WideString);
StartStoring corresponds to the function of the Store button (Figure 83). It starts storing or arms the trigger if one is set.
Parameters:
FileName is the name of the file where data is stored to including its path and its extension (.dsd). If FileName is empty, the settings of DEWESoft are used.

Figure 83 Store Button

procedure Stop;
Stop stops the application and switches to the stop-screen. This function is similar to clicking the Stop button (Figure 84).

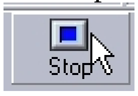


Figure 84 Stop Button

procedure StopModuleScan;
StopModuleScan allows to stop the scanning of the modules. Use StartModuleScan to restart scanning again.

procedure UpdateHardwareSetup;
UpdateHardwareSetup has to be called after making changes within the hardware setup.

procedure WriteErrorMessage(const ErrorMsg: WideString);
WriteErrorMessage writes an error message to the caption of DEWESoft. The purpose of this procedure is mainly for debugging applications. (In future the message will be written to some kind of debug-window.)

procedure ZeroAllAutoChannels(Zero: WordBool);
ZeroAllAutoChannels works for all DaqChannels having AutoZero set to True.
Parameters:
Zero: If Zero is set to True, the function sets the offset to perform zeroing. If Zero is False, it clears the offset from selected channels.

7.6.2 Properties of IApp

property ActiveScreen: Integer; (read-only)
ActiveScreen is the number of the active screen, which can be one of the following:
0…“Overview”,
1…“Scope”,
2…“Recorder”,
3…”FFT”,
4…”Video”,
5…”GPS”,

6…"Vert. rec.",
7…"Power".

property Acquiring: WordBool; (read-only)
Acquiring informs about whether DEWESoft is currently acquiring data. (If this property is True, DEWESoft is not necessarily storing.)

property ActualRunMode: Integer; (read-only)
ActualRunMode is the actual mode in which the application is currently running. This can be one of the following:
0…none,
1…measure,
2…analyse.
At startup ActualRunMode will be 0 in most cases (except "Automatically start acquisition" is checked in the "Starting setup" within the "General settings"). If the application is in measure mode, ActualRunMode is 1. If it is in analyze mode, ActualRunMode will be 2. The application can be set back to ActualRunMode of 0 by calling Stop.

property AISetupScreen: IAISetupScreen
See 7.1 IAISetupScreen for details.

property Alarms: IAlarms; (read-only)
Alarms provides the interface for the alarm settings. See IAlarms.

property AlwaysEnableTrigger: WordBool
AlwaysEnableTrigger allows to show the Trigger tab in setup screen anyway.

property AOGroup: IAOGroup; (read-only)
See IAOGroup for detailed information.

property AveragedCPB: IAveragedFFT (read-only)
AveragedCPB provides an interface for the calculation of octave analysis data. See IAveragedFFT.

property CAN: ICAN; (read-only)
See ICAN for detailed information.

property Data: IData; (read-only)
Data provides the measurement data and corresponding methods and properties. See IData for detailed information.

property DataLost: WordBool; (read-only)
DataLost is true when loss of data has occurred during measurement.

property DaqGroup: IDaqGroup; (read-only)
DaqGroup is the group of analog input channels. See IDaqGroup.

property DisableStoring: WordBool;
DisableStoring will hide the Store button, if it is set to True.

Property Enabled: WordBool;
Enabled allows enabling and disabling DEWESoft. If Enabled is False, it is not possible to manipulate DEWESoft's controls by the mouse.

property EventList: IEventList; (read-only)
EventList contains a list of events occurred during measurement. See IEventList for detailed information.

property FileNameSettings: IFileNameSettings; (read-only)
FileNameSettings provides access to the settings of the naming options of the data files.

property Height: Integer;
Height is the height of the application window expressed in pixels.

property IniFileDir: WideString;
IniFileDir is the path of the directory where the ini-file of DEWESoft resides.

property IsSetupMode: WordBool; (read-only)
IsSetupMode imforms about whether DEWESoft is currently in the setup mode or not.

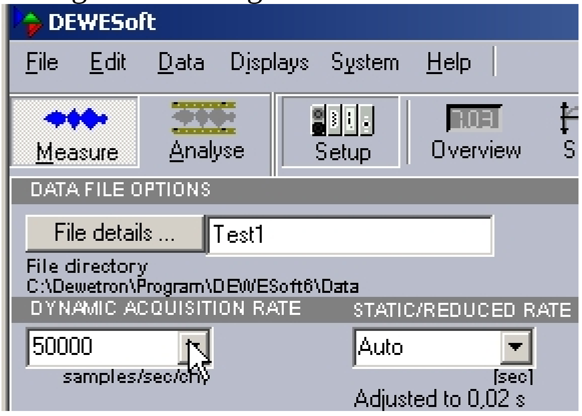
property Left: Integer;
Left is the distance between the left border of the application window and the left border of the screen expressed in pixels.

property LoadEngine: ILoadEngine; (read-only)
LoadEngine has to be used for handling data files acquired by DEWESoft. See ILoadEngine for detailed information.

property MainDataDir: WideString; (read-only)
MainDataDir is the path of the directory where measurement data is stored (By default this path would be C:\Dewetron\Program\DEWESoft6\Data).

property MasterClock: IMasterClock; (read-only)
MasterClock provides the current time from the A/D-board before the data is transferred. The unit of this time is seconds. MasterClock can be used for times-tamping asynchronous data.

property MeasureSampleRate: Integer;
MeasureSampleRate is the sample rate for data acquisition. This property corresponds to the setting shown in Figure 85 where the mouse cursor is placed.



Test1

File directory
C:\Dewetron\Program\DEWESoft6\Data

50000
samples/sec/ch

Auto
[sec]
Adjusted to 0,02 s

Figure 85 Acquisition Rate

property Modules: IModules; (read-only)
Modules provides the interface to the modules. See IModules.

property Parent: Integer;
Parent serves for referencing the windows handle of a form as the parent for DEWESoft.

property Screens: IScreens; (read-only)
Screens allows handling the different screens of DEWESoft. These screens can be e.g. an "Overview", a "Recorder", etc. Moreover, each of these categories can have more than one screen. See IScreens and IScreen for detailed information.

property ShowPropertyFrame: WordBool;
ShowPropertyFrame allows to show or to hide the property fame of a measurement screen. The property frame is the frame on the left side of a measurement screen. (Cf. Figure 86, where the property frame is shown on the left screen shot and not shown on the right one.)

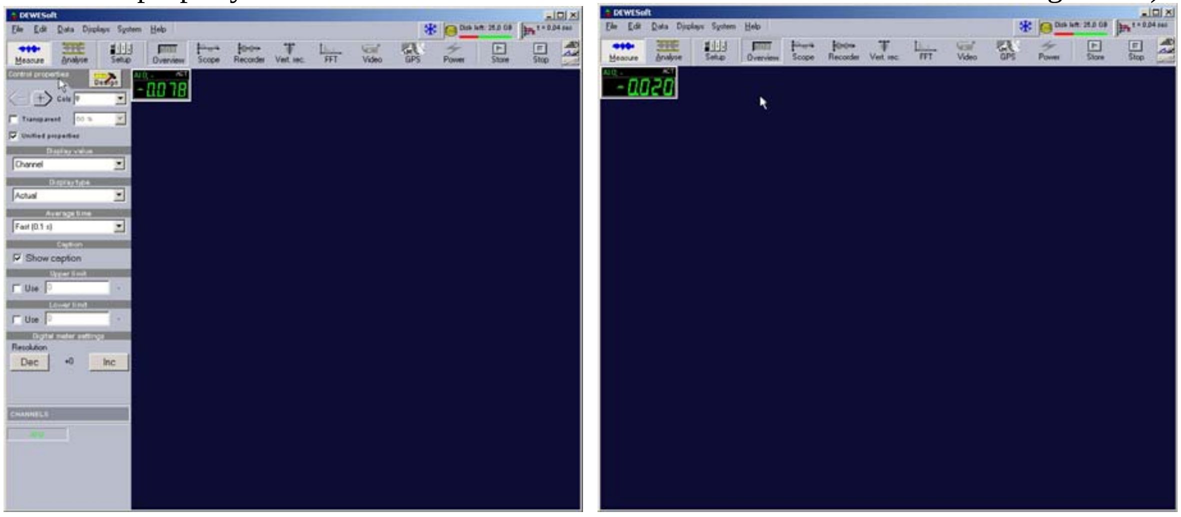


Figure 86 Property Frame of Measurement Screen

property ShowSROptions: WordBool;
ShowSROptions allows to show or to hide the sampling rate options within the setup screen. (Cf. Figure 87, where the sampling rate options are shown on the left screen shot and not shown on the right one.)

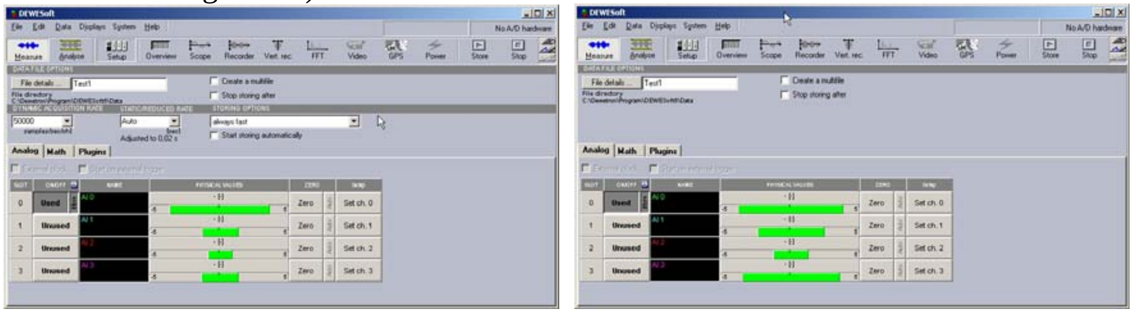


Figure 87 Sampling Rate Options

property ShowStoreOptions: WordBool;
ShowStoreOptions allows to show or to hide the store options within the setup screen. (Cf. Figure 88, where the sampling rate options are shown on the left screen shot and not shown on the right one.)

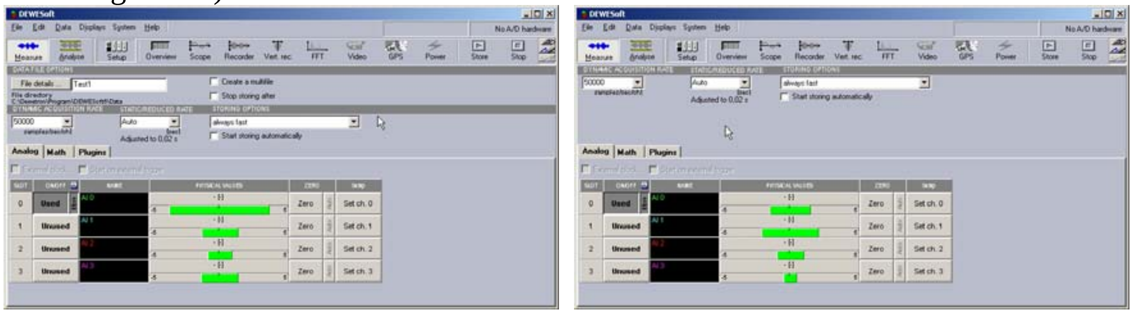


Figure 88 Store Options

property ShowStyle: Integer;
ShowStyle is for changing the appearance of the menu bar.
0…hides the whole menu bar (hides menu and buttons, cf. Figure 89),

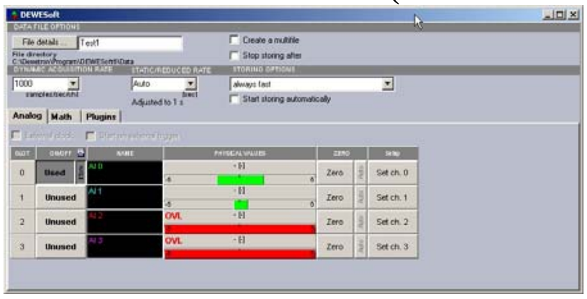


Figure 89 ShowStyle = 0
1…shows the menu bar but hides the drop-down menu bar (shows only buttons, cf. Figure 90),

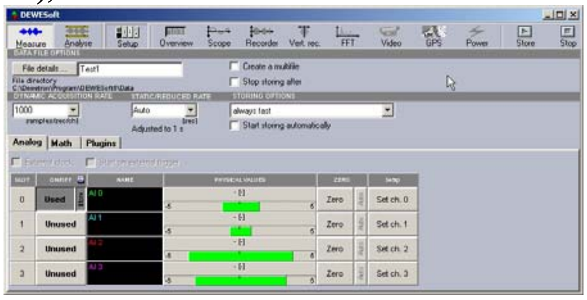


Figure 90 ShowStyle = 1
2… shows the menu bar including the drop-down menu bar (shows menu and buttons, cf. Figure 91).

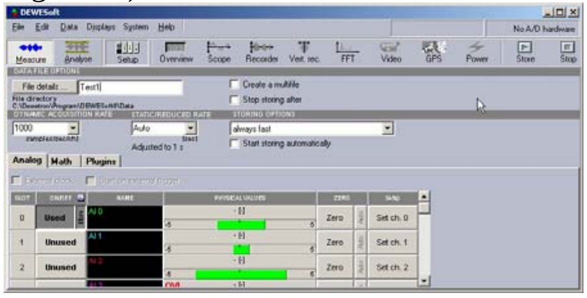


Figure 91 ShowStyle = 2

property StayOnTop: WordBool;
StayOnTop allows setting whether the application window should stay on top or not. If set to True, DEWESoft will remain in the foreground of the screen.

property StoreEngine: IStoreEngine; (read-only)
StoreEngine has to be used for handling the storing of data. See IStoreEngine for detailed information.

property TimerInterval: Integer
TimerInterval is the timer interval for DEWESoft acquisition in ms. The default value is 33.

property Top: Integer;
Top is the distance between the top border of the application window and the top border of the screen expressed in pixels.

property Trigger: ITrigger; (read-only)
See 7.36 for details.

property UsedDatafile: WideString;
UsedDatafile is the file name of the data file including its path and file extension (.dsd) where acquired data will be stored to. E.g. this could be “C:\Dewetron\Program\DEWESoft6\Data\Test.dsd”.

property UsedSetupfile: WideString; (read-only)
UsedSetupfile is the file name of the setup file currently used, including its path and file extension (.dss). E.g. this could be “C:\Dewetron\Program\DEWESoft6\Setups\MySetup.dss”.

property Version: WideString; (read-only)
Version is the number of the currently used DEWESoft version. E.g. this can be “6.2”.

property Visible: WordBool;
Visible defines whether the application window should be visible or not.

property Width: Integer;
Width is the width of the application window expressed in pixels.

7.6.3 Events ⚡ of IApp

procedure OnAlarm;
OnAlarm will be raised when an alarm occurs. Alarms can be set in the Alarms tab of the setup screen which is available when activated within the hardware setup.

procedure OnDataLost;
OnDataLost occurs when data lost is reported by the DAQ object.

procedure OnEvent(Reason: EventReason; Param: OleVariant);
OnEvent will be raised in case of several events.
Parameters:
EventReason denotes the type of event, which can be one of the following:
1 indicating “start”,
2 indicating “stop”,
3 indicating “trigger”,
11 indicating “VStart” (video start storing)
12 indicating “VStop” (video stop storing)
20 indicating “Keyboard”
21 indicating “Notice”
22 indicating “Voice”
24 indicating “Module” (example: Bridge shunt on/off).
Param can contain certain data of this event. E.g. in case of a “Notice”-event, this will be of the type String.

procedure OnException(const Str: WideString);
OnException will be raised when an error occurs within DEWESoft.
Parameters:
Str contains the message according to the exception that has happened.

procedure OnExit;
OnExit is raised on exiting DEWESoft.

procedure OnGetData;
OnGetData will be raised every time new data is added to a channel. This event happens about every 40ms.

procedure OnGetTime(var TimeLo: Integer; var TimeHi: Integer);
At OnGetTime a clock can be provided to DEWESoft, if no AD-hardware is used.
Parameters:
TimeLo and TimeHi are used because a 64-bit value is necessary to provide the time value.

procedure OnStartStoring;
OnStartStoring will be raised when the storing of data is started or a trigger for storing data is armed.

procedure OnStopStoring;
OnStopStoring will be raised when the storing of data is stopped or a trigger for storing data is unarmed.

procedure OnTrigger(Mid: Integer; Dir: Integer; Time: Double);
OnTrigger will be raised when a trigger event happens.
Parameters:
Mid is the number of complete data blocks currently acquired.
Dir is the position within the current data block.
Time is the current measurement time.

procedure OnTriggerStop(Mid: Integer; Dir: Integer; Time: Double);
OnTriggerStop will be raised when a stop-trigger happens.
Parameters:
Mid is the number of complete data blocks currently acquired.
Dir is the position within the current data block.
Time is the current measurement time.

7.6.4 Types / Constants of IApp

```
type
  MenuItems = TOleEnum;
const
  ItemSaveSetup = 0;
  ItemSaveSetupAs = 1;
  ItemLoadSetup = 2;
```

7.7 IAveragedFFT

7.7.1 Methods of IAveragedFFT

function CalculateFromPos(Mid: Integer; Dir: Integer): WordBool;
CalculateFromPos calculates the octave values according to the previously set properties. The function returns True if enoght data was available for the calculation; False otherwise.
Parameters:
Mid: The intermediate buffer position.
Dir: The direct buffer position.
Mid and Dir define the position within the data file from where the calculation should be started.

procedure GetChannels(Channels: OleVariant);
GetChannels provides a list of all channels used in AveragedFFT.
Parameters:
Channels is an OleVariant contaning an array of channels of the type IChannel.

procedure GetData(ChNo: Integer; OctaveDivider: Integer; Weighting: Integer; out BandCount: Integer; out Data: OleVariant);
GetData retrieves the calculated octave analysis data after having called CalculateFromPos.
Parameters:
ChNo is the number of the channel which data should be retrieved.
OctaveDivider is the devider for octave analysis. It is e.g. 3 in case of 1/3 ocatve analysis. The Divider can be 1, 3, 12 or 24.
Weighting is the weighting curve which is applied (cf. Figure 92). This can be one of the following:
0…Linear
1…A-weighting
2…B-weighting
3…C-weighting
4…D-weighting
BandCount returns the number of frequency bands.
Data returns the array of the calculated data. The length of the array depends on the BandCount.

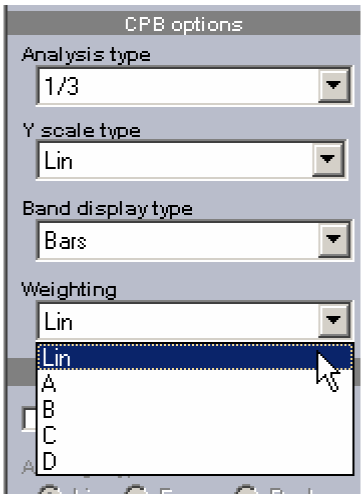

CPB options
Analysis type: 1/3
Y scale type: Lin
Band display type: Bars
Weighting: Lin

Figure 92 AveragedCPB Weighting

7.7.2 Properties of IAveragedFFT

property AveCount: Integer;
AveCount is the number of averages which will be used for the calculation (cf. Figure 93).

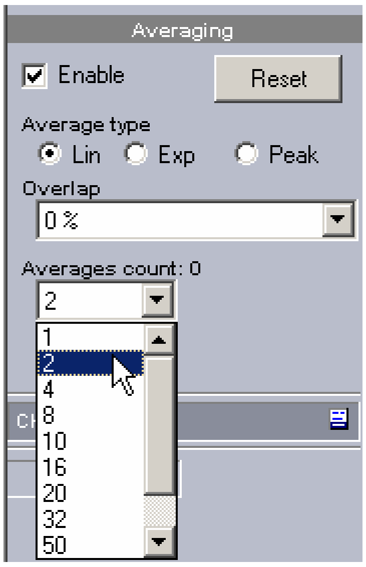


Figure 93 AveragedFFT Averages Count

property AverageType: Integer;
AverageType defines the type of averaging which should be applied to the calculation (cf. Figure 94).

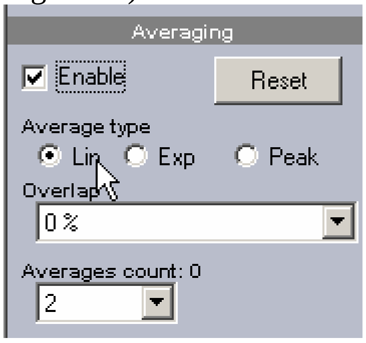


Figure 94 AveragedFFT Averaging Type

property Lines: Integer;
Lines defines the number of FFT lines used for the calculation (cf. Figure 95).

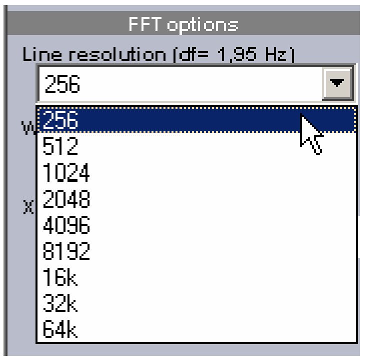
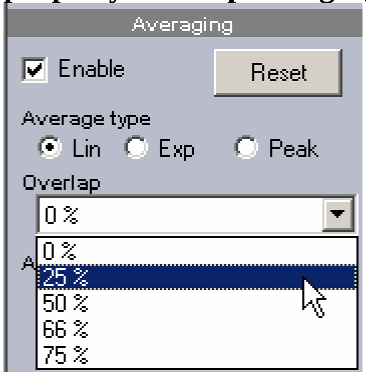


Figure 95 Averaged FFT Lines

property Overlap: Integer;



property Window: Integer;
Window defines the kind of window which will be applied (cf. Figure 96). The value can be one of the following:
0…Rectangular
1…Hanning
2…Hamming
3…Flat top
4…Triangle
5…Blackman
6…Exponent down.

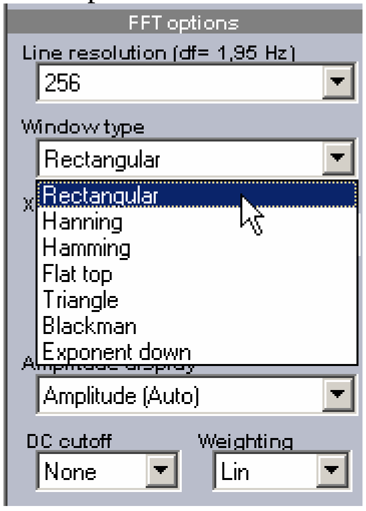


Figure 96 Averaged FFT Window Type

7.8 ICAN

7.8.1 Properties of ICAN

property SupportsOutput: WordBool; (read-only)
SupportsOutput gives information about whether a CAN-port is capable to output data.

property Count: Integer; (read-only)
Count gives the number of available CAN-ports.

property Item[Index: Integer]: ICANPort; (read-only)
Item is one of the CAN-ports which is of the type ICANPort (See 7.9 for details). Index specifies the which CAN-port to use. Index ranges from 0 up to Count–1.

7.9 ICANPort

7.9.1 Methods of ICANPort

procedure EnableOutput(Enable: WordBool);
EnableOutput enables or disables a CAN-port for output.
Parameter:
Enable must be True to enable the output, False otherwise.

procedure EndRead;
EndRead should be called after reading of data from the CAN-port. (See ReadMessage.)

function GetBaudRate: Integer;
GetBaudRate returns the Baud rate currently applied to a CAN-port.

function GetBaudRateList: OleVariant;
GetBaudRateList returns an array of available Baud rates. The data type of the array elements is String.

function ReadMessage(var TimeStamp: Double; var ArbId: Integer; var DataLo: Integer; var DataHi: Integer): WordBool;
ReadMessage reads a CAN-message.
Parameters:
TimeStamp is the timestamp of the read message, which is of the type Double.
Note:
If this function is used e.g. within a plug-in, it should be used within the procedure OnGetData and in conjunction with StartRead and StopRead. It should be called as often as defined by MessageCount. So, the code could look somehow like the following:

```
procedure Plugin.OnGetData;
var   i, ArbId, DataLo, DataHi: Integer;
      TimeStamp: Double;
begin
      MyCanPort.StartRead;
      for i:= 0 to MyCanPort.MessageCount – 1 do
      begin
            ReadMessage(TimeStamp, ArbId, DataLo, DataHi);
            //do something with TimeStamp, ArbId, DataLo and DataHi.
      end;
      MyCanPort.EndRead;
end;
```

function SendFrame(Extended: WordBool; ArbId: Integer; Data: T_CANFrame): WordBool;
SendFrame sends a message on the CAN-port.
Parameters:
Extended specifies whether extended identifiers are used or not.
ArbId is the identifier of the CAN-message.
Data is the data of the message which is of the type T_CANFrame. T_CANFrame is a record of 8 bytes (Byte0…Byte7).

procedure SetBaudRate(BaudRate: Integer);
SetBaudRate sets the baud rate of a CAN-port.
Parameters:
BaudRate is the Baud rate to be set.

procedure StartRead;
StartRead should be called before starting the reading of data from the CAN-port. (See ReadMessage.)

7.9.2 Properties of ICANPort

property MessageCount: Integer; (read-only)
MessageCount gives the number of messages available to read from the CANport.

7.10 IChannel

7.10.1 Methods of IChannel

procedure AddAsyncByteSample(Value: Byte; TimeStamp: Double);
AddAsyncByteSample adds an asynchronous data sample of the type Byte and its corresponding timestamp to a channel.
Parameters:
Value is the sample value to add to the channel.
TimeStamp is the corresponding timestamp of the sample.

procedure AddAsyncDoubleSample(Value: Double; TimeStamp: Double);
AddAsyncDoubleSample adds an asynchronous data sample of the type Double and its corresponding timestamp to a channel.
Parameters:
Value is the sample value to add to the channel.
TimeStamp is the corresponding timestamp of the sample.

procedure AddAsyncSingleSample(Value: Single; TimeStamp: Double);
AddAsyncSingleSample adds an asynchronous data sample of the type Single and its corresponding timestamp to a channel.
Parameters:
Value is the sample value to add to the channel.
TimeStamp is the corresponding timestamp of the sample.

procedure AddSingleSample(Value: Single);
AddSingleSample adds a synchronous data sample of the type Single to a channel.
Parameters:
Value is the sample value to add to the channel.

procedure AddSingleSamples(Count: Integer; Data: OleVariant; Timestamps: OleVariant);
AddSingleSamples allows adding a number of samples to a channel.
Parameters:
Count is the number of samples which will be added.
Data must contain a 1D-array of values of the type Single. The length of the array must be equal to Count.
TimeStamps must contain a 1D-array of timestamps of the type Double. The length of the array must be equal to Count.

function CreateConnection: IChannelConnection;
CreateConnection creates a connection to a channel in order to access its data. See IChannelConnection for detailed information.

function GetDBAddress: Integer;
GetDBAddress returns the pointer of the next position in the direct buffer to be written to.

procedure GetIBValues(Pos: Integer; out Min: Single; out Max: Single; out Ave: Single; out Rms: Single);
GetIBValues reads the intermediate buffer values Min, Max, Ave and RMS at the position specified by Pos. The property IChannel.IBPos can be read to determine the current position of the intermediate buffer. This procedure is implemented for automation applications developed in LabVIEW because, due to its data type, the property IChannel.IBValues is not accessible by LabVIEW. In programming environments other than LabVIEW the property IBValues can be used instead of GetIBValues.
Parameters:
Pos specified the position within the intermediate buffer to read from.
Min is the minimum value.
Max is the maximum value.
Ave is the average value.
Rms is the RMS value.

procedure GetIndex1(out IndexLevel: Integer; out I1: Integer; out I2: Integer; out I3: Integer; out I4: Integer; out I5: Integer);
GetIndex1 reads the indexes specifying a channel. This procedure is implemented for automation applications developed in LabVIEW because, due to its data type, the property IChannel.Index cannot be read by LabVIEW. In programming environments other than LabVIEW the property IChannel.Index can be used instead of GetIndex1.
Parameters:
IndexLevel defines the number of used indexes specifying a channel. This number depends on the kind of a channel.
Index1 defines the group a channel belongs to.
Index2…Index5 are group dependant indexes.
See Table 2 for detailed information on those indexes.

procedure GetRBValues(Pos: Integer; out Min: Single; out Max: Single; out Ave: Single; out Rms: Single); safecall;
GetRBValues reads the reduced buffer values Min, Max, Ave and RMS at the position specified by Pos. The property IChannel.RBPos can be read to determine the current position of the reduced buffer. This procedure is implemented for automation applications developed in LabVIEW because, due to its data type, the property IChannel.RBValues is not accessible by LabVIEW. In programming environments other than LabVIEW the property RBValues can be used instead of GetRBValues.
Parameters:
Pos specified the position within the reduced buffer to read from.
Min is the minimum value.
Max is the maximum value.
Ave is the average value.
Rms is the RMS value.

function GetScaledData: OleVariant;
GetScaledData returns scaled data from the direct buffer. The data type of scaled data is always “Single”. The return value is an OleVariant containing an array of data points.
If GetScaledData is used during measurement, the size of the array returned will increase with the measurement time because all the acquired data of the channel within the buffer is returned.

If GetScaledData is used for reloading acquired data from a dsd file in conjunction with ILoadEngine and ReloadBlock, GetScaledData returns an array which size is equal to IData.Samples. See ILoadEngine and IData for detailed information.

function GetScaledDataEx(Start: Integer; Count: Integer): OleVariant;
GetScaledDataEx is an extended version of GetScaledData. GetScaledDataEx returns scaled data from the direct buffer which is already linearized and allows specifying the start position from where to start retrieving data and the desired amount of data. The data type of scaled data is always “Single”.
Parameters:
Start defines the position within the direct buffer from where to start retrieving data.
Count is the amount of data being retrieved.

procedure GetScaledDataEx1(Start: Integer; Count: Integer; out Data: OleVariant);
GetScaledDataEx1 offers functionality equal to GetScaledDataEx but is implemented as a procedure and not as a function.
Start defines the position within the direct buffer from where to start retrieving data.
Count is the amount of data being retrieved.
Data contains the data retrieved from the direct buffer.

function GetTSData: OleVariant;
GetTSData retrieves the timestamp data of asynchronous channels. This function should be used in conjunction with GetScaledData for asynchronous channels. The returned OleVariant contains an array of timestamp values of the data type “Double”. The size of the array will be equal to the size of the array returned by GetScaledData.

function GetTSDataEx (Start: Integer; Count: Integer): OleVariant;
GetTSDataEx retrieves the timestamp data of asynchronous channels. This function should be used in conjunction with GetScaledDataEx or GetUnscaledDataEx for asynchronous channels. Similar to GetScaledDataEx, the function allows to specifying the start position from where to start retrieving timestamp data and the desired amount of timestamp data. The data type of timestamp data is “Double”.
Parameters:
Start defines the position within the direct buffer from where to start retrieving timestamp data.
Count is the amount of timestamp data being retrieved.

procedure GetTSDataEx1(Start: Integer; Count: Integer; out Data: OleVariant);
GetTSDataEx1 is similar to GetTSDataEx, but is implemented as a procedure and not as a function.
Parameters:
Start defines the position within the direct buffer from where to start retrieving timestamp data.
Count is the amount of timestamp data being retrieved.
Data contains the timestamp data retrieved from the direct buffer.

function GetUnscaledData: OleVariant;
GetUnscaledData returns unscaled data from the direct buffer. The return value is an OleVariant containing an array of data points.
If GetScaledData is used during measurement, the size of the array returned will increase with the measurement time because all the acquired data of the channel within the buffer is returned.

If GetUnscaledData is used for reloading acquired data from a dsd file in conjunction with ILoadEngine and ReloadBlock, GetUnscaledData returns an array which size is equal to IData.Samples. See ILoadEngine and IData for detailed information.

function GetUnscaledDataEx(Start: Integer; Count: Integer): OleVariant;
GetUnscaledDataEx is an extended version of GetUnscaledData. GetScaledDataEx returns unscaled data from the direct buffer which is already linearized and allows specifying the start position from where to start retrieving data and the desired amount of data.
Parameters:
Start defines the position within the direct buffer from where to start retrieving data.
Count is the amount of data being retrieved.

procedure GetUnscaledDataEx1(Start: Integer; Count: Integer; out Data: OleVariant);
GetUnscaledDataEx1 offers functionality equal to GetUnscaledDataEx but is implemented as a procedure and not as a function.
Start defines the position within the direct buffer from where to start retrieving data.
Count is the amount of data being retrieved.
Data contains the data retrieved from the direct buffer.

procedure IncDBSamples(Count: Integer);
To increment the position of the direct buffer.
Parameters:
Count is the amount of samples the direct buffer position will be increased.

function ScaleValue(Value: Single): Single;
ScaleValue returns the scaled value of the given unscaled value.
Parameters:
Value is the unscaled value which should be scaled.

procedure SetSRDiv(SRDiv: Integer);
SetSRDiv allows setting the sample rate divider.
Parameters:
SRDiv is the sample rate divider to be set.

procedure SetSRDivType(AType: TSRDivType);
SetSRDivType allows setting the type of sample rate divider.
Parameters:
AType is type to be applied which is of the data type TSRDivType. AType can be one of the following:
0…sdSkip
1…sdAverage
2…sdFilter4th
3…sdFilter6th
4…sdFilter8th

7.10.2 Properties of IChannel

property Async: WordBool; (read-only)
Async is True when the channel is asynchronous, otherwise it is False.

property Bytes: Integer; (read-only)
Bytes is the number of bytes used for one data sample of the specific channel. The number of bytes depends on the data type used for the channel.

property DataType: Integer; (read-only)
DataType is the data type of data samples of a channel. The data type can be one of the following:
0…Byte,
1…ShortInt,
2…SmallInt,
3…Word,
4…Integer,
5…Single,
6…Int64,
7…Double,

property DBBufSize: Integer; (read-only)
DBBUfSize is the size of the direct buffer. (See 3.1 The Buffer Structure)

property DBDataSize: Integer; (read-only)
DBDataSize is the size of the data stored in the direct buffer.

property DBPos: Integer; (read-only)
DBPos is the next position within the direct buffer to be written to.

property DBTimeStamp[Index: Integer]: Double; (read-only)
DBTimeStamp[Index] is the timestamp data of asynchronous data in the direct buffer. Index is the position within the direct buffer.

property DBValues[Index: Integer]: Single; (read-only)
DBValues[Index] is the data value in the direct buffer. Index is the position within the direct buffer.

property Description: WideString;
Description is a comment to a channel. Description is equal to "property Measurement". Description correspond to the description entered within the channel setup dialog (see mouse cursor in Figure 98) which can be reached via the setup button of a channel (see mouse cursor in Figure 97).

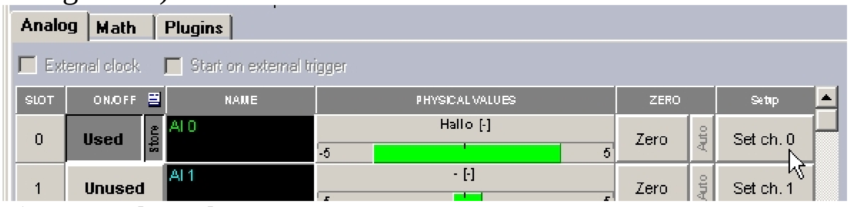
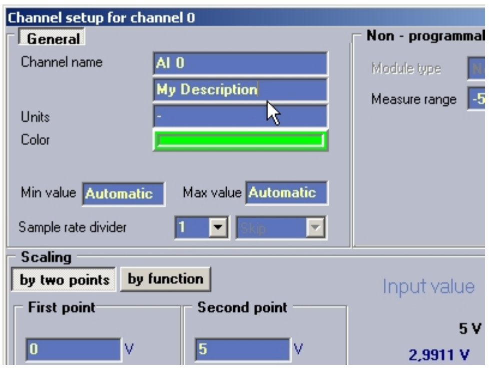


Figure 97 Channel Setup Button


Channel setup for channel 0

General

Channel name: AI 0

My Description

Units: -

Min value: Automatic Max value: Automatic

Sample rate divider: 1

Non - programmal

Module type

Measure range: -5

Scaling: by two points | by function

First point: 0 V

Second point: 5 V

Input value: 5 V, 2,9911 V

Figure 98 Channel Setup

property ExportOrder: Integer
ExportOrder allows specifying the order in which the channels are exported. The default value is –1 meaning not ordered. Channels having the default ExportOrder set to –1 will always be listed after any channel having a customized ExportOrder.
So, e.g. the order of channels can be similar to the following list of ExportOrder properties:
3
4
7
1000
7000
-1
-1
-1
…
This order will be applied to the channel lists shown in any screen too.
Using “Data >> Store Settings and Events”, these settings can be stored to the data file.

property Group: IChannelGroup; (read-only)
Group is a group containing the channels. E.g. channels of a plug-in have to be mounted to group 8.

property IBBufSize: Integer; (read-only)
IBBufSize is the size of the intermediate buffer. (See 3.1 The Buffer Structure)

property IBDataSize: Integer; (read-only)
IBDataSize is the size of the data within the intermediate buffer.

property IBPos: Integer; (read-only)
IBPos is the next position in the intermediate buffer to be written to.

property IBValues[Index: Integer]: T_ReducedRec; (read-only)
IBValues[Index] is a record of the intermediate buffer values at Index. Index defines the position within the intermediate buffer. The record contains the values Min, Max, Ave and RMS.
The property IBValues does not work in conjunction with LabVIEW. It is recommended to use the procedure IChannel.GetIBValues instead.

property Index: T_ChIndex; (read-only)
Index is a record of indexes specifying a channel. See T_ChIndex in 7.10.3 Types / Constants of IChannel.

The property Index is not available in LabVIEW. It is recommended to use the procedure IChannel.GetIndex1 instead.

property MainDisplayColor: Int64;
MainDiplayColor specifies the color of a channel. The default colors of DEWESoft and their order are as follows:
clLime, clAqua, clRed, clFuchsia, clBlue, clTeal, clOlive, clGreen,
$00FF8080, $00FF80FF, $0080FF00, $00FFFF80, $00D6DAD6, $0081ABB1, $00CF63C7, $0065CD92,
$0043E7EF, $005ED5D5, $00953AC9, $00F19A9C, $00AFE6A6, $00A0ADEB, $0055CAB9, $0013E6E6,
$0010CBCB, $00EA738A, $002E04EE, $004D5ED2, $009F9FA2, $00C67362, $00C67362, $00AA027C,
$00AA027C, $00BDFFED, clWhite, $005E0DE3, $008F8D61, $0021E4AA, $002C49D3, $00BC05B3,
$002E03BE, $0093FDF0, $00026458, $00BC4151, $0040A293, $00656569, $00B37015, $0002DBCB,
$0021FEF3, $0040B09A, $008CFFDF, $009A8BFE, $00716F4D, $005B49FE, $0033079A, $0001DA3D,
$00020DD2, $0074852E, $003F41A0, $00727070, clYellow, clSilver, clActiveCaption, clBtnFace;

property Measurement: WideString;
Measurement is equal to “property Description”. See there for detailed information.

property Name: WideString;
Name is the name of a channel.

property Offset: Double;
Offset is the value of the channel offset.

property RBBufSize: Integer; (read-only)
RBBufSize is the size of the reduced buffer. (See 3.1 The Buffer Structure)

property RBDataSize: Integer; (read-only)
RBDataSize is the size of the data within the reduced buffer.

property RBPos: Integer; (read-only)
RBPos is the next position in the reduced buffer to be written to.

property RBValues[Index: Integer]: T_ReducedRec; (read-only)
RBValues[Index] is a record of the reduced buffer values at Index. Index defines the position within the reduced buffer. The record contains the values Min, Max, Ave and RMS.
The property RBValues does not work in conjunction with LabVIEW. It is recommended to use the procedure IChannel.GetRBValues instead.

property Scale: Double;
Scale is the scaling value of a channel.

property Shown: WordBool;
Shown defines whether a channel is shown or not.

property SRDiv: Integer; (read-only)
SRDiv is the sample rate divider set for a specific channel. SRDiv corresponds to the setting within the channel setup shown at the position of the mouse cursor in Figure 99.

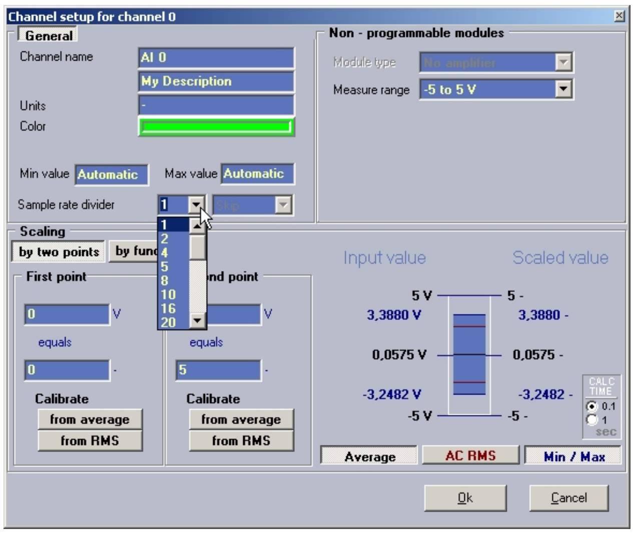


Figure 99 Sample rate divider

property TypicalMaxValue: Single; (read-only)
TypicalMaxValue is the maximum value of the y-axis of a graph displaying a certain channel.

property TypicalMinValue: Single; (read-only)
TypicalMinValue is the minimum value of the y-axis of a graph displaying a certain channel.

property Unit_: WideString;
Unit_ defines the measurement unit of a channel.

property Used: WordBool;
Used specifies whether a channel is used or not, corresponding to the “Used” button at each channel in the measurement setup (Cf. Figure 100).

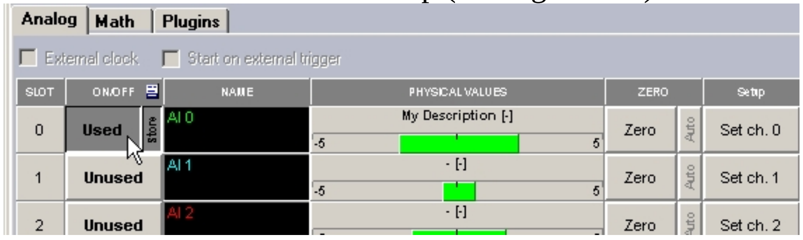


Figure 100 "Used" Button

property UserScaleMax: Single;
UserScaleMax offers the ability to change the maximum value of the scaling of the y-axis of a channel.

property UserScaleMin: Single;
UserScaleMin offers the ability to change the minimum value of the scaling of the y-axis of a channel.

7.10.3 Types / Constants of IChannel

T_ChIndex
T_ChIndex = packed record
IndexLevel: Smallint;
Index1: Integer;
Index2: Integer;
Index3: Integer;

Index4: Integer;
Index5: Integer;
end;
T_ChIndex is a record of indexes containing information about a channel.
IndexLevel defines the number of used indexes, which depends on the kind of a channel.
Index1 always defines the group a channel belongs to. Index2 to Index5 are group dependant indexes, which can be interpreted as described in Table 2:

Table 2 T_ChIndex

Channel type	IndexLevel (number of used indexes)	Index1	Index2	Index3	Index4	Index5
AI	2	1	channel number (0…number of channels - 1)			
	Example: Channel “AI 3” → (2; 1, 3)					
DI	3	100	port number	bit number		
	Example: Channel “bit no 3 on DI port 2” → (3; 100, 2, 3)					
Counter	2	200	counter number (0…number of counters - 1)			
	Example: Channel “counter 0” → (2; 200, 0)					
PAD	3	1000	slot number	channel number on module		
	Example: Channel “PAD slot 3/ Ch 2” – (3; 1000, 3, 2)					
CAN bus message	3	2000	port number	message id		
	Example: channel “RPM in Motor message on CAN port 0” → (3; 2000, 0, 33 [Motor message id])					
CAN bus signal	4	2000	port number	message id	start bit in message	
	Example: channel “RPM in Motor message on CAN port 0” → (4; 2000, 0, 33 [Motor message id], 16 [start bit of RPM channel])					
MATH (old, DEWESoft 6.3.x)	2	3000	math channel number			
	Example: Channel “Math 5” → (2; 3000, 5)					
MATH (new, DEWESoft 6.4)	4	7000	number of the math object. (The line in the math channel setup.)	Module number. E.g. basic statistics has as much modules as input channels.	The output channel number of one math module.	
	Example: Channel “Math object 5, module 0, output channel 1” → (4; 7000, 5, 0, 1)					
GPS	2	4000	GPS channel (0…X absolute, 1…Y absolute, …, 7…used satellites;)			
	Example: channel “Y absolute” → (2; 4000, 1)					

Power	5	6000	Module number (0…#Modules-1)	Phase
				Phase 1 = 1
				Phase 2 = 2
				Phase 3 = 3
				Total = 4
					Voltages = 0;
					Currents = 1;
					Power = 2;
					THD = 3;
					Unbalance= 5;
					Disturbances= 6;
					Impedances= 7;
					Flicker= 8;
						0…Number of channels

	Example: (5; 6000; 0; 1; 3; 0) THD 0 of 1st power module.					
AO	3	200000	AO channel sub-type: 10000…AO/Freq	AO channel number.		

			0…AO 1000…AO/Ampl 2000…AO/Phase 3000…AO/Offset			
	Example: (3; 200000, 2000, 0) Phase of AO 0.					

T_ReducedRec
T_ReducedRec = packed record
Min: Single;
Max: Single;
Ave: Single;
Rms: Single;
end;
T_ReducedRec is a record of reduced data values of the following kinds: Minimum, Maximum, Average and RMS.

7.11 IChannelConnection

7.11.1 Methods of IChannelConnection

function GetDataBlocks(NumBlocks: Integer): OleVariant;
GetDataBlocks returns a certain number of data blocks from the direct buffer. The return value is an OleVariant containing an array of values of the type “Single”. The size of the array will be NumBlocks times the size of one data block. The size of the data blocks and the position within the direct buffer from where data is retrieved are dependant on the properties AType and BlockSize. See there for detailed information.
Parameters:
NumBlocks defines the number of data blocks to read.

procedure GetDataBlocks1(NumBlocks: Integer; out Data: OleVariant);
GetDataBlocks1 offers the same functionality as GetDataBlocks but is implemented as a procedure instead of a function.
Parameters:
NumBlocks defines the number of data blocks to read.
Data contains an array of values of the type “Single”.

function GetDataValues(NumValues: Integer): OleVariant;
GetDataValues returns the specified amount of data values from the direct buffer. The return value is an OleVariant containing an array of values of the type “Single”. The size of the array will be equal to NumValues. If GetDataValues is used, IChannelConnection.AType would be set to ctLast in most cases. If IChannelConnection.AType is set to ctNew, data will be returned only if the specified amount of new data (not read yet) is available. If GetDataValues is used for reading asynchronous data, the corresponding timestamps can be read by GetTSValues. If those two functions are used, GetTSValues has to be called before GetDataValues is called.
Parameters:
NumValues specifies the amount of data values to retrieve from the direct buffer.

procedure GetDataValues1(NumValues: Integer; out Data: OleVariant);
GetDataValues1 offers the same functionality as GetDataValues but is implemented as a procedure instead of a function. If GetDataValues is used, IChannelConnection.AType would be set to ctLast in most cases. If IChannelConnection.AType is set to ctNew, data will be

returned only if the specified amount of new data (not read yet) is available. If GetDataValues1 is used for reading asynchronous data, the corresponding timestamps can be read by GetTSValues1. If those two functions are used, GetTSValues1 has to be called before GetDataValues1 is called.
Parameters:
NumValues specifies the amount of data values to retrieve from the direct buffer.
Data contains an array of values of the type "Single".

procedure Reset;
To reset a ChannelConnection at a certain moment. In the case of multiple connections, calling StartDataSync before resetting the connections and calling EndDataSync after having reset each connection would be necessary for synchronization. (Remark: Reset is called by CreateConnection.)

procedure Start;
Start resets the ChannelConnections to the start of the acquisition.

function GetTSBlocks(NumBlocks: Integer): OleVariant;
GetTSBlocks has to be used in conjunction with GetDataBlocks in case of asynchronous channels. When those two functions are used, GetTSBlocks has to be called before GetDataBlocks. The return value is an OleVariant containing an array of timestamp values of the type "Double".
Parameters:
NumBlocks specifies the number of blocks to read.

procedure GetTSBlocks1(NumBlocks: Integer; out Data: OleVariant);
GetTSBlocks1 offers the same functionality as GetTSBlocks, but is implemented as a procedure instead of a function.
Parameters:
NumBlocks specifies the number of blocks to read.
Data contains an array of timestamps of the type "Double".

function GetTSValues(NumValues: Integer): OleVariant;
GetTSValues returns the specified amount of timestamp values from the direct buffer. The return value is an OleVariant containing an array of timestamps of the type "Double". The size of the array will be equal to NumValues. If GetTSValues is used, IChannelConnection.AType would be set to ctLast in most cases. If IChannelConnection.AType is set to ctNew, data will be returned only if the specified amount od new data (not read yet) is available. GetTSValues is intended to be used in conjunction with GetDataValues. When both of these functions are used, GetTSValues has to be called before GetDataValues.
Parameters:
NumValues specifies the amount of timestamps to retrieve from the direct buffer.

procedure GetTSValues1(NumValues: Integer; out Data: OleVariant);
GetTSValues1 offers the same functionality as GetTSValues but is implemented as a procedure instead of a function. If GetSTValues1 is used, IChannelConnection.AType has to be set to ctLast. GetTSValues1 is intended to be used in conjunction with GetDataValues1. When both of these functions are used, GetTSValues1 has to be called before GetDataValues1.
Parameters:
NumValues specifies the amount of timestamps to retrieve from the direct buffer.

Data contains an array of timestamps of the type “Double”.

7.11.2 Properties of IChannelConnection

property AType: ConnTypes;
AType defines the type of the ChannelConnection and therefore alters the behavior of GetDataBlocks and GetDataBlocks1. AType can be one of the following:
0… ctLast… for reading a certain amount (specified by “BlockSize”) of the last data; Figure 101 illustrates two subsequent calls of GetDataBlocks when AType is set to ctLast.
1… ctOverlap… for reading blocks (size specified by “BlockSize”) of data; the percentage of an overlap of the data blocks can be specified by Overlap; Figure 102 illustrates two subsequent calls of GetDataBlocks when AType is set to ctOverlap and Overlap is set to 0%. In Figure 103 Overlap is set to 50%.
2… ctTrigger… for reading the data which is displayed in the Scope-screen on triggers (use App.SetScopeParams before, in order to setup the trigger of the scope and activate the scope trigger by calling App.SetScopeTriggerUsed(True); see chapter 7.6.1 Methods of IApp for detailed information.).
3…ctNew… for reading a certain amount (specified by “BlockSize” or “NumValues”) of the last data only if such an amount of new data (not read yet) is available.

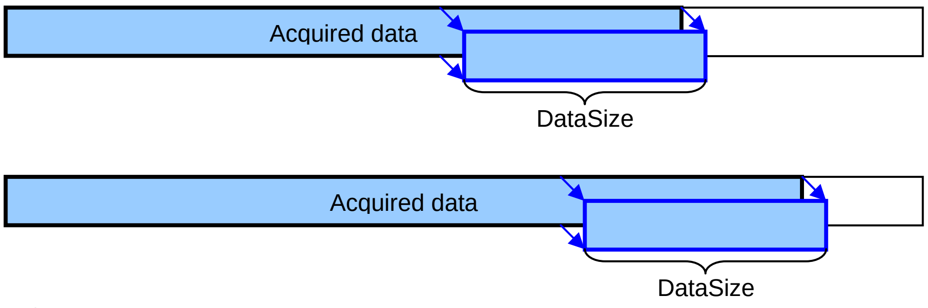


Figure 101 AType set to ctLast

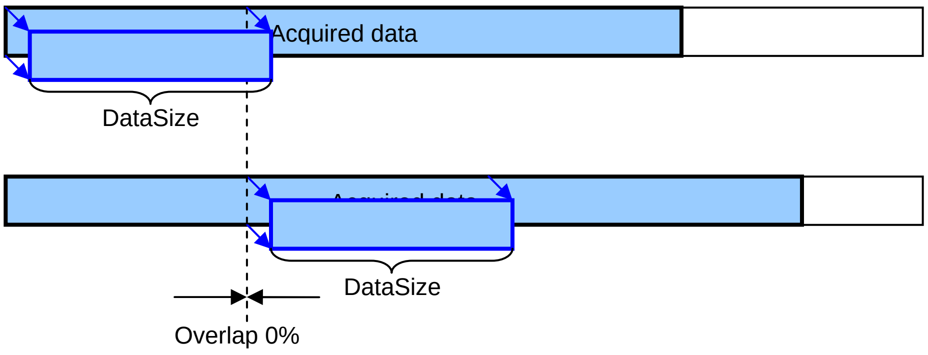


Figure 102 AType set to ctOverlap, Overlap set to 0

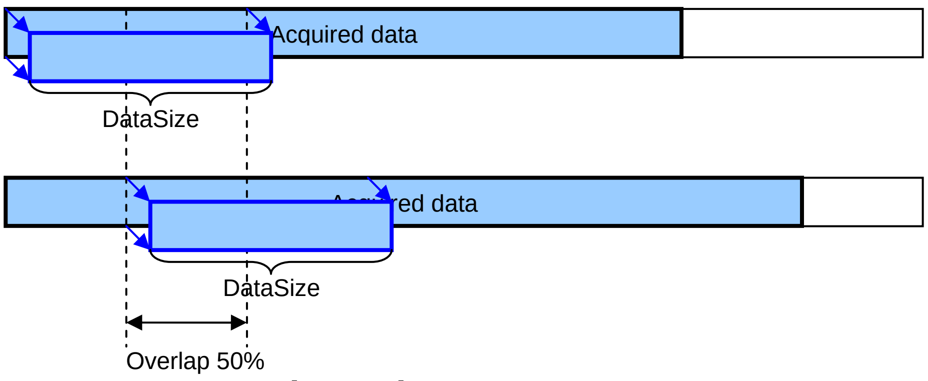


Figure 103 AType set to ctOverlap, Overlap set to 50

property BlockSize: Integer;
BlockSize specifies the size of the data block(s) to read using GetDataBlocks or GetDataBlocks1.

property Channel: IChannel; (read-only)
See IChannel for detailed information.

property NumBlocks: Integer; (read-only)
NumBlocks is the number of available data blocks of the size specified by BlockSize within the direct buffer not read yet.

property NumValues: Integer; (read-only)
NumValues is the number of available values within the direct buffer not read yet.

property Overlap: Integer;
Overlap defines how much the data blocks read by two subsequent calls of GetDataBlocks or GetDataBlocks1 should overlap each other, when AType is set to ctOverlap. See Figure 102 and Figure 103 for illustration.

7.11.3 Types / Constants of IChannelConnection

```
type
  ConnTypes = TOleEnum;
const
  ctLast 0;
  ctOverlap = 1;
  ctTrigger = 2;
..ctNew = 3;
```

7.12 IChannelGroup

7.12.1 Methods of IChannelGroup

function GetIndexName(Index: T_ChIndex): WideString;
GetIndexName returns the name of the ChannelGroup specified by Index.

7.12.2 Properties of IChannelGroup

property Count: Integer; (read-only)
Count is the number of channels in a group.

property ExportRate: Integer;
ExportRate defines the sample rate for exporting asynchronous data to DIAdem. This property can be set separately for each group.

property Item[Index: Integer]: IChannel; (read-only)
Item[Index] is the channel item of a group. This item is of the type IChannel.

property Name: WideString; (read-only)
Name is the name of a group. The name of a group cannot be changed.

7.13 IChannelGroup2

IChannelGroup2 is derived from IChannelGroup, so it provides the properties and methods of IChannelGroup including those of IChannelGroup2

7.13.1 Methods of IChannelGroup2

function GetIndexNameShort(Index: T_ChIndex): WideString;
GetIndexNameShort returns the name of the index in an abbreviated version.

7.14 IChannelGroups

7.14.1 Properties of IChannelGroups

property Count: Integer; (read-only)
Count indicates the number of channel groups.

property Item[Index: Integer]: IChannelGroup; (read-only)
Item[Index] is one of the channel groups which is of the type IChannelGroup

7.15 IChannelList

7.15.1 Properties of IChannelList

property Count: Integer; (read-only)
Count is the number of channels within the channel list.

property Item[Index: Integer]: IChannel; (read-only)
Item[Index] is one of the channels from the channel list. The item is of the type IChannel.

7.16 IDaqChannel

In order to access the properties of IDaqChannel a channel of the type IChannel must be casted to the type IDaqChannel. The properties of IDaqChannel are available only for analog channels.

7.16.1 Methods of IDaqChannel

procedure SetCardGain(Gain: Single);
SetCardGain sets the gain of a DAQ card.
Parameters:
Gain is the new gain value to be set.

7.16.2 Properties of IDaqChannel

property AutoZero: WordBool;
AutoZero defines whether the zero function of a DAQ channel is set to automatic or not. Cf. to the Auto button of a DAQ channel at the mouse cursor position in Figure 104.

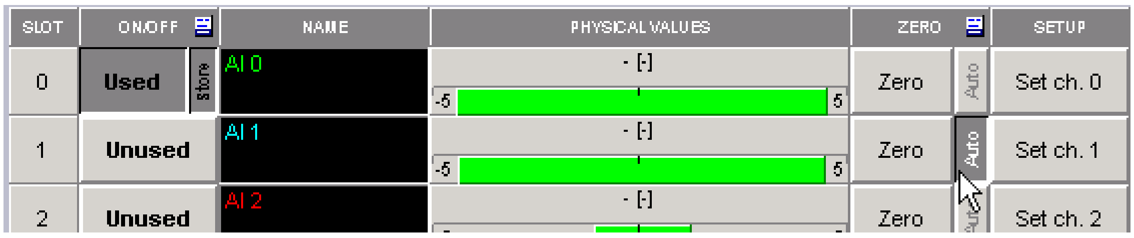


Figure 104 AutoZero

property CardBitResolution: Integer; (read-only)
CardBitResolution returns the resolution of a DAQ card in bits.

property CardGain: Double; (read-only)
Card Gain is the gain value whch is set at a DAQ card.

property CardOffset: Double; (read-only)
CardOffset is the offset value setfor a DAQ card.

property ModuleGain: Double; (read-only)
ModuleGain is the gain value set at a module.

property ModuleOffset: Double; (read-only)
ModuleOffset is the offset value set at a module.

property ModuleType: Integer; (read-only)
ModuleType is the type of a module.
The association of the number to the modules is as follows:
2… DAQP-Bridge
5… DAQP-Charge
8… DAQP-Freq
9… DAQP-Acc
23… DAQP-Charge-A
24… DAQP-Bridge-A
26… DAQP-Freq-A
27… DAQP-ACC-A
30… DAQP-Charge-B
31… DAQP-Bridge-B
34… DAQP-V-A
35… DAQP-V-B
36… MDAQ

7.17 IDaqData

7.17.1 Methods of IDaqData

function CopyToString: WideString;
CopyToString copies the module parameters to a string.

function CopyUnitToString: WideString;
CopyUnitToString copies the unit of measurement to a string.

function ModuleAmpl: Single;
ModuleAmpl returns the module scale factor.

function ModuleOffset: Single;
ModuleOffset returns the module offset factor.

function ShortCopyToString: WideString;
ShortCopyToString copies the module parameters in a short version (only the values) to a string.

7.17.2 Properties of IDaqData

property Address: Smallint;
Address defines the address of the DAQ-module.

property CurrentSource: Byte;
CurrentSource defines the current source of ACC modules. Valid values for this property are the following:
0…4mA
1…8mA

property DaqNNames[ANameCode: Smallint]: WideString; (read-only)
DaqNNames provides the name of a DAQN-module referenced by ANameCode.

property DaqNNamesCount: Integer; (read-only)
DaqNNamesCount is the number of available DAQN-modules.

property Filters[AFilterCode: Byte]: WideString; (read-only)
Filters provides the name of a filter corresponding to AFilterCode.

property FilterCode: Byte;
FilterCode defines the filter of a DAQ-module.

property FiltersCount: Integer; (read-only)
FiltersCount is the number of filters available for DAQ-modules.

property FREQAInputCoupling: Shortint;
FREQAInputCoupling defines the input coupling-type of a FREQ-A-module, which can be:
0…AC
1…DC

property FREQAOutputFilter: Shortint;
FREQAOutputFilter defines the output filter-type of a FREQ-A-module, which can be:
0…slow
1…fast

property FREQATriggerLevel: Single;
FREQATriggerLevel defines the trigger level of a FREQ-A-module.

property HighpassType: Byte;
HighpassType defines the type of the highpass-filter set at a DAQ-module. Possible values are dependent on the module type.

property ICPInput: Byte;
ICPinput defines whether a charge-module is set to ICP or to charge. The meaning is dependant on the module-type.

property ModuleError: Byte; (read-only)
ModuleError is the error code of a module. Refer to the technical reference of the modules for dtailed information.

property ModuleType: Smallint; (read-only)
ModuleType defines the type of a module.
The association of the number to the modules is as follows:
2… DAQP-Bridge
5… DAQP-Charge
8… DAQP-Freq
9… DAQP-Acc
23… DAQP-Charge-A
24… DAQP-Bridge-A
26… DAQP-Freq-A
27… DAQP-ACC-A
30… DAQP-Charge-B
31… DAQP-Bridge-B
34… DAQP-V-A
35… DAQP-V-B
36… MDAQ

property Name: WideString; (read-only)
Name provides the name of a module.

property Overflow: Byte;
Overflow will be 1 if an overflow has occurred.

property RangeCode: Byte;
RangeCode is the code denoting the measurement range of the module. Refer to the technical reference of the modules for detailed information.

property Ranges[ARangeCode: Integer]: WideString; (read-only)
Ranges allows to retrieve the range settings as a string according to the range code.
ARangeCode must be in the range of 0…RangesCount–1.

property RangesCount: Integer; (read-only)
RangesCount is the number of valid ranges of a module.

property Remote: Byte;
Remote denotes whether the module is set to local or remote mode. Refer to the technical reference of the modules for detailed information.

property VRange: Byte;
VRange is only used with DAQP-FREQ-module to indicate the trigger voltage input range.

7.18 IDaqGroup

7.18.1 Properties of IDaqGroup

property Count: Integer; (read-only)
Count is the number of channels of the type IDaqChannel within DaqGroup.

property Item[Index: Integer]: IDaqChannel; (read-only)
Item[] is the list of DaqGroups channel items of the type IDaqChannel. Index is the list index, which is in the range of 0…Count-1;

7.19 IData

7.19.1 Methods of IData

procedure ApplyChannels;
ApplyChannels has to be called after channels of a plug-in are mounted by any user-action (button-click).

procedure BuildChannelList;
BuildChannelList updates the properties AllChannels and UsedChannels. This procedure should be called before reading one of these properties after some channels have been set to used or not.

procedure EndDataSync;
EndDataSync has to be used together with StartDataSync in conjunction with IChannelConnection for synchronous manipulation of multiple channels. See 7.11 for details.

function FindChannel(const Name: WideString): IChannel;
FindChannel allows finding a channel by means of its name.
Parameters:
Name is the name of the requested channel. It must be the exact name of an existing channel in order to find the channel. The function is case-sensitive.

function FindChannelByIndex(Index: T_ChIndex): IChannel;
FindChannelByIndex allows finding a channel by its indexes.
Parameters:
Index is of the type T_ChIndex. See T_ChIndex in 7.10.3 Types / Constants.
FindChannelByIndex does not work in conjunction with LabVIEW. It is recommended to use IData.FindChannelByIndex1 instead.

function FindChannelByIndex1(IndexLevel: Integer; I1: Integer; I2: Integer; I3: Integer; I4: Integer; I5: Integer): IChannel;
FindChannelByIndex1 has the same functionality as FindChannelByIndex, namely finding a channel by its indexes. It is implemented because FindChannelByIndex does not work in conjunction with LabVIEW.
Parameters:
IndexLevel defines the number of used indexes for specifying a channel.
I1 is the group a channel corresponds to.
I2…I5 are the channel specific indexes.
See Table 2 for detailed information about the channel indexes.

function GetIndexName(Index: T_ChIndex): WideString;
GetIndexName returns the identifier specified by Index.
Parameters:
Index is of the type T_ChIndex. See T_ChIndex in 7.10.3 Types / Constants.
GetIndexName does not work in conjunction with LabVIEW. It is recommended to use IData.GetIndexName1 instead.

function GetIndexName1(IndexLevel: Integer; I1: Integer; I2: Integer; I3: Integer; I4: Integer; I5: Integer): WideString;
GetIndexName1 retunes the identifier specified by its index.
Parameters:
IndexLevel specifies which index' name should be returned. If IndexLevel is 1, the name of Index1 is returned, and so on.
I1 is the group a channel corresponds to.
I2…I5 are the channel specific indexes.
See Table 2 for detailed information about the channel indexes.

function GetIndexNameShort(ChIndex: T_ChIndex): WideString;
GetIndexNameShort returns the short identifier specified by Index.
Parameters:
Index is of the type T_ChIndex. See T_ChIndex in 7.10.3 Types / Constants .
GetIndexName does not work in conjunction with LabVIEW. It is recommended to use IData.GetIndexNameShort1 instead.

function GetIndexNameShort1(IndexLevel: Integer; I1: Integer; I2: Integer; I3: Integer; I4: Integer; I5: Integer): WideString;
GetIndexNameShort1 retunes the identifier specified by its index.
Parameters:
IndexLevel specifies which index' name should be returned. If IndexLevel is 1, the name of Index1 is returned, and so on.
I1 is the group a channel corresponds to.
I2…I5 are the channel specific indexes.
See Table 2 for detailed information about the channel indexes.

procedure GetSamplesAcquired(out Mid: Integer; out Dir: Integer);
GetSamplesAcquired returns information about the amount of data acquired.
Parameters:
Mid is the number of complete data blocks of the size specified by the property Samples. (See Properties of IData)

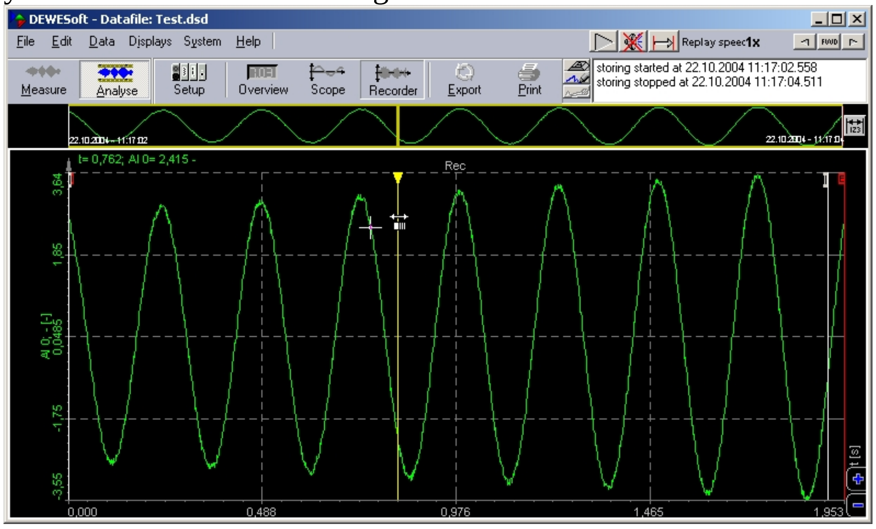
procedure StartDataSync;
StartDataSync has to be used together with EndDataSync in conjunction with IChannelConnection for synchronous manipulation of multiple channels. See 7.11 for details.

7.19.2 Properties of IData

property ActiveChannels: IChannelList; (read-only)
Do **not** use ActiveChannels because it is implemented for internal use only. Use AllChannels or UsedChannels instead!
ActiveChannels is a list of the active channels, which is of the type IChannelList. Active channels are those channels which are available according to the hardware setup.

property AllChannels: IChannelList; (read-only)
AllChannels is a list of all available channels, which is of the type IChannelList. (Cf. IData.UsedChannels). Call IData.BuildChannelList before reading the property to update it, if some channels have been set to used or not before.

property CurrentPos: T_RecordPosition;
CurrentPos is used in analyze mode of DEWESoft. It corresponds to the position of the yellow cursor as shown in Figure 105.



storing started at 22.10.2004 11:17:02.558
storing stopped at 22.10.2004 11:17:04.511

Figure 105 Current Position Cursor

property EndStamp: T_RecordPosition;
EndStamp is the last possible position of the yellow cursor.

property ExternalClock: Integer; (read-only)
ExternalClock is the external clock definition in clocks per revolution.

property Groups: IChannelGroups; (read-only)
Groups are the groups which data is dedicated to. Each kind of channel belongs to a certain group. E.g. the data group for plug-in data is 8.

property SampleRate: Integer; (read-only)
SampleRate is the sampling rate not depending on the factor of the “reduced rate”.

property Samples: Integer; (read-only)
Samples is the number of data samples within one data block.

property ShownChannels: IChannelList; (read-only)
ShownChannels is **not** recommended to use! Use AllChannels or Used Channels instead! ShownChannels is a list of the channels being set to “used” (See the mouse cursor in Figure 106 pointing at the “Used” button). This property is accurate only when the application is in measure mode and the setup screen is already left. This list is of the type IChannelList.

property StartStamp: T_RecordPosition;
StartStamp is the first possible position of the yellow cursor within the analyse mode of DEWESoft.

property StartStoreTime: TDateTime; (read-only)
StartStoreTime is the absolute time of the start of storing.

property StartStoreTimeUTC: TDateTime; (read-only)
StartStoreTimeUTC is the UTC of the start of storing.

property UsedChannels: IChannelList; (read-only)
UsedChannels is a list of the channels being set to “used” (See the mouse cursor in Figure 106 pointing at the “Used” button). This list is of the type IChannelList. (Cf. IData.AllChannels) Call IData.BuildChannelList before reading the property to update it, if some channels have been set to used or not before.

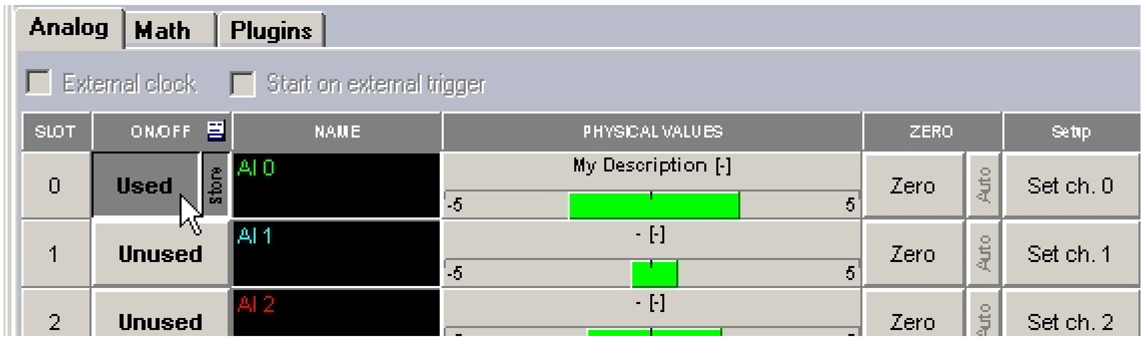


Figure 106 "Used" Button

7.20 IDataSection

7.20.1 Methods of IDataSection

function ReadData(const Channel: IChannel; out Timestamps: OleVariant): OleVariant;
ReadData reads the whole set of data of a section. A section is the part between a start and a stop event. The data section is independent of the data blocks. The data is returned as an OleVariant containing an array of data values of the type Single.
Parameters:
Channel defines the channel of which data should be retrieved.
Timestanps returns an OleVariant containing an array of timestamp data of the section. Timestamps are of the type Double.

procedure ReadData1(const Channel: IChannel; out Data: OleVariant; out Timestamps: OleVariant);
ReadData1 is similar to ReadData but is implemented as a procedure instead of a function.
Parameters:
Channel defines the channel of which data should be retrieved.
Data returns an OleVariant containing an array of measurement data of the section. The array consists of values of the type Single.
Timestanps returns an OleVariant containing an array of timestamp data of the section. Timestamps are of the type Double.

7.20.2 Properties of IDataSection

property DataCount: Integer; (read-only)
DataCount is the number of samples within a DataSection.

property Time: TDateTime; (read-only)
Time is the absolute start time of a section.

property TrigPos: Integer; (read-only)
TrigPos is the number of the sample where the trigger event occurred, counted from the start of the section.

7.21 IDataSections

7.21.1 Properties of IDataSections

property Count Integer; (read-only)
Count is the number of data sections where measurement data is stored.

property Item[Index: Integer]: IDataSection; (read-only)
Item[Index] is a section where data is stored, which is of the type IDataSection. Cf. 7.20.

7.22 IEvent

7.22.1 Properties of IEvent

property Data: OleVariant; (read-only)
Data contains special information about an event. E.g. this can be text or voice data. (This function is not implemented yet.)

property PosDir: Integer; (read-only)
PosDir is the position of an event within a data block.

property PosMid: Integer; (read-only)
PosMid is the number of the data block where the event has happened.

property TimeStamp: Double; (read-only)
TimeStamp is the time stamp of an event in seconds.

property Type_: Integer; (read-only)
Type_ defines the type of an event which can be one of the following:
1 indicating “start”,
2 indicating “stop”,
3 indicating “trigger”,
11 indicating “VStart” (video start storing)
12 indicating “VStop” (video stop storing)
20 indicating “Keyboard”
21 indicating “Notice”
22 indicating “Voice”
24 indicating “Module” (example: Bridge shunt on/off).

7.23 IEventList

7.23.1 Properties of IEventList

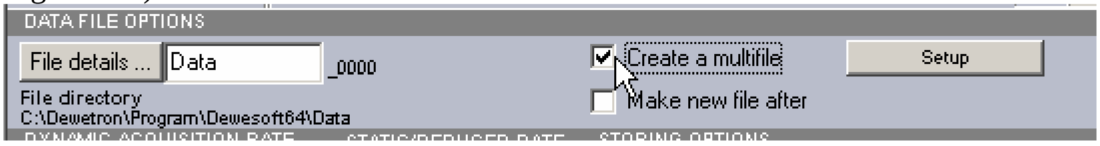
property Count: Integer; (read-only)
Count is the number of events.

property Item[Index: Integer]: IEvent; (read-only)
Item[Index] is the even item of the type IEvent. See 7.22.

7.24 IFileNameSettings

7.24.1 Properties of IFileNameSettings

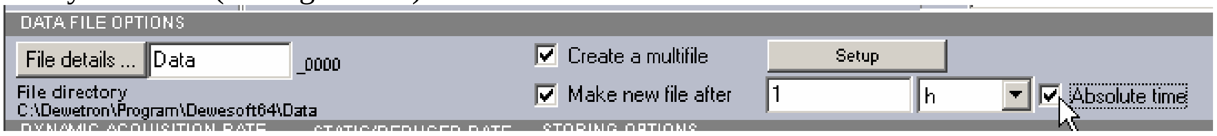
property AutoCreate: WordBool;
AutoCreate determines whether a data file name should be generated automatically (Cf. Figure 107).



Data _0000
File directory C:\Dewetron\Program\Dewesoft64\Data
☑ Create a multifile
☐ Make new file after

Figure 107 Data File Options – Create a multifile

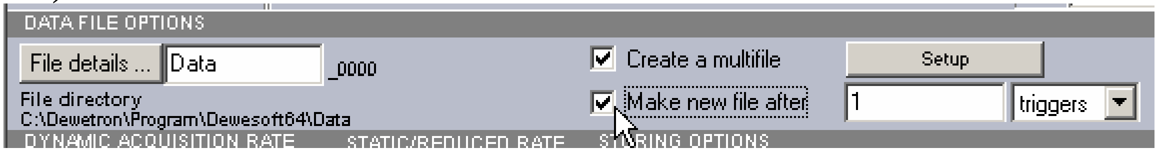
property AutoFlipAbsTime: WordBool;
AutoFlipAbsTime defines whether absolute time is used or not, in case of the Unit being set to any time unit (Cf. Figure 108).



Data _0000
File directory C:\Dewetron\Program\Dewesoft64\Data
☑ Create a multifile
☑ Make new file after 1 h
☑ Absolute time

Figure 108 Data File Options – Absolute time

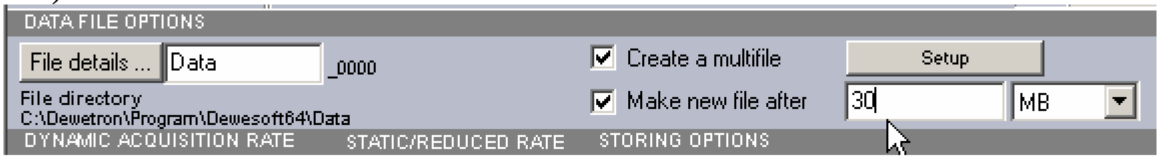
property AutoFlipFile: WordBool;
AutoFlipFile defines whether a new file should be created after a certain criterion (cf. Figure 109).



Data _0000
File directory C:\Dewetron\Program\Dewesoft64\Data
☑ Create a multifile
☑ Make new file after 1 triggers

Figure 109 Data File Options – Make new file after

property AutoFlipSize: Single;
AutoFlipSize is the size of a certain unit after which a new file will be created (cf. Figure 110).



Data _0000
File directory C:\Dewetron\Program\Dewesoft64\Data
☑ Create a multifile
☑ Make new file after 30 MB

Figure 110 Data File Options – Size

property AutoFlipUnit: Integer;
AutoFlipUnit is the unit of the AutoFlipSize which can be MB, h, min, sec or triggers (cf. Figure 111). The value assigned to AutoFlipUnit refers to the list index of the possible selections:
0…MB
1…h
2…min
3…sec
4…triggers

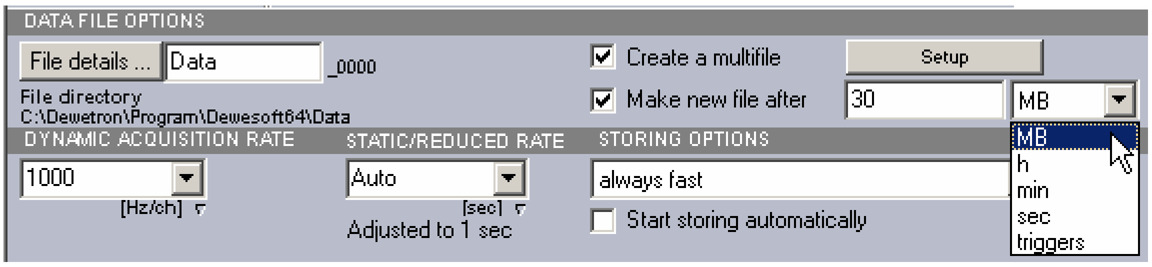

DATA FILE OPTIONS
File details ... Data _0000
File directory
C:\Dewetron\Program\Dewesoft64\Data
☑ Create a multifile
Setup
☑ Make new file after 30 MB
MB
h
min
sec
triggers
DYNAMIC ACQUISITION RATE
1000 [Hz/ch]
STATIC/REDUCED RATE
Auto [sec]
Adjusted to 1 sec
STORING OPTIONS
always fast
☐ Start storing automatically

Figure 111 Data File Options – Unit

property BaseFileName: WideString;

BaseFileName is the file name prefix for automatic naming of multi files (cf. Figure 112).

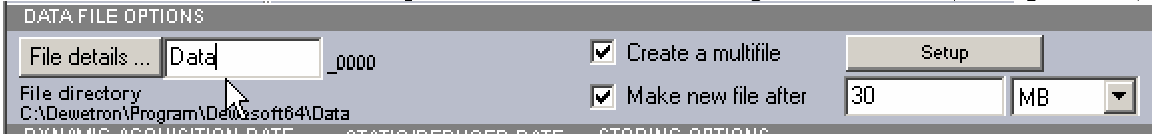

DATA FILE OPTIONS
File details ... Data _0000
File directory
C:\Dewetron\Program\Dewesoft64\Data
☑ Create a multifile
Setup
☑ Make new file after 30 MB

Figure 112 Data File Options – File name

property MultiFileStartIndex: Integer;

MultiFileStartIndex is the number to start from for the naming of multi files (cf. Figure 113).

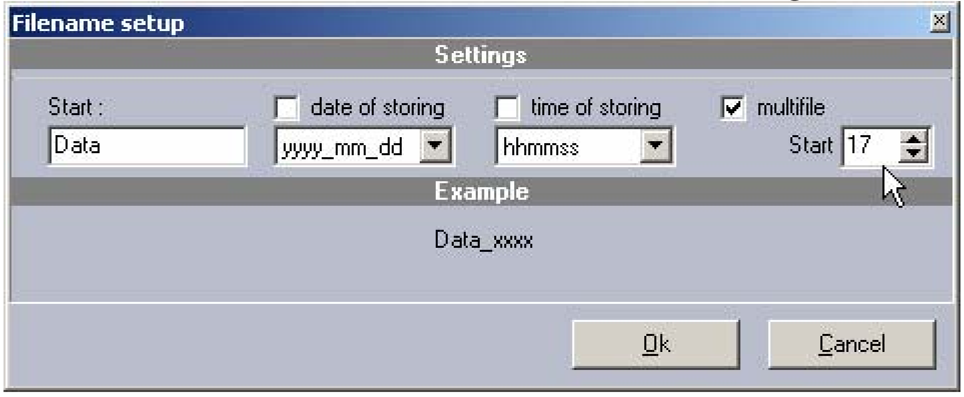

Filename setup
Settings
Start : Data
☐ date of storing yyyy_mm_dd
☐ time of storing hhmmss
☑ multifile
Start 17
Example
Data_xxxx
Ok
Cancel

Figure 113 Filename setup – Start index

property UseDate: WordBool;

UseDate determines whether the date should be included in the file name (cf. Figure 114).

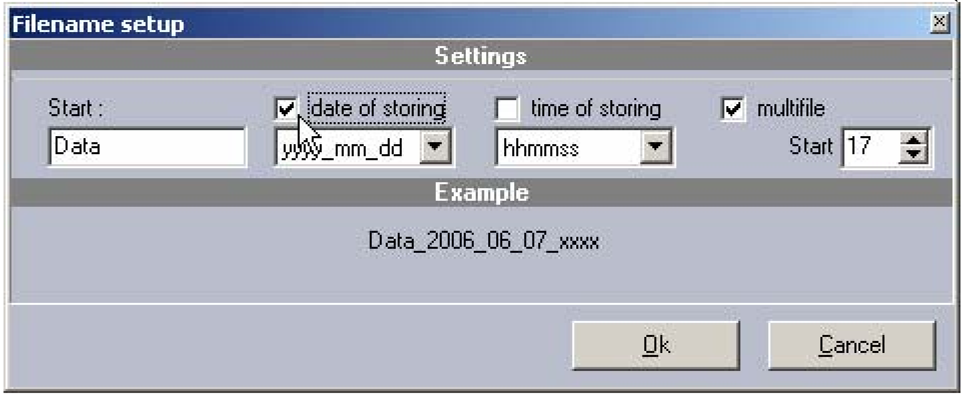
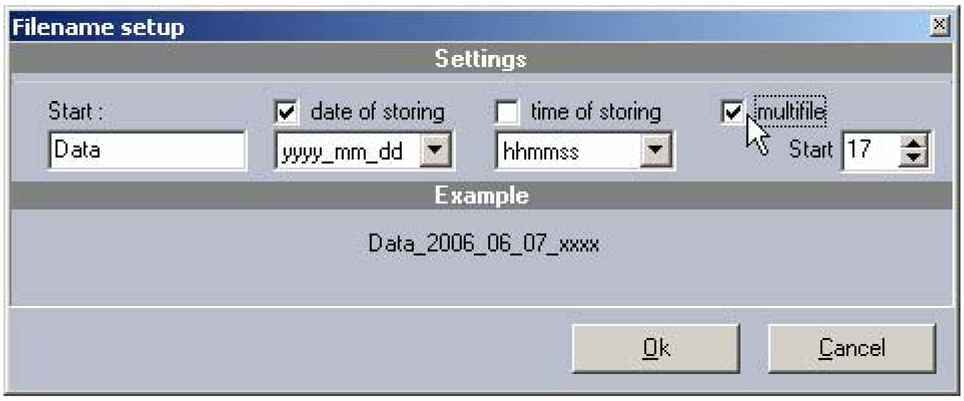

Filename setup
Settings
Start : Data
☑ date of storing yyyy_mm_dd
☐ time of storing hhmmss
☑ multifile
Start 17
Example
Data_2006_06_07_xxxx
Ok
Cancel

Figure 114 Filename setup – Date of storing

property UseMultiFile: WordBool;

UseMultiFile determines whether to use the multi file option or not (cf. Figure 115).


Filename setup

Settings

Start : Data

☑ date of storing: yyyy_mm_dd

☐ time of storing: hhmmss

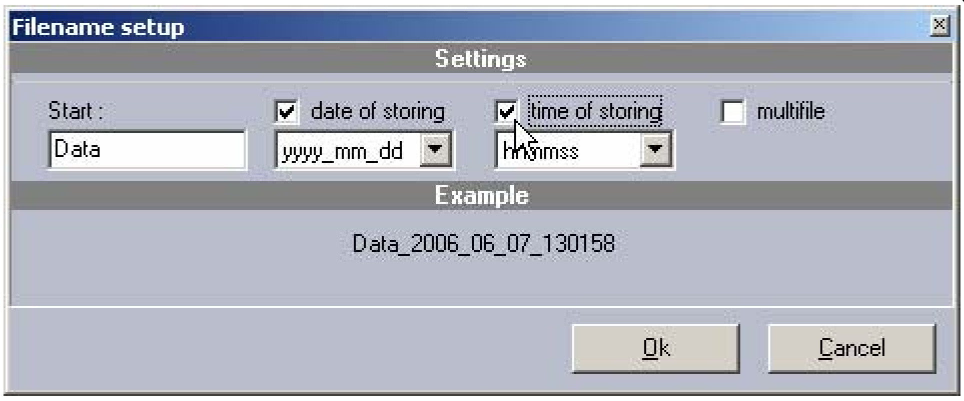
☑ multifile, Start: 17

Example

Data_2006_06_07_xxxx

Figure 115 Filename setup – Multifile

property UseTime: WordBool;
UseTime determines whether the time should be included in the file name (cf. Figure 116).


Filename setup

Settings

Start : Data

☑ date of storing: yyyy_mm_dd

☑ time of storing: hhmmss

☐ multifile

Example

Data_2006_06_07_130158

Figure 116 Filename setup – Time of storing

7.25 ILoadEngine

7.25.1 Methods of ILoadEngine

procedure CloseFile;
CloseFile closes a measurement file.

function NextDBLoad: WordBool;
NextDBLoad is used in conjunction with StartDBLoad. It proceeds to the next data block of the direct buffer. The function returns True, if a data block is available; False otherwise.

procedure ReloadBlock(Num: Integer);
ReloadBlock reloads the specified data block.
Parameters:
Num defines the data bloc to reload.

procedure Reload(Start: T_RecordPosition; Stop: T_RecordPosition);
Reload reloads a specific section of data.
Parameters:
Start is the beginning of the data section to reload.
Stop is the end of the data section to reload.

procedure ShrinkFile(const FileName: WideString);
ShrinkFile makes an export back to DEWESoft. Prior to using this function the properties Data.StartStamp amd Data.EndStamp (see Properties of IData) have to be set defining the boundaries of the new file.
Parameters:
FileName is the name of the file including the path to where the new file is stored to.

procedure StartDBLoad(Start: T_RecordPosition; Stop: T_RecordPosition; BlockSize: Integer);
StartDBLoad starts the loading of datablocks beginning at a certain position within the direct buffer, to a certain position within the direct buffer, of a certain blocksize. The function NextDBLoad has to be used to proceed to the next data block.
Parameters:
Start is the start position from where the reloading of data will begin.
Stop is the end position up to which the reloading of data will proceed.
BlockSize is the size of the data blocks being retrieved.

7.25.2 Properties of ILoadEngine

property DataSections: IDataSections; (read-only)
DataSections provides information about data sections which are the parts between a start and a stop event.

property FileOpened: WordBool; (read-only)
FileOpened specifies whether a measurement file is currently opened or not.

property NumBlocks: Integer; (read-only)
NumBlocks is the number of data blocks within a measurement file.

property ReducedOnly: WordBool; (read-only)
ReducedOnly specifies whether data has been stored using reduced rate only.

property VideoLoadEngines: IVideoLoadEngines; (read-only)
See IVideoLoadEngines;

7.26 IMasterClock

7.26.1 Methods of IMasterClock

function GetCurrentTime: Double;
GetCurrentTime returns the current time from the A/D-board before the data is transferred. The unit of this time is seconds. MasterClock can be used for time stamping asynchronous data.

7.27 IModule

7.27.1 Methods of IModule

procedure ClearModule;
ClearModule sets the address of a PAD-module back to 0.

procedure DetectModule(Address: Integer);
DetectModule detects a module at a certain address.
Parameter:
Address is the address at which the module has to be detected.

function FillModule(Address: Integer; Timeout: Integer): WordBool;
FillModule tries to set a PAD-module having address 0 to the given address and waits for pressing the button on the module.
Parameters:
Address is the new address to set for the PAD-module.
Timeout is the time to wait for the operator pressing the button on the module or the time it may take to detect the module at the given address. Its unit is ms.

procedure FREQAFindTriggerLevel;
FREQAFindTriggerLevel activates the auto-trigger of the FREQA-module.

procedure GetPadData;
GetPadData retrieves the current data from the PAD-module. The data can be read afterwards via the propertie PadData.

function GetSerialNumber: WideString;
GetSerialNumber reads out the serial number of the module. The return-value is a string.

function SetDaq: WordBool;
SetDaq applies the configuration to a PAD-module after previously having changed its properties.

procedure SetDaqAddress(Address: Integer; Timeout: Integer);
SetDaqAddress changes the address of the DAQ-module.
Parameters:
Address is the new address to set for the module. The values have to be in the range 0…254.
Timeout specifies the timeout-limit. Its unit is [ms].

procedure SetModule(SetType: Integer);
SetModule applies the properties of a module which were specified before.
Parameters:
SetType has to be 2 always. (Other values of SetType are for internal use only.)

procedure SetPad(NewAddress: Integer);
SetPad sets the address and applies the configuration to a PAD-module.
Parameter:
NewAddress is the address to set for the PAD-module. The values have to be within the range 0…255.

7.27.2 Properties of IModule

property DaqData: IDaqData; (read-only)
DaqData provides the data of a DAQ-module.

property Index: Smallint; (read-only)
Index is the index of the module which is in the range 0…IModules.Count–1.

property ModuleType: Smallint;
ModuleType denotes the type of a module. Its value can be one of the following:
0…none
1…PAD
2…DAQ

property PadData: IPadData; (read-only)
PadData is the data of the PAD-module. See 7.29 IPadData.

7.28 IModules

7.28.1 Properties of IModules

property Count: Integer; (read-only)
Count is the number of available modules.

property Item[Index: Integer]: IModule; (read-only)
Item is the reference to one of the modules of the type IModule. Valid indexes are in the range 0…Count–1. See 7.27 IModule.

7.29 IPadData

7.29.1 Methods of IPadData

function CopyToString: WideString;
CopyToString returns the data of the PAD-module as a string.

function CopyUnitToString: WideString;
CopyUnitToString returns the unit of the PAD-module as a string.

function ModuleAmpl: Single;
ModuleAmpl returns module scale factor.

function ModuleOffset: Single;
ModuleOffset returns the offset of the PAD-module.

function ShortCopyToString: WideString;
ShortCopyToString returns the data of the PAD-module in a short version (only the values) as a string.

7.29.2 Properties of IPadData

property Address: Smallint;
Address is the PAD-module's address.

property ConfigCode: Smallint; (read-only)
ConfigCode is the configuration code of a PAD-module.

property Data[Index: Integer]: Single; (read-only)
Data is the measurement data of the PAD-module as an array of Single values.

property ModuleType: Smallint; (read-only)
ModuleType is the type of the PAD-module which can be one of the following:

0…Unknown
1…PAD-V8
2…PAD-VTH
3…PAD-TH8
4…PAD-RTD3
5…PAD-AO1
6…PAD-CNT
7…PAD-DI8
8…PAD-DO7
9…PAD-V8-P
10…PAD-TH8-P

property Name: WideString; (read-only)
Name is the name of the PAD-Module.

property RangeCode: Smallint;
RangeCode is the range code of the PAD-module.

property RangeIndex[Index: Integer]: Integer; (read-only)
RangeIndex gives the range code of choosen range at Index.

property Ranges[Index: Integer]: WideString; (read-only)
Ranges gives the description of a range at the index.

property RangesCount: Integer; (read-only)
RangesCount defines the number of ranges available for the module.

property SpeedCode: Smallint; (read-only)
SpeedCode denotes the baudrate set on the module. The code is within the range 3…10 and having the following meaning:
3…1200 baud
4…2400 baud
5…4800 baud
6…9600 baud
7…19200 baud
8…38400 baud
9…57600 baud
10…115200 baud

7.30 IPlugin2

7.30.1 Methods of IPlugin2

procedure ClearChannelsInstance;
ClearChannelsInstance is called by DEWESoft when all channels are destroyed in DEWESoft. Therefore all instances of plug-in channels have to be set to “nil” “null” or “nothing”, depending on the programming language used for development of the plugin.

procedure Configure;
Configure is called by DEWESoft when a user presses the “Configure” button within the plug-in tab of the measurement setup screen. This button will appear, when the plug-in has no frame which could be embedded in the setup screen. At the procedure Configure a dialog

window could be opened in order to be able to make some settings to the plug-in. See Figure 4 on page 11.

procedure HideFrame;
HideFrame is called by DEWESoft when the measurement setup screen is left. This can be when switched to analyze mode or when switched to the stop screen.

procedure Initiate(const DeweApp: IApp);
Initiate is called by DEWESoft every time DEWESoft is started and the plug-in is checked in the hardware configuration (See Figure 2 on page 10) or when the hardware configuration window is closed and the plug-in is activated.

procedure LoadSetup(Data: OleVariant);
LoadSetup is called by DEWESoft every time a channel setup is loaded. This corresponds to the menu item “File >> Load Setup”. When switched from analyze mode to measurement setup, LoadSetup will also be called.
Parameters:
Data can contain any information about special settings of the plug-in if it has been stored to the setup file (.dss) at SaveSetup.

procedure NewSetup;
NewSetup is called by DEWESoft when a new setup is created. This corresponds to the menu item “File >> New Setup”. NewSetup happens also at startup of DEWESoft.

procedure OnGetData;
OnGetData is called by DEWESoft every time new data is added to a channel which happens about every 40 ms. At OnGetData it is not necessary to add new data.

procedure OnOleMsg(Msg: Integer; Param: Integer);
OnOleMsg will be called by DEWESoft after App.MainWndMessage has been called by an additional thread, other than the main thread, of the plug-in. When a plug-in has additional threads, these cannot send messages directly to DEWESoft. Therefore additional threads have to use App.MainWndMessage.
Parameters:
Msg and Param correspond to the arguments of MainWndMessage which are named equal. These parameters can be used to specify the message which has to be sent to DEWESoft.

procedure OnStartAcq;
OnStartAcq is called by DEWESoft when it is switched to one of the measurement screens e.g. “Overview”, “Recorder” etc. Implicitly this is done by starting storing when OnStartAcq has not been called by switching to one of the measurement screens before.

procedure OnStartStoring;
OnStartStoring is called by DEWESoft when the storing of data is started or a trigger for storing is armed, corresponding to the Store button.

procedure OnStopAcq;
OnStopAcq is called by DEWESoft when the acquisition is stopped. This can be by clicking the Stop button, switching to the setup screen or to the analyze mode, opening the “General setup” or the “Hardware setup” dialog, or by closing DEWESoft.

procedure OnStopStoring;
OnStopStoring is called by DEWESoft when the storing of data is stopped or a trigger for storing data is unarmed. This can be by clicking the Stop button, switching to the setup screen or to the analyze mode, opening the “General setup” or the “Hardware setup” dialog, or by closing DEWESoft.

procedure OnTrigger(Time: Double);
OnTrigger is called by DEWESoft every time a trigger occurs.
Parameters:
Time is the measurement time in seconds corresponding to the trigger event.

procedure SaveSetup(var Data: OleVariant);
SaveSetup is called by DEWESoft when a channel setup is saved. This corresponds to “File >> Save Setup”.
Parameters:
Data is of the type OleVariant and can be used to specific settings for the plug-in to a setup file (.dss). These data will read back on LoadSetup.

function ShowFrame(Parent: Integer): WordBool;
ShowFrame is called by DEWESoft when the setup screen is entered. The Boolean return value has to be true if the plug-in includes any setup frame to show, otherwise false.
Parameters:
Parent is the ID of the parent for the setup frame.

procedure UpdateFrame;
UpdateFrame is called by DEWESoft, when switched to the Plugins tab of the measurement setup screen. If there are more than one plug-ins activated in the hardware configuration, UpdateFrame will be called only for the plug-in which is currently marked in the list on the left side of the plug-ins tab.

7.30.2 Properties of IPlugin2

property Id: WideString; (read-only)
Id is the GUID (Globally Unique IDentifier) of the plug-in library. This identifier is automatically generated by the development environment.

property Used: WordBool;
Used defines whether a plug-in is used or not. This corresponds to the check box in the Plugins tab of the measurement setup screen.

7.31 IPlugin3

7.31.1 Methods of IPlugin3

function GetDWTypeLibVersion: Integer;
At GetDWTypeLibVersion the version number of the DEWESoft type library should be returned. It is defined in DEWESoft’s type library as DEWESoftMinorVersion.

procedure OnAfterCalcMath;
OnAfterCalcMath is called after the Math-channels are calculated. This procedure should be used if the results of the Math-channels are relevant for the plug-in. This procedure is similar

to OnGetData of IPlugin2, but is called after calculation of the Math-channels. (DEWESoft channels are processed in the following order: Devices, Plug-ins, Math.)

procedure OnAfterStartAcq;
OnAfterStartAcquisition is called by DEWESoft on starting the acquisition. This procedure is equal to OnStartAcq of IPlugin2.

procedure OnAfterStopAcq;
OnAfterStopAcq is called by DEWESoft on stopping the Acquisition. This procedure is equal to OnStopAcq of IPlugin2.

procedure OnAlarm(CondIndex: Integer; Status: WordBool);
OnAlarm is called by DEWESoft on the occurrence of an alarm.
Parameters:
CondIndex gives the index of the related alarm condition.
Status tells whether the alarm condition has become true of false.

procedure OnBeforeStartAcq(var AllowStart: WordBool);
OnBeforeStartAcq is called by DEWESoft before the acquisition is started. Within this procedure some kind of initialization can be done by the plug-in.
Parameters:
AllowStart defines whther the acquisition is allowed to start or not. Its default value is True.

procedure OnBeforeStopAcq(var AllowStop: WordBool);
OnBeforeStopAcq is called before the acquisition is stopped.
Parameters:
AllowStop defines whether the acquisition is allowed to stop or not. Its default value is True.

procedure OnBigListLoad(const TextSetup: WideString);
OnBigListLoad is only for DEWESoft's internal use.

procedure OnExit;
OnExit is called by DEWESoft on closing.

procedure OnGetClock(var ClockLow: Integer; var ClockHigh: Integer);
At OnGetClock a clock can be provided to DEWESoft, if no AD-hardware is used.
Parameters:
ClockLow and ClockHigh are used because a 64-bit value is necessary to provide the time value.

procedure OnGetSetupData;
OnGetSetupData is similar to OnGetData, but will be called periodically when the setup screen is active. OnGetSetupData can be used for displaying acquired data during setup.

procedure OnRepaintFrame;
OnRepaintFrame is called 10 times per second and can be used to update some information displayed in the plug-in frame.

procedure OnResizeFrame(Width: Integer; Height: Integer);
OnResizeFrame allows to resize the setup frame of a plug-in according to the current size of the DEWESoft's setup screen.
Parameters:
Width is the width of the setup screen.

Heigth is the hight if the setup screen.

procedure OnStartSetup;
OnStartSetup is called by DEWESoft on entering the setup screen.

procedure OnStopSetup;
OnStopSetup is called by DEWESoft on leaving the setup screen.

procedure OnTriggerStop(Time: Double; TrigDuration: Double);
OnStopTrigger is called by DEWESoft when the stop-trigger event happens.
Parameters:
Time is the timestamp when the stop-trigger happens.
TrigDuration is the duration of the trigger event.

procedure ProvidesClock(var Value: WordBool);
ProvidesClock sets whether the plug-in provides a clock to DEWESoft or not.
Parameters:
Value must be set to True if the plug-in provides a clock. The default value is False.

procedure SetCANPort(Port: Integer);
SetCANPort allows setting a CAN-port for using it for the plug-in.

7.32 IPluginGroup

7.32.1 Methods of IPluginGroup

procedure ClearAllChannels;
ClearAllChannels clears all mounted plug-in-channels.

function GetIndexName(Index: T_ChIndex): WideString;
GetIndexName returns the name of a channel index.
Parameters:
Index specifies the index of a channel which is of the type T_ChIndex. For detailed information on T_ChIndex please see 7.10.3 Types / Constants of IChannel.

function MountChannel(DataType: Integer; Async: WordBool; DBSize: Integer): IChannel;
MountChannel mounts a channel to a certain group and returns the reference to a channel of the type IChannel.
Parameters:
DataType specifies the type of data the channel contains. This can be one of the following:
0…dtByte,
1…dtShortInt,
2…dtSmallInt,
3…dtWord,
4…dtInteger,
5…dtSingle,
6…dtInt64,
7…dtDouble.
Async defines whether the channel contains asynchronous or synchronous data. It hast to be True if it is an asynchronous channel, otherwise False.

DBSize specifies the size of the direct buffer. If DBSize is set to -1, the default buffer size is applied.

procedure UnmountChannel(var Channel: IChannel);
UnmountChannel allows removing one channel separately.

7.32.2 Properties of IPluginGroup

property Count: Integer; (read-only)
Count is the number of mounted channels within the plug-in group.

property ExportRate: Integer;
ExportRate is the rate which is used for exporting asynchronous data to DIAdem.

property Item[Index: Integer]: IChannel; (read-only)
Item[Index] is one of the plug-in channels which is of the type IChannel.

property Name: WideString; (read-only)
Name is the name of a channel item within the plug-in group.

7.33 IScreen

7.33.1 Methods of IScreen

procedure Show;
To show one specific screen.

7.33.2 Properties of IScreen

property Id: Integer; (read-only)
Id is the identification number of a screen. A screen can be e.g. an “Overview”, a “Scope”, a “Recorder”, etc. Each of these categories can have more then one screen.

property IsCurrent: WordBool; (read-only);
IsCurrent is True if the screen is currently shown.

property Name: WideString; (read-only)
Name is the name of the screen how it appears at the buttons for selecting one of the screens. This can be e.g. “Overview”, “Overview 2”, “Scope”, etc.

7.34 IScreens

7.34.1 Properties of IScreens

property Count: Integer; (read-only)
Count is the number of screens. A screen can be e.g. an “Overview”, a “Scope”, a “Recorder”, etc. Each of these categories can have more then one screen

property Item[Index: Integer]: IScreen; (read-only)
Item[Index] is one of the screen items which is of the type IScreen.

7.35 IStoreEngine

7.35.1 Methods of IStoreEngine

procedure AddNewEvent(Type_: EventType; Data: OleVariant);
AddNewEvent adds an event during storing of data.
Parameters:
EventType defines the type of event to add. This can be etText (= $00000015) or etKeyboard (= $00000014).
Data is a Sring, if the EventType is set to etText.

7.35.2 Properties of IStoreEngine

property IsTriggering: WordBool; (read-only)
IsTriggering indicates whether the application is currently triggering. This corresponds to the state between a start- and a stop-trigger. E.g., if the storing option is set to “fast on trigger”, IsTriggering will be True while data is stored, otherwise False.

property FileSize: Largeuint; (read-only)
FileSize is the current size of the measurement file in bytes.

property StoreMode: Integer; (read-only)
StoreMode corresponds to the store options that can be set in the measurement setup See Figure 117. This mode can be set to one of the following values:
0… “always fast”
1… “always slow”
2… “fast on trigger”
3… “fast on trigger, slow otherwise”

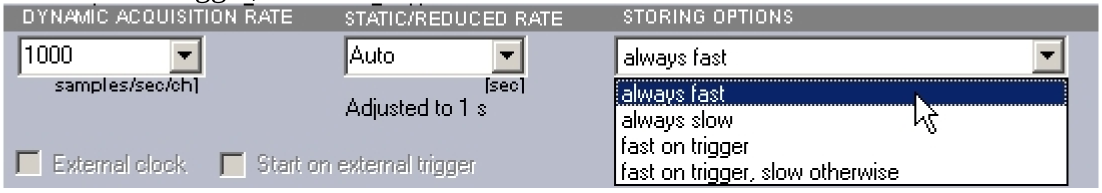


Figure 117 Storing Options

property Storing: WordBool; (read-only)
Storing is true, if storing is currently in progress; false otherwise.

7.36 ITrig

7.36.1 Properties of ITrig

property NotOrList: ITriggerCondList; (read-only)
NotOrList provides a trigger condition list of the type ITriggerCondList. The trigger conditions of this list are equal to the “Don’t store” conditions as found in DEWESoft’s measurement setup.

property OrList: ITriggerCondList; (read-only)
OrList provides a trigger condition list of the type ITriggerCondList. The trigger conditions of this list are OR-combined.

property TrigIndex: Integer; (read-only)
TrigIndex is the Index of the condition in the OrList which caused the trigger.

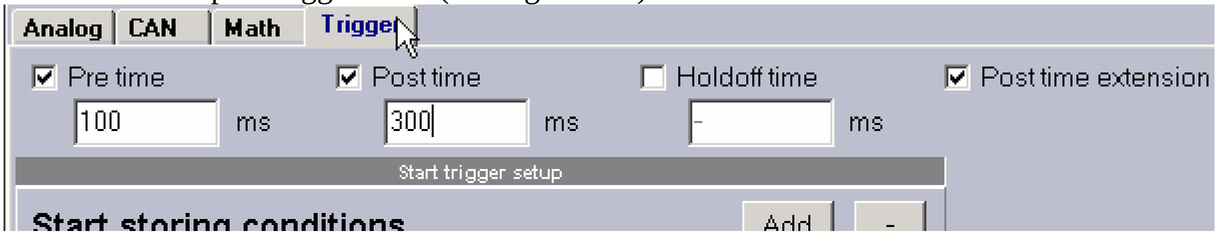
7.37 ITrigger

7.37.1 Properties of ITrigger

property HoldoffTime: Single
HoldoffTime is the holdoff time in ms which is set for the trigger.

property HoldoffTimeUsed: WordBool;
HoldoffTimeUsed defines whether the holdoff time is used or not.

property PostTime: Single
PostTime is the post trigger time. (Cf. Figure 118)



☑ Pre time 100 ms
☑ Post time 300 ms
☐ Holdoff time - ms
☑ Post time extension

Figure 118 Trigger Properties

property PostTimeUsed: WordBool
PostTimeUsed determines whether the PostTime will be used or not. (Cf. Figure 118)

property PostTimeExtensionUsed: WordBool
PostTimeExtensionUsed defines whether the post time extension is used or not.

property PreTime: Single
PreTime is the pre trigger time. (Cf. Figure 118)

property PreTimeUsed: WordBool
PreTimeUsed determines whether the PreTime will be used or not. (Cf. Figure 118)

property StartTrig: ITrig (read-only)
Start trigger

property StopTrig: ITrig (read-only)
Stop trigger

7.38 ITriggerCondition

7.38.1 Properties of ITriggerCondition

property Direction: Integer;
Direction defines the direction of a trigger condition. Valid values of this property are dependant on the Mode.
For modes 0,1,4,5:
0…positive
1…negative

2…any edge
For modes 2, 3:
0…enter
1…leave

property Level1: Single;
Level1 is the first trigger level.If only one of the trigger levels is available, this has to be assigned. So, it will be used in conjunction with any trigger mode if the trigger type is 0 (“on data”).

property Level2: Single;
Level2 is an additional trigger level which will be used in conjunction with the trigger modes tmFilteredEdge (1), tmWindow (2) and tmWindowPulsewidth (4), when the trigger type is 0 (“on data”).

property Mode: Integer;
Mode defines the trigger mode of the trigger condition, which can be one of the following:
0… tmSimpleEdge
1… tmFilteredEdge
2… tmWindow
3… tmPulsewidth
4… tmWindowPulsewidth
5… tmSlope

property TrigType: Integer;
TrigType defines the type of a trigger which can be one of the following:
0…on data
1…on time
2…on FFT

property TrigValue: Integer;
TrigValue determines which kind of value is used for the recognition of any trigger condition. This can be one of the following:
0…real data
1…average
2…RMS

7.39 ITriggerCondList

7.39.1 Properties of ITriggerCondList

property Count: Integer; (read-only)
Count is the number of trigger conditions within the trigger condition list.

property Item[Index: Integer]: ITriggerCondition; (read-only)
Item[] is an array of trigger conditions of the type ITriggerCondition. The index can be within the range of 0…Count–1.

7.39.2 Methods of ITriggerCondList

function Add: ITriggerCondition;
Add adds a trigger condition of the type ITriggerCondition to the list of trigger conditions.

procedure Remove(Ind: Integer);
Remove removes the trigger condition specified by the index Ind. Ind has to be within the range 0... Count–1.

7.40 IVideoLoadEngine

7.40.1 Methods of IVideoLoadEngine

procedure GetFramesInfo(out Frames: OleVariant);
GetFramesInfo gives information about the timestamps of the video-frames of one video file.
Parameters:
Frames is an OleVariant containing an array of timestamps of the type Double. The index of the array matches to the frame-index of the video file.

7.41 IVideoLoadEngines

7.41.1 Properties of IVideoLoadEngines

property Count: Integer; (read-only)
Count is the number of VideoLoadEngines which is equal to the number of videos captured.

property Item[Index: Integer]: IVideoLoadEngine; (read-only)
Item is one of the VideoLoadEngines which is of the type IVideoLoadEngine. The Index must be in the range 0…Count – 1.

8 DEWESoft DCOM Interface Tree

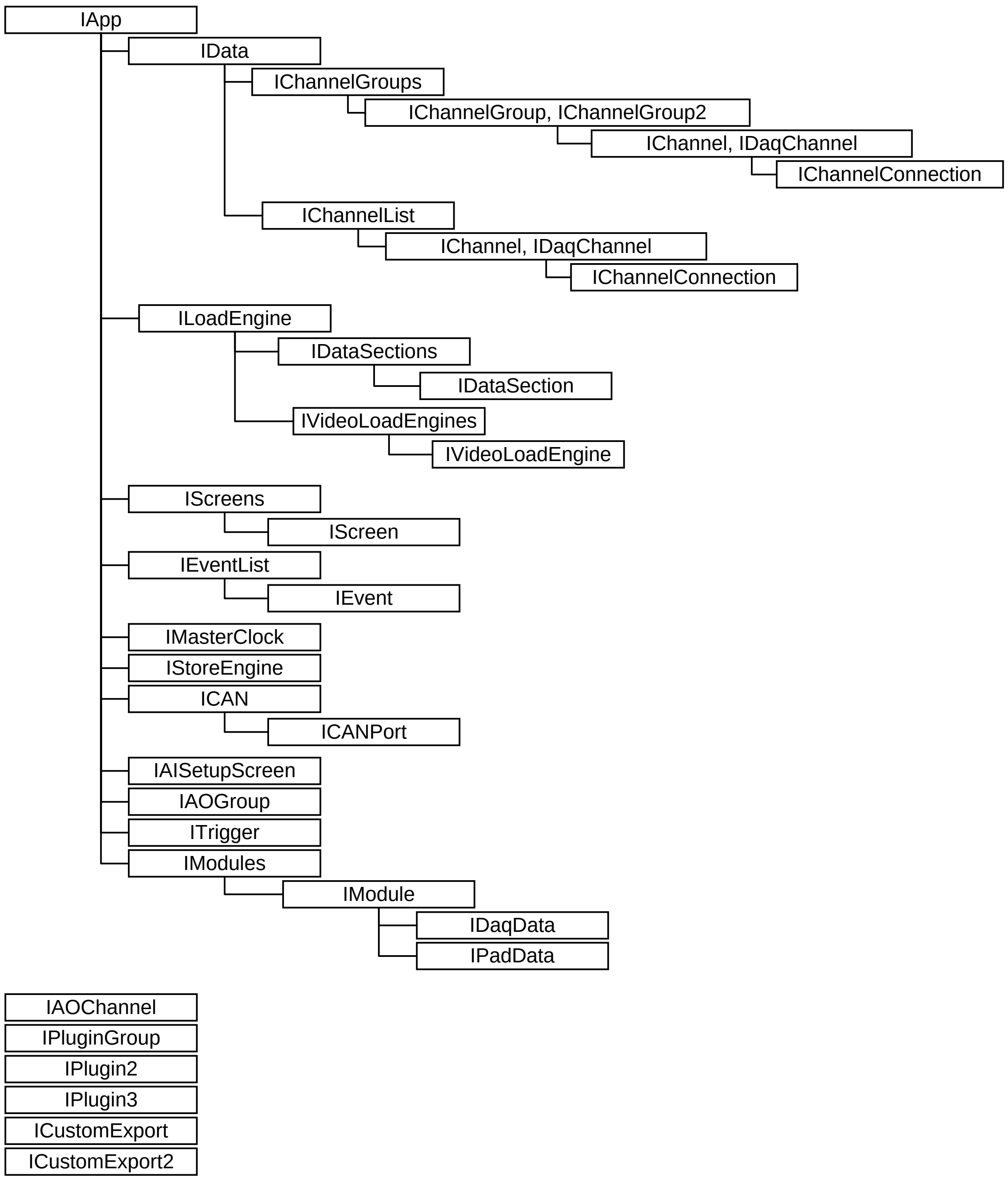


Figure 119 DEWESoft DCOM Interface Tree

9 DEWESoft DCOM Interface – Properties and Methods

▦ …Property
● …Method
⚡ …Event
▤ …Constant
▣ …Enumeration
☑ …Type
autom.…Automation

Table 3 List of Properties and Methods

Type	*Name*		*Access*	*Use only for*	*Description*
IAISetupScreen					
●	AISetupScreen.ShowChannelSetup	procedure **ShowChannelSetup(**ChNo: Integer**);**			To open the setup dialog of the channel ChNo.
●	AISetupScreen.SetColumnVisible	procedure **SetColumnVisible(**ColNo: Integer; Visible: WordBool**);**			To show or hide column of the setup screen specified by ColNo. (only for analog channels)
IAlarms					
▦	Alarms.ActiveCount	**ActiveCount:** Integer;	read-only		ActiveCount is the number of alarm conditions which are currently active.
▦	Alarms.ActiveItem[ ]	**ActiveItem[I: Integer]:** IAlarmCond;	read-only		ActiveItem[I] is the active alarm condition at the index I. I is in the range of 0…ActiveCount–1. ActiveItem list is consistent only during measurement.
▦	Alarms.Count	**Count:** Integer;	read-only		Count is the number of alarm conditions which are set.
▦	Alarms.Item[ ]	**Item[I: Integer]:** IAlarmCond;	read-only		Item[I] is the alarm condition at index I. I is in the range of 0…Count–1.
IAlarmCond					
●	AlarmCond.EndAlarm	procedure **EndAlarm;**			EndAlarm can be called in order to stop an alarm having StopOption set to 1 (manual).
▦	AlarmCond.Index	**Index:** Integer;	read-only		Returns the Index of the alarm condition.
▦	AlarmCond.Name	**Name:** WideString	read-only		The name assigned to an alarm condition.

	AlarmCond.Status	**Status:** WordBool	read-only		Status is true if an alarm ondition is met and the alarm is currently still active.
	AlarmCond.StopOption	**StopOption:** Integer;			StopOption defines how and whether an alarm condition is stopped. This can be one of the following: 0…never 1…manual 2…on stop condition 3…on time
	AlarmCond.StopTime	**StopTime:** Single;			StopTime is the time after which an alarm is stopped if the StopOption is set to 3 (on time). The unit of StopTime is seconds.
	AlarmCond.StopTrigger	**StopTrigger:** ITrig;	read-only		StopTrigger is the combination of trigger conditions deactivating an alarm, if StopOption is set to 2.
	AlarmCond.Trigger	**Trigger:** ITrig;	read-only		Trigger is the combination of trigger conditions activating an alarm.
IAOChannel					
	AOChannel.Ampl	**Ampl:** Single	read/write		The amplitude of the output signal.
	AOChannel.FilterFreq1	**FilterFreq1:** Single	read/write		The first frequency of a filter.
	AOChannel.FilterFreq2	**FilterFreq2:** Single	read/write		The second frequency of a filter.
	AOChannel.FilterType	**FilterType:** Integer	read/write		The type of filter.
	AOChannel.FilterOrder	**FilterOrder:** Integer	read/write		The order of a low/band pass filter.
	AOChannel.FilterProtoType	**FilterProtoType:** Integer	read/write		The filter prototype.
	AOChannel.Offset	**Offset:** Single	read/write		The offset of the output signal.
	AOChannel.Phase	**Phase:** Single	read/write		The phase angle of the output signal.
	AOChannel.Range	**Range:** Integer	read/write		The voltage range of the analog output channel.
	AOChannel.WaveForm	**WaveForm:** AOWaveForm	read/write		The type of waveform to output. 0…Sine 1…Triangular 2…Rectangular 3…Saw 4…Noise 5…Arbitrary
IAOGroup					
	AOGroup.AmplChangeFactor	**AmplChangeFactor:** Single			The rate for amplitude changes. [V/s]

	AOGroup.AOChannels	**AOChannels:** IChannelList	read-only		The list of analog output channels.
	AOGroup.ControlsClock	**ControlsClock:** WordBool	read-only		True, if the analog output-board is the master clock.
	AOGroup.DCChangeFactor	**DCChangeFactor:** Single			The rate for DC value changes. [V/s]
	AOGroup.DeltaFreq	**DeltaFreq:** Single			The frequency step-size for a “Step sweep” signal.
	AOGroup.Freq	**Freq:** Single			The frequency of a signal.
	AOGroup.FreqChangeFactor	**FreqChangeFactor:** Single			The rate for frequency changes. [Hz/s]
	AOGroup.LogSweep	**LogSweep:** WordBool			Logarithmic frequency change of a sweep.
	AOGroup.OperationMode	**OperationMode:** AOOperationMode			The mode of the function generator 0…Fixed Frequency 1…Sweep 2…Step sweep 3…Burst 4…Chirp
	AOGroup.PhaseChangeFactor	**PhaseChangeFactor:** Single			The rate for phase changes. [°/s]
	AOGroup.SampleRate	**SampleRate:** Integer			The sample rate for analog output.
	AOGroup.ShowInfoChannels	**ShowInfoChannels:** WordBool			Whether channels indicationg Ampl”, “Phase”, “Offset” and “Freq” should be available.
	AOGroup.StartFreq	**StartFreq:** Single			The start frequency of a signal.
	AOGroup.StartTime	**StartTime:** Single			The startup time of a waveform.
	AOGroup.StopFreq	**StopFreq:** Single			The end frequency of a signal.
	AOGroup.StopTime	**StopTime:** Single			The fall time of a waveform.
	AOGroup.SweepMode	**SweepMode:** AOSweepMode			Repeated or single output. 0…Loop 1…Single
IApp					
	App.ActiveScreen	**ActiveScreen:** Integer	read-only		Returns number of the active screen. This can be 0…“Overview”, 1… “Scope”, 2… “Recorder”, 3…”FFT”, 4…”Video”, 5…”GPS”, 6…”Vert. rec.”, 7…”Power”.
	App.Acquiring	**Acquiring:** WordBool	read-only		Whether DEWESoft is acquiring data.

	App.ActualRunMode	**ActualRunMode:** Integer	read-only		Returns the actual run-mode which can be 0…none, 1…measure, 2…analyse.
	App.AISetupScreen	**AISetupScreen:** IAISetupScreen	read-only		See IAISetupScreen.
	App.Alarms	**Alarms:** IAlarms	read-only		See IAlarms
	App.AlwaysEnableTrigger	**AlwaysEnableTrigger:** WordBool	read/write		To show the Trigger tab in setup screen anyway.
	App.AOGroup	**AOGroup:** IAOGroup	read-only		See IAOGroup.
	App.AOGetManualAvail	function **AOGetManualAvail:** WordBool;			Whether the manual start/stop feature of analog output is available.
	App.AOSetManual	procedure **AOSetManual;**			To manually start or stop the AO.
	App.AveragedCPB	**AveragedCPB:** IAveragedFFT	read-only		See IAveragedFFT.
	App.CAN	**CAN:** ICAN	read-only		See ICAN.
	App.ChangeDaqType	procedure **ChangeDaqType(**DaqType: Integer**);**			Changes the divice type for analog acquisition. 0…No A/D 1…Dewetron DAQ 2…Dewetron DSA 3…Spectrum 4…National Instruments 5…National Instruments MX 6…National Instruments DSA 7…Data Translation 8…Microstar DAP
	App.ChangeComPort	**ChangeComPort(**ComPort: Integer**)**;			To change the com port being used for the modules. (0… com1; 1… com2;…)
	App.Data	**Data:** IData	read-only		Returns the Measurement Data; see IData.
	App.DataLost	**DataLost:** WordBool	read-only		Whether loss of data has occurred.
	App.DaqGroup	**DaqGroup:** IDaqGroup	read-only		The analog input group. See IDaqGroup.
	App.DisableStoring	**DisableStoring:** WordBool	read/write		To hide the Store button.
	App.Enabled	**Enabled:** WordBool	read/write		Whether DEWESoft is enabled or disabled.
	App.EventList	**EventList:** IEventList	read-only		Returns the list of events.

●	App.ExportData	procedure **ExportData(**ExportType: Integer; const FileName: WideString**)**;			Call to export data of a certain type to a specified file. The ExportType can be: 0…Flexpro (*.fpd) 1…Excel (.xls) 2…DIAdem (*.dat) 3…Matlab (*.mat) 4…Universal File Format 58 (*.unv) 5…FAMOS (*.dat) 6…NSoft time series (*.dac) 7…Text (‘.txt) 8…Sony (*.log) 9…RPCIII (*.rsp) 10…Comptade (*.cfg) FileName specifies the target-file path.
▤	App. FileNameSettings	FileNameSettings: IFileNameSettings	read-only		See IFileNameSettings.
●	App.GetInterfaceVersion	procedure **GetInterfaceVersion(**var Major: Integer; var Minor: Integer; var Revision: Integer**)**;			To find out the version number of DEWESoft’s DCOM interface.
●	App.HardwareSetup	procedure **HardwareSetup(**Plugins: WordBool**)**;			Starting the dialog window of the hardware setup. When Plugins is set to true, the windows registry will be searched for plug-ins and all the plug-ins will be reinitialized, like on startup of DEWESoft.
▤	App.Height	**Height:** Long	read/write		Height of the application window expressed in pixels.
▤	App.IniFileDir	**IniFileDir**: String	read/write		Path of the ini-file directory.
●	App.Init	procedure **Init**;			To start the initialization of the application.
▤	App.IsSetupMode	**IsSetupMode:** WordBool	read-only		Whether DEWSoft is in setup mode or not.
▤	App.Left	**Left:** Integer	read/write		Left property of the application window expressed in pixels.
●	App.LoadDBC	**LoadDBC(**PortNo: Integer; const FileName: WideString**)**;			To load the DBC file FileName for the port specified by PortNo (first port is 0).
●	App.LoadDisplaySetup	procedure **LoadDisplaySetup(**const FileName: WideString**)**;			To load a display setup.
▤	App.LoadEngine	**LoadEngine:** ILoadEngine	read-only		See ILoadEngine.
●	App.LoadFile	procedure **LoadFile(**const FileName: WideString**)**;			To load a data file specified by FileName.

🟩	App.LoadModuleSetup	LoadModuleSetup**(**const FileName: WideString**);**			To load a DEWESoft setup for modules only.
🟩	App.LoadSetup	procedure **LoadSetup(**const FileName: WideString**);**			To load a setup file specified by FileName.
▤	App.MainDataDir	**MainDataDir:** WideString	read-only		The path of the data directory.
🟩	App.MainWndMessage	procedure **MainWndMessage(**Msg: Integer; WParam: Integer; Wait: WordBool**);**			This is the only allowed message to be called from any thread within a plug-in. If it is necessary to send a message to DEWESoft from an additional thread of a plug-in, MainWndMessage has to be used. “Msg” and “Param” will be available at “IPlugin2.OnOleMsg” which will follow. On this, the message has to be sent to DEWESoft from the main thread of the plug-in. “Wait” specifies whether the thread from which MainWndMessage was called will wait or continue execution.
🟩	App.ManualStart	procedure **ManualStart**;			To cause a manual start trigger.
🟩	App.ManualStop	procedure **ManualStop**;			To cause a manual stop trigger.
▤	App.MasterClock	**MasterClock:** IMasterClock	read-only		Returns the current time from the A/D-board in seconds before the data is transferred. (used for time-stamping asynchronous data)
🟩	App.Measure	procedure **Measure**;			To switch to measure mode.
▤	App.MeasureSampleRate	**MeasureSampleRat**e**:** Integer	read/write		The sample rate for acquisition.
🟩	App.MenuClick	procedure **MenuClick(**Item: MenuItems**);**			To cause a menu item to be clicked. See MenuItems.
▤	App.Modules	**Modules:** IModules	read-only		See IModules.
🟩	App.NewSetup	procedure **NewSetup**;			Equals to menu selection “File >> New Setup”
⚡	App.OnAlarm	procedure **OnAlarm**;		autom.	When an alarm occurs.
⚡	App.OnDataLost	procedure **OnDataLost**;		autom.	When data lost is reported by the DAQ object.

⚡	App.OnEvent	procedure **OnEvent(**Reason: EventReason; Param: OleVariant**);**		autom.	On several events specified by Reason which can be 1 indicating “start”, 2 indicating “stop”, 3 indicating “trigger”, 11 indicating “VStart” (video start storing) 12 indicating “VStop” (video stop storing) 20 indicating “Keyboard” 21 indicating “Notice” 22 indicating “Voice” 24 indicating “Module” (example: Bridge shunt on/off).
⚡	App.OnException	procedure **OnException(**const Str: WideString**);**		autom.	When an exception occurs.
⚡	App.OnExit	procedure **OnExit;**		autom.	When DEWESoft is closed.
⚡	App.OnGetData	procedure **OnGetData;**		autom.	When new data is added to a channel. This happens every 40 ms.
⚡	App.OnGetTime	procedure **OnGetTime(**var TimeLo: Integer; var TimeHi: Integer**);**		autom.	To provide a clock to DEWESoft, if no AD-hardware is used.
⚡	App.OnStartStoring	procedure **OnStartStoring;**		autom.	When the storing of data is started.
⚡	App.OnStopStoring	procedure **OnStopStoring;**		autom.	When the storing of data is stopped.
⚡	App.OnTrigger	procedure **OnTrigger(**Mid: Integer; Dir: Integer; Time: Double**);**		autom.	When a trigger occurs. Returns: Mid…the number of the current data block, Dir…the position within the current data block and Time…the current time in seconds.
⚡	App.OnTriggerStop	procedure **OnTriggerStop(**Mid: Integer; Dir: Integer; Time: Double**);**		autom.	When the stop trigger occurs.
▤	App.Parent	**Parent:** Integer	read/write	autom.	To reference the windows handle of a form as the DEWESoft parent.
●	App.PauseStoring	procedure **PauseStoring**;			To pause storing of data.
●	App.PrintScreen	procedure **PrintScreen(**ShowDialog: WordBool**);**			To print the current screen.
●	App.ResumeStoring	procedure **ResumeStoring**;			To resume storing of data.
●	App.SaveSetup	procedure **SaveSetup(**const FileName: WideString**);**			To save a channel setup to the file specified by FileName. (corresponds to File >> Save Setup)
▤	App.Screens	**Screens** As IScreens	read-only		See IScreens.

	App.SendCommand	function **SendCommand(**const Cmd: WideString; Timeout: Integer**):** WideString;			To send a command to a hardware module via the (internal) module bus. Returns the answer of the device.
	App.SetFullScreen	procedure **SetFullScreen(**Full: WordBool**)**;			To toggle between full screen display and standard display mode. (similar to pressing <CTRL> + <F>)
	App.SetHeaderData	procedure **SetHeaderData(**const Caption: WideString; const Header: WideString**)**;			To set header information to a measurement file.
	App.SetInstrument	procedure **SetInstrument(**Id: Integer**)**;			To set a screen. Id can be 0…"Overview", 1… "Scope", 2… "Recorder", 3…"FFT", 4…"Video", 5…"GPS", 6…"Vert. rec.", 7…"Power".
	App.SetMainDataDir	procedure **SetMainDataDir(**const DataDir: WideString**)**;			For setting the main data dorectory used by DEWESoft.
	App.SetScopeParams	function **SetScopeParams(**PreTime: TDateTime; PostTime: TDateTime; const Channel: IChannel; Level: Single**)**: Integer;			To set the pre- and the post-trigger time, the trigger channel and the trigger level of the scope. Returns the number of samples from the whole shot.
	App.SetScopeUsed	procedure **SetScopeUsed(**Value: WordBool**)**;			To specify whether the scope trigger is used or not.
	App.SetScreenIndex	procedure **SetScreenIndex(**Index: Integer**)**;			To set the screen index of active screen. (If more than one screen exists of any kind.)
	App.SetStoreMode	procedure **SetStoreMode(**Mode: Integer**)**;			To set the storing option.
	App.SetupScreen	procedure **SetupScreen**;			To switch to the setup screen.
	App.ShowPropertyFrame	**ShowPropertyFrame:** WordBool	read/write		To show or hide the property frame in measure mode.
	App.ShowSensorEditor	**ShowSensorEditor;**			To open the sensor editor window.
	App.ShowSROptions	**ShowSROptions:** WordBool	read/write		To show or hide the sample rate options in measure mode.
	App.ShowStoreOptions	**ShowStoreOptions:** WordBool	read/write		To show or hide the store options in measure mode.

▤	App.ShowStyle	**ShowStyle:** Integer	read/write		To change the appearance of the menu bar. 0…hides the whole menu bar (hides menu and buttons), 1…shows the menu bar but hides the drop-down menu bar (shows only buttons), 2… shows the menu bar including the drop-down menu bar (shows menu and buttons).
●	App.StartModuleScan	procedure **StartModuleScan**;			To (re)start scanning of the module.
●	App.StartStoring	procedure **StartStoring(**const FileName: WideString**)**;			To start storing. FileName is the path to the target data file. If FileName is empty, then the settings of DEWESoft are used.
▤	App.StayOnTop	**StayOnTop:** WordBool	read/write		Set whether the application window should stay on top or not.
●	App.Stop	procedure **Stop**;			To stop the application. Goes to stop-screen. (cf. stop-button)
●	App.StopModuleScan	procedure **StopModuleScan**;			To stop the scanning of the modules.
▤	App.StoreEngine	**StoreEngine:** IStoreEngine	read-only		See IStoreEngine.
▤	App.TimerInterval	**TimerInterval:** Integer	read/write		The timer interval for DEWESoft acquisition in ms. The default is 33.
▤	App.Top	**Top:** Integer	read/write		The top property of the application window expressed in pixels.
▤	App.Trigger	**Trigger:** ITrigger;	read-only		See ITrigger.
●	App.UpdateHardwareSetup	procedure **UpdateHardwareSetup**;			Has to be called after changes within the hardware setup. See App.HarwareSetup().
▤	App.UsedDatafile	**UsedDatafile:** WideString	read/write		The data file path to use.
▤	App.UsedSetupfile	**UsedSetupfile:** WideString	read-only		The currently used setup file.
▤	App.Version	**Version:** WideString	read-only		Returns the version number of the application.
▤	App.Visible	**Visible:** WordBool	read/write		Whether the application should be visible or not.
▤	App.Width	**Width:** Integer	read/write		The width of the application window expressed in pixels.
●	App.WriteErrorMessage	procedure **WriteErrorMessage(**const ErrorMsg: WideString**)**;			To write error messages to the caption of DEWESoft.
●	App.ZeroAllAutoChannels	procedure **ZeroAllAutoChannels(**Zero: WordBool**)**;			To zero all DaqChannels having AutoZero set to True. Zero determines whether the offset will be set or cleared.
IAveragedFFT					
▤	IAveragedFFT.AveCount	**AveCount:** Integer	read/write		The number of averages for FFT/CPB calculation.

▦	IAveragedFFT.AverageType	**AverageType:** Integer	read/write		The averaging type for FFT/CPB calculation.
●	IAveragedFFT.CalculateFromPos	function **CalculateFromPos(**Mid: Integer; Dir: Integer**)**: WordBool;			For performing the FFT/CPB calculation from the defined start position.
●	IAveragedFFT.GetChannels	procedure **GetChannels(**Channels: OleVariant**)**;			Provides a list of all channels used in AveragedFFT.
●	IAveragedFFT.GetData	procedure **GetData(**ChNo: Integer; OctaveDivider: Integer; Weighting: Integer; out BandCount: Integer; out Data: OleVariant**)**;			To retrieve the calculated FFT/CPB data.
▦	IAveragedFFT.Lines	**Lines:** Integer	read/write		The number of FFT lines.
▦	IAveragedFFT.Overlap	**Overlap:** Integer	read/write		The amount of overlap for averaging in percent.
▦	IAveragedFFT.Window	**Window:** Integer	read/write		The window type for averaging.
ICAN					
▦	CAN.SupportsOutput	**SupportsOutput:** WordBool	read-only		Whether output is supported.
▦	CAN.Count	**Count:** Integer	read-only		The number of available CAN ports.
▦	CAN.Item	**Item[**Index: Integer**]:** ICANPort	read-only		See ICANPort.
ICANPort					
●	CANPort.EnableOutput	procedure **EnableOutput(**Enable: WordBool**)**;			Enable CANPort for output.
●	CANPort.EndRead	procedure **EndRead**;			Must be called after reading of CAN data.
●	CANPort.GetBaudRate	function **GetBaudRate:** Integer;			Returns the Baud rate of a CAN-port.
●	CANPort.GetBaudRateList	function **GetBaudRateList:** OleVariant;			Returns an array of Strings of available Baud rates.
▦	CANPort.MessageCount	**MessageCount**: Integer	read-only		The number of messages available.
●	CANPort.ReadMessage	function **ReadMessage(**var TimeStamp: Double; var ArbId: Integer; var DataLo: Integer; var DataHi: Integer**):** WordBool;			Read a meassage from the CAN-port
●	CANPort.SendFrame	function **SendFrame(**Extended: WordBool; ArbId: Integer; Data: T_CANFrame; Size: Integer**):** WordBool;			Send a message on the CAN port.
●	CANPort.SetBaudRate	procedure **SetBaudRate(**BaudRate: Integer**)**;			Sets the Baud rate of a CAN-port.
●	CANPort.StartRead	procedure **StartRead**;			Must be called after reading of CAN data.
ConnTypes ▤					
▤	ConnTypes.ctLast	Const **ctLast** = 0			See IChannelConnection.AType.
▤	ConnTypes.ctOverlap	Const **ctOverlap** = 1			See IChannelConnection.AType.
▤	ConnTypes.ctTrigger	Const **ctTrigger** = 2			See IChannelConnection.AType.
▤	ConnTypes.ctNew	Const **ctNew** = 3			See IChannelConnection.AType.

ExportTypes					
▤	ExportTypes.etChannelBased	Const **etChannelBased** = 1			Export type for exporting each channel after the other. (all data of one channel before all data of the next channel)
▤	ExportTypes.etValueBased	Const **etValueBased** = 0			Export type for exporting each value after the other. (one sample of each channel before next sample of each channel)
IChannel					
●	Channel.AddAsyncByteSample	procedure **AddAsyncByteSample(**Value: Byte; TimeStamp: Double**);**			Called for adding asynchronous samples of the type byte.
●	Channel.AddAsyncDoubleSample	procedure **AddAsyncDoubleSample(**Value: Double; TimeStamp: Double**);**			Called for adding asynchronous samples of the type Double.
●	Channel.AddAsyncSingleSample	procedure **AddAsyncSingleSample(**Value: Single; TimeStamp: Double**);**			Called for adding asynchronous samples of the type Single.
●	Channel.AddSingleSample	procedure **AddSingleSample(**Value: Single**);**			Called for adding synchronous samples of the type Single.
●	Channel.AddSingleSamples	procedure **AddSingleSamples(**Count: Integer; Data: OleVariant; Timestamps: OleVariant**);**			Called for adding multiple synchronous samples of the type Single.
▦	Channel.Async	**Async:** WordBool	read-only		Whether a channel is asynchronous or not.
▦	Channel.Bytes	**Bytes:** Integer	read-only		The number of bytes used for one data sample (depending on the data type).
●	Channel.CreateConnection	function **CreateConnection:** IChannelConnection;			To create a connection to a channel to access its data. See IChannelConnection.
▦	Channel.DataType	**DataType:** Integer	read-only		The data type, e.g. Single, Double, etc.
▦	Channel.DBBufSize	**DBBufSize:** Integer	read-only		The size of the direct buffer.
▦	Channel.DBDataSize	**DBDataSize:** Integer	read-only		The size of the data stored in the direct buffer.
▦	Channel.DBPos	**DBPos:** Integer	read-only		The next position in the direct buffer to be written to.
▦	Channel.DBTimeStamp	**DBTimeStamp[**Index: Integer**]:** Double	read-only		The time stamp data of asynchronous data in the direct buffer.
▦	Channel.DBValues	**DBValues[**Index: Integer**]:** Single	read-only		The data values within the direct buffer.
▦	Channel.Description	**Description:** WideString	read/write		The description of a channel.
▦	Channel.ExportOrder	property **ExportOrder:** Integer	read/write		The order of the channels for exporting.
●	Channel.GetDBAddress	function **GetDBAddress:** Integer;			Returns the pointer of the next position in direct buffer to be written to.

	Channel.GetIBValues	procedure **GetIBValues(**Pos: Integer; out Min: Single; out Max: Single; out Ave: Single; out Rms: Single**)**;			Similar to reading the property IChannel.IBValues. The property IBValues cannot be read by LabVIEW. So this procedure can be used instead.
	Channel.GetIndex1	procedure **GetIndex1(**out IndexLevel: Integer; out I1: Integer; out I2: Integer; out I3: Integer; out I4: Integer; out I5: Integer**)**;			Similar to reading the property IChannel.Index. The property Index cannot be read by LabVIEW, so this procedure can be used instead.
	Channel.GetRBValues	procedure **GetRBValues(**Pos: Integer; out Min: Single; out Max: Single; out Ave: Single; out Rms: Single**)**;			Similar to reading the property IChannel.RBValues. The property RBValues cannot be read by LabVIEW. So this procedure can be used instead.
	Channel.GetScaledData	function **GetScaledData:** OleVariant;			Returns scaled data from the circular buffer. (The data type of scaled data is always "Single")
	Channel.GetScaledDataEx	function **GetScaledDataEx(**Start: Integer; Count: Integer**):** OleVariant;			Returns scaled data from the direct buffer which is already linearized.
	Channel.GetScaledDataEx1	procedure **GetScaledDataEx1(**Start: Integer; Count: Integer; out Data: OleVariant**)**;			Same as above implemented as a procedure.
	Channel.GetTSData	function **GetTSData:** OleVariant;			Returns the time stamp data.
	Channel.GetTSDataEx	function **GetTSDataEx(**Start: Integer; Count: Integer**):** OleVariant;			Returns a certain amount of the time stamp data, specified by Count, from a certain position, specified by Start.
	Channel.GetTSDataEx1	procedure **GetTSDataEx1(**Start: Integer; Count: Integer; out Data: OleVariant**)**;			Same as above implemented as a procedure.
	Channel.GetUnscaledData	function **GetUnscaledData:** OleVariant;			Returns unscaled data from the circular buffer.
	Channel.GetUnscaledDataEx	function **GetUnscaledDataEx(**Start: Integer; Count: Integer**):** OleVariant;			Returns unscaled data from the direct buffer, which is already linerized.
	Channel.GetUnscaledDataEx1	procedure **GetUnscaledDataEx1(**Start: Integer; Count: Integer; out Data: OleVariant**)**;			Same as above implemented as a procedure.
	Channel.Group	**Group:** IChannelGroup	read-only		A group containing the channels. (Channels of a plug-in have to be mounted to the group 8)
	Channel.IBBufSize	**IBBufSize:** Integer	read-only		The size of the intermediate buffer.
	Channel.IBDataSize	**IBDataSize:** Integer	read-only		The size of the data within the intermediate buffer.
	Channel.IBPos	**IBPos:** Integer	read-only		The next position in the intermediate buffer to be written to.

▨	Channel.IBValues	**IBValues[**Index: Integer**]:** T_ReducedRec	read-only		The data values within the intermediate buffer. (Min, Max, Ave and RMS)
●	Channel.IncDBSamples	procedure **IncDBSamples(**Count: Integer**)**;			To increment the direct buffer position by a certain amount of samples.
▨	Channel.Index	**Index:** T_ChIndex	read-only		See T_ChIndex and Table 2
▨	Channel.MainDisplayColor	**MainDisplayColor:** Int64	read/write		Specifies the color of a channel.
▨	Channel.Measurement	**Measurement:** WideString	read/write		The description of a channel. Equal to "Description".
▨	Channel.Name	**Name:** WideString	read/write		The name of a channel.
▨	Channel.Offset	**Offset:** Double	read/write		The offset value of a channel.
▨	Channel.RBBufSize	**RBBufSize:** Integer	read-only		The size of the reduced buffer.
▨	Channel.RBDataSize	**RBDataSize:** Integer	read-only		The size of the data within the reduced buffer.
▨	Channel.RBPos	**RBPos:** Integer	read-only		The next position in the reduced buffer to be written to.
▨	Channel.RBValues	**RBValues[**Index: Integer**]:** T_ReducedRec	read-only		The data values within the reduced buffer. (Min, Max, Ave and RMS).
▨	Channel.Scale	**Scale:** Double;	read/write		The scaling value of a channel.
●	Channel.ScaleValue	function **ScaleValue(**Value: Single**):** Single;			To scale an unscaled value.
●	Channel.SetSRDiv	procedure **SetSRDiv(**SRDiv: Integer**)**;			To set the sample rate devider.
●	Channel.SetSRDivType	procedure **SetSRDivType(**AType: TSRDivType**)**;			To set the type of sample rate divider.
▨	Channel.Shown	**Shown:** WordBool	read/write		If it is possible to show a channel.
▨	Channel.SRDiv	**SRDiv:** Integer	read-only		The sample rate divider of a channel.
▨	Channel.TypicalMaxValue	**TypicalMaxValue:** Single	read-only		Maximum value of the y-axis of a graph.
▨	Channel.TypicalMinValue	**TypicalMinValue:** Single	read-only		Minimum value of the y-axis of a graph.
▨	Channel.Unit	**Unit_:** WideString	read/write		The unit assigned to a channel.
▨	Channel.Used	**Used:** WordBool	read/write		Whether a channel is used or not.
▨	Channel.UserScaleMax	**UserScaleMax:** Single	read/write		User defined maximum value of the y-axis.
▨	Channel.UserScaleMin	**UserScaleMin:** Single	read/write		User defined minimum value of the y-axis.

IChannelConnection					
▤	ChannelConnection.AType	**AType:** ConnTypes	read/write		The type of IChannelConnection can be **ctLast**… for reading a certain amount (specified by "BlockSize") of the last data; **ctOverlap**… for reading blocks (size specified by "BlockSize") of data; The percentage of an overlap of the data blocks can be specified by Overlap; **ctTrigger**… for reading the data which is displayed in the Scope-screen on triggers (Before, the trigger has to be set up manually. Setting up the trigger via a plug-in will be possible in future.). **ctNew**… for reading a certain amount (specified by "BlockSize" or "NumValues") of the last data, but only if such an amout of new data (not read yet) is available;
▤	ChannelConnection.BlockSize	**BlockSize:** Integer	read/write		The size of the data block(s) to read.
▤	ChannelConnection.Channel	**Channel** As IChannel	read-only		See IChannel.
🟩	ChannelConnection.GetDataBlocks	function **GetDataBlocks(**NumBlocks: Integer**):** OleVariant;			Returns the amount of data blocks specified by "NumBlocks".
🟩	ChannelConnection.GetDataBlocks1	procedure **GetDataBlocks1(**NumBlocks: Integer; out Data: OleVariant**)**;			Same as above implemented as a procedure.
🟩	ChannelConnection.GetDataValues	function **GetDataValues(**NumValues: Integer**):** OleVariant;			Returns a certain amount (specified by "NumValues") of data starting at the position where data has not been read yet.
🟩	ChannelConnection.GetDataValues1	procedure **GetDataValues1(**NumValues: Integer; out Data: OleVariant**)**;			Same as above implemented as a procedure.
▤	ChannelConnection.NumBlocks	**NumBlocks:** Integer	read-only		The number of available blocks of the size specified by "BlockSize" not read yet.
▤	ChannelConnection.NumValues	**NumValues:** Integer	read-only		The number of values not read yet.
▤	ChannelConnection.Overlap	**Overlap:** Integer	read/write		The percentage of how much the data blocks to read overlap each other.
🟩	ChannelConnection.Reset	procedure **Reset**;			To reset a ChannelConnection to a certain moment. (Sets back "NumValues" to 0.)
🟩	ChannelConnection.Start	procedure **Start**;			Sets the ChannelConnection to the start of the acquisition.

	ChannelConnection.GetTSBlocks	function **GetTSBlocks(**NumBlocks: Integer**):** OleVariant;			Returns the amount of timestamp blocks specified by “NumBlocks”.
	ChannelConnection.GetTSBlocks1	procedure **GetTSBlocks1(**NumBlocks: Integer; out Data: OleVariant**);**			Same as above implemented as a procedure.
	ChannelConnection.GetTSValues	function **GetTSValues(**NumValues: Integer**):** OleVariant;			Returns a certain amount (specified by “NumValues”) of timestamps starting at the position where timestamp data has not been read yet.
	ChannelConnection.GetTSValues1	procedure **GetTSValues1(**NumValues: Integer; out Data: OleVariant**);**			Same functionality as above, but implemented as a procedure.
IChannelGroup					
	ChannelGroup.Count	**Count:** Integer	read-only		The number of channels in a group.
	ChannelGroup.ExportRate	**ExportRate:** Integer	read/write		Sample rate for exporting asynchronous data to DIAdem. (can be set for each group type separately)
	ChannelGroup.GetIndexName	function **GetIndexName(**Index: T_ChIndex**):** WideString;			Returns the name of the ChannelGroup specified by Index.
	ChannelGroup.Item	**Item[**Index: Integer**]:** IChannel**;**	read-only		The items of a group (These items are the channels of the type IChannel).
	ChannelGroup.Name	**Name:** WideString	read-only		The name of a group.
IChannelGroup2 (includs the properties and methods of IChannelGroup)					
	ChannelGroup2.GetIndexNameShort	function **GetIndexName(**Index: T_ChIndex**):** WideString;			Returns the name of the index in an abbreviated version.
IChannelGroups					
	ChannelGroups.Count	**Count:** Integer	read-only		The number of channel groups.
	ChannelGroups.Item	**Item(**Index As Long**)** As IChannelGroup	read-only		The group items (These items are of the type IChannelGroup, containing the channels of the type IChannel).
IChannelList					
	ChannelList.Count	**Count:** Integer	read-only		The number of channels within a channel list.
	ChannelList.Item	**Item[**Index: Integer**]:** IChannel	read-only		One of the channels within the channel list.
ICustomExport					
	CustomExport.AbsoluteTime	**AbsoluteTime:** WordBool		Custom Export	Whether the absolute or relative time data is exported.
	CustomExport.DataCount	**DataCount:** Largeuint		Custom Export	The number of samples of a channel.
	CustomExport.EndChannel	function **EndChannel:** HResult;		Custom Export	Ends the export of a channel. (ExportType = etChannelBased)

	CustomExport.EndDataFolder	function **EndDataFolder:** HResult;		Custom Export	Ends a data folder.
	CustomExport.EndExport	function **EndExport:** HResult;		Custom Export	Ends the export.
	CustomExport.EndHeader	function **EndHeader:** HResult;		Custom Export	Ends the export of the header information.
	CustomExport.EndInfo	function **EndInfo:** HResult;		Custom Export	Ends the export of general information.
	CustomExport.EndValue	function **EndValue:** HResult;		Custom Export	Ends the export of one value of each channel. (ExportType = etValueBased)
	CustomExport.ExportType	**ExportType:** ExportTypes	read-only	Custom Export	Whether the export will be performed value- or channel-based. (etValueBased = 0; etChannelBased = 1)
	CustomExport.Extension	**Extension:** WideString	read-only	Custom Export	The file extension for the export file.
	CustomExport.FileName	**FileName:** WideString		Custom Export	The name of the export file.
	CustomExport.StartAbsValue	function **StartAbsValue(**DateTime: TDateTime**):** HResult;		Custom Export	Starts the export of one absolute value of eachchannel. (ExportType = etValueBased; absolute time axis)
	CustomExport.StartChannel	function **StartChannel(**const FieldName: WideString; const FieldUnit: WideString; Async: WordBool**):** HResult;		Custom Export	Starts the export of one channel. (ExportType = etChannelBased)
	CustomExport.StartDataField	function **StartDataField(**const FieldName: WideString; const FieldUnit: WideString; ExportRate: Integer**):** HResult;		Custom Export	Exporting the name, the unit and the rate for each channel.
	CustomExport.StartDataFolder	function **StartDataFolder(**const FolderName: WideString; AbsTime: TDateTime**):** HResult;		Custom Export	Exporting the name and the absolute start time of a data folder at the beginning of the export of a data folder.
	CustomExport.StartExport	function **StartExport(**const App: IApp**):** HResult;		Custom Export	Initializing the export.
	CustomExport.StartInfo	function **StartInfo(**const Info: WideString**):** HResult;		Custom Export	Start of exporting general information.
	CustomExportStartTimeField	function **StartTimeField(**const FieldName: WideString; const FieldUnit: WideString**):** HResult;		Custom Export	Exports the name and the unit of the time channel.
	CustomExport.StartValue	function **StartValue(**TimeValue: Double**):** HResult;		Custom Export	Starts the export of one value of each channel. (ExportType = etValueBased; relative time axis)

▣	CustomExport.SupportsAsync	**SupportsAsync:** WordBool	read-only	Custom Export	Whether export of asynchronous data is supported or not.
▣	CustomExport.SupportsSRDiv	**SupportsSRDiv:** WordBool	read-only	Custom Export	Whether sample rate dividers are supported or not.
▣	CustomExport.TimeIncrease	**TimeIncrease:** Double	read/write	Custom Export	The interval between two subsequent values of synchronous data.
●	CustomExport.WriteAsyncValue	function **WriteAsyncValue(**TimeStamp: Double; Value: Single**):** HResult;		Custom Export	Export of one asynchronous value. (ExportType = etChannelBased)
●	CustomExport.WriteInfoDate	function **WriteInfoDate(**const Description: WideString; Value: TDateTime**):** HResult;		Custom Export	Exports date information to the header of the export file.
●	CustomExport.WriteInfoInteger	function **WriteInfoInteger(**const Description: WideString; Param: Integer**):** HResult;		Custom Export	Exports Integer data to the header of the export file.
●	CustomExport.WriteInfoSingle	function **WriteInfoSingle(**const Description: WideString; Value: Single**):** HResult;		Custom Export	Exports Single data to the header of the export file.
●	CustomExport.WriteInfoString	function **WriteInfoString(**const Description: WideString; const Value: WideString**):** HResult;		Custom Export	Exports String data to the header of the export file.
●	CustomExport.WriteValue	function **WriteValue(**Value: Single**):** HResult;		Custom Export	Exports one value. (ExportType = etValueBased, or for synchronous values when ExportType = etChannelBased)
ICustomExport2					
●	CustomExport2.GetDWTypeLibVersion	function **GetDWTypeLibVersion(**out Value: Integer**):** HResult;		Custom Export	The DEWESoft type library version (DEWESoftMinorVersion) should be returned.
●	CustomExport2.Get_SupportsDouble	function **Get_SupportsDouble(**out Value: WordBool**):** HResult;		Custom Export	Whether the export supports values of the type Double.
●	CustomExport2.SetAbsMax	function **SetAbsMax(**AbsMax: Single**):** HResult;		Custom Export	Sets the absolute maximum value of a channel.
●	CustomExport2.SetAbsMin	function **SetAbsMin(**AbsMin: Single**):** HResult;		Custom Export	Sets the absolute minimum value of a channel.
●	CustomExport2.SetApp	function **SetApp(**const App: IApp**):** HResult;		Custom Export	Provides a reference to IApp.
●	CustomExport2.SetChannel	function **SetChannel(**const Ch: IChannel**)**: HResult;		Custom Export	Provides a reference to IChannel.
●	CustomExport2.SetChannelColor	function **SetChannelColor(**Color: Integer**):** HResult;		Custom Export	Sets the color of the channel.
●	CustomExport2.SetDoubleFloat	function **SetDoubleFloat(**DoubleFloat: WordBool**):** HResult;		Custom Export	Sets whether a channel contains data of the type Double.

	CustomExport2.SetRangeMax	function **SetRangeMax(**Value: Single**):** HResult;		Custom Export	Sets the maximum measurement range.
	CustomExport2.SetRangeMin	function **SetRangeMin(**Value: Single**):** HResult;		Custom Export	Sets the minimum measurement range.
	CustomExport2.SetTrigOffset	function **SetTrigOffset(**TrigTime: Double**):** HResult;		Custom Export	Sets the pretrigger time of a trigger.
	CustomExport2.StartEvents	function **StartEvents:** HResult;		Custom Export	Indicates the start of writingthe events.
	CustomExport2.StopEvents	function **StopEvents**: HResult;		Custom Export	Indicates the end of writing the events.
	CustomExport2.WriteAsyncDoubleValue	function **WriteAsyncDoubleValue(**TimeStamp: Double; Value: Double**)**: HResult;		Custom Export	Writes an asynchronous value of the type Double.
	CustomExport2.WriteDoubleValue	function **WriteDoubleValue(**Value: Double**)**: HResult;		Custom Export	Writes a synchronous value of the type Double.
	CustomExport2.WriteEvent	function **WriteEvent(**EventType: Integer; const EventTypeString: WideString; Time: Double; const Comment: WideString**)**: HResult;		Custom Export	Writes an event.
IDaqChannel					
	DaqChannel.AutoZero	**AutoZero:** WordBool;	read/write		Whether zeroing of a channel is set to automatic.
	DaqChannel.CardBitResolution	**CardBitResolution:** Integer	read-only		The resolution of the DAQ card in bits.
	DaqChannel.CardGain	**CardGain:** Double;	read-only		The gain value set at a DAQ card.
	DaqChannel.CardOffset	**CardOffset:** Double;	read-only		The offset value set at a DQ card.
	DaqChannel.ModuleGain	**ModuleGain:** Double;	read-only		The gain value set at a module.
	DaqChannel.ModuleOffset	**ModuleOffset:** Double;	read-only		The offset value set at a module.
	DaqChannel.ModuleType	**ModuleType:** Double;	read-only		The type number of a module.
	DaqChannel.SetCardGain	procedure **SetCardGain(**Gain: Single**)**;			To set the gain of the DAQ card.
IDaqData					
	DaqData.Address	**Address:** Smallint			The address of a DAQ-module.
	DaqData.CopyToString	function **CopyToString:** WideString;			Copies the module parameters to a string.
	DaqData.CopyUnitToString	function **CopyUnitToString:** WideString;			Copies the unit of measurement to a string.
	DaqData.CurrentSource	**CurrentSource:** Byte;			The type of current source of ACC modules. 0…4mA 1…8mA
	DaqData.DaqNNames	**DaqNNames[**ANameCode: Smallint**]:** WideString	read-only		The name of the DAQN-module.
	DaqData.DaqNNamesCount	**DaqNNamesCount:** Integer	read-only		The number of available DAQN-modules

▤	DaqData.Filters	**Filters[**AFilterCode: Byte**]:** WideString	read-only		The name of a filter.
▤	DaqData.FilterCode	**FilterCode:** Byte			The filter of a DAQ-module.
▤	DaqData.FiltersCount	**FiltersCount:** Integer	read-only		The number of available filters.
▤	DaqData.FREQAInputCoupling	**FREQAInputCoupling:** Shortint			The input coupling type of a FREQ-A-module. 0…AC 1…DC
▤	DaqData.FREQAOutputFilter	**FREQAOutputFilter:** Shortint			The output filter type of a FREQ-A-module. 0…slow 1…fast
▤	DaqData.FREQATriggerLevel	**FREQATriggerLevel:** Single			The trigger level of a FREQ-A-module.
▤	DaqData.HighpassType	**HighpassType:** Byte			The highpass-filter at a DAQ-module.
▤	DaqData.ICPInput	**ICPInput:** Byte			Whether a charge-module is set to ICP or charge.
●	DaqData.ModuleAmpl	function **ModuleAmpl:** Single;			Returns the module scale factor.
▤	DaqData.ModuleError	**ModuleError:** Byte	read-only		The error code of a module.
●	DaqData.ModuleOffset	function **ModuleOffset:** Single;	read-only		Returns the module offset.
▤	DaqData.ModuleType	**ModuleType:** Smallint			The type of a module. 2… DAQP-Bridge 5… DAQP-Charge 8… DAQP-Freq 9… DAQP-Acc 23… DAQP-Charge-A 24… DAQP-Bridge-A 26… DAQP-Freq-A 27… DAQP-ACC-A 30… DAQP-Charge-B 31… DAQP-Bridge-B 34… DAQP-V-A 35… DAQP-V-B 36… MDAQ
▤	DaqData.Name	**Name:** WideString	read-only		The name of a module.
▤	DaqData.Overflow	**Overflow:** Byte			If an overflow has occurred it will be 1.
▤	DaqData.RangeCode	**RangeCode:** Byte			The code defining the range of a module.
▤	DaqData.Ranges	**Ranges[**ARangeCode: Integer**]:** WideString	read-only		The range settings according to the range code.
▤	DaqData.RangesCount	**RangesCount:** Integer	read-only		The number of valid ranges of a module.
▤	DaqData.Remote	**Remote:** Byte			Whether the module is set to remote mode.
●	DaqData.ShortCopyToString	function **ShortCopyToString:** WideString;			Copies the module parameters to a string.

●	DaqData.ThermLinearize	function **ThermLinearize(**InputVoltage: Double**):** Double;			Calculates the thermal linearized value of the input voltage
▤	DaqData.VRange	**VRange:** Byte			Is only used with DAQP-FREQ-module to indicate the trigger voltage input range.
IDaqGroup					
▤	DaqGroup.Count	**Count:** Integer	read-only		The number of channels within DaqGroup.
▤	DaqGroup.Item	**Item[**Index: Integer**]:** IDaqChannel	read-only		The list of items of the type IDaqChannel within DaqGroup.
IData					
▤	Data.ActiveChannels	**ActiveChannels:** IChannelList	read-only		All channels available according to the hardware setup. (only for internal use)
▤	Data.AllChannels	**AllChannels:** IChannelList	read-only		All channels available according to the hardware setup.
●	Data.ApplyChannels	procedure **ApplyChannels;**			Has to be called after channels of a plug-in are mounted by any user-action.
●	Data.BuildChannelList	procedure **BuildChannelList;**			To update the properties AllChannels and UsedChannels.
▤	Data.CurrentPos	**CurrentPos:** T_RecordPosition	read/write		Used in analyze mode; (corresponds to the yellow line)
●	Data.EndDataSync	procedure **EndDataSync;**			Used together with StartDataSync and IChannelConnection for synchronous manipulation of multiple channels.
▤	Data.EndStamp	**EndStamp:** T_RecordPosition	read/write		Last possible position of the cursor.
▤	Data.ExternalClock	**ExternalClock** As Long	read-only		Returns the external clock definition in clocks per revolution.
●	Data.FindChannel	function **FindChannel(**const Name: WideString**):** IChannel;			To find a channel by its name (Must be the exact name; function is case-sensitive).
●	Data.FindChannelByIndex	function **FindChannelByIndex(**Index: T_ChIndex**):** IChannel;			To find a channel by its indexes.
●	Data.FindChannelByIndex1	function **FindChannelByIndex1(**IndexLevel: Integer; I1: Integer; I2: Integer; I3: Integer; I4: Integer; I5: Integer**):** IChannel;			To find a channel by its indexes. This function is similar to IData.FindChannelByIndex. It is implemented because FindChannelByIndex does not work in conjunction with LabVIEW.
●	Data.GetIndexName	function **GetIndexName(**Index: T_ChIndex**):** WideString;			Returns the identifier of a certain index.

🟩	Data.GetIndexName1	function **GetIndexName1(**IndexLevel: Integer; I1: Integer; I2: Integer; I3: Integer; I4: Integer; I5: Integer**)**: WideString;			Returnes the identifier of a certain index. This function is similar to IData.GetIndexName. GetIndexName1 is implemented because GetIndexName dose not work in conjunction with LabVIEW.
🟩	Data.GetIndexNameShort	function **GetIndexNameShort(**ChIndex: T_ChIndex**):** WideString;			Returns the short identifier of a certain index.
🟩	Data.GetIndexNameShort1	function **GetIndexNameShort1(**IndexLevel: Integer; I1: Integer; I2: Integer; I3: Integer; I4: Integer; I5: Integer**)**: WideString;			Returnes the short identifier of a certain index. This function is similar to IData.GetIndexNameShort. GetIndexName1 is implemented because GetIndexNameShort dose not work in conjunction with LabVIEW.
🟩	Data.GetSamplesAcquired	procedure **GetSamplesAcquired(**out Mid: Integer; out Dir: Integer**);**			Called to get information about the amount of acquired data.
▨	Data.Groups	**Groups:** IChannelGroups	read-only		The groups to which data is dedicated. E.g. the data group for plug-in data is 8.
▨	Data.SampleRate	**SampleRate:** Integer	read-only		The sampling rate; not depending on the factor of the “reduced rate”.
▨	Data.Samples	**Samples:** Integer	read-only		The number of samples within a data block.
▨	Data.ShownChannels	**ShownChannels:** IChannelList	read-only		The list of channels which can be shown (e.g. CAN-bus messages cannot be shown).
🟩	Data.StartDataSync	procedure **StartDataSync;**			Used together with EndDataSync and IChannelConnection for synchronous manipulation of multiple channels.
▨	Data.StartStamp	**StartStamp:** T_RecordPosition	read/write		First possible position of the cursor.
▨	Data.StartStoreTime	**StartStoreTime**: TDateTime	read-only		Absolute time of the start of storing.
▨	Data.StartStoreTimeUTC	**StartStoreTime**: TDateTime	read-only		Absolute UTC of the start of storing.
▨	Data.UsedChannels	**UsedChannels:** IChannelList	read-only		A list of the channels which are set to used.
IDataSection					
▨	DataSection.DataCount	**DataCount:** Integer	read-only		The number of samples within a DataSection.
🟩	DataSection.ReadData	function **ReadData(**const Channel: IChannel; out Timestamps: OleVariant**):** OleVariant;			Reading the whole data of a section (part between a start and a stop event), independent of the data blocks.
🟩	DataSection.ReadData1	procedure **ReadData1(**const Channel: IChannel; out Data: OleVariant; out Timestamps: OleVariant**);**			Same as above implemented as a procedure.
▨	DataSection.Time	**Time:** TDateTime	read-only		Absolute start time of a section.

	DataSection.TrigPos	**TrigPos:** Integer	read-only		The sample position of the trigger event within a section, counted from the start of the section.
IDataSections					
	DataSections.Count	**Count:** Integer	read-only		The number of sections where data is stored.
	DataSections.Item	**Item[**Index: Integer**]:** IDataSection	read-only		The sections where data is stored.
IEvent					
	Event.Data	**Data:** OleVariant	read-only		The data of an event. This can be e.g. text or voice data. (not yet implemented)
	Event.PosDir	**PosDir:** Integer	read-only		Position within a data block.
	Event.PosMid	**PosMid:** Integer	read-only		Number of the data block.
	Event.TimeStamp	**TimeStamp:** Double	read-only		The time stamp of an event in seconds.
	Event.Type_	**Type_:** Integer	read-only		The type of an event. This can be: 1 indicating “start”, 2 indicating “stop”, 3 indicating “trigger”, 11 indicating “VStart” (video start storing) 12 indicating “VStop” (video stop storing) 20 indicating “Keyboard” 21 indicating “Notice” 22 indicating “Voice” 24 indicating “Module” (example: Bridge shunt on/off).
IEventList					
	EventList.Count	**Count:** Integer	read-only		The number of events.
	EventList.Item	**Item[**Index: Integer**]:** IEvent	read-only		The event items.
IExportFrame					
	ExportFrame.Apply	procedure **Apply**;			When the “Export data” button is clicked.
	ExportFrame.HideFrame	procedure **HideFrame**;			When the custom export is left.
	ExportFrame.ShowFrame	procedure **ShowFrame(**Handle: Integer; out FrameHeight: Integer**)**;			When the setup frame of the custom export is shown.
IFileNameSettings					
	FileNameSettings.AutoCreate	**AutoCreate:** WordBool	read/write		Whether a data file name should be generated automatically.
	FileNameSettings.AutoFlipAbsTime	**AutoFlipAbsTime:** WordBool	read/write		Whether absolute time is used or not.
	FileNameSettings.AutoFlipFile	**AutoFlipFile:** WordBool	read/write		Whether a new file should be created after a certain criterion.
	FileNameSettings.AutoFlipSize	**AutoFlipSize:** Single	read/write		The size of a certain unit after which a new file will be created.

▦	FileNameSettings.AutoFlipUnit	**AutoFlipUnit:** Integer	read/write		The unit of the AutoFlipSize which can be MB, h, min, sec or triggers.
▦	FileNameSettings.BaseFileName	**BaseFileName:** WideString	read/write		The file name prefix for automatic naming.
▦	FileNameSettings.MultiFileStartIndex	**MultiFileStartIndex:** Integer	read/write		The number to start from for the naming of multi files.
▦	FileNameSettings.UseDate	**UseDate:** WordBool	read/write		Whether the date should be included in the file name.
▦	FileNameSettings.UseMultiFile	**UseMultiFile:** WordBool	read/write		Whether to use the multi file option or not.
▦	FileNameSettings.UseTime	**UseTime:** WordBool	read/write		Whether the time should be included in the file name.
ILoadEngine					
●	LoadEngine.CloseFile	procedure **CloseFile**;			To close a measurement file.
▦	LoadEngine.DataSections	**DataSections:** IDataSections	read-only	autom.	See IDataSection.
▦	LoadEngine.FileOpened	**FileOpened:** WordBool	read-only	autom.	Whether a measurement file is opened or not.
●	LoadEngine.NextDBLoad	function **NextDBLoad:** WordBool;		autom.	Proceeds to the next data block of the direct buffer. (See StartDBLoad)
▦	LoadEngine.NumBlocks	**NumBlocks:** Integer	read-only	autom.	Returns the number of data blocks.
▦	LoadEngine.ReducedOnly	**ReducedOnly:** WordBool	read-only	autom.	Whether data has been stored reduced only.
●	LoadEngine.Reload	procedure **Reload(**Start: T_RecordPosition; Stop: T_RecordPosition**)**;		autom.	Reloads a specific section of data.
●	LoadEngine.ReloadBlock	procedure **ReloadBlock(**Num: Integer**)**;		autom.	Called to reload a data block.
●	LoadEngine.ShrinkFile	procedure **ShrinkFile(**const FileName: WideString**)**;		autom.	For exporting a part of a measurement limited by Data.StartStamp und Data.EndStamp.
●	LoadEngine.StartDBLoad	procedure **StartDBLoad(**Start: T_RecordPosition; Stop: T_RecordPosition; BlockSize: Integer**)**;		autom.	Starts the loading of datablocks beginning at a certain position within the direct buffer, to a certain position within the direct buffer, of a certain blocksize. (See NextDBLoad)
▦	LoadEngine.VideoLoadEngine	**VideoLoadEngines:** IVideoLoadEngines	read-only	autom.	See IVideoLoadEngines.
IMasterClock					
●	MasterClock.GetCurrentTime	function **GetCurrentTime:** Double;			The current time stamp during measurement.
IModule					
●	Module.ClearModule	procedure **ClearModule**;			Sets the address of a PAD-module back to 0.
▦	Module.DaqData	**DaqData:** IDaqData	read-only		See IDaqData.
●	Module.DetectModule	procedure **DetectModule(**Address: Integer**)**;			Detects a module at a certain address.
●	Module.FillModule	function **FillModule(**Address: Integer; Timeout: Integer**):** WordBool;			Tries to set a PAD-module having address 0 to the given address and waits for pressing the button on the module.

🟢	Module.FREQAFindTriggerLeve	procedure **FREQAFindTriggerLevel**;			Activates the auto-trigger of the FREQA-module.
🟢	Module.GetPadData	procedure **GetPadData**;			Retrieves the current data from the PAD-module.
🟢	Module.GetSerialNumber	function **GetSerialNumber:** WideString;			Reads out the serial number of the module.
▨	Module.Index	property **Index:** Smallint	read-only		The index of the module.
▨	Module.ModuleType	property **ModuleType:** Smallint			The type of a module. 0…none 1…PAD 2…DAQ
▨	Module.PadData	property **PadData:** IPadData	read-only		See IPadData.
🟢	Module.SetDaq	function **SetDaq:** WordBool;			Applies the configuration to a PAD-module.
🟢	Module.SetDaqAddress	procedure **SetDaqAddress(**Address: Integer; Timeout: Integer**)**;			Changes the address of the DAQ-module.
🟢	Module.SetModule	procedure **SetModule(**SetType: Integer**)**;			To apply the setting to a module. SetType always has to 2.
🟢	Module.SetPad	procedure **SetPad(**NewAddress: Integer**)**;			Sets the address and applies the configuration to a PAD-module.
IModules					
▨	Modules.Count	**Count:** Integer	read-only		The number of available modules.
▨	Modules.Item	**Item[**Index: Integer**]:** IModule	read-only		The array of modules of the type IModule.
IPadData					
▨	PadData.Address	**Address:** Smallint			The PAD-module’s address.
▨	PadData.ConfigCode	**ConfigCode:** Smallint	read-only		The configuration code of a PAD-module
🟢	PadData.CopyToString	function **CopyToString:** WideString;			Returns the data of the PAD-module as a string.
🟢	PadData.CopyUnitToString	function **CopyUnitToString:** WideString;			Returns the unit of the PAD-module as a string.
▨	PadData.Data[ ]	**Data[**Index: Integer**]:** Single	read-only		The measurement data of the PAD-module as an array of Single values.
🟢	PadData.ModuleAmpl	function **ModuleAmpl:** Single;			Returns module scale factor.
🟢	PadData.ModuleOffset	function **ModuleOffset:** Single;			Returns the offset of the PAD-module.

▦	PadData.ModuleType	**ModuleType:** Smallint	read-only		the type of the PAD-module which can be one of the following: 0…Unknown 1…PAD-V8 2…PAD-VTH 3…PAD-TH8 4…PAD-RTD3 5…PAD-AO1 6…PAD-CNT 7…PAD-DI8 8…PAD-DO7 9…PAD-V8-P 10…PAD-TH8-P
▦	PadData.Name	**Name:** WideString	read-only		The name of the PAD-Module.
▦	PadData.RangeCode	**RangeCode:** Smallint			The range code of the PAD-module.
▦	PadData.RangeIndex[ ]	**RangeIndex[**Index: Integer**]:** Integer	read-only		Gives the range code of choosen range at Index.
▦	PadData.Ranges[ ]	**Ranges[**Index: Integer**]:** WideString	read-only		Gives the description of a range at the index.
▦	PadData.RangesCount	**RangesCount:** Integer	read-only		The number of ranges available for the module.
●	PadData.ShortCopyToString	function **ShortCopyToString:** WideString;			Returns the data of the PAD-module in a short version (only the values) as a string.
▦	PadData.SpeedCode	**SpeedCode:** Smallint	read-only		The baudrate set on the module. The code is within the range 3…10 and having the following meaning: 3…1200 baud 4…2400 baud 5…4800 baud 6…9600 baud 7…19200 baud 8…38400 baud 9…57600 baud 10…115200 baud
IPlugin2					
●	Plugin2.ClearChannelsInstance	procedure **ClearChannelsInstance**;		Plug-in	Before all channels are destroyed in DEWESoft. All instances have to be set to “nil” or “nothing”.
●	Plugin2.Configure	procedure **Configure**;		Plug-in	When the “Configure”-Button is clicked. This button appears if there is no plug-in frame. Clicking can e.g. cause a configuration dialog to be opened.

	Plugin2.HideFrame	procedure **HideFrame**;		Plug-in	When the setup screen is left.
	Plugin2.Id	**Id:** WideString	read-only	Plug-in	GUID (Globally Unique IDentifier) of the plug-in library.
	Plugin2.Initiate	procedure **Initiate(**const DeweApp: IApp**)**;	read-only	Plug-in	When DEWESoft is started and the plug-in is checked in the hardware configuration.
	Plugin2.LoadSetup	procedure **LoadSetup(**Data: OleVariant**)**;		Plug-in	When a channel setup is loaded (corresponds to File >> Load Setup). Data is an OleVariant and will contain plug-in specific settings if such settings are stored using SaveSetup(Data).
	Plugin2.NewSetup	procedure **NewSetup**;		Plug-in	When a new channel setup is created (corresponds to File >> New Setup).
	Plugin2.OnGetData	procedure **OnGetData**;		Plug-in	When new data is added to a channel. This happens every 40 ms. It is not necessary to add any new data.
	Plugin2.OnOleMsg	procedure **OnOleMsg(**Msg: Integer; Param: Integer**)**;		Plug-in	When a plug-in has additional threads, these threads cannot directly send messages to DEWESoft at any time. Therefore additional threads have to use “App.MainWndMessage”. “OnOleMsg” will follow in order to be able to send the message to DEWESoft from within the main thread. (See App.MainWndMessage)
	Plugin2.OnStartAcq	procedure **OnStartAcq**;		Plug-in	When data acquisition starts.
	Plugin2.OnStartStoring	procedure **OnStartStoring**;		Plug-in	When the storing of data is started.
	Plugin2.OnStopAcq	procedure **OnStopAcq**;		Plug-in	When data acquisition is stopped.
	Plugin2.OnStopStoring	procedure **OnStopStoring**;		Plug-in	When the storing of data is stopped.
	Plugin2.OnTrigger	procedure **OnTrigger(**Time: Double**)**;		Plug-in	When a trigger occurs. Returns the corresponding measurement time in seconds.
	Plugin2.SaveSetup	procedure **SaveSetup(**var Data: OleVariant**)**;		Plug-in	When a channel setup is saved (corresponds to File >> Save Setup). The OleVariant Data can be used to store plug-in specific settings, which will will be read again at LoadSetup(Data).
	Plugin2.ShowFrame	function **ShowFrame(**Parent: Integer**):** WordBool;		Plug-in	When entering the setup screen. True if there is any frame to show; otherwise false. Returns “Parent”. Only useful with programming environments supporting frames (e.g. not MS Visual Basic).

●	Plugin2.UpdateFrame	procedure **UpdateFrame**;		Plug-in	When the frame is updated. This happens every 0.1 seconds when the application is in setup mode.
■	Plugin2.Used	**Used:** WordBool	read/write	Plug-in	Corresponds to the check box in the hardware configuration in the plug-in tab.
IPlugin3					
●	Plugin3.GetDWTypeLibVersion	function **GetDWTypeLibVersion**: Integer;		Plug-in	To return the DEWESoft type library version.
●	Plugin3.OnAfterCalcMath	procedure **OnAfterCalcMath**;		Plug-in	Similar to IPlugin2.OnGetData but called after calculation of Math channels, to be used if the results of the Math channels are relevant for the Plug-in.
●	Plugin3.OnAfterStartAcq	procedure **OnAfterStartAcq**;		Plug-in	Called by DEWESoft on starting the acquisition (equal to IPlugin2.OnStartAcq).
●	Plugin3.OnAfterStopAcq	procedure **OnAfterStopAcq**;		Plug-in	Called by DEWESoft on stopping the acquisition (equal to IPlugin2.OnStopAcq).
●	Plugin3.OnAlarm	procedure **OnAlarm(**CondIndex: Integer; Status: WordBool**)**;		Plug-in	Called by DEWESoft on the occurrence of an alarm.
●	Plugin3.OnBeforeStartAcq	procedure **OnBeforeStartAcq(**var AllowStart: WordBool**)**;		Plug-in	Called by DEWESoft before the acquisition is started. (For initialization of the plug-in.)
●	Plugin3.OnBeforeStopAcq	procedure **OnBeforeStopAcq(**var AllowStop: WordBool**)**;		Plug-in	Called by DEWESoft before the acquisition is stopped. AllowStop determines whether tha acquisition is allowed to stop (default: True).
●	Plugin3.OnBigListLoad	procedure **OnBigListLoad(**const TextSetup: WideString**)**;		Plug-in	For internal use only.
●	Plugin3.OnExit	procedure **OnExit**;		Plug-in	Called by DEWESoft on closing.
●	Plugin3.OnGetClock	procedure **OnGetClock(**var ClockLow: Integer; var ClockHigh: Integer**)**;		Plug-in	To provide a clock to DEWESoft, if no AD-hardware is used.
●	Plugin3.OnGetSetupData.	procedure **OnGetSetupData**;		Plug-in	Similar to OnGetData, but will be called periodically when the setup screen is active.
●	Plugin3.OnRepaintFrame	procedure **OnRepaintFrame**;		Plug-in	Is called by DEWESoft 10 times per second for updating some information displayed in the plug-in frame.
●	Plugin3.OnStartSetup	procedure **OnStartSetup**;		Plug-in	Called by DEWESoft when the setup screen is entered.
●	Plugin3.OnStopSetup	procedure **OnStopSetup**;		Plug-in	Called by DEWESoft, when the setup screen is left.
●	Plugin3.OnTriggerStop	procedure **OnTriggerStop(**Time: Double; TrigDuration: Double**)**;		Plug-in	When the stop-trigger event happens

●	Plugin3.ProvidesClock	procedure **ProvidesClock(**var Value: WordBool**);**		Plug-in	Sets whether the plug-in provides a clock to DEWESoft or not.
●	Plugin3.SetCANPort	procedure **SetCANPort(**Port: Integer**);**		Plug-in	Setting a CAN-port for using it for the plug-in.
IPluginGroup					
●	PluginGroup.ClearAllChannels	procedure **ClearAllChannels**;		Plug-in	Clear all mounted channels.
▨	PluginGroup.Count	**Count:** Integer	read-only	Plug-in	The number of mounted channels within the plug-in group.
▨	PluginGroup.ExportRate	**ExportRate:** Integer	read/write	Plug-in	Sample rate for exporting asynchronous data to DIAdem.
●	PluginGroup.GetIndexName	function **GetIndexName(**Index: T_ChIndex**):** WideString;		Plug-in	Returns the name of an index of a channel.
▨	PluginGroup.Item	**Item[**Index: Integer**]:** IChannel	read-only	Plug-in	The item of a PluginGroup specified by Index.
●	PluginGroup.MountChannel	function **MountChannel(**DataType: Integer; Async: WordBool; DBSize: Integer**):** IChannel;		Plug-in	To mount a channel to a certain group. DataType can be: 0…dtByte, 1…dtShortInt, 2…dtSmallInt, 3…dtWord, 4…dtInteger, 5…dtSingle, 6…dtInt64, 7…dtDouble. Async hast to be True if it is an asynchronous channel, otherwise False. DBSize specifies the size of the direct buffer. If DBSize is set to -1, the default buffer size is applied.
▨	PluginGroup.Name	**Name:** WideString	read-only	Plug-in	The name of a channel within the plug-in group.
●	PluginGroup.UnmountChannel	procedure **UnmountChannel(**var Channel: IChannel**);**		Plug-in	Allows removing one channel separately.
IScreen					
▨	Screen.Id	**Id:** Integer	read-only		The ID of a screen e.g. an “Overview”, a “Scope”, a “Recorder”, etc. (each of these categories can have more then one screen).
▨	Screen.IsCurrent	**IsCurrent:** WordBool	read-only		True, if the screen is currently shown.
▨	Screen.Name	**Name:** WideString	read-only		The name of a screen.
●	Screen.Show	procedure **Show**;			To show one specific screen.

IScreens					
	Screens.Count	**Count:** Integer	read-only		The number of screens.
	Screens.Item	**Item[**Index: Integer**]:** IScreen	read-only		The screen items.
IStoreEngine					
	StoreEngine.AddNewEvent	procedure **AddNewEvent(**Type_: EventType; Data: OleVariant**);**			To add an event during storing of data.
	StoreEngine.FileSize	**FileSize:** Largeuint	read-only		The current file size in bytes.
	StoreEngine.IsTriggering	**IsTriggering:** WordBool	read-only		The state between a start- and a stop-trigger.
	StoreEngine.StoreMode	**StoreMode:** Integer	read-only		The Storing options like in the measurement setup. 0… "always fast" 1… "always slow" 2… "fast on trigger" 3… "fast on trigger, slow otherwise"
	StoreEngine.Storing	**Storing:** WordBool	read-only		Whether storing is currently active or not.
ITrig					
	Trig.NotOrList	**NotOrList:** ITriggerCondList;	read-only		NotOrList provides a trigger condition list of the type ITriggerCondList. The trigger conditions of this list are equal to the "Don't store" conditions as found in DEWESoft's measurement setup.
	Trig.OrList	**OrList:** ITriggerCondList;	read-only		OrList provides a trigger condition list of the type ITriggerCondList. The trigger conditions of this list are OR-combined.
	Trig.TrigIndex	**TrigIndex:** Integer	read-only		The index of the condition in the OrList which caused the trigger.
ITrigger					
	Trigger.HoldoffTime	**HoldoffTime**: Single	read/write		The holdoff time in ms.
	Trigger.HoldoffTimeUsed	**HoldoffTimeUsed**: WordBool	read/write		Whether the holdoff time is used or not.
	Trigger.PostTime	**PostTime**: Single	read/write		The post trigger time.
	Trigger.PostTimeUsed	**PostTimeUsed**: WordBool	read/write		Whether the PostTime is used or not.
	Trigger.PostTimeExtensionUsed	**PostTimeExtensionUsed**: WordBool	read/write		Whether the post time extension is used or not.
	Trigger.PreTime	**PreTime**: Single	read/write		The pre trigger time.
	Trigger.PreTimeUsed	**PreTimeUsed**: WordBool	read/write		Whether the PreTime is used or not.
	Trigger.StartTrig	**StartTrig**: ITrig	read-only		Start trigger
	Trigger.StopTrig	**StopTrig**: ITrig	read-only		Stop trigger

ITriggerCondition					
	TriggerCondition.Direction	**Direction:** Integer;			Direction defines the direction of a trigger condition. Valid values of this property are dependant on the Mode. For modes 0,1,4,5: 0…positive 1…negative 2…any edge For modes 2, 3: 0…enter 1…leave
	TriggerCondition.Level1	**Level1:** Single;			Level1 is the first trigger level.If only one of the trigger levels is available, this has to be assigned. So, it will be used in conjunction with any trigger mode if the trigger type is 0 (“on data”).
	TriggerCondition.Level2	**Level2:** Single;			Level2 is an additional trigger level which will be used in conjunction with the trigger modes tmFilteredEdge (1), tmWindow (2) and tmWindowPulsewidth (4), when the trigger type is 0 (“on data”).
	TriggerCondition.Mode	**Mode:** Integer;			Mode defines the trigger mode of the trigger condition, which can be one of the following: 0… tmSimpleEdge 1… tmFilteredEdge 2… tmWindow 3… tmPulsewidth 4… tmWindowPulsewidth 5… tmSlope
	TriggerCondition.TrigType	**TrigType:** Integer;			TrigType defines the type of a trigger which can be one of the following: 0…on data 1…on time 2…on FFT

	TriggerCondition.TrigValue	**TrigValue:** Integer;			TrigValue determines which kind of value is used for the recognition of any trigger condition. This can be one of the following: 0…real data 1…average 2…RMS
ITriggerCondList					
	TriggerCondList.Add	function **Add:** ITriggerCondition;			Adds a trigger condition of the type ITriggerCondition to the list of trigger conditions.
	TriggerCondList.Count	Count: Integer;	read/write		The number of trigger conditions within the trigger condition list.
	TriggerCondList.Item[ ]	Item[Index: Integer]: ITriggerCondition;	read/write		An array of trigger conditions of the type ITriggerCondition. The index can be within the range of 0…Count–1.
	TriggerCondList.Remove	procedure **Remove(**Ind: Integer**)**;			Removes the trigger condition specified by the index Ind. Ind has to be within the range 0...Count–1.
IVideoLoadEngine					
	VideoLoadEngine.GetFramesInfo	procedure **GetFramesInfo(**out Frames: OleVariant**)**;			Gives information about the timestamps of the video-frames of one video file.
IVideoLoadEngines					
	VideoLoadEngines.Count	**Count**: Integer	read-only		The number of VideoLoadEngines.
	VideoLoadEngines.Item	**Item[**Index: Integer**]**: IVideoLoadEngine	read-only		One of the VideoLoadEngines which is of the type IVideoLoadEngine.
AOOperationMode					
	AOOperationMode.aomFixed	aomFixed = 0			
	AOOperationMode.aomSweep	aomSweep = 1			
	AOOperationMode.aomStepSweep	aomStepSweep = 2			
	AOOperationMode.aomBurst	aomBurst = 3			
	AOOperationMode.aomChirp	aomChirp = 4			
AOSweepMode					
	AOSweepMode.aosSingle	aosSingle = 0			
	AOSweepMode.aosLoop	aosLoop = 1			
AOWaveForm					
	AOWaveForm.aowSine	aowSine = 0			
	AOWaveForm.aowTrian	aowTrian = 1			
	AOWaveForm.aowRect	aowRect = 2			

	AOWaveForm.aowSaw	aowSaw = 3			
	AOWaveForm.aowWhiteNoise	aowWhiteNoise = 4			
	AOWaveForm.aowArbitrary	aowArbitrary = 5			
MenuItems					
	MenuItems.ItemLoadSetup	ItemLoadSetup = 2			
	MenuItems.ItemSaveSetup	ItemSaveSetup = 0			
	MenueItem.ItemSaveSetupAs	ItemSaveSetupAs = 1			
EventReason					
	EventReason.erGetData	erGetData = 0			
	EventReason.erStartStoring	erStartStoring = 1			
	EventReason.erStopStoring	erStopStoring = 2			
	EventReason.erTrigger	erTrigger = 3			
	EventReason.erException	erException = 4			
	EventReason.erTriggerStop	erTriggerStop = 5			
	EventReason.erAlarm	erAlarm = 6			
	EventReason.erExit	erExit = 7			
	EventReason.erDataLost	erDataLost = 8			
EventType					
	EventType.etStart	etStart = 1			
	EventType.etStop	etStop = 2			
	EventType.etTrigger	etTrigger = 3			
	EventType.etKeyboard	etKeyboard = 20			
	EventType.etText	etText = 21			
	EventType.etVoice	etVoice = 22			
	EventType.etPicture	etPicture = 23			
	EventType.etModule	etModule = 24			
	EventType.etVStart	etVStart = 11			
	EventType.etVStop	etVStop = 12			
ExportTypes					
	ExportTypes.etValueBased	etValueBased = 0			
	ExportTypes.etChannelBased	etChannelBased = 1			
T_CANFrame					
	T_CANFrame.Byte0	Byte0: Byte			
	T_CANFrame.Byte1	Byte1: Byte			
	T_CANFrame.Byte2	Byte2: Byte			
	T_CANFrame.Byte3	Byte3: Byte			
	T_CANFrame.Byte4	Byte4: Byte			

	T_CANFrame.Byte5	Byte5: Byte			
	T_CANFrame.Byte6	Byte6: Byte			
	T_CANFrame.Byte7	Byte7: Byte			
T_ChIndex					
	T_ChIndex.Index1	**Index1:** Integer			Group
	T_ChIndex.Index2	**Index2:** Integer			Group dependant value.
	T_ChIndex.Index3	**Index3:** Integer			Group dependant value.
	T_ChIndex.Index4	**Index4:** Integer			Group dependant value.
	T_ChIndex.Index5	**Index5:** Integer			Group dependant value.
	T_ChIndex.IndexLevel	**IndexLevel:** Smallint			Number of used indexes.
T_RecordPosition					
	T_RecordPosition.Mid	**Mid:** Integer			
	T_RecordPosition.Dir	**Dir:** Integer			
T_ReducedRec					
	T_ReducedRec.Ave	**Ave:** Single	-		Average value in the reduced buffer.
	T_ReducedRec.Max	**Max:** Single	-		Maximum value in the reduced buffer.
	T_ReducedRec.Min	**Min:** Single	-		Minimum value in the reduced buffer.
	T_ReducedRec.Rms	**Rms:** Single	-		RMS value in the reduced buffer.
TSRDivType					
	TSRDivType.sdSkip	sdSkip = 0			
	TSRDivType.sdAverage	sdAverage = 1			
	TSRDivType.sdFilter4th	sdFilter4th = 2			
	TSRDivType.sdFilter6th	sdFilter6th = 3			
	TSRDivType.sdFilter8th	sdFilter8th = 4			

Index