Agilent 81910A

Photonic All-Parameter Analyzer

Programmer’s Guide

Agilent Technologies

Notices

Manual Part Number

81910-90A61

Edition

Second edition, September 2004

First edition, April 2003

Printed in Germany

Agilent Technologies, Deutschland GmbH
Herrenberger Straße 130
71034 Böblingen, Germany.

Warranty

Technology Licenses

Restricted Rights Legend

Safety Notices

CAUTION

A **CAUTION** notice denotes a hazard. It calls attention to an operating procedure, practice, or the like that, if not correctly performed or adhered to, could result in damage to the product or loss of important data. Do not proceed beyond a **CAUTION** notice until the indicated conditions are fully understood and met.

WARNING

A WARNING notice denotes a hazard. It calls attention to an operating procedure, practice, or the like that, if not correctly performed or adhered to, could result in personal injury or death. Do not proceed beyond a WARNING notice until the indicated conditions are fully understood and met.

In This Guide....

This Programming Guide contains information about using the Agilent Photonic Foundation Library (PFL) Component Object Model (COM) Application Program Interface (API), and descriptions of all the objects and collections in the library.

This API is designed to perform all-parameter measurements using the optical test head supplied with the Agilent 81910A Photonic All-Parameter Analyzer

Contents

1
Understanding the Agilent PFL COM API

The Agilent Photonic Foundation Library (PFL) includes a COM (Component Object Model) API (Application Programming Interface) designed to perform all-parameter measurements using the optical test head supplied with the Agilent 81910A Photonic All-Parameter Analyzer.

This chapter outlines how the Agilent PFL COM API is structured and used.

About the PFL COM API

The Agilent PFL COM API allows you to automate the measurements offered by the Agilent 81910A Photonic All-Parameter Analyzer. Optical measurement result (*.omr) files can be created and read by both the PFL API and PAT.

Programming environments: The PFL COM API is compatible with the following programming environments:

- Visual Basic,
- Visual C/C++,
- LabView.

Software programmed using the COM API is both simple and readable.

Distribution: This COM API is included as PFLAPI.dll with the PAT distribution, and is registered in the Windows registry by its installer.

After registration, all programming environments that support the use of COM automation servers can use the PFL. No additional files, such as headers, are required.

Measurement Procedure

Generally, a measurement procedure requires a sequence of three steps.

To perform a measurement procedure:

Step	Action	Notes
1 Set up	**a** Define the instruments required, such as a tunable laser, power meters, and so on. **b** Define the measurement parameters, such as start wavelength and stop wavelength. **c** Verify the measurement parameters against the instruments' capabilities. **d** Store the measurement parameters to file, if required.	• Referred to as *resources* here.
2 Perform reference measurements	**a** Perform reference measurements. **b** Store the reference measurement results to file, if required.	• Different reference measurements are required for different measurement settings (For example, reflection, termination).
3 Run DUT Measurements	**a** Characterize the DUT based on the preceeding reference measurements. **b** Store the measurement results to file, if required.	

Interface model

When programming using the Agilent PFL COM API, you deal with a number of key interfaces, each supporting a partial functional area.

This release of the Agilent PFL COM API is structured around the measurements required for All-Parameter Passive Component Test, which is abbreviated to ApPact in several of the identifiers used (abbreviations preceeded by I refer to interface objects).

Table 1 Interface Model

Dependence	Key Interface	Provides
Independent Interfaces that are independent of measurement type.	IPflManager Interface	PFL house-keeping functions
	IResourceManager Interface	Bus scanning and attached instrument detection functions
IApPact Interfaces that are designed to support All-Parameter Passive Component Testing.	IApPactMeasurementControl Interface	Measurement process control functions
	IApPactMeasurementData Interface	An environment for handling user settings and measurement results

The reference chapters of this Guide are structured according to these key interfaces, and the additional COM classes (these contain properties only, so are equivalent to structs) that are provided to support them.

In the same way, additional reference chapters document the enumerations used.

See also: "Independent Interfaces - Summary" on page 83

"IApPact Interfaces - Summary" on page 165.

"Enumerations - Details" on page 144

"ApPact Enumerations - Details" on page 242.

PFL COM Classes

The most important PFL COM classes used for All-Parameter Passive Component Test are:

1. *PflManager*, which you use to initialize and close (deinitialize) the PFL.
2. *ApPactMeasurementControl*, which you use to initialize the instruments with measurement parameters, perform reference measurements, perform the DUT measurement, and retrieve the measurement data.
3. *ApPactMeasurementData,* which you use to hold all information about the measurement. This includes measurement parameters, reference curves, result curves, and general information such as DUT serial number, operator ID, and so on.

   You also use *ApPactMeasurementData* to save and load measurement data, and to perform ASCII exports.
4. *ApPactUserParams*, which is used to hold all the measurement parameters that you can specify.
5. *ApPactSystemParams*, which is used to hold measurement parameters determined by the system.
6. *Curve*, which is a container used to hold an array representing equidistant wavelength points on a spectral curve.

Class Identifiers: For some environments, it is important to know under which programmatic identifiers (ProgIDs) the COM classes that implement particular interfaces are registered.

For the PFL COM API, each class implements one PFL interface. Its class identifier is always **PFL.** followed by the interface, for example: **PFL.ApPactUserParams**

The examples for "Visual C++" on page 64 demonstrate when and how to use the class identifier.

See Also: "IPflManager Interface" on page 85

"IMeasurementControl Interface" on page 116

"IApPactMeasurementControl Interface" on page 167

"IMeasurementData Interface" on page 130

"IApPactMeasurementData Interface" on page 188

"IApPactUserParams Interface" on page 202

"IApPactSystemParams Interface" on page 225

"ICurve Interface" on page 121.

A First Program

This program, written in Visual Basic 6, runs a measurement using parameters loaded from a .omr file.

NOTE

The program assumes that suitable reference and calibration data is available in an .omr file stored via, for example, the Agilent Photonic Analysis Toolbox (PAT) GUI.

The parameters and reference measurements file

To create a Parameters and Reference Measurements file:

Step	Notes
1 Start the Photonic Analysis Toolbox.	
2 Specify the measurement parameters.	
3 Make all the reference measurements required.	
4 Store the results.	• In **C:\MyMeasurements\MyMeasurement.omr**

Visual Basic 6: A first program

The program demonstrates how to run a measurement based on parameters and reference measurements loaded from a file. The result is stored in another file.

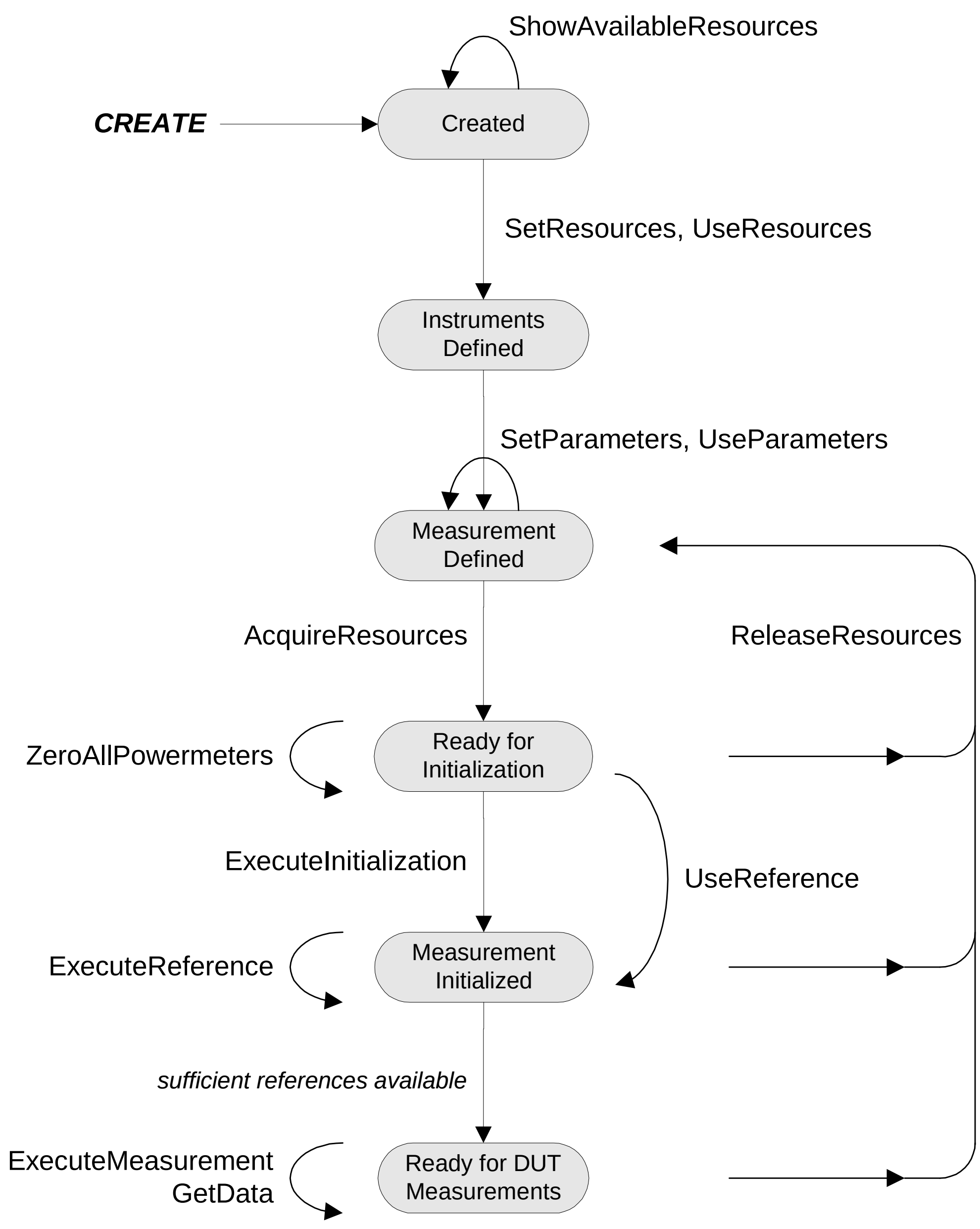
```
Private Sub Form_Load()
 ' This code loads an existing .omr measurement file into
 ' the PFL and uses its measurement parameters and references to do
 ' a simple DUT measurement.
 ' Note: The instruments specified in the file must match the real ones

 ' Initialize the PFL
 PflManager.Initialize "",TRUE

 ' Instantiate a new measurement data object
 Dim measurementData As New ApPactMeasurementData
 ' Load measurement data that contains
 '   - instrument specification
 '   - measurement parameters
 '   - reference curves
 measurementData.Load "C:\MyMeasurements\MyMeasurement.omr"

 ' Instantiate a new measurement control object for
 ' all parameter passive component test...
 Dim measurementControl As New ApPactMeasurementControl

 ' ... and attach the loaded instruments specification to use
 measurementControl.UseResources measurementData
 ' ... hand over the loaded parameters to the measurement control
 measurementControl.UseParameters measurementData

 ' ... capture the instruments and download the settings
 measurementControl.AcquireResources
 ' ... and attach the loaded reference curves
 measurementControl.UseReference measurementData

 ' Execute DUT measurements. Depending on the parameters set,
 ' CD and Mueller post-processing performed automatically
 measurementControl.ExecuteMeasurement 1

 ' Get the measurement results
 'Dim measurementData As ApPactMeasurementData
 Set measurementData = measurementControl.GetData

 ' Release the instruments
 measurementControl.ReleaseResources

 ' Save the measurement results to a file in binary format (.omr)
 measurementData.Save "C:\MyMeasurements\" _
 & "MyMeasurement2.omr"

 ' Deinitialize the PFL
 PflManager.Close
End Sub
```

See Also: "The parameters and reference measurements file" on page 20.

About the Measurement Control Sequence

Generally, the *IApPactMeasurementControl* interface is central to controlling measurements made with the Agilent 81910A Photonic All-Parameter Analyzer.

To ensure that all the parameters required for your program are specified correctly, follow this sequence:



About Initializing the PFL

First, initialize the PFL , which invokes a scan of the GPIB and LAN:

```
PflManager.Initialize "",TRUE
```

If you do not intend to access any instruments, for example when examining measurements stored in .omr files, use:

```
PflManager.Initialize "",FALSE
```

See also: "IPflManager::Initialize Method" on page 86.

About Instrument Resources

Before starting a measurement, find and acquire the required instrument resources.

IMeasurement Control: For a typical all-parameter measurement, use the *IMeasurementControl* interface to show all the available resources that are valid for an all-parameter test.

IResource Manager: The *IResourceManager* interface examines all the resources found on the GP-IB and LAN, including those not used by a measurement.

Currently, the *IResourceManager* interface offers no means of controlling instruments not used for a measurement.

See also: "IMeasurementControl::ShowAvailableResources Method" on page 117

"IResourceManager Interface" on page 87.

Visual Basic 6: Acquiring instrument resources

This code snippet demonstrates how to find and acquire the resources for a measurement control object via the *IMeasurementControl* interface.

```
' Instantiate a new measurement control object for
' all parameters passive component test...
Dim measurementControl As New ApPactMeasurementControl
' Instantiate collection for resources
Dim res As New Resources
' Ask measurement control object for connected resources
Set res = measurementControl.ShowAvailableResources

' Set resource to use
measurementControl.ConfigureResources res
```

See also: "IMeasurementControl::ShowAvailableResources Method" on page 117

"IMeasurementControl::ConfigureResources Method" on page 118.

Visual Basic 6: Utilizing resources

This code snippet demonstrates how to add resources to a collection.

```
' Instantiate a new measurement control object for
' all parameters passive component test...
Dim measurementControl As New ApPactMeasurementControl
' Instantiate collection for resources
Dim resources As New Resources
' Add resources to the collection
resources.Add "GPIB0::25::INSTR
resources.Add "GPIB0::20::INSTR::0"
resources.Add "GPIB0::20::INSTR::1"
resources.Add "GPIB0::20::INSTR::2"
resources.Add "GPIB0::20::INSTR::3"
resources.Add "LAN0::192.168.250.2::INSTR "

' Set the resources to use
measurementControl.ConfigureResources resources
```

See also: "IResources Interface" on page 97

"IResources::Add Method" on page 99.

About Reference Values

Reference values are required for accurate DUT measurement results that meeet the instruments' performance specifications. The particular reference values needed depends on the parameters measured.

Where reference values are shown as optional, they are required to deliver absolute accuracy but not for relative measurements.

CAUTION Any changes to your test setup may invalidate reference measurements stored in a file.

Reference Measurement Type	Initialization	TX	RX mirror	RX termination
Fast Loss TX	Required	[optional]		
Loss/PDL TX	Required	Required		
Fast GD TX	Required	[optional]		
GD/DGD TX	Required	Required		
Fast Loss RX	Required		[optional]	[optional]
Loss/PDL RX	Required		Required	[optional]
Fast GD RX	Required		[optional]	
GD/DGD RX	Required		Required	
Realtime Loss/GD TX	Required	[optional]		
Realtime Loss/GD RX	Required		[optional]	[optional]
Realtime PDL/DGD TX	Required	Required		
Realtime PDL/DGD RX	Required		Required	[optional]

Obtain reference values by measurement, or load them from a file:

- To measure a reference value, use the *ExecuteReference* function.
- To use a reference value loaded from a file (included in the ApPactMeasurementData object), use the *UseReference* function.

In either case, determine whether all the reference values required are available using, for example, the *CheckReference* and *IsReadyForDUTMeasurement* functions.

See also: "IApPactMeasurementControl Interface" on page 167

"IApPactMeasurementControl::ExecuteReference Method" on page 174

"IApPactMeasurementControl::UseReference Method" on page 171

"IApPactMeasurementControl::CheckReference Method" on page 185

"IApPactMeasurementControl::IsReadyForDUTMeasurement Method" on page 186

"IApPactMeasurementData::ReferenceParams Property" on page 190.

Visual Basic 6: Reference measurements

This Visual Basic 6 snippet demonstrates how to execute reference measurements:

```
' Execute initialization reference measurement
measurementControl.ExecuteReference ApPactReferenceTypeInitialization, 0
' Execute transmission reference measurement
measurementControl.ExecuteReference ApPactReferenceTypeTransmission, 0
' Execute reflection reference measurement
measurementControl.ExecuteReference ApPactReferenceTypeReflection, 0
' Execute termination reference measurement
measurementControl.ExecuteReference ApPactReferenceTypeTermination, 0
```

See also: "IApPactMeasurementControl::ExecuteReference Method" on page 174.

About Measurement Parameters

Set up the user-definable loss and phase parameters required for your measurement with the *IApPactUserParams* interface, utilizing its *LossParams* and *PhaseParams* properties.

Access particular properties via the *IApPactUserLossParams* and *IApPactUserPhaseParams* interfaces.

Use the *IApPactMeasurementControl::ConfigureParameters* method to apply these parameters to a measurement, or the *UseParameters* method to apply the parameters stored in another measurement data object.

See also: "IApPactUserParams Interface" on page 202

"IApPactUserParams::LossParams Property" on page 203

"IApPactUserParams::PhaseParams Property" on page 203

"IApPactUserLossParams Interface" on page 205

"IApPactUserPhaseParams Interface" on page 216

"IApPactMeasurementControl::ConfigureParameters Method" on page 169

"IApPactMeasurementControl::UseParameters Method" on page 170.

Visual Basic 6: Setting user-defined parameters

This code snippet shows how to set up the user-defineable parameters for a measurement.

```
' Instantiate object for user parameters
Dim params As New ApPactUserParams

' Set up parameters for the loss measurements, all parameters have
' sensible defaults
With params.LossParams
  .StartWavelength = 0.000001549
  .StopWavelength = 0.0000015515
  .StepWavelength = 0.00000000001
  .MeasurementType = ApPactLossMeasurementTypePDL
  .MeasureReflection = False
  .MaxAveragingTime = AveragingTimeAuto
```

```
 .SweepCount = SweepCountOneSweep
 .EnableCoherenceControl = False
 .MaxSweepSpeed = SweepSpeedAuto
 .TransmissionStartRange = PowerRangeAuto
 .TransmissionRangeDecrement = RangeDecrementAuto
End With

' Set parameters for the phase measurements. All parameters have
' sensible defaults
With params.PhaseParams
 .StartWavelength = 0.000001549
 .StopWavelength = 0.0000015515
 .StepWavelength = 0.00000000001
 .MeasurementType = ApPactPhaseMeasurementTypeDGD
 .SupplyJonesMatrix = True
 .MeasureReflection = True
 .AveragingCount = 1
 .AveragingWindowWidth = 1
 .CDAveragingWindowWidth = 1
End With

' Hand over the parameters to the measurement control object
measurementControl.ConfigureParameters params
```

See also: "IApPactUserParams Interface" on page 202

"IApPactUserParams::LossParams Property" on page 203

"IApPactUserParams::PhaseParams Property" on page 203

"IApPactMeasurementControl::ConfigureParameters Method" on page 169.

About Measurement Data

ApPactMeasurementData contains the measurement results and all the data needed to repeat a DUT or reference measurement.

The following information can be retrieved from *ApPactMeasurementData*:

- The selected instruments,
- Information provided by the programmer about the DUT,
- The measurement parameters,
- The reference measurements,
- The measurement results.

See also: "IApPactMeasurementData Interface" on page 188

"IMeasurementData Interface" on page 130.

About Curves

The most important part of the measurement results are *Curves*. A curve is an array of floating point values where index points (which always represent wavelength) are on the X-axis and values are on the Y-axis.

Each curve is specified by a set of parameters consisting of *LightPath*, *Role* and *PolState*.

See also: "ICurve Interface" on page 121

"ICurves Interface" on page 127.

"Light Path" on page 33

"Role" on page 34

"Polarization State" on page 35.

Visual Basic 6: Accessing Curves:

Direct Access The *IApPactMeasurementData Interface* includes functions that access the most commonly required curves.

NOTE **channel** The first channel index is 1, not 0.

```
GetDUTPARAMCurve(direction, channel)

' Get Loss curves
Set curve = GetDUTLossCurve(DirectionTX, 1)
Set curve = GetDUTLossCurve(DirectionRX, 1)

' Get PDL curves
Set curve = GetDUTPDLCurve(DirectionTX, 1)
Set curve = GetDUTPDLCurve(DirectionRX, 1)

' Get GD curves
Set curve = GetDUTGDCurve(DirectionTX, 1)
Set curve = GetDUTGDCurve(DirectionRX, 1)

' Get DGD curves
Set curve = GetDUTDGDCurve(DirectionTX, 1)
Set curve = GetDUTDGDCurve(DirectionRX, 1)
```

```
' Get CD curves
Set curve = GetDUTCDCurve(DirectionTX, 1)
Set curve = GetDUTCDCurve(DirectionRX, 1)
```

See also: "IApPactMeasurementData::GetDUTLossCurve Method" on page 197

"IApPactMeasurementData::GetDUTPDLCurve Method" on page 198

"IApPactMeasurementData::GetDUTGDCurve Method" on page 199

"IApPactMeasurementData::GetDUTDGDCurve Method" on page 200

"IApPactMeasurementData::GetDUTCDCurve Method" on page 201.

Indirect access: You can also access the curve data array, which is a SAFEARRAY.

NOTE **Limitation** SAFEARRAYs are the standard array data type for COM automation, but do not work reliably in LabView so may cause a program crash.

See also: "ICurve::DataArray Property" on page 126.

GetSingleCurve To access, for example, raw Mueller Matrix data, or reference measurement curves, use the more complex *GetSingleCurve* function.

See also: "IMeasurementData::GetSingleCurve Method" on page 134.

Light Path

A LightPath specifies the way light propagates through the DUT.

For the transmission measurement of a DUT with one input and one output channel, specify the *LightPath*:

```
lightpath.DestinationDutPort.Number = 1
lightpath.DestinationDutPort.Group = DutPortGroupOut
lightpath.SourceDutPort.Number = 1
lightpath.SourceDutPort.Group = DutPortGroupIn
```

For a reflection measurement, the system measures the light reflected by the DUT, so specify the *LightPath*:

```
lightpath.DestinationDutPort.Number = 1
lightpath.DestinationDutPort.Group = DutPortGroupIn
lightpath.SourceDutPort.Number = 1
lightpath.SourceDutPort.Group = DutPortGroupIn
```

For channel N of an m-channel DUT in transmission, specify the *LightPath*:

```
lightpath.DestinationDutPort.Number = N
lightpath.DestinationDutPort.Group = DutPortGroupOut
lightpath.SourceDutPort.Number = 1
lightpath.SourceDutPort.Group = DutPortGroupIn
```

NOTE

1xN DUTs only The PFL current release allows for measurements on DUTs with 1 input and N outputs, so:

lightpath.SourceDutPort.Number must = 1

And, for reflection measurements, both:

- **lightpath.SourceDutPort.Number** must = 1, and
- **lightpath.DestinationDutPort.Number** must = 1.

See also: "ILightPath Interface" on page 112

"ILightPath::SourceDUTPort Property" on page 113

"ILightPath::DestinationDUTPort Property" on page 113

"IDUTPort Interface" on page 114

"IDUTPort::Group Property" on page 115

"IDUTPort::Number Property" on page 115.

Role

A 'role' describes how the array of measurement data that constitutes a curve is obtained.

Curve Role	Description
CurveRoleDUTPwrWvl	Power (over wavelength) obtained from DUT measurement.
CurveRoleRefPwrWvl	Power (over wavelength) obtained from reference measurement.
CurveRoleLossWvl	Insertion Loss over wavelength.
CurveRolePDLWvl	Polarization Dependent Loss over wavelength.
CurveRoleLossMinWvl	Minimum Insertion Loss over wavelength.
CurveRoleLossMaxWvl	Maximum Insertion Loss over wavelength.
CurveRoleLossAvgWvl	Average Insertion Loss over wavelength.
CurveRoleRefPhaseWvl	Reference phase over wavelength.
CurveRoleGDWvl	Group Delay (over wavelength) obtained from DUT measurement.
CurveRoleDGDWvl	Differential Group Delay (over wavelength) obtained from DUT measurement.
CurveRoleCDWvl	Chromatic Dispersion over wavelength.
CurveRoleRefPwrWvlRT	Power (over wavelength) obtained from reference. For realtime loss measurements
CurveRoleRefTermPwrWvl	Termination power (over wavelength) obtained from reference.

See also: "ICurve::Role Property" on page 123

"CurveRoleEnum Enumeration" on page 151.

"Polarization State" on page 35

"Role and Polarization State" on page 36.

Polarization State

The following polarization states are defined:

Polarization State	Description
PolStateUnknown	The light is polarized, but the exact polarization state is unknown.
PolStateUnpolarized	The light is unpolarized.
PolStateLH0	A polarization state of linear horizontal.
PolStateLDP45	A polarization state of linear diagonal positive 45.
PolStateLDN45	A polarization state of linear diagonal negative 45.
PolStateLV90	A polarization state of linear vertical 90 (defined but currently not used).
PolStateLHC	Polarization state is left hand circular (defined but currently not used).
PolStateRHC	Polarization state is right hand circular.
PolStateUndefined	The definition of a polarization state makes no sense in this context.

See also: "IPolStates Interface" on page 103

"PolStateEnum Enumeration" on page 149.

"Role" on page 34

"Role and Polarization State" on page 36.

Role and Polarization State

Each Curve Role is associated with a number of valid Polarization States. Other Polarization States are not valid in terms of the measurement that the curve represents.

For example:

- *CurveRoleDUTPwrWvl* represents the raw curves for a Mueller Matrix for a PDL measurement, which are generated at four different polarization states.
- The Polarization State of *CurveRoleGDWvl* is **Unknown** if only GD is measured. However, if obtained from a DGD measurement, the Polarization State is **Unpolarized** since GD is calculculated from averaged values.

	Polarization State						
Curve Role	**Un-known**[*]	**Un-polarized**[†]	**LH0**	**LDP 45**	**LDN 45**	**RHC**	**Un-defined**[‡]
CurveRoleDUTPwrWvl			valid	valid	valid	valid	
CurveRoleRefPwrWvl			valid	valid	valid	valid	
CurveRoleLossWvl			valid	valid	valid	valid	
CurveRolePDLWvl							valid
CurveRoleLossMinWvl							valid
CurveRoleLossMaxWvl							valid
CurveRoleLossAvgWvl		valid					
CurveRoleRefPhaseWvl	valid	valid					
CurveRoleGDWvl	valid	valid					
CurveRoleDGDWvl							valid
CurveRoleCDWvl	valid	valid					
CurveRoleRefPwrWvlRT	valid						
CurveRoleRefTermPwrWvl			valid	valid	valid	valid	

[*] **Unknown**: Measured at some unknown polarization state.

[†] **Unpolarized**: Derived from measurements at a number of different polarization states.

[‡] **Undefined**: Refers to a parameter that is inherently polarization dependent, such as PDL.

See also: "Role" on page 34

"Polarization State" on page 35.

"CurveRoleEnum Enumeration" on page 151

"PolStateEnum Enumeration" on page 149.

Visual Basic 6: Utilizing a curve object

This code snippet demonstrates how to set the parameters for phase measurement, execute reference and DUT measurements, and place the results in a curve object.

```
' Instantiate object for user parameters
Dim params As New ApPactUserParams

' Set parameters for the phase measurements.
' All parameters have sensible defaults
With params.PhaseParams
  .StartWavelength = 0.000001549
  .StopWavelength = 0.0000015515
  .StepWavelength = 0.00000000001
  .MeasurementType = ApPactPhaseMeasurementTypeDGD
  .SupplyJonesMatrix = True
  .MeasureReflection = False
  .AveragingCount = 1
  .AveragingWindowWidth = 1
  .CDAveragingWindowWidth = 1
End With

' Hand over the parameters to the measurement control object
measurementControl.ConfigureParameters params
' Capture the instruments and download the settings
measurementControl.AcquireResources

' Execute initialisation reference measurement
measurementControl.ExecuteReference _
ApPactReferenceTypeInitialization
' Execute transmission reference measurement
measurementControl.ExecuteReference _
ApPactReferenceTypeTransmission, 0
' Execute DUT measurements, acquires instruments if not already
' locked. Depending on the parameters set, CD and Mueller post-
' processing performed automatically
measurementControl.ExecuteMeasurement 1

' Instantiate a data object to get the results of the measurement
Dim measurementData As ApPactMeasurementData
' Get the measurement results
```

```
Set measurementData = measurementControl.GetData

' Declare a curve object
Dim curve As curve
' Instantiate a light path object
Dim lightpath As New lightpath

' Specify light path
lightpath.DestinationDutPort.Number = 1
lightpath.DestinationDutPort.Group = DutPortGroupOut
lightpath.SourceDutPort.Number = 1
lightpath.SourceDutPort.Group = DutPortGroupIn

' Get GD curve
Set curve = measurementData.GetSingleCurve(lightpath, _
CurveRoleGdWvl, PolStateUnpolarized)

' Release the instruments
measurementControl.ReleaseResources
```

See also: "IApPactUserParams Interface" on page 202

"IApPactUserParams::PhaseParams Property" on page 203

"IApPactMeasurementControl::ConfigureParameters Method" on page 169

"IApPactMeasurementControl::AcquireResources Method" on page 172

"IApPactMeasurementControl::ExecuteReference Method" on page 174

"IApPactMeasurementControl::ExecuteMeasurement Method" on page 176

"IApPactMeasurementControl::GetData Method" on page 184

"ILightPath Interface" on page 112

"IMeasurementData::GetSingleCurve Method" on page 134

"IApPactMeasurementControl::ReleaseResources Method" on page 183.

Visual Basic 6: Exporting data

This code snippet demonstrates how to save curve data in a form readable by the Photonic Analysis Toolkit (PAT) and to export them into a text format readable by, for example, Microsoft® Excel[*].

```
' Instantiate object for user parameters
Dim params As New ApPactUserParams

' Set parameters for the phase measurements.
' All parameters have sensible defaults
With params.PhaseParams
 .StartWavelength = 0.000001549
 .StopWavelength = 0.0000015515
 .StepWavelength = 0.00000000001
 .MeasurementType = ApPactPhaseMeasurementTypeDGD
 .SupplyJonesMatrix = True
 .MeasureReflection = False
 .AveragingCount = 1
 .AveragingWindowWidth = 1
 .CDAveragingWindowWidth = 1
End With

' Hand over the parameters to the measurement control object
measurementControl.ConfigureParameters params
' capture the instruments and download the settings
measurementControl.AcquireResources

' Execute initialisation reference measurement
measurementControl.ExecuteReference _
ApPactReferenceTypeInitialization
' Execute transmission reference measurement
measurementControl.ExecuteReference _
ApPactReferenceTypeTransmission, 0
' Execute DUT measurements, acquires instruments if not already
' locked. Depending on the parameters set, CD and Mueller post-
' processing performed automatically
measurementControl.ExecuteMeasurement 1

' Instantiate a data object to get the results of the measurement
Dim measurementData As ApPactMeasurementData
' Get the measurement results
Set measurementData = measurementControl.GetData

' Release the instruments
```

* Microsoft is a U.S. registered trademark of Microsoft Corporation.

```
measurementControl.ReleaseResources

' Save measurement results - readable by PAT
measurementData.Save "c:\measurement.omr"
' Save measurement results - text export
measurementData.Export "c:\measurement.txt"
' Save measurement results - Jones
measurementData.ExportJonesMatrix "c:\jones.txt"
```

See also: "IApPactUserParams Interface" on page 202

"IApPactUserParams::PhaseParams Property" on page 203

"IApPactMeasurementControl::ConfigureParameters Method" on page 169

"IApPactMeasurementControl::AcquireResources Method" on page 172

"IApPactMeasurementControl::ExecuteReference Method" on page 174

"IApPactMeasurementControl::ExecuteMeasurement Method" on page 176

"IApPactMeasurementControl::GetData Method" on page 184

"IApPactMeasurementControl::ReleaseResources Method" on page 183

"IMeasurementData::Save Method" on page 140

"IApPactMeasurementData::Export Method" on page 195

"IApPactMeasurementData::ExportJonesMatrix Method" on page 196.

Additional Information for Visual C++ Programmers

Using a COM API in, for example, Visual C++ requires more effort than Visual Basic. As a guide, this section describes some features of Microsoft Visual C++.

NOTE

Results depend on your chosen envionment Please review your chosen development environment's reference material for complete information. The implementation of features such as wrappers may look very different in another environment.

Wrappers

A very convenient way to deal with COM-interfaces in Visual C++ is to use the import statement:

```
#import "pflapi.dll" no_namespace
```

This statement instructs the compiler to generate a wrapper that allows easier access to the functions in the specified interface.

This approach includes exception handling, so the return value of the wrapped functions is no longer an **HRESULT** but the [out, retval] specified by the interface.

Therefore the signature of the COM API functions differs from the specification provided in this Programmer's Guide, even though the interfaces are described correctly.

Example: Using a Wrapper

The Get Single curve function is specified:

```
HRESULT GetSingleCurve(
    ILightPath* lightPath,
    CurveRoleEnum role,
    PolStateEnum polState,
    Icurve** curve);
```

However, the COM wrapper creates a function signature that looks like this:

```
ICurve * GetSingleCurve(
    ILightPath* lightPath,
    CurveRoleEnum role,
    PolStateEnum polState);
```

See also: "IMeasurementData::GetSingleCurve Method" on page 134.

Smart Pointers

Visual C++ SmartPointers allow you to access properties directly, without calling the associated Get...() function.

Example: Using a SmartPointer

Instead of programming:

startWavelength = pUserParam->PhaseParams->GetStartWavelength();

Visual C++ allows you to use:

startWavelength = pUserParam->PhaseParams->StartWavelength;

See also: "IApPactUserParams Interface" on page 202

"IApPactUserParams::PhaseParams Property" on page 203.

COM Class ProgIDs

You cannot instantiate a COM object for a particular interface without knowing the ProgID under which the interface is registered.

To derive the ProgID from the interface type, simply:

Step	Notes
1 Delete the leadling **I** and the trailing **Ptr**	• **Ptr** (from the smart pointer)
2 Attach **PFL.**	

All PFL COM classes are registered using this scheme.

Example: Using a ProgID

IApPactUserParamsPtr pUserParam("PFL.ApPactUserParams");

Error Handling

Since a Visual C++ wrapper uses exception handling to propagate errors to the user, write an error handling routine that catches these exceptions.

Example: Error handling

This program snippet demonstrates how errors can be handled:

```
try
{
   // Test: IPflManager::Initialize
   pPflManager->Initialize("", VARIANT_FALSE);
}
catch(_com_error& e)
{
   BSTR errStr;
   HRESULT lHres = e.Error();
   printf("Caught COM error, code 0x%8x\n", lHres);
   IErrorInfo * lErrorInfo = e.ErrorInfo();
   if (NULL != lErrorInfo)
   {
      // there is additional information from PFL available
      lErrorInfo->GetDescription(&errStr);
      printf("Error info: %S\n", errStr);
   }
   else
   {
      // no additional information from PFL available
      printf("No error info available; %s\n", e.ErrorMessage());
   }
}
```

This error handling method is may not be applicable outside the Microsoft Visual C++ environment. However, the **ErrorInfo** provided by the PFL COM API follows generic COM guidelines so it should be accessible from all development environments that support COM.

See also: "IPflManager Interface" on page 85.

2
Example Programs

The Agilent 81910A Photonic All-Parameter Analyzer CD-ROM includes a number of Example Programs written in Visual Basic 6, Visual C++, and LabView.

These resources are intended to aid the development of your own programs by demonstrating principles. They are not prescriptive, or necessarily illustrative of best-practice for your organization.

This chapter contains, where possible, complete code (rather than the code snippets provided in Chapter 1, “Understanding the Agilent PFL COM API).

Visual Basic

This section includes:

- Visual Basic: Master, which is a template for program development.
- Visual Basic: Example 1, which demonstrates how to load an existing PAT file, utilize its resource and measurement data to execute a DUT measurement, then store the measurement results to file.
- Visual Basic: Example 2, which demonstrates how to configure the resources for a measurement, execute reference measurements, then store the results to file.
- Visual Basic: Example 3, This program demonstrates how to import the raw data for a measurement, then separate it into its curve components.

Visual Basic: Master

This program is provided on the Agilent 81910A Photonic All-Parameter Analyzer CD-ROM in
\\Example Programs\COM API Examples\VB6 Examples\Master

Use this code as a template for program development.

```
VERSION x.xx

Begin VB.Form Form1
  Caption = "Form1"
  ClientHeight = 3195
  ClientLeft = 60
  ClientTop = 45
  ClientWidth = 4680
  LinkTopic = "Form1"
  ScaleHeight = 3195
  ScaleWidth = 4680
  StartUpPosition = 3 'Windows Default
End

Attribute VB_Name = "Form1"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False

Private Sub Form_Load()
```

```
PflManager.Initialize
' Instantiate a new measurment data object
Dim measurementData As New ApPactMeasurementData
' Load a measurement file that contains:
'   - measurement parameter
'   - reference curves
' measurementData.Load ("C:\MyMeasurements\MyMeasurement.omr")
measurementData.Load ("C:\Test.omr")

' Instantiate a new measurement control object for
' all parameter passive component test...
Dim measurementControl As New ApPactMeasurementControl

' ... and attach the loaded instruments specification to use
' measurementControl.UseResources measurementData

' ... hand over the loaded parameters to the measurement control
' measurementControl.UseParameters measurementData

' ... capture the instruments and download the settings
' measurementControl.AcquireResources

' ... and also the loaded reference curves
' measurementControl.UseReference measurementData

' Execute DUT measurements,
' Depending on the parameters set, CD and Mueller post-processing
' performed automatically
' measurementControl.ExecuteMeasurement (1)

' Get the measurement results
' Dim measurementData As ApPactMeasurementData
' Set measurementData = measurementControl.GetData()
' Release the instruments
measurementControl.ReleaseResources

' Save the measurement results to a file in binary format (.omr)
' measurementData.Save ("C:\MyMeasurements\MyMeasurement2.omr")
measurementData.Save ("C:\Test2.omr")

' Deinitialize the PFL
PflManager.Close
End Sub
```

Visual Basic: Example 1

This program is provided on the Agilent 81910A Photonic All-Parameter Analyzer CD-ROM in
\\Example Programs\COM API Examples\VB6 Examples\Example1

This program demonstrates how to load an existing .omr file, utilize its resource and measurement data to execute a DUT measurement, then store the measurement results to file.

```
VERSION x.xx

Begin VB.Form CtrlPannel
  Caption = "API Demo"
  ClientHeight = 6720
  ClientLeft = 0
  ClientTop = 345
  ClientWidth = 8385
  LinkTopic = "Form1"
  ScaleHeight = 6720
  ScaleWidth = 8385
  StartUpPosition = 3 'Windows Default
  Begin VB.ListBox Tracer
    BeginProperty Font
      Name = "Courier New"
      Size = 8.25
      Charset = 0
      Weight = 400
      Underline = 0 'False
      Italic = 0 'False
      Strikethrough = 0 'False
    EndProperty
    Height = 5940
    Left = 120
    TabIndex = 1
    Top = 120
    Width = 8175
  End
  Begin VB.CommandButton DoMeasurement
    Caption = "Run Demo"
    Height = 75
    Left = 120
    TabIndex = 0
    Top = 6240
    Width = 8175
  End
End
```

```
Attribute VB_Name = "CtrlPannel"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Private Sub DoMeasurement_Click()
On Error GoTo ErrorHandling
DoMeasurement.Enabled = False
Tracer.Clear

Dim nIdx As Integer

'instantiate a new measurement control object
'for all parameter passive component test...
Dim measurementControl As New ApPactMeasurementControl
'instantiate a measurement data object
Dim measurementData_File As New ApPactMeasurementData
'load setup from file
measurementData_File.Load "Setup.omr"

TRACE "Configure the resources for measurement using the file setup.omr"

'use resources from file
measurementControl.UseResources measurementData_File

TRACE "Configure the parameter for measurement using the file setup.omr"

'use parameter from file
measurementControl.UseParameters measurementData_File

TRACE "Acquire Resources"

'acquire resources
measurementControl.AcquireResources

TRACE "Use the references for measurement using the file setup.omr"

'use parameter from file
measurementControl.UseReference measurementData_File

Dim msgRes As Integer

'execute reference measurements, automatically acquires instruments:
msgRes = MsgBox("Connect DUT to the optical input and output. Press Ok to start the" _
& "DUT measurement.", vbOKCancel + vbInformation, "User action requiered !")
If (msgRes = vbCancel) Then GoTo Abort
TRACE "Doing transmission DUT measurement..."
```

```
'execute DUT measurements, acquires instruments if not already locked
'Depending on parameters set, CD and Mueller post-processing performed automatically:
measurementControl.ExecuteMeasurement 1

TRACE "   " + "...Done"
TRACE "Get measurement results"

'instantiate a data object to get the results of the measurement
Dim measurementData As ApPactMeasurementData
'get the measurement results
Set measurementData = measurementControl.GetData

Set params = measurementData.UserParams

TRACE "Save measurement results ..."

'save measurement results - readable by PAT
measurementData.Save "APIDemo1.omr"

TRACE "   " + "... PAT readable - APIDemo1.omr"

'save measurement results - text export
measurementData.Export "APIDemo1Curves.txt"

TRACE "   " + "... Text export  - APIDemo1Curves.txt"

'save measurement results - Jones
measurementData.ExportJonesMatrix "APIDemo1Jones.txt"

TRACE "   " + "... Jones - APIDemo1Jones.txt"

'save measurement results - Mueller
measurementData.ExportFirstRowMuellerMatrix "APIDemo1Mueller.txt"

TRACE "   " + "... Mueller - APIDemo1Mueller.txt"

TRACE "Release resources"

'release the instruments, optional
measurementControl.ReleaseResources

TRACE "-------------------- Measurement done --------------------"

'clean up objects
Set measurementData = Nothing
Set params = Nothing
Set measurementControl = Nothing
Set Crv = Nothing

DoMeasurement.Enabled = True
GoTo Ready

ErrorHandling:
  TRACE_ERROR Err.Description
  DoMeasurement.Enabled = True
  GoTo Ready
Abort:
  TRACE "### " + "Measurement aborted by user"
  DoMeasurement.Enabled = True

Ready:
End Sub

Private Sub Form_Load()
'initialize Pfl
PflManager.Initialize "", True
Tracer.ForeColor = &H80000008
TRACE "PFL Initialized"
End Sub

Private Sub Form_Terminate()
'deinitialize Pfl
PFL.Close
TRACE "PFL Deinitialized"
End Sub

Private Sub TRACE(s As String)
Tracer.AddItem s
Tracer.ListIndex = Tracer.ListCount - 1
End Sub

Private Sub TRACE_ERROR(s As String)
Tracer.ForeColor = &HFF&
TRACE "################################################"
TRACE "###" + s
End Sub
```

Visual Basic: Example 2

This program is provided on the Agilent 81910A Photonic All-Parameter Analyzer CD-ROM in
\\Example Programs\COM API Examples\VB6 Examples\Example2

This program demonstrates how to configure the resources for a measurement, execute reference measurements, then store the results to file.

```
VERSION x.xx

Begin VB.Form CtrlPannel
  Caption = "API Demo"
  ClientHeight = 6720
  ClientLeft = 60
  ClientTop = 345
  ClientWidth = 8385
  LinkTopic = "Form1"
  ScaleHeight = 6720
  ScaleWidth = 8385
  StartUpPosition = 3  'Windows Default
  Begin VB.ListBox Tracer
    BeginProperty Font
      Name = "Courier New"
      Size = 8.25
      Charset = 0
      Weight = 400
      Underline = 0   'False
      Italic =   0 'False
      Strikethrough = 0 'False
    EndProperty
    Height = 5940
    Left = 120
    TabIndex = 1
    Top = 120
    Width = 8175
  End
  Begin VB.CommandButton DoMeasurement
    Caption = "Run Demo"
    Height = 375
    Left = 120
    TabIndex = 0
    Top = 6240
    Width = 8175
  End
End
```

```
Attribute VB_Name = "CtrlPannel"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Private Sub DoMeasurement_Click()
On Error GoTo ErrorHandling
DoMeasurement.Enabled = False
Tracer.Clear

Dim nIdx As Integer

'instantiate a new measurement control object for all parameter passive component test...
Dim measurementControl As New ApPactMeasurementControl
'declare array for resources
Dim res As New Resources

TRACE "Determining available resources"

'ask measurement control object for connected instruments
Set res = measurementControl.ShowAvailableResources

For nIdx = 0 To res.Count - 1
  TRACE "   " + res.Item(nIdx)
Next

TRACE "Configure the resources for measurement"

'set instruments
measurementControl.ConfigureResources res

'instantiate object for user parameters
Dim params As New ApPactUserParams

TRACE "Configure parameters for the loss measurements"

'set up parameters for the loss measurements, all parameters have sensible defaults
With params.LossParams
  .StartWavelength = 0.000001549
  .StopWavelength = 0.0000015515
  .StepWavelength = 0.00000000001
  .MeasurementType = ApPactLossMeasurementTypePDL
  .MaxAveragingTime = AveragingTimeAuto
  .SweepCount = SweepCountOneSweep
  .EnableCoherenceControl = False
  .MaxSweepSpeed = SweepSpeedAuto
  .TransmissionStartRange = PowerRangeAuto
  .TransmissionRangeDecrement = RangeDecrementAuto
  .ReflectionStartRange = PowerRangeAuto
  .ReflectionRangeDecrement = RangeDecrementAuto
End With

TRACE "Configure parameters for the phase measurements"

'set up parameters for the phase measurements, all parameters have sensible defaults
With params.PhaseParams
  .StartWavelength = 0.000001549
  .StopWavelength = 0.0000015515
  .StepWavelength = 0.00000000001
  .MeasurementType = ApPactPhaseMeasurementTypeDGD
  .SupplyJonesMatrix = True
  .MeasureReflection = True
  .AveragingCount = 1
  .AveragingWindowWidth = 1
  .CDAveragingWindowWidth = 1
End With

TRACE "Set configured parameters to do the measurements"

'hand the parameters over to the measurement control
measurementControl.ConfigureParameters params

TRACE "Acquire Resources"

'acquire resources
measurementControl.AcquireResources

Dim msgRes As Integer
'execute reference measurements, automatically acquires instruments:
msgRes = MsgBox("Disconnect any devices from the optical input and output. Press Ok" _
& " to start the Initialization.", vbOKCancel + vbInformation, "User action requiered !")
If (msgRes = vbCancel) Then GoTo Abort
'execute initialisation reference measurement
TRACE "Doing initialisation reference measurement..."

'execute initialisation reference measurement
measurementControl.ExecuteReference ApPactReferenceTypeInitialization

TRACE "   " + "...Done"
msgRes = MsgBox("Connect the optical input and output with a patch cord. Press Ok " _
& "to start the reference measurement.", vbOKCancel + vbInformation, " User action " _
& "required !")
If (msgRes = vbCancel) Then GoTo Abort
TRACE "Doing transmission reference measurement..."

'execute transmission reference measurement
```

```
measurementControl.ExecuteReference ApPactReferenceTypeTransmission, 0

TRACE "   " + "...Done"
msgRes = MsgBox("Connect The optical input and output with a mirror. Press Ok to "_
& "start the DUT measurement.", vbOKCancel + vbInformation, "User action requiered !")
If (msgRes = vbCancel) Then GoTo Abort
TRACE "Doing reflection reference measurement..."

'execute reflection reference measurement
measurementControl.ExecuteReference ApPactReferenceTypeReflection, 0

'TRACE "Doing transmission reference measurement..."

TRACE "   " + "...Done"
msgRes = MsgBox("Connect DUT to the optical input and output. Press Ok to start the "_
& "DUT measurement.", vbOKCancel + vbInformation, "User action requiered !")
If (msgRes = vbCancel) Then GoTo Abort
TRACE "Doing transmission DUT measurement..."

'execute DUT measurements, acquires instruments if not already locked,
'CD and Mueller post-processing performed automatically:
measurementControl.ExecuteMeasurement 1

TRACE "   " + "...Done"
TRACE "Get measurement results"

'instantiate a data object to get the results of the measurement
Dim measurementData As ApPactMeasurementData
'get the measurement results
Set measurementData = measurementControl.GetData

Set params = measurementData.UserParams

TRACE "Save measurement results ..."

'save measurement results - readable by PAT
measurementData.Save "APIDemo2.omr"

TRACE "   " + "... PAT readable - APIDemo2.omr"

'save measurement results - text export
measurementData.Export "APIDemo2Curves.txt"

TRACE "   " + "... Text export  - APIDemo2Curves.txt"

'save measurement results - Jones
measurementData.ExportJonesMatrix "APIDemo2Jones.txt"
```

```
TRACE "   " + "... Jones       - APIDemo2Jones.txt"

'save measurement results - Mueller
measurementData.ExportFirstRowMuellerMatrix "APIDemo2Mueller.txt"

TRACE "   " + "... Mueller      - APIDemo2Mueller.txt"

TRACE "Release resources"

'release the instruments, optional
measurementControl.ReleaseResources

TRACE "-------------------- Measurement done --------------------"

'clean up objects
Set measurementData = Nothing
Set params = Nothing
Set measurementControl = Nothing
Set Crv = Nothing

DoMeasurement.Enabled = True
GoTo Ready

ErrorHandling:
  TRACE_ERROR Err.Description
  DoMeasurement.Enabled = True
  GoTo Ready
Abort:
  TRACE "### " + "Measurement aborted by user"
  DoMeasurement.Enabled = True

Ready:
End Sub

Private Sub Form_Load()
'initialize Pfl
PflManager.Initialize "", True
Tracer.ForeColor = &H80000008
TRACE "PFL Initialized"
End Sub

Private Sub Form_Terminate()
'deinitialize Pfl
PFL.Close
TRACE "PFL Deinitialized"
End Sub

Private Sub TRACE(s As String)
```

```
Tracer.AddItem s
Tracer.ListIndex = Tracer.ListCount - 1
End Sub

Private Sub TRACE_ERROR(s As String)
Tracer.ForeColor = &HFF&
TRACE "################################################"
TRACE "###" + s
End Sub
```

Visual Basic: Example 3

This program is provided on the Agilent 81910A Photonic All-Parameter Analyzer CD-ROM in
\\Example Programs\COM API Examples\VB6 Examples\Example3

This program demonstrates how to import the raw data for a measurement, then separate it into its curve components.

```
VERSION x.xx

Begin VB.Form CtrlPannel
  Caption = "API Demo"
  ClientHeight = 6720
  ClientLeft = 60
  ClientTop = 345
  ClientWidth = 8385
  LinkTopic = "Form1"
  ScaleHeight = 6720
  ScaleWidth = 8385
  StartUpPosition = 3 'Windows Default
  Begin VB.ListBox Tracer
    BeginProperty Font
      Name = "Courier New"
      Size = 8.25
      Charset = 0
      Weight = 400
      Underline = 0   'False
      Italic = 0 'False
      Strikethrough = 0   'False
    EndProperty
    Height = 5940
    Left = 120
    TabIndex = 1
    Top = 120
    Width = 8175
  End
  Begin VB.CommandButton DoMeasurement
    Caption = "Run Demo"
    Height = 375
    Left = 120
    TabIndex = 0
    Top = 6240
    Width = 8175
  End
End
```

```
Attribute VB_Name = "CtrlPannel"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Private Sub DoMeasurement_Click()
On Error GoTo ErrorHandling
DoMeasurement.Enabled = False
Tracer.Clear

Dim nIdx As Integer

'instantiate a measurement data object
Dim measurementData_File As New ApPactMeasurementData

TRACE "Load file setup.omr"

'load setup from file
measurementData_File.Load "Setup.omr"

Dim PS As PolStateEnum
Dim CR As CurveRoleEnum
Dim LP As New LightPath
Dim c As Curve

'Set light path for transmission 1st port
LP.SourceDUTPort.Group = DUTPortGroupIn
LP.SourceDUTPort.Number = 1
LP.DestinationDUTPort.Group = DUTPortGroupOut
LP.DestinationDUTPort.Number = 1

Dim CurveCount As Integer
CurveCount = 0

TRACE "-------------------------------------------------------"
TRACE "Search data for curves in transmission"
TRACE "-------------------------------------------------------"

For CR = CurveRoleDUTPwrWvl To CurveRoleRefTermPwrWvl
  For PS = PolStateUnpolarized To PolStateUndefined
    Set c = measurementData_File.GetSingleCurve(LP, CR, PS)
    If Not c Is Nothing Then
      CurveCount = CurveCount + 1
      TRACE "Found Curve..."
      TRACE "      Role : " + CurveRole2String(CR)
      TRACE "  Polstate : " + Polstate2String(PS)
      TRACE "      Unit : " + Unit2String(c.Unit)
      TRACE "     Count : " + Str(c.Size)
    End If
  Next
Next

'Set light path for reflection 1st port
LP.SourceDUTPort.Group = DUTPortGroupIn
LP.SourceDUTPort.Number = 1
LP.DestinationDUTPort.Group = DUTPortGroupIn
LP.DestinationDUTPort.Number = 1

TRACE "--------------------------------------------------------"
TRACE "Search data for curves in reflection"
TRACE "--------------------------------------------------------"

For CR = CurveRoleDUTPwrWvl To CurveRoleRefTermPwrWvl
  For PS = PolStateUnpolarized To PolStateUndefined
    Set c = measurementData_File.GetSingleCurve(LP, CR, PS)
    If Not c Is Nothing Then
      CurveCount = CurveCount + 1
      TRACE "Found Curve..."
      TRACE "     Role : " + CurveRole2String(CR)
      TRACE "  Polstate : " + Polstate2String(PS)
      TRACE "     Unit : " + Unit2String(c.Unit)
      TRACE "    Count : " + Str(c.Size)
    End If
  Next
Next

If CurveCount = 0 Then
  TRACE "No Curves found"
Else
  Dim sTrace As String

  sTrace = Str(CurveCount)
  TRACE "Found" + sTrace + " curves"
End If

'clean up objects
Set measurementData = Nothing

DoMeasurement.Enabled = True
GoTo Ready

ErrorHandling:
  TRACE_ERROR Err.Description
  DoMeasurement.Enabled = True
  GoTo Ready
Abort:
  TRACE "### " + "Measurement aborted by user"
  DoMeasurement.Enabled = True

Ready:
End Sub

Private Sub Form_Load()
'initialize Pfl
PflManager.Initialize "", False
Tracer.ForeColor = &H80000008
TRACE "PFL Initialized"
End Sub

Private Sub Form_Terminate()
'deinitialize Pfl
PFL.Close
TRACE "PFL Deinitialized"
End Sub

Private Sub TRACE(s As String)
Tracer.AddItem s
Tracer.ListIndex = Tracer.ListCount - 1
End Sub

Private Sub TRACE_ERROR(s As String)
Tracer.ForeColor = &HFF&
TRACE "##################################################"
TRACE "###" + s
End Sub

Private Function Polstate2String(p As PolStateEnum) As String
  Select Case p
    Case PolStateUnpolarized
      Polstate2String = "unpolarized"
    Case PolStateLH0
      Polstate2String = "LH0"
    Case PolStateLDP45
      Polstate2String = "LDP45"
    Case PolStateLDN45
      Polstate2String = "LDN45"
    Case PolStateLV90
      Polstate2String = "LV90"
    Case PolStateLHC
      Polstate2String = "left hand circular"
    Case PolStateRHC
      Polstate2String = "right hand circular"
    Case PolStateUndefined
      Polstate2String = "undefined"
  End Select
End Function

Private Function CurveRole2String(c As CurveRoleEnum) As String
  Select Case c
    Case CurveRoleDUTPwrWvl
      CurveRole2String = "DUT - Power over wavelength"
    Case CurveRoleRefPwrWvl
      CurveRole2String = "Reference - Power over wavelength"
    Case CurveRoleLossWvl
      CurveRole2String = "DUT - Loss over wavelength"
    Case CurveRolePDLWvl
      CurveRole2String = "DUT - Polarization dependent loss over wavelength"
    Case CurveRoleLossMinWvl
      CurveRole2String = "DUT - Minimum loss over wavelength"
    Case CurveRoleLossMaxWvl
      CurveRole2String = "DUT - Maximum loss over wavelength"
    Case CurveRoleLossAvgWvl
      CurveRole2String = "DUT - Average loss over wavelength"
    Case CurveRoleRefPhaseWvl
      CurveRole2String = "Reference - Phase over wavelength"
    Case CurveRoleGDWvl
      CurveRole2String = "DUT - Group delay over wavelength"
    Case CurveRoleDGDWvl
      CurveRole2String = "DUT - Differential group delay over wavelength"
    Case CurveRoleCDWvl
      CurveRole2String = "DUT - Cromatic dispersion over wavelength"
    Case CurveRoleRefPwrWvlRT
      CurveRole2String = "Reference - Real time power over wavelength"
    Case CurveRoleRefTermPwrWvl
      CurveRole2String = "Reference - Termination power over wavelength"
  End Select
End Function

Private Function Unit2String(u As CurveUnitEnum) As String
  Select Case u
    Case CurveUnitdB
      Unit2String = "[dB]"
    Case CurveUnitW
      Unit2String = "[W]"
    Case CurveUnitm
      Unit2String = "[m]"
    Case CurveUnits
      Unit2String = "[s]"
    Case CurveUnitspm
      Unit2String = "[m/s]"
  End Select
End Function
```

Visual C++

This section includes:

- Visual C++: Headers, which describes the standard program headers required.
- Visual C++: LoadFile, which demonstrates how to load an existing file and retrieve measurement parameters and data.
- Visual C++: RepeatMeasurement, which demonstrates how to load an existing file, use its measurement parameters and reference data, execute a measurement, retrieve the results and save them into a file.
- Visual C++: SetupMeasurement, This program demonstrates how to create a measurement setup, perform an all-parameter measurement and save the results into a file.

Visual C++: Headers

These headers are part of each Visual C++ program provided on the Agilent 81910A Photonic All-Parameter Analyzer CD-ROM in **\\Example Programs\COM API Examples\VC++ Examples\...**

Standard include files: stafx.ccp

```
// stdafx.cpp : source file that includes just the standard includes
// LoadFile.pch is the pre-compiled header
// stdafx.obj contains the pre-compiled type information

#include "stdafx.h"
```

Standard include files: stafx.h

```
// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently but
// changed infrequently
//

#pragma once

#define WIN32_LEAN_AND_MEAN// Exclude rarely-used stuff from Windows headers
#include <stdio.h>
#include <tchar.h>

// TODO: reference additional headers your program requires here
```

Visual C++: LoadFile

This program is provided on the Agilent 81910A Photonic All-Parameter Analyzer CD-ROM in
\\Example Programs\COM API Examples\VC++ Examples\LoadFile

```
/*-----------------------------------------------------------
Module: LoadFile.cpp
Copyright: Agilent Technologies Deutschland GmbH
Purpose: Demonstrate how to load an existing file and
retrieve measurement parameters and data
Date: Mar. 24th, 2003
-----------------------------------------------------------*/

#include <stdafx.h>
#import "pflapi.dll" no_namespace

#include<iostream>
#include<string>
#include<atlbase.h>
#include<atlsafe.h>
#include<comdef.h>
#include<objbase.h>

void main(int argc, char** argv)
{
  CoInitializeEx(NULL,COINIT_MULTITHREADED);
  IPflManagerPtr pPflManager;

  IApPactMeasurementDataPtr pMeasData;

  char filename[256];

  if(argc<2) strcpy(filename, "test.omr");
  else strcpy(filename, argv[1]);

  // create new COM objects
  HRESULT lRes = pPflManager.CreateInstance("PFL.PflManager");
  lRes = pMeasData.CreateInstance("PFL.ApPactMeasurementData");
  try
  {
    // Test: IPflManager::Initialize
    printf("initializing PFL ...\n");
    pPflManager->Initialize("", VARIANT_FALSE);

    printf("File: %s\n", filename);
```

```
 pMeasData->Load(filename);

 // read parameters.
 IApPactUserParamsPtr pUserParam("Pfl.ApPactUserParams");

 pUserParam = pMeasData->UserParams;

 // start / stop / step ...
 printf("Measurement Parameters:\n");
 printf("Start: %fnm, Stop: %fnm, Step: %fnm\n",
   pUserParam->PhaseParams->StartWavelength*1e9,
   pUserParam->PhaseParams->StopWavelength*1e9,
   pUserParam->PhaseParams->StepWavelength*1e9);

 // determine meas. parameter & directions ...
 bool amplRX  = (pUserParam->LossParams->MeasureReflection == VARIANT_TRUE);
 bool amplTX  = (pUserParam->LossParams->MeasureTransmission ==
                                                               VARIANT_TRUE);
 bool phaseRX = (pUserParam->PhaseParams->MeasureReflection ==
                                                               VARIANT_TRUE);
 bool phaseTX = (pUserParam->PhaseParams->MeasureTransmission ==
                                                               VARIANT_TRUE);
 bool Pdl = pUserParam->LossParams->MeasurementType ==
                                           ApPactLossMeasurementTypePDL;
 bool Loss = (pUserParam->LossParams->MeasurementType ==
                                     ApPactLossMeasurementTypeLoss) || Pdl;
 bool DGD = pUserParam->PhaseParams->MeasurementType ==
                                          ApPactPhaseMeasurementTypeDGD;
 bool GD = (pUserParam->PhaseParams->MeasurementType ==
                                   ApPactPhaseMeasurementTypeGD) || DGD;

 printf("LossRX %c LossTX %c, PdlRX %c PdlTX %c, GDRX %c GDTX %c, DGDRX %c
DGDTX %c\n",
    (amplRX && Loss)? 'Y' : 'N', (amplTX && Loss)? 'Y' : 'N',
    (amplRX && Pdl)? 'Y' : 'N', (amplTX && Pdl)? 'Y' : 'N',
    (phaseRX && GD)? 'Y' : 'N', (phaseTX && GD)? 'Y' : 'N',
    (phaseRX && DGD)? 'Y' : 'N', (phaseTX && DGD)? 'Y' : 'N');

 // get curves (only channel 1!) ...
 ICurvePtr pCurve;
 pCurve.CreateInstance("Pfl.Curve");

 // declare data array
 CComSafeArray<DOUBLE> data;

 pCurve = pMeasData->GetDUTLossCurve(DirectionTX, 1);  // Loss TX
 if(pCurve)
 {
  printf("Loss TX, data points: %ld\n", pCurve->Size);
  double min=100.0, max=0.0;
  // copy data array from PFL
  data.CopyFrom(pCurve->DataArray);
  // find min/max
  for(int i=0; i<pCurve->Size; i++)
  {
    if(min > data[i]) min = data[i];
    if(max < data[i]) max = data[i];
  }
  printf("--> Min: %.2lf dB, Max: %.2lf dB\n", min, max);
}
pCurve = pMeasData->GetDUTLossCurve(DirectionRX, 1); // Loss RX
if(pCurve)
  printf("Loss RX, data points: %ld\n", pCurve->Size);

pCurve = pMeasData->GetDUTPDLCurve(DirectionTX, 1); // PDL TX
if(pCurve)
  printf("PDL TX, data points: %ld\n", pCurve->Size);
pCurve = pMeasData->GetDUTPDLCurve(DirectionRX, 1); // PDL RX
if(pCurve)
  printf("PDL RX, data points: %ld\n", pCurve->Size);

pCurve = pMeasData->GetDUTGDCurve(DirectionTX, 1);  // GD TX
if(pCurve)
  printf("GD TX, data points: %ld\n", pCurve->Size);
pCurve = pMeasData->GetDUTGDCurve(DirectionRX, 1);  // GD RX
if(pCurve)
  printf("GD RX, data points: %ld\n", pCurve->Size);

pCurve = pMeasData->GetDUTDGDCurve(DirectionTX, 1); // DGD TX
if(pCurve)
  printf("DGD TX, data points: %ld\n", pCurve->Size);
pCurve = pMeasData->GetDUTDGDCurve(DirectionRX, 1); // DGD RX
if(pCurve)
  printf("DGD RX, data points: %ld\n", pCurve->Size);

pCurve = pMeasData->GetDUTCDCurve(DirectionTX, 1);  // CD TX
if(pCurve)
  printf("CD TX, data points: %ld\n", pCurve->Size);
pCurve = pMeasData->GetDUTCDCurve(DirectionRX, 1);  // CD RX
if(pCurve)
  printf("CD RX, data points: %ld\n", pCurve->Size);

// wait for user input to terminate ...
printf("Hit <return> to terminate!\n");
char ret;
scanf("%c", &ret);
```

```
    pUserParam = NULL;
    pPflManager->Close();
    CoUninitialize();
  }
  catch(_com_error& e)
  {
    BSTR errStr;
    HRESULT lHres = e.Error();
    printf("Caught COM error, code 0x%8x\n", lHres);
    IErrorInfo * lErrorInfo = e.ErrorInfo();
    if (NULL != lErrorInfo)
    {
      lErrorInfo->GetDescription(&errStr);
      printf("Error info: %S\n", errStr);
    }
    else
    {
      printf("No error info available; %s\n", e.ErrorMessage());
    }
  }
  pPflManager = NULL;
}
```

Visual C++: RepeatMeasurement

This program is provided on the Agilent 81910A Photonic All-Parameter Analyzer CD-ROM in
\\Example Programs\COM API Examples\VC++ Examples\Repeat Measurement

```
/*-----------------------------------------------------------
Module: RepeatMeasurement.cpp
Copyright: Agilent Technologies Deutschland GmbH
Purpose: Demonstrate how to load an existing file, use
its measurement parameters and reference data,
execute a measurement, retrieve the results and
save them into a file
Date: Mar. 24th, 2003
-----------------------------------------------------------*/

#include <stdafx.h>
#import "pflapi.dll" no_namespace

#include<iostream>
#include<string>
#include<atlbase.h>
#include<atlsafe.h>
#include<comdef.h>
#include<objbase.h>

void main(int argc, char** argv)
{
  CoInitializeEx(NULL,COINIT_MULTITHREADED);
  IPflManagerPtr pPflManager;
  IApPactMeasurementControlPtr pMeasControl("PFL.ApPactMeasurementControl");

  IApPactMeasurementDataPtr pMeasData;

  char filename[256];

  if(argc<2) strcpy(filename, "1nmrxtxallparams.omr");
  else strcpy(filename, argv[1]);

  // create new COM objects
  HRESULT lRes = pPflManager.CreateInstance("PFL.PflManager");
  lRes = pMeasData.CreateInstance("PFL.ApPactMeasurementData");
  try
  {
    // Test: IPflManager::Initialize
    printf("initializing PFL ...\n");
    pPflManager->Initialize("", VARIANT_TRUE);

    printf("File: %s\n", filename);
    pMeasData->Load(filename);

    // get available resources
    printf("configuring resources ...\n");
    IResourcesPtr IResources = pMeasControl->ShowAvailableResources();
    pMeasControl->ConfigureResources(IResources);

    // read parameters.
    IApPactUserParamsPtr pUserParam("Pfl.ApPactUserParams");

    pUserParam = pMeasData->UserParams;

    // start / stop / step ...
    printf("Measurement Parameters:\n");
    printf("Start: %fnm, Stop: %fnm, Step: %fnm\n",
      pUserParam->PhaseParams->StartWavelength*1e9,
      pUserParam->PhaseParams->StopWavelength*1e9,
      pUserParam->PhaseParams->StepWavelength*1e9);

    // determine meas. parameter & directions ...
    bool amplRX  = (pUserParam->LossParams->MeasureReflection == VARIANT_TRUE);
    bool amplTX  = (pUserParam->LossParams->MeasureTransmission ==
    VARIANT_TRUE);
    bool phaseRX = (pUserParam->PhaseParams->MeasureReflection ==
    VARIANT_TRUE);
    bool phaseTX = (pUserParam->PhaseParams->MeasureTransmission ==
    VARIANT_TRUE);
    bool Pdl = pUserParam->LossParams->MeasurementType ==
    ApPactLossMeasurementTypePDL;
    bool Loss = (pUserParam->LossParams->MeasurementType ==
    ApPactLossMeasurementTypeLoss) || Pdl;
    bool DGD = pUserParam->PhaseParams->MeasurementType ==
    ApPactPhaseMeasurementTypeDGD;
    bool GD = (pUserParam->PhaseParams->MeasurementType ==
    ApPactPhaseMeasurementTypeGD) || DGD;

    printf("LossRX %c LossTX %c, PdlRX %c PdlTX %c, GDRX %c GDTX %c, DGDRX %c
  DGDTX %c\n",
      (amplRX && Loss)? 'Y' : 'N', (amplTX && Loss)? 'Y' : 'N',
      (amplRX && Pdl)? 'Y' : 'N', (amplTX && Pdl)? 'Y' : 'N',
      (phaseRX && GD)? 'Y' : 'N', (phaseTX && GD)? 'Y' : 'N',
      (phaseRX && DGD)? 'Y' : 'N', (phaseTX && DGD)? 'Y' : 'N');

    // initialize measurement ...
    printf("setting parameters & references ...\n");
    pMeasControl->UseParameters(pMeasData);
```

```
pMeasControl->AcquireResources();

pMeasControl->UseReference(pMeasData);

// execute measurement ...
printf("starting measurement ...\n");
pMeasControl->ExecuteMeasurement(1, 0); // no timeout!

printf("measurement done, retrieving data ...\n");
pMeasData = pMeasControl->GetData();

// get curves (only channel 1!) ...
ICurvePtr pCurve;
pCurve.CreateInstance("Pfl.Curve");

// declare data array
CComSafeArray<DOUBLE> data;

pCurve = pMeasData->GetDUTLossCurve(DirectionTX, 1);  // Loss TX
if(pCurve)
{
  printf("Loss TX, data points: %ld\n", pCurve->Size);
  double min=100.0, max=0.0;
  // copy data array from PFL
  data.CopyFrom(pCurve->DataArray);
  // find min/max
  for(int i=0; i<pCurve->Size; i++)
  {
    if(min > data[i]) min = data[i];
    if(max < data[i]) max = data[i];
  }
  printf("--> Min: %.2lf dB, Max: %.2lf dB\n", min, max);
}

pCurve = pMeasData->GetDUTGDCurve(DirectionTX, 1);  // GD TX
if(pCurve)
{
  printf("GD TX, data points: %ld\n", pCurve->Size);
  double min=100000.0, max=0.0;
  // copy data array from PFL
  data.CopyFrom(pCurve->DataArray);
  // find min/max
  for(int i=0; i<pCurve->Size; i++)
  {
    if(min > data[i]) min = data[i];
    if(max < data[i]) max = data[i];
  }
```

```
      printf("--> Min: %.4lf ps, Max: %.4lf ps\n", min*1e12, max*1e12);
    }

    // save the data ...
    printf("saving data to output.omr!\n");
    pMeasData->Save("output.omr");

    // wait for user input to terminate ...
    printf("Hit <return> to terminate!\n");
    char ret;
    scanf("%c", &ret);
    pUserParam = NULL;
    pPflManager->Close();
    CoUninitialize();
  }
  catch(_com_error& e)
  {
    BSTR errStr;
    HRESULT lHres = e.Error();
    printf("Caught COM error, code 0x%8x\n", lHres);
    IErrorInfo * lErrorInfo = e.ErrorInfo();
    if (NULL != lErrorInfo)
    {
      lErrorInfo->GetDescription(&errStr);
      printf("Error info: %S\n", errStr);
    }
    else
    {
      printf("No error info available; %s\n", e.ErrorMessage());

    }
  }
  pPflManager = NULL;
}
```

Visual C++: SetupMeasurement

This program is provided on the Agilent 81910A Photonic All-Parameter Analyzer CD-ROM in
\\Example Programs\COM API Examples\VC++ Examples\Setup Measurement

```
/*-----------------------------------------------------------
Module: SetupMeasurement.cpp
Copyright: Agilent Technologies Deutschland GmbH
Purpose: Demonstrate how to create a measurement setup,
perform an all-parameter measurement and save the
results into a file
Date: Mar. 24th, 2003
-----------------------------------------------------------*/

#include <stdafx.h>
#import "pflapi.dll" no_namespace

#include<iostream>
#include<string>
#include<atlbase.h>
#include<atlsafe.h>
#include<comdef.h>
#include<objbase.h>

void main(int argc, char** argv)
{
  CoInitializeEx(NULL,COINIT_MULTITHREADED);
  IPflManagerPtr pPflManager;
  IApPactMeasurementControlPtr pMeasControl("PFL.ApPactMeasurementControl");

  IApPactMeasurementDataPtr pMeasData;

  char filename[256];

  if(argc<2) strcpy(filename, "output.omr");
  else strcpy(filename, argv[1]);

  // create new COM objects
  HRESULT lRes = pPflManager.CreateInstance("PFL.PflManager");
  lRes = pMeasData.CreateInstance("PFL.ApPactMeasurementData");
  try
  {
    // Test: IPflManager::Initialize
    printf("initializing PFL ...\n");
    pPflManager->Initialize("", VARIANT_TRUE);
```

```
// get available resources
printf("configuring resources ...\n");
IResourcesPtr IResources = pMeasControl->ShowAvailableResources();
pMeasControl->ConfigureResources(IResources);

// configure parameters.
printf("configuring parameters ...\n");
IApPactUserParamsPtr pUserParam("Pfl.ApPactUserParams");

pUserParam->LossParams->StartWavelength = 1550e-9;
pUserParam->LossParams->StopWavelength = 1560e-9;
pUserParam->LossParams->StepWavelength = 1e-12;
pUserParam->LossParams->SweepCount = SweepCountTwoSweeps;
pUserParam->PhaseParams->StartWavelength = 1550e-9;
pUserParam->PhaseParams->StopWavelength = 1560e-9;
pUserParam->PhaseParams->StepWavelength = 1e-12;
pUserParam->PhaseParams->AveragingCount = 10;
pUserParam->PhaseParams->AveragingWindowWidth = 10;
pUserParam->PhaseParams->CDAveragingWindowWidth = 50;
pUserParam->PhaseParams->MeasureTransmission = VARIANT_TRUE;
pUserParam->LossParams->MeasureTransmission = VARIANT_TRUE;
pUserParam->PhaseParams->MeasureReflection = VARIANT_FALSE;
pUserParam->LossParams->MeasureReflection = VARIANT_FALSE;
pUserParam->PhaseParams->MeasurementType =
ApPactPhaseMeasurementTypeDGD;
pUserParam->LossParams->MeasurementType = ApPactLossMeasurementTypePDL;
pUserParam->PhaseParams->SupplyJonesMatrix = VARIANT_TRUE;

// initialize measurement ...
ApPactParamCheckHintEnum lossHint;
ApPactParamCheckHintEnum phaseHint;
PflStatusEnum status;
printf("setting parameters ...\n");
// the following ...GetInterface indirection is required
// because ConfigureParameters needs the real interface, not
// the IApPactUserParamsPtr SmartPointer
pMeasControl->ConfigureParameters(&(pUserParam.GetInterfacePtr()),
  &lossHint, &phaseHint, &status);

pMeasControl->AcquireResources();

// take reference measurements ...
printf("Performing intialization reference, disconnect all devices and\n");
printf("hit <return> to continue ...\n");
char ret;
scanf("%c", &ret);
pMeasControl->ExecuteReference(ApPactReferenceTypeInitialization, 0);
```

```
printf("Performing transmission reference, connect reference patchcord and\n");
printf("hit <return> to continue ...\n");
scanf("%c", &ret);
pMeasControl->ExecuteReference(ApPactReferenceTypeTransmission, 0);

// execute measurement ...
printf("starting measurement ...\n");
pMeasControl->ExecuteMeasurement(1, 0); // no timeout!

printf("measurement done, retrieving data ...\n");
pMeasData = pMeasControl->GetData();

// get curves (only channel 1!) ...
ICurvePtr pCurve;
pCurve.CreateInstance("Pfl.Curve");

// declare data array
CComSafeArray<DOUBLE> data;

pCurve = pMeasData->GetDUTLossCurve(DirectionTX, 1);  // Loss TX
if(pCurve)
{
  printf("Loss TX, data points: %ld\n", pCurve->Size);
  double min=100.0, max=0.0;
  // copy data array from PFL
  data.CopyFrom(pCurve->DataArray);
  // find min/max
  for(int i=0; i<pCurve->Size; i++)
  {
    if(min > data[i]) min = data[i];
    if(max < data[i]) max = data[i];
  }
  printf("--> Min: %.2lf dB, Max: %.2lf dB\n", min, max);
}

pCurve = pMeasData->GetDUTGDCurve(DirectionTX, 1);  // GD TX
if(pCurve)
{
  printf("GD TX, data points: %ld\n", pCurve->Size);
  double min=100000.0, max=0.0;
  // copy data array from PFL
  data.CopyFrom(pCurve->DataArray);
  // find min/max
  for(int i=0; i<pCurve->Size; i++)
  {
    if(min > data[i]) min = data[i];
    if(max < data[i]) max = data[i];
```

```
      }
      printf("--> Min: %.4lf ps, Max: %.4lf ps\n", min*1e12, max*1e12);
    }

    // save the data ...
    printf("saving data to %s!\n", filename);
    pMeasData->Save(filename);

    // wait for user input to terminate ...
    printf("Hit <return> to terminate!\n");
    scanf("%c", &ret);
    pUserParam = NULL;
    pPflManager->Close();
    CoUninitialize();
  }
  catch(_com_error& e)
  {
    BSTR errStr;
    HRESULT lHres = e.Error();
    printf("Caught COM error, code 0x%8x\n", lHres);
    IErrorInfo * lErrorInfo = e.ErrorInfo();
    if (NULL != lErrorInfo)
    {
      lErrorInfo->GetDescription(&errStr);
      printf("Error info: %S\n", errStr);
    }
    else
    {
      printf("No error info available; %s\n", e.ErrorMessage());
    }
  }
  pPflManager = NULL;
}
```

Lab View 6

This section includes:

- LabView 6: Example 1 (Simple measurement).
- LabView 6: Example 2 (Display curve).
- LabView 6: Example 3 (Set measurement parameters).

LabView 6: Example 1 (Simple measurement)

This program is provided on the Agilent 81910A Photonic All-Parameter Analyzer CD-ROM in
\\Example Programs\COM API Examples\Lab_View_Examples\Example 1 (Simple measurement)

LabView 6: Example 2 (Display curve)

This program is provided on the Agilent 81910A Photonic All-Parameter Analyzer CD-ROM in
\\Example Programs\COM API Examples\Lab_View_Examples\Example 2 (Display curve)

LabView 6: Example 3 (Set measurement parameters)

This program is provided on the Agilent 81910A Photonic All-Parameter Analyzer CD-ROM in
\\Example Programs\COM API Examples\Lab_View_Examples\Example 3 (Set measurement parameters)

Notes

[This page provided for your own notes]

3
Independent Interfaces

This chapter describes all the Agilent PFL COM interfaces that are independent of the measurement interfaces.

Independent Interfaces - Summary

This section provides a tabulated summary of the interfaces (collections of properties, methods and events) available.

Table 2 Independent Interface Summary

Interface	Description
"IPflManager Interface" on page 85	Contains PFL housekeeping functions.
"IResourceManager Interface" on page 87	Features services associated with the measurement instruments, named resources.
"IResourceDetails Interface" on page 94	Used to get detailed information about a resource.
"IResources Interface" on page 97	Represents a collection of resources.
"IPowerRanges Interface" on page 100	Represents a collection of power ranges.
"IPolStates Interface" on page 103	Represents a collection of polarization states.
"IBuses Interface" on page 106	Represents a collection of buses.
"IValues Interface" on page 109	Represents a collection of curve values.
"ILightPath Interface" on page 112	Represents a light path.
"IDUTPort Interface" on page 114	Represents a port of the device under test (DUT).
"IMeasurementControl Interface" on page 116	Features operations common to all measurement control classes. This is the base interface for other interfaces that are specific to a particular type of measurement.
"ICurve Interface" on page 121	Represents a collection of points (x, y). The x values are described by a start/step pair which means that they are equidistant. The y values are stored in an array.
"ICurves Interface" on page 127	Represents a collection of curves.
"IMeasurementData Interface" on page 130	Features access functions to the database that stores all relevant data for a measurement in one place. This is the base interface for other interfaces that are specific to a particular type of measurement.

Independent Interfaces and Objects - Detail

The following sections list the properties, methods and events associated with each interface collection.

A short description of each property and method is provided, together with its syntax, parameters and return values.

IPflManager Interface

This interface contains PFL housekeeping functions.

Properties: None.

Methods:

IPflManager::Initialize Method	This method initializes the current instance of PFL.
IPflManager::Close Method	This method closes the current instance of PFL.

Events: None.

IPflManager::Initialize Method

This method initializes the current instance of PflManager.
HRESULT Initialize(BSTR GPIBBuses,VARIANT_BOOL busScan);

Parameters: *GPIBBuses* [in]

A comma separated list that specifies the GPIB buses to be scanned. If an empty string is supplied, all buses are scanned.

busScan [in]

Specifies if a bus scan will be made or not.

Return value: *S_OK*

The value returned if successful.

E_PFL_INIT_FAILED

The value returned if the initialization failed.

E_PFL_CMD_DISALLOWED

PFL is already initialized.

See also: "IPflManager::Close Method" on page 86

"IPflManager Interface" on page 85.

IPflManager::Close Method

This method closes the current instance of PflManager.
HRESULT Close(void);

Parameters: None.

Return value: *S_OK*

The value returned if successful.

E_PFL_CMD_ORDER_INVALID

Operation was called before the IPflManager::Initialize Method

See also: "IPflManager::Initialize Method" on page 86

"IPflManager Interface" on page 85.

IResourceManager Interface

Features services around the instruments, called resources here, which are used for the measurements.

Properties: None.

Methods:

IResourceManager::ScanBuses Method	This method scans the buses passed as an argument.
IResourceManager::ShowAllResources Method	This method returns all resources detected during the most recent scan.
IResourceManager::GetResourceDetails Method	Returns all details of the specified resource.
IResourceManager::GetParentResourceSpec Method	Used to query for the parent of a resource. Returns a resource specification.
IResourceManager::GetChildResourceSpecs Method	Used to query for the child resources. Returns a list of resource specifications.
IResourceManager::AcquireResourceHandle Method	Returns a VISA session handle of the VXIplug&play driver used by the PFL.
IResourceManager::ReleaseResourceHandle Method	Releases a previously acquired VISA session handle.

Events: None.

IResourceManager::ScanBuses Method

This method scans the buses passed as an argument.

HRESULT ScanBuses(IBuses buses);**

Parameters:

buses [in]

The buses to be scanned. If *buses* is empty, all known buses will be scanned.

Return value:

S_OK

The value returned if successful.

E_POINTER

The value returned if *ppBuses* is Null.

E_PFL_CMD_DISALLOWED

Currently at most one MeasurementControl object may exist at a time. Rescans are not allowed with currently active measurements. Please terminate all measurements before rescanning the interfaces.

See also: "IResourceManager Interface" on page 87

"IBuses Interface" on page 106.

IResourceManager::ShowAllResources Method

This method returns all resources detected during the most recent scan.

HRESULT ShowAllResources(IResources resources);**

Parameters:

resources [out, retval]

The resource list.

Return value:

S_OK

The value returned if successful.

E_POINTER

The value returned if *resources* is Null.

E_PFL_CMD_DISALLOWED

The operation was called before the IPflManager::Initialize Method.

See also: "IResourceManager Interface" on page 87

"IResources Interface" on page 97.

IResourceManager::GetResourceDetails Method

Returns all details of the specified resource. Details contains module types, firmware revision information etc.

```
HRESULT GetResourceDetails(
    BSTR resourceSpec,
    IResourceDetails** details
);
```

Parameters: *resourceSpec* [in]

The resource. For example, GPIB::20::INSTR.

details [out]

The resource details.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *details* is Null.

E_PFL_RSRC_NOT_EXISTING

The module specified in the resource specification does not exist. Please verify that the resource specification is correct, and that the module is correctly installed and in working order.

See also: "IResourceDetails Interface" on page 94

"IResourceManager::GetParentResourceSpec Method" on page 90

"IResourceManager::GetChildResourceSpecs Method" on page 91

"IMeasurementData::GetResourceDetails Method" on page 136

"IResourceManager Interface" on page 87.

IResourceManager::GetParentResourceSpec Method

Used to query for the parent of a resource. Returns a resource specification.

```
HRESULT GetParentResourceSpec(
    BSTR resourceSpec,
    BSTR* parent
);
```

Parameters:

resourceSpec [in]

The specification string of the resource for which the parent is to be queried.

parent [out, retval]

The specification string of the parent resource.

Return value:

S_OK

The value returned if successful.

E_POINTER

The value returned if *parent* is Null.

E_PFL_RSRC_NOT_EXISTING

The resource specified is not used in the measurement.

See also:

"IResourceManager::GetResourceDetails Method" on page 89

"IResourceManager::GetChildResourceSpecs Method" on page 91

"IResourceManager::GetParentResourceSpec Method" on page 90

"IResourceManager Interface" on page 87.

IResourceManager::GetChildResourceSpecs Method

Used to query for the child of a resources. Returns a list of resource specifications.

```
HRESULT GetChildResourceSpecs(
    BSTR resourceSpec,
    IResources** children
);
```

Parameters: *resourceSpec* [in]

The specification string of the resource for which child resources are to be queried. For example, GPIB::20::INSTR.

children [out, retval]

The specification strings of the child resources.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *children* is Null.

E_PFL_RSRC_NOT_EXISTING

The resource specified is not used in the measurement.

See also: "IResourceManager::GetResourceDetails Method" on page 89

"IResourceManager::GetParentResourceSpec Method" on page 90

"IResourceManager::GetChildResourceSpecs Method" on page 91

"IResourceManager Interface" on page 87

"IResources Interface" on page 97.

IResourceManager::AcquireResourceHandle Method

Returns a VISA session handle of the VXIplug&play driver used by the PFL.

HRESULT AcquireResourceHandle(
BSTR resourceSpec,
ViSession* session
);

Parameters: *resourceSpec* [in]

The resource specification for which the session is requested.

session [out, retval]

The VISA session handle belonging to the resource.

Return value: *S_OK*

The value returned if successful.

E_PFL_RSRC_NOT_EXISTING

The instrument in the string does not exist.

E_PFL_LOCK_FAILED

Requested Instrument is already locked.

E_PFL_CMD_ORDER_INVALID

Init must be called before this function.

E_POINTER

The value returned if *session* is Null.

E_PFL_GENERAL_INSTR

The selected instrument has an error.

Remarks: Can be used to call VXIplug&play driver commands directly that are used by the PFL. Concurrency handling applies (that is, PFL uses internal locking). This method is only available if PFL is used as in-process server.

See also: "IResourceManager::ReleaseResourceHandle Method" on page 93

"IResourceManager Interface" on page 87.

IResourceManager::ReleaseResourceHandle Method

Releases a previously acquired VISA session handle.

HRESULT ReleaseResourceHandle(ViSession session);

Parameters: *session* [in]

The VISA session handle belonging to the resource.

Return value: *S_OK*

The value returned if successful.

E_PFL_PARAM_INVALID

The given handle is invalid.

E_PFL_CMD_ORDER_INVALID

Init must be called before this function

E_PFL_GENERAL_INSTR

The selected instrument has an error.

Remarks: This method is only available if PFL is used as in-process server.

See also: "IResourceManager::AcquireResourceHandle Method" on page 92

"IResourceManager Interface" on page 87

IResourceDetails Interface

Used to get detailed information about a resource.

Properties:

IResourceDetails::ModelNumber Property	Returns the model number of a resource. Read-only.
IResourceDetails::SerialNumber Property	Returns the serial number of a resource. Read-only.
IResourceDetails::SoftwareRevision Property	Returns the software revision of a resource. Read-only.
IResourceDetails::Manufacturer Property	Returns the manufacturer of a resource. Read-only.

Methods: None.

Events: None.

IResourceDetails::ModelNumber Property

Returns the model number of a resource.
HRESULT get_ModelNumber(BSTR* modelNumber);

Parameters: *modelNumber* [out, retval]

The model number of the resource.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *modelNumber* is Null.

Remarks: String. The property is read only.

See also: "IResourceDetails Interface" on page 94.

IResourceDetails::SerialNumber Property

Returns the serial number of a resource.
HRESULT get_SerialNumber(BSTR* serialNumber);

Parameters: *serialNumber* [out, retval]

The serial number of the resource.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *serialNumber* is Null.

Remarks: String. The property is read only.

See also: "IResourceDetails Interface" on page 94.

IResourceDetails::SoftwareRevision Property

Returns the software revision of a resource.
HRESULT get_SoftwareRevision(BSTR* softwareRevision);

Parameters: *softwareRevision* [out, retval]

The software revision of the resource.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *softwareRevision* is Null.

Remarks: String. The property is read only.

See also: "IResourceDetails Interface" on page 94.

IResourceDetails::Manufacturer Property

Returns the manufacturer of a resource.
HRESULT get_Manufacturer(BSTR* manufacturer);

Parameters: *manufacturer* [out, retval]

The manufacturer of the resource.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *manufacturer* is Null.

Remarks: String. The property is read only.

See also: "IResourceDetails Interface" on page 94.

IResources Interface

Represents a collection of resources.

Properties:

IResources::Count Property	Indicates the number of resources in the collection. Read-only.
IResources::Item Property	Allows random access to individual resources within the collection. Read-only.

Methods:

IResources::Add Method	Adds a resource to the collection.
IResources::RemoveAll Method	Removes all resources from the collection.

Events: None.

See also: "IMeasurementControl::ShowAvailableResources Method" on page 117

"IMeasurementControl::ConfigureResources Method" on page 118

"IMeasurementData::GetResourceSelection Method" on page 135

"IMeasurementData::GetChildResourceSpecs Method" on page 138

"IResourceManager::ShowAllResources Method" on page 88

"IResourceManager::GetChildResourceSpecs Method" on page 91

IResources::Count Property

Indicates the number of resources in the collection.

HRESULT get_Count(long* count);

Parameters: *count* [out, retval]

The number of resources in the collection.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *count* is Null.

Remarks: This is a read-only property.

See also: "IResources Interface" on page 97.

IResources::Item Property

Allows random access to individual resources within the collection.

```
HRESULT get_Item(
    long index,
    BSTR* resource
);
```

Parameters: *index* [in]

The index of the resource to be queried.

resource [out, retval]

The resource with the specified index (or Null if the index is out of range). For example, GPIB::20::INSTR

Return value: *S_OK*

The value returned if successful.

E_INVALIDARG

The value returned if the index is out of range.

E_POINTER

The value returned if *resource* is Null.

Remarks: This is a read-only property.

See also: "IResources Interface" on page 97.

IResources::Add Method

Adds a resource to the collection.

HRESULT Add(BSTR resource);

Parameters: *resource* [in]

The resource to add to the collection. For example, GPIB::20::INSTR.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *count* is Null.

See also: "IResources Interface" on page 97.

IResources::RemoveAll Method

Removes all resources from the collection.

HRESULT RemoveAll();

Parameters: None.

Return value: *S_OK*

The value returned if successful.

See also: "IResources Interface" on page 97.

IPowerRanges Interface

Represents a collection of power ranges.

Properties:

IPowerRanges::Count Property	Indicates the number of power ranges in the collection. Read-only.
IPowerRanges::Item Property	Allows random access to individual power ranges within the collection. Read-only.

Methods:

IPowerRanges::Add Method	Adds a power range to the collection.
IPowerRanges::RemoveAll Method	Removes all power ranges from the collection.

Events: None.

See also: "IApPactSystemLossParams::TransmissionRanges Property" on page 230

"IApPactSystemLossParams::ReflectionRanges Property" on page 231.

IPowerRanges::Count Property

Indicates the number of power ranges in the collection.

HRESULT get_Count(long* count);

Parameters: *count* [out, retval]

The number of power ranges in the collection.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *count* is Null.

Remarks: This is a read-only property.

See also: "IPowerRanges Interface" on page 100.

IPowerRanges::Item Property

Allows random access to individual power ranges within the collection.

```
HRESULT get_Item(
    long index,
    PowerRangeEnum* powerRange
);
```

Parameters: *index* [in]

The index of the power range to be queried.

powerRange [out, retval]

The power ranges with the specified index or Null if the index is out of range.

Return value: *S_OK*

The value returned if successful.

E_INVALIDARG

The value returned if the index is out of range.

E_POINTER

The value returned if *powerRange* is Null.

Remarks: This is a read-only property.

See also: "IPowerRanges Interface" on page 100

"PowerRangeEnum Enumeration" on page 147.

IPowerRanges::Add Method

Adds a power range to the collection.

HRESULT Add(PowerRangeEnum powerRange);

Parameters: *powerRange* [in]

The power range to add to the collection.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *count* is Null.

See also: "IPowerRanges Interface" on page 100

"PowerRangeEnum Enumeration" on page 147.

IPowerRanges::RemoveAll Method

Removes all power ranges from the collection.

HRESULT RemoveAll();

Return value: *S_OK*

The value returned if successful.

See also: "IPowerRanges Interface" on page 100.

IPolStates Interface

Represents a collection of polarization states.

Properties:

IPolStates::Count Property	Indicates the number of polarization states in the collection. Read-only.
IPolStates::Item Property	Allows random access to individual polarization states within the collection. Read-only.

Methods:

IPolStates::Add Method	Adds a polarization state to the collection.
IPowerRanges::RemoveAll Method	Removes all polarization states from the collection.

Events: None.

See also: "IApPactSystemLossParams::PolStates Property" on page 232.

IPolStates::Count Property

Indicates the number of polarization states in the collection.

HRESULT get_Count(long* count);

Parameters: *count* [out, retval]

The number of polarization states in the collection.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *count* is Null.

Remarks: This is a read-only property.

See also: "IPolStates Interface" on page 103.

IPolStates::Item Property

Allows random access to individual polarization states within the collection.

```
HRESULT get_Item(
    long index
    PolStateEnum* polState
);
```

Parameters: *index* [in]

The index of the polarization state to be queried.

polState [out, retval]

The polarization state with the specified index, or Null if the index is out of range.

Return value: *S_OK*

The value returned if successful.

E_INVALIDARG

The value returned if the index is out of range.

E_POINTER

The value returned if *polState* is Null.

Remarks: This is a read-only property.

See also: "IPolStates Interface" on page 103

"PolStateEnum Enumeration" on page 149.

IPolStates::Add Method

Adds a polarization state to the collection.

HRESULT Add(PolStateEnum polState);

Parameters: *polState* [in]

The polarization state to add to the collection.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *count* is Null.

See also: "IPolStates Interface" on page 103

"PolStateEnum Enumeration" on page 149.

IPowerRanges::RemoveAll Method

Removes all polarization states from the collection.

HRESULT RemoveAll();

Parameters: None.

Return value: *S_OK*

The value returned if successful.

See also: "IPolStates Interface" on page 103

IBuses Interface

Represents a collection of buses.

Properties:

IBuses::Count Property	Indicates the number of buses in the collection. Read-only.
IBuses::Item Property	Allows random access to individual buses within the collection. Read-only.

Methods:

IBuses::Add Method	Adds a bus to the collection.
IBuses::RemoveAll Method	Removes all buses from the collection.

Events: None.

See also: "IResourceManager::ScanBuses Method" on page 88.

IBuses::Count Property

Indicates the number of buses in the collection.

HRESULT get_Count(long* count);

Parameters: *count* [out, retval]

The number of buses in the collection.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *count* is Null.

Remarks: This is a read-only property.

See also: "IBuses Interface" on page 106.

IBuses::Item Property

Allows random access to individual buses within the collection.

```
HRESULT get_Item(
    long index,
    STR* bus
);
```

Parameters: *index* [in]

The index of the bus to be queried.

bus [out, retval]

The bus with the specified index, or Null if the index is out of range. For example, GPIB0, ASRL1

Return value: *S_OK*

The value returned if successful.

E_INVALIDARG

The value returned if the index is out of range.

E_POINTER

The value returned if *bus* is Null.

Remarks: This is a read-only property.

See also: "IBuses Interface" on page 106.

IBuses::Add Method

Adds a bus to the collection.

HRESULT Add(BSTR bus);

Parameters: *bus* [in]

The bus to add to the collection. For example, GPIB0, ASRL1

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *count* is Null.

See also: "IBuses Interface" on page 106.

IBuses::RemoveAll Method

Removes all buses from the collection.

HRESULT RemoveAll();

Parameters: None

Return value: *S_OK*

The value returned if successful.

See also: "IBuses Interface" on page 106.

IValues Interface

Represents a collection of curve values.

Properties:

IValues::Count Property	Indicates the values of buses in the collection. Read/write.
IValues::Item Property	Allows random access to individual values within the collection. Read/write.

Methods:

IValues::Add Method	Adds a value to the collection.
IValues::RemoveAll Method	Removes all values from the collection.

Events: None.

See also: "ICurve::Values Property" on page 122.

IValues::Count Property

Indicates the number of buses in the collection.

HRESULT get_Count(long* count);
HRESULT put_Count(long count);

Parameters: *count* [out, retval] [in]

The number of values in the collection.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *count* is Null.

Remarks: This is a read/write property.

See also: "IValues Interface" on page 109.

IValues::Item Property

Allows random access to individual values within the collection.

```
HRESULT get_Item(
    long index,
    DOUBLE* value
);
HRESULT put_Item(
    long index,
    DOUBLE value
);
```

Parameters: *index* [in]

The index of the value to be queried.

value [out, retval] [in]

The value with the specified index, or Null if the index is out of range.

Return value: *S_OK*

The value returned if successful.

E_INVALIDARG

The value returned if the index is out of range.

E_POINTER

The value returned if *bus* is Null.

Remarks: This is a read/write property.

See also: "IValues Interface" on page 109.

IValues::Add Method

Adds a value to the collection.
HRESULT Add(DOUBLE value);

Parameters: *value* [in]

The value to add to the collection.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *count* is Null.

See also: "IValues Interface" on page 109.

IValues::RemoveAll Method

Removes all values from the collection.
HRESULT RemoveAll();

Parameters: None

Return value: *S_OK*

The value returned if successful.

See also: "IValues Interface" on page 109.

ILightPath Interface

Represents a light path.

Properties:

ILightPath::SourceDUTPort Property	The source DUT port. Read/write.
ILightPath::DestinationDUTPort Property	The destination DUT port. Read/write.

Methods: None.

Events: None.

See also: "ICurve::LightPath Property" on page 122

"IMeasurementData::GetSingleCurve Method" on page 134

IMeasurementData::GetCurves Method.

ILightPath::SourceDUTPort Property

The source DUT port.

HRESULT get_SourceDUTPort(IDUTPort sourceDUTPort);**
HRESULT put_SourceDUTPort(IDUTPort* sourceDUTPort);

Parameters: *sourceDUTPort* [out, retval] [in]

The source DUT port.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *sourceDUTPort* is Null.

Remarks: This is a read/write property.

See also: "IDUTPort Interface" on page 114

"ILightPath::DestinationDUTPort Property" on page 113

"ILightPath Interface" on page 112

ILightPath::DestinationDUTPort Property

The destination DUT port.

HRESULT get_DestinationDUTPort(IDUTPort destinationDUTPort);**
HRESULT put_DestinationDUTPort(IDUTPort* destinationDUTPort);

Parameters: *destinationDUTPort* [out, retval] [in]

The destination DUT port.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *destinationDUTPort* is Null.

Remarks: This is a read/write property.

See also: "IDUTPort Interface" on page 114

"ILightPath::SourceDUTPort Property" on page 113

"ILightPath Interface" on page 112

IDUTPort Interface

Represents a port of the device under test.

Properties:

IDUTPort::Group Property	The DUT port group. Read/write.
IDUTPort::Number Property	The DUT port number. Read/write.

Methods: None.

Events: None.

IDUTPort::Group Property

The DUT port group.

HRESULT get_Group(DUTPortGroupEnum* group);
HRESULT put_Group(DUTPortGroupEnum group);

Parameters: *group* [out, retval] [in]

The DUT port group.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *group* is Null.

Remarks: This is a read/write property.

See also: "DUTPortGroupEnum Enumeration" on page 154

"IDUTPort Interface" on page 114.

IDUTPort::Number Property

The DUT port number.

HRESULT get_Number(long* number);
HRESULT put_Number(long number);

Parameters: *number* [out, retval] [in]

The DUT port number.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *number* is Null.

Remarks: This is a read/write property.

See also: "IDUTPort Interface" on page 114.

IMeasurementControl Interface

Features operations common to all measurement control classes. This is the base interface for measurement type specific interfaces.

Properties: None.

Methods:

IMeasurementControl::ShowAvailableResources Method	Provides a list of instruments that can be used to run the measurement.
IMeasurementControl::ConfigureResources Method	Defines the selection of instruments to be used.
IMeasurementControl::UseResources Method	Reuses the selection of instruments of a MeasurementData object.

Events: None.

IMeasurementControl::ShowAvailableResources Method

Provides a list of instruments that can be used to run an all-parameter measurement.

HRESULT ShowAvailableResources(IResources resources);**

Parameters: *resources* [out, retval]

A list of available resources for this kind of measurement.

Return value: *S_OK*

The value returned if successful.

E_PFL_INSTR_SETUP_INSUFF

There are not enough modules to support the type of measurement. The available modules are listed in *resources* but these are not sufficient to perform a measurement. The correct types of modules must exist and have a supported firmware version before they are reported. Please check any unreported modules to ensure they are working and have the correct firmware version installed.

E_PFL_CMD_ORDER_INVALID

The IPflManager::Initialize Method must be called before this function

E_POINTER

The value returned if *resources* is Null.

See also: "IResourceManager::ShowAllResources Method" on page 88

"IMeasurementControl Interface" on page 116

"IResources Interface" on page 97.

IMeasurementControl::ConfigureResources Method

Defines the selection of instruments to be used.

HRESULT ConfigureResources(IResources* resources);

Parameters:

resources [in]

Contains the selection of resources

Return value:

S_OK

The value returned if successful.

E_PFL_CMD_ORDER_INVALID

The IPflManager::Initialize Method must be called before this function.

E_PFL_RSRC_SEL_INVALID

The resource selection isn't valid for this measurement.

E_PFL_RSRC_NOT_EXISTING

Some resources defined aren't available.

E_PFL_STRING_FORMAT_INVALID

The selection string format is wrong.

E_PFL_INSTR_TIMEOUT

A module or mainframe failed to respond within the required time period.

E_PFL_GPIB_COMM

Communication on the GPIB bus failed.

E_PFL_LAN_COMM

Communication on the LAN failed.

E_PFL_INSTR_SETUP

An instrument failed to respond correctly.

E_POINTER

The value returned if *resources* is Null.

E_INVALIDARG

The value returned if *resources* is not a valid array descriptor or if the array has more than one dimension.

E_OUTOFMEMORY

Insufficient Memory.

E_FAIL

An error has occurred. Refer to the event log file.

See also: "IMeasurementControl Interface" on page 116

"IResources Interface" on page 97.

IMeasurementControl::UseResources Method

Reuses the selection of instruments of an ApPactMeasurementData object.

HRESULT UseResources(IMeasurementData* measurementData);

Parameters: *measurementData* [in]

Only the resource selection of this data is read.

Return value: *S_OK*

The value returned if successful.

E_PFL_CMD_ORDER_INVALID

The IPflManager::Initialize Method must be called before this function

E_PFL_RSRC_SEL_INVALID

The resource selection isn't valid for this measurement.

E_PFL_RSRC_NOT_EXISTING

Some resources defined aren't available.

E_PFL_INSTR_TIMEOUT

A module or mainframe failed to respond within the required time period.

E_PFL_GPIB_COMM

Communication on the GPIB bus failed.

E_PFL_LAN_COMM

Communication on the LAN failed

E_PFL_INSTR_SETUP

An instrument failed to respond correctly.

E_POINTER

The value returned if *measurementData* is Null.

E_OUTOFMEMORY

Insufficient Memory.

E_FAIL

An error has occurred. Refer to the event log file.

See also: "IMeasurementData Interface" on page 130

"IMeasurementControl Interface" on page 116.

ICurve Interface

Represents a collection of points (x, y). The x values are described by a start/step pair which means that they are equidistant. The y values are stored in an array.

Properties:

ICurve::Values Property	Returns the curve data as an IValues collection. Read/Write.
ICurve::LightPath Property	Returns the light path that the curve belongs to. Read only.
ICurve::Role Property	Returns the curve role. Read only.
ICurve::PolState Property	Returns the polarization state. Read only.
ICurve::Unit Property	Returns the unit of the y axis. Read only.
ICurve::Start Property	Returns the start value of the X axis. Read only.
ICurve::Step Property	Returns the step size between two adjacent X axis values. Read only.
ICurve::Size Property	Returns the number of data points. Read only.
ICurve::DataArray Property	Returns the curve data as an array. Read only.

Methods: None.

Events: None.

See also: "IMeasurementData::GetSingleCurve Method" on page 134

"IValues Interface" on page 109.

ICurve::Values Property

Returns the curve data as array.

HRESULT get_Values(IValues values);**
HRESULT put_Values(IValues* values);

Parameters: *values* [out, retval] [in]

The list of values.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *values* is Null.

Remarks: This is a read/write property.

See also: "IValues Interface" on page 109.

"ICurve Interface" on page 121

ICurve::LightPath Property

Returns the light path that the curve belongs to.

HRESULT get_LightPath(ILightPath lightPath);**

Parameters: *lightPath* [out, retval]

The light path.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *lightPath* is Null.

Remarks: The property is read only.

See also: "ILightPath Interface" on page 112

"ICurve Interface" on page 121.

ICurve::Role Property

Returns the role.
HRESULT get_Role(CurveRoleEnum* role);

Parameters: *role* [out, retval]

The role.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *role* is Null.

Remarks: The property is read only.

See also: "CurveRoleEnum Enumeration" on page 151

"ICurve Interface" on page 121.

ICurve::PolState Property

Returns the polarization state.
HRESULT get_PolState(PolStateEnum* polState);

Parameters: *polState* [out, retval]

The polarization state.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *polState* is Null.

Remarks: The property is read only.

See also: "PolStateEnum Enumeration" on page 149

"ICurve Interface" on page 121.

ICurve::Unit Property

Returns the unit of the y axis.
HRESULT get_Unit(CurveUnitEnum* unit);

Parameters: *unit* [out, retval]

The unit of the y axis.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *unit* is Null.

Remarks: The property is read only.

See also: "CurveUnitEnum Enumeration" on page 153

"ICurve Interface" on page 121.

ICurve::Start Property

Returns the start value of the X axis.
HRESULT get_Start(double* start);

Parameters: *start* [out, retval]

The start value of the X axis.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *start* is Null.

Remarks: The property is read only.

See also: "ICurve Interface" on page 121.

ICurve::Step Property

Returns the step size between two adjacent X axis values.
HRESULT get_Step(double* step);

Parameters: *step* [out, retval]

The step size.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *step* is Null.

Remarks: The property is read only.

See also: "ICurve Interface" on page 121.

ICurve::Size Property

Returns the number of data points.
HRESULT get_Size(long* size);

Parameters: *size* [out, retval]

The number of data points.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *size* is Null.

Remarks: The property is read only.

See also: "ICurve Interface" on page 121.

ICurve::DataArray Property

Returns the number of data points.

HRESULT get_DataArray(SAFEARRAY(double)* array);

Parameters: *array* [out, retval]

The number of data points.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *size* is Null.

Remarks: The property is read only.

See also: "ICurve Interface" on page 121.

ICurves Interface

Represents a collection of curves.

Properties:

ICurves::Count Property	Indicates the number of curves in the collection. Read-only.
ICurves::Item Property	Allows random access to individual curves within the collection. Read-only.

Methods: None.

Events: None.

See also: "IApPactMeasurementControl::ExecuteRealtimeMeasurement" on page 178

ICurves::Count Property

Indicates the number of curves in the collection.

HRESULT get_Count(long* count);

Parameters: *count* [out, retval]

The number of curves in the collection.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *count* is Null.

Remarks: This is a read-only property.

See also: "ICurves Interface" on page 127.

ICurves::Item Property

Allows random access to individual curves within the collection.

```
HRESULT get_Item(
    long index,
    ICurve** curve
);
```

Parameters: *index* [in]

The index of the curve to be queried.

curve [out, retval]

The curve with the specified index or Null if the index is out of range.

Return value: *S_OK*

The value returned if successful.

E_INVALIDARG

The value returned if the index is out of range.

E_POINTER

The value returned if *curve* is Null.

Remarks: This is a read-only property.

See also: "ICurve Interface" on page 121

"ICurves Interface" on page 127.

IMeasurementData Interface

Features access functions to the data base that stores all relevant data for a measurement in one single place. This interface is the base interface for measurement type specific interfaces.

Properties:

IMeasurementData::TimeStamp Property	Returns the time when the measurement was started. Read only.
IMeasurmentData::Duration Property	Returns the measurement duration. Read only.
IMeasurementData::OperatorID Property	Can be used to store a string that identifies the operator who did the measurement. Read/write.
IMeasurementData::DUTTemperature Property	Can be used to store a floating point number that specifies the temperature of the device under test.Read/write.
IMeasurementData::DUTDescription Property	Can be used to store a string that describes the device under test. Read/write.
IMeasurementData::Comment	Can be used to store a comment. Read/write.

Methods:

IMeasurementData::GetSingleCurve Method	Queries for specific curves. The first matching curve is returned.
IMeasurementData::GetResourceSelection Method	Used to query for the resources used in the measurement. Returns a list of resource specifications.
IMeasurementData::GetResourceDetails Method	Returns additional information for the resource specified by a resource specification.
IMeasurementData::GetParentResourceSpec Method	Used to query for the parent of a resource. Returns a resource specification.
IMeasurementData::GetChildResourceSpecs Method	Used to query for child resources. Returns a list of resource specifiations.
IMeasurementData::Load Method	Loads a measurement data object from the specified file.
IMeasurementData::Save Method	Saves a measurement data object to the specified file.

Events: None.

IMeasurementData::TimeStamp Property

Returns the time when the measurement was started.
HRESULT get_TimeStamp(DATE* timeStamp);

Parameters: *timeStamp* [out, retval]

The time stamp when the measurement was started.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *timeStamp* is Null.

Remarks: This is a read only property. The time stamp is set by the corresponding measurement control object during a measurement.

See also: "IMeasurementData Interface" on page 130.

IMeasurmentData::Duration Property

Returns the measurement duration.
HRESULT get_Duration(double* duration);

Parameters: *duration* [out, retval]

The measurement duration.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *duration* is Null.

Remarks: This is a read only property. The duration is set by the corresponding measurement control object during a measurement.

See also: "IMeasurementData Interface" on page 130.

IMeasurementData::OperatorID Property

Can be used to store a string that identifies the operator who did the measurement.

HRESULT get_OperatorID(BSTR* operatorID);
HRESULT put_OperatorID(BSTR operatorID);

Parameters: *operatorID* [out, retval] [in]

The operator identification.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *operatorID* is Null.

Remarks: This is a read/write property.

See also: "IMeasurementData Interface" on page 130.

IMeasurementData::DUTTemperature Property

Can be used to store a floating point number that specifies the temperature of the device under test.

HRESULT get_DUTTemperature(double* temperature);
HRESULT put_DUTTemperature(double* temperature);

Parameters: *temperature* [out, retval] [in]

The temperature of the device under test.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *temperature* is Null.

Remarks: This is a read/write property.

See also: "IMeasurementData Interface" on page 130.

IMeasurementData::DUTDescription Property

Can be used to store a string that describes the device under test.

HRESULT get_DUTDescription(BSTR* description);
HRESULT put_DUTDescription(BSTR description);

Parameters: *description* [out, retval] [in]

The description of the device under test.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *description* is Null.

Remarks: This is a read/write property.

See also: "IMeasurementData Interface" on page 130.

IMeasurementData::Comment

Can be used to store a comment.

HRESULT get_Comment(BSTR* comment);
HRESULT put_Comment(BSTR comment);

Parameters: *comment* [out, retval] [in]

The comment.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *comment* is Null.

Remarks: This is a read/write property.

See also: "IMeasurementData Interface" on page 130.

IMeasurementData::GetSingleCurve Method

This method queries for specific curves. The first matching curve is returned.

```
HRESULT GetSingleCurve(
    ILightPath* lightPath,
    CurveRoleEnum role,
    PolStateEnum polState,
    ICurve** curve
);
```

Parameters: *lightPath* [in]

The light path of the curve to be queried.

role [in]

The role of the curve to be queried.

polState [in]

The polarization state of the curve to be queried.

curve [out, retval]

The first curve matching the query expression, or Null if no match is found.

Return value: *S_OK*

The value returned if successful.

S_FALSE

The value returned if there is no match.

E_POINTER

The value returned if *lightPath* or *curve* is Null.

Remarks: The *GetSingleCurve* method returns the first matching curve.

See also: "ILightPath Interface" on page 112

"CurveRoleEnum Enumeration" on page 151

"PolStateEnum Enumeration" on page 149

"IMeasurementData Interface" on page 130

"ICurve Interface" on page 121.

IMeasurementData::GetResourceSelection Method

Used to query for the resources used in the measurement. Returns a list of resource specifications.

HRESULT GetResourceSelection(IResources resourceSpecs);**

Parameters:

resourceSpecs [out, retval]

The specifications of the resources used in the measurement. If no resources are assigned, returns an empty list.

Return value:

S_OK

The value returned if successful.

E_POINTER

The value returned if *resourceSpecs* is Null.

See also:

"IResources Interface" on page 97

"IMeasurementData Interface" on page 130

"IMeasurementData::GetResourceDetails Method" on page 136

"IMeasurementData::GetParentResourceSpec Method" on page 137

"IMeasurementData::GetChildResourceSpecs Method" on page 138.

IMeasurementData::GetResourceDetails Method

Returns additional information for the resource specified by a resource specification.

```
HRESULT GetResourceDetails(
    BSTR resourceSpec,
    IResourceDetails** details
);
```

Parameters:

resourceSpec [in]

The specification string of the resource for which the details are to be queried. For example, GPIB::20::INSTR

details [out, retval]

The details of the resource.

Return value:

S_OK

The value returned if successful.

E_INVALIDARG

The value returned if the format of resource specification string is not valid.

E_PFL_RSRC_NOT_EXISTING

The resource specified is not used in the measurement.

E_POINTER

The value returned if *details* is Null.

See also:

"IResourceDetails Interface" on page 94

"IMeasurementData Interface" on page 130

"IMeasurementData::GetResourceSelection Method" on page 135

"IMeasurementData::GetResourceDetails Method" on page 136

"IMeasurementData::GetChildResourceSpecs Method" on page 138

"IResourceManager::GetResourceDetails Method" on page 89.

IMeasurementData::GetParentResourceSpec Method

Used to query for the parent of a resource. Returns a resource specification.

HRESULT GetParentResourceSpec(
BSTR resourceSpec,
BSTR* parent
);

Parameters: *resourceSpec* [in]

The specification string of the resource for which the details are to be queried. For example, GPIB::20::INSTR

parent [out, retval]

The specification string of the parent resource. For example, GPIB::20::INSTR

Return value: *S_OK*

The value returned if successful.

E_INVALIDARG

The value returned if the format of resource specification string is not valid.

E_PFL_RSRC_NOT_EXISTING

The resource specified is not used in the measurement.

E_POINTER

The value returned if *parent* is Null.

See also: "IMeasurementData Interface" on page 130

"IMeasurementData::GetResourceSelection Method" on page 135

"IMeasurementData::GetResourceDetails Method" on page 136

"IMeasurementData::GetChildResourceSpecs Method" on page 138

"IResourceManager::GetResourceDetails Method" on page 89.

IMeasurementData::GetChildResourceSpecs Method

Used to query for the child resources. Returns a list of resource specifications.

```
HRESULT GetChildResourceSpecs(
    BSTR resourceSpec,
    IResources** children
);
```

Parameters:

resourceSpec [in]

The specification string of the resource for which child resources are to be queried. For example, GPIB::20::INSTR

children [out, retval]

The specification strings of the child resources.

Return value:

S_OK

The value returned if successful.

E_INVALIDARG

The value returned if the format of resource specification string is not valid.

E_PFL_RSRC_NOT_EXISTING

The resource specified is not used in the measurement.

E_POINTER

The value returned if *children* is Null.

See also:

"IResources Interface" on page 97

"IMeasurementData Interface" on page 130

"IMeasurementData::GetResourceDetails Method" on page 136

"IMeasurementData::GetParentResourceSpec Method" on page 137

"IMeasurementData::GetResourceSelection Method" on page 135

IMeasurementData::Load Method

Loads a measurement data object from the specified file.

HRESULT Load(BSTR filename);

Parameters: *filename* [in]

The name of the file from which the data is loaded.

Return value: *S_OK*

The value returned if successful.

E_OUTOFMEMORY

The value returned if there is not enough memory to perform the operation.

E_PFL_FILE_INCONSISTENT

The value returned if the file specified does not contain valid data.

E_PFL_FILE_HANDLING

The value returned if the file could not be opened.

E_FAIL

The value returned if the data could not be loaded.

See also: "IMeasurementData::Save Method" on page 140

"IMeasurementData Interface" on page 130.

IMeasurementData::Save Method

Saves a measurement data object to the specified file.

HRESULT Save(BSTR filename);

Parameters: *filename* [in]

The name of the file to which the data is saved.

Return value: *S_OK*

The value returned if successful.

E_OUTOFMEMORY

The value returned if there is not enough memory to perform the operation.

E_PFL_FILE_HANDLING

The value returned if the file could not be opened.

E_FAIL

The value returned if the data could not be saved.

See also: "IMeasurementData::Load Method" on page 139

"IMeasurementData Interface" on page 130.

4
Enumerations

Enumerations (enumerated data types) are used to define and list all possible values that can be placed in particular variables. No other values are valid.

This chapter describes the enumerations used that are independent of measurement type.

Enumerations - Summary

This section provides a tabulated summary of the enumerations used.

Table 3 Enumeration Summary

Enumeration	Description
"SweepSpeedEnum Enumeration" on page 145	This enumeration defines the possible sweep speeds of the tunable laser.
"AveragingTimeEnum Enumeration" on page 146	This enumeration defines the possible averaging times of the power meters.
"SweepCountEnum Enumeration" on page 147	This enumeration defines the sweep count of the tunable laser.
"PowerRangeEnum Enumeration" on page 147	This enumeration defines the possible range settings of the power meters.
"RangeDecrementEnum Enumeration" on page 149	This enumeration defines the range decrement of the power meters.
"PolStateEnum Enumeration" on page 149	This enumeration defines polarization states.
"CurveRoleEnum Enumeration" on page 151	This enumeration defines the role of a curve.
"CurveUnitEnum Enumeration" on page 153	This enumeration defines curve units.
"DUTPortGroupEnum Enumeration" on page 154	This enumeration defines the port group of the device under test (DUT).
"ExportSampleCountEnum Enumeration" on page 155	This enumeration specifies the number of data points in each exported curve.
"PflStatusEnum Enumeration" on page 156	This enumeration defines the PFL status codes.
"DirectionEnum Enumeration" on page 161	This enumeration defines the measurement direction.

Enumerations - Details

This section provides a short description of each enumeration, its syntax, and lists any defined constants.

SweepSpeedEnum Enumeration

This enumeration defines the possible sweep speeds of the tunable laser.

```
enum SweepSpeedEnum
{
    SweepSpeed500pmps = 500,
    SweepSpeed5nmps = 5000,
    SweepSpeed10nmps = 10000,
    SweepSpeed20nmps = 20000,
    SweepSpeed40nmps = 40000,
    SweepSpeed80nmps = 80000,
    SweepSpeedAuto = -1
};
```

Constants: *SweepSpeed500pmps ... SweepSpeed80nmps*

Specifies a sweep speed between 500 pm/s and 80 nm/s.

SweepSpeedAuto

Specifies that the sweep speed is automatically determined by the system.

Remarks: None.

See also: "IApPactUserLossParams::MaxSweepSpeed Property" on page 210

"IApPactSystemLossParams::SweepSpeed Property" on page 228

"IApPactUserPhaseParams::SweepSpeed Property" on page 223

"IApPactSystemPhaseParams::SweepSpeed Property" on page 234

AveragingTimeEnum Enumeration

This enumeration defines the possible averaging times of the power meters.

```
enum AveragingTimeEnum
{
    AveragingTimeAuto = 0
    AveragingTime25us = 25
    AveragingTime100us = 100
    AveragingTime200us = 200
    AveragingTime500us = 500
    AveragingTime1ms = 1000
    AveragingTime2ms = 2000
    AveragingTime5ms = 5000
    AveragingTime10ms = 10000
    AveragingTime20ms = 20000
    AveragingTime50ms = 50000
    AveragingTime100ms = 100000
    AveragingTime200ms = 200000
    AveragingTime500ms = 500000
    AveragingTime1s = 1000000
    AveragingTime2s = 2000000
    AveragingTime5s = 5000000
    AveragingTime10s = 10000000
};
```

Constants: *AveragingTimeAuto*

Specifies that the averaging time is determined automatically by the system.

AveragingTime25 ms ... AveragingTime10 s

Specifies an averaging time between 25 µs and 10 s in 1-2-5 steps.

Remarks: None.

See also: "IApPactUserLossParams::MaxAveragingTime Property" on page 211

"IApPactSystemLossParams::AveragingTime Property" on page 229.

SweepCountEnum Enumeration

This enumeration defines the sweep count of the tunable laser.

```
enum SweepCountEnum
{
    SweepCountOneSweep = 1,
    SweepCountTwoSweeps = 2,
    SweepCountThreeSweeps = 3
};
```

Constants: *SweepCountAuto*

Specifies that the sweep count is determined automatically by the system.

SweepCountOneSweep ... SweepCountThreeSweeps

Specifies a sweep count between one and three.

Remarks: None.

See also: "IApPactUserLossParams::SweepCountProperty" on page 212

PowerRangeEnum Enumeration

This enumeration defines the possible range settings of the power meters.

```
enum PowerRangeEnum
{
    PowerRangeAuto = 9999,
    PowerRange50dBm = 50,
    PowerRange40dBm = 40,
    PowerRange30dBm = 30,
    PowerRange20dBm = 20,
    PowerRange10dBm = 10,
    PowerRange0dBm = 0,
    PowerRangeM10dBm = -10,
    PowerRangeM20dBm = -20,
    PowerRangeM30dBm = -30,
    PowerRangeM40dBm = -40,
    PowerRangeM50dBm = -50,
    PowerRangeM60dBm = -60,
    PowerRangeM70dBm = -70,
    PowerRangeM80dBm = -80,
    PowerRangeM90dBm = -90,
    PowerRangeM100dBm = -100,
    PowerRangeM110dBm = -110,
    PowerRangeM120dBm = -120,
    PowerRangeM130dBm = -130,
    PowerRangeM140dBm = -140,
    PowerRangeM150dBm = -150,
    PowerRangeM160dBm = -160,
    PowerRangeM170dBm = -170,
    PowerRangeM180dBm = -180,
    PowerRangeM190dBm = -190,
    PowerRangeM200dBm = -200,
};
```

Constants: *PowerRangeAuto*

Specifies that the power range is determined automatically by the system.

PowerRange50dBm ... PowerRangeM200dBm

Specifies a power range between 50 dBm and –200 dBm, in 10 dB steps.

Remarks: None.

See also: "IApPactUserLossParams::TransmissionStartRange Property" on page 212

"IApPactUserLossParams::ReflectionStartRange Property" on page 213

"IApPactSystemLossParams::TransmissionRanges Property" on page 230

"IApPactSystemLossParams::ReflectionRanges Property" on page 231

RangeDecrementEnum Enumeration

This enumeration defines the range decrement of the power meters.

```
enum RangeDecrementEnum
{
    RangeDecrementAuto = 0
    RangeDecrement10dB = 10
    RangeDecrement20dB = 20
    RangeDecrement30dB = 30
    RangeDecrement40dB = 40
};
```

Constants: *RangeDecrementAuto*

Specifies that the range decrement is determined automatically by the system.

RangeDecrement10dB ... RangeDecrement40dB

Specifies a range decrement between 10 dB and 40 dB, in 10 dB steps.

Remarks: None.

See also: "IApPactUserLossParams::TransmissionRangeDecrement Property" on page 214

"IApPactUserLossParams::ReflectionRangeDecrement Property" on page 215

PolStateEnum Enumeration

This enumeration defines polarization states.

```
enum PolStateEnum
{
    PolStateUnknown = 0,
    PolStateUnpolarized = 1,
    PolStateLH0 = 2,
    PolStateLDP45 = 3,
    PolStateLDN45 = 4,
    PolStateLV90 = 5,
    PolStateLHC = 6,
    PolStateRHC = 7,
    PolStateUndefined = 8,
    PolStateAll = -1
};
```

Constants: *PolStateUnknown*

Specifies that the light is polarized, but the exact polarization state is unknown.

PolStateUnpolarized

Specifies that the light is unpolarized.

PolStateLH0

Specifies a polarization state is Linear Horizontal 0°.

PolStateLDP45

Specifies a polarization state is Linear Diagonal +45°.

PolStateLDN45

Specifies a polarization state of Linear Diagonal -45°.

PolStateLV90

Specifies a polarization state of Linear Vertical 90°.

PolStateLHC

Specifies a polarization state is Left Hand Circular.

PolStateRHC

Specifies a polarization state is Right Hand Circular.

PolStateUndefined

Specifies that the definition of a polarization state makes no sense in this context.

PolStateALL

Specifies any polarization state.

Remarks: None.

See also: "ICurve::PolState Property" on page 123

"IApPactSystemLossParams::PolStates Property" on page 232

"IMeasurementData::GetSingleCurve Method" on page 134

CurveRoleEnum Enumeration

This enumeration defines the role of a curve.

```
enum CurveRoleEnum
{
    CurveRoleDUTPwrWvl = 0,
    CurveRoleRefPwrWvl = 1,
    CurveRoleLossWvl = 2,
    CurveRolePDLWvl = 3,
    CurveRoleLossMinWvl = 4,
    CurveRoleLossMaxWvl = 5,
    CurveRoleLossAvgWvl = 6,
    CurveRoleRefPhaseWvl = 7,
    CurveRoleGDWvl = 8,
    CurveRoleDGDWvl = 9,
    CurveRoleCDWvl = 10,
    CurveRoleRefPwrWvlRT = 11,
    CurveRoleRefTermPwrWvl = 12,
};
```

Constants:

CurveRoleDUTPwrWvl

Power (over wavelength) obtained from DUT measurement.

CurveRoleRefPwrWvl

Power (over wavelength) obtained from reference measurement.

CurveRoleLossWvl

Insertion loss over wavelength.

CurveRolePDLWvl

Polarization dependent loss over wavelength.

CurveRoleLossMinWvl

Minimum insertion loss over wavelength.

CurveRoleLossMaxWvl

Maximum insertion loss over wavelength.

CurveRoleLossAvgWvl

Average insertion loss over wavelength.

CurveRoleRefPhaseWvl

Reference phase over wavelength.

CurveRoleGDWvl

Group delay (over wavelength) obtained from DUT measurement.

CurveRoleDGDWvl

Differential group delay (over wavelength) obtained from DUT measurement.

CurveRoleCDWvl

Chromatic dispersion over wavelength.

CurveRoleRefPwrWvlRT

Power (over wavelength) obtained from reference. For realtime loss measurements.

CurveRoleRefTermPwrWvl

Termination power (over wavelength) obtained from reference.

Remarks: None.

See also: "ICurve::Role Property" on page 123

"IMeasurementData::GetSingleCurve Method" on page 134.

CurveUnitEnum Enumeration

This enumeration defines curve units.

```
enum CurveUnitEnum
{
    CurveUnitdB = 0,
    CurveUnitW = 1,
    CurveUnitm = 2,
    CurveUnits = 3,
    CurveUnitspm = 4
};
```

Constants: *CurveUnitdB*

Specifies a unit of dB.

CurveUnitW

Specifies a unit of Watts.

CurveUnitm

Specifies a unit of Meters.

CurveUnits

Specifies a unit of Seconds.

CurveUnitspm

Specifies a unit of seconds per meter.

Remarks: None.

See also: "ICurve::Unit Property" on page 124

DUTPortGroupEnum Enumeration

This enumeration defines the port group of the device under test.

```
enum DUTPortGroupEnum
{
    DUTPortGroupIn = 0,
    DUTPortGroupOut = 1,
    DUTPortGroupUndefined = 2,
    DUTPortGroupAll = -1
};
```

Constants: *DUTPortGroupIn*

Specifies that the port group of the device under test is *In*.

DUTPortGroupOut

Specifies that the port group of the device under test is *Out*.

DUTPortGroupUndefined

Specifies that the port group of the device under test is undefined.

DUTPortGroupAll

Specifies all port groups

Remarks: None.

See also: "IDUTPort::Group Property" on page 115

ExportSampleCountEnum Enumeration

Specifies the number of data points that exported curves have.

```
enum ExportSampleCountEnum
{
    ExportSampleCountAll = -1,
    ExportSampleCount50000 = 50000,
    ExportSampleCount10000 = 10000,
    ExportSampleCount5000 = 5000,
    ExportSampleCount1000 = 1000,
    ExportSampleCount500 = 500
};
```

Constants: *ExportSampleCountAll*

Specifies that exported curves have the same number of samples as the curves stored in the data object.

ExportSampleCount50000 ... *ExportSampleCount500*

Specify the number of samples.

Remarks: None.

See also: "IApPactMeasurementData::Export Method" on page 195

PflStatusEnum Enumeration

This enumeration defines the PFL status codes.

```
enum PflStatusEnum
{
    S_PFL_SUCCESS = 0x0,
    E_PFL_ABORTED = 0x80040201,
    S_PFL_PARAMS_CORRECTED = 0x4021D,
    S_PFL_ENVIRONMENT_INCORRECT = 0x40204,
    E_PFL_NO_LICENSE = 0x80040202,
    E_PFL_FEATURE_NOT_SUPPORTED = 0x80040203,
    E_PFL_FIRMWARE_REV_INSUFF = 0x80040205,
    E_PFL_INSTR_SETUP_INSUFF = 0x80040206,
    E_PFL_RSRC_SEL_INVALID = 0x80040207,
    E_PFL_RSRC_NOT_EXISTING = 0x80040208,
    E_PFL_STRING_FORMAT_INVALID = 0x80040209,
    E_PFL_TYPE_INVALID = 0x8004020A,
    E_PFL_FILE_HANDLING = 0x8004020B,
    E_PFL_FILE_INCONSISTENT = 0x8004020C,
    E_PFL_INSTR_SETUP = 0x8004020D,
    E_PFL_GPIB_COMM = 0x8004020E,
    E_PFL_LAN_COMM = 0x8004020F,
    E_PFL_INSTR_SERVER_COMM = 0x80040210,
    E_PFL_INSTR_LASER_LOCK = 0x80040211,
    E_PFL_INSTR_TIMEOUT = 0x80040212,
    E_PFL_INSTR_OUT_OF_SPEC = 0x80040213,
    E_PFL_OPTICAL_CABLING = 0x80040214,
    E_PFL_ELECTRICAL_CABLING = 0x80040215,
    E_PFL_DUT_OUT_OF_SPEC = 0x80040216,
    E_PFL_LOCK_FAILED = 0x80040217,
    E_PFL_MATH = 0x80040218,
    E_PFL_REFERENCE_INCOMPATIBLE = 0x80040219,
    E_PFL_REFERENCE_MISSING = 0x8004021A,
    E_PFL_PARAMS_INCONSISTENT = 0x8004021B,
    E_PFL_PARAM_INVALID = 0x80070057,
    E_PFL_PWM_ZERO_FAILED = 0x8004021E,
    E_PFL_TRIGGER_LOST = 0x8004021F,
    E_PFL_INIT_FAILED = 0x80040220,
    E_PFL_CMD_DISALLOWED = 0x80040221,
    E_PFL_CMD_ORDER_INVALID = 0x80040222,
    E_PFL_DATA_INCONSISTENT = 0x80040223,
    E_PFL_DATA_INCOMPLETE = 0x80040224,
    E_PFL_NOT_FOUND = 0x80040225,
    E_PFL_ALREADY_STORED = 0x80040226,
    E_PFL_SIZE_MISMATCH = 0x80040227,
    E_PFL_MEM_INSUFF = 0x8007000E,
    E_PFL_GENERAL = 0x80004005,
    E_PFL_GENERAL_INSTR = 0x8004022A,
    E_PFL_TIMEOUT = 0x800405E8,
};
```

Constants: *S_PFL_SUCCESS*

Operation completed successfully.

S_PFL_ABORTED

The current operation has been aborted by the user.

S_PFL_PARAM_CORRECTED

The parameters were invalid but it was possible to propose corrected values.

S_PFL_ENVIRONMENT_INCORRECT

The PFL can only be used with a compatible operating system.

E_PFL_NO_LICENSE

A license for the specified feature is not available.

E_PFL_FEATURE_NOT_SUPPORTED

Correct versions of the DLLs required for the specified feature are not available.

E_PFL_FIRMWARE_REV_INSUFF

The firmware revision of at least one necessary module is too low to support the specified feature.

E_PFL_INSTR_SETUP_INSUFF

The available instruments are not adequate to support the specified measurement.

E_PFL_RSRC_SEL_INVALID

The module selection provided is not valid for the specified measurement feature.

E_PFL_RSRC_NOT_EXISTING

At least one module in the selection string describes a module that does not exist.

E_PFL_STRING_FORMAT_INVALID

The syntax of the selection string is not valid.

E_PFL_TYPE_INVALID

For example, the type of a DataSet transferred to the Transformation is invalid.

E_PFL_FILE_HANDLING

File handling error.

E_PFL_FILE_INCONSISTENT

The file contents are inconsistent.

E_PFL_INSTR_SETUP

Instrument manipulated in some way, such as: Module unplugged, optical head detached, ...

E_PFL_GPIB_COMM

Communication to an instrument connected via GPIB failed completely.

E_PFL_LAN_COMM

Communication to an instrument connected via LAN failed completely.

E_PFL_INSTR_SERVER_COMM

Communication with a shared agent failed (For example, server has shut down, LAN disconnected, ...)

E_PFL_INSTR_LASER_LOCK

Attempt to turn on the laser failed due to an active “RemoteInterlock” or “Softlock”.

E_PFL_INSTR_TIMEOUT

Call to an instrument command timed out.

E_PFL_INSTR_OUT_OF_SPEC

Device characteristics don´t meet the requirements (For example, the tunable laser has a “wavelength continuity issue” within the scan range specified in the specifications, or power levels in the Optical Testhead are out of specification).

E_PFL_OPTICAL_CABLING

Optical cabling error, such that the light power level in the optical system is too low.

E_PFL_ELECTRICAL_CABLING

Electrical cabling error in the system, such as swapped data cables between OTH and DAQ cards.

E_PFL_DUT_OUT_OF_SPEC

For example, DUT too long.

E_PFL_LOCK_FAILED

Locking of a necessary instrument has failed - or somebody else has locked this instrument.

E_PFL_MATH

A computational error occurred.

E_PFL_REFERENCE_INCOMPATIBLE

UseReference is called and the data in the passed measurement data object is incompatible with the data used for opening the measurement.

E_PFL_REFERENCE_MISSING

Execute measurement is called but no reference is available.

E_PFL_PARAMS_INCONSISTENT

Each parameter provided is valid, but not in combination.

E_PFL_PARAM_INVALID

At least one parameter is invalid.

E_PFL_PWM_ZERO_FAILED

Attempt to zero the powermeters failed – usually due to light on the sensor.

E_PFL_TRIGGER_LOST

The receivers did not get enough triggers.

E_PFL_INIT_FAILED

Initialization of a PFL component failed.

E_PFL_CMD_DISALLOWED

Command is not allowed in the current context.

E_PFL_CMD_ORDER_INVALID

Command cannot be executed unless other commands are executed beforehand.

E_PFL_DATA_INCONSISTENT

The content of the measurement data object is inconsistent.

E_PFL_DATA_INCOMPLETE

The content of the measurement data object is incomplete.

E_PFL_NOT_FOUND

The requested item could not be found.

E_PFL_ALREADY_STORED

The item is already stored.

E_PFL_SIZE_MISMATCH

Size mismatch.

E_PFL_MEM_INSUFF

Insufficient memory.

E_PFL_GENERAL

An unspecified error occured.

E_PFL_GENERAL_INSTR

Something went wrong with the instruments.

E_PFL_TIMEOUT

A call to the PFL layer timed out.

Remarks: None.

See also: "IApPactMeasurementControl::ConfigureParameters Method" on page 169

"ApPact Enumerations - Details" on page 242

DirectionEnum Enumeration

This enumeration defines the measurement direction.

```
enum DirectionEnum
{
    DirectionRX = 0,
    DirectionTX =1,
    DirectionBoth = 2
};
```

Constants: *DirectionRX*

The measurement direction Reflection.

DirectionTX

The measurement direction Transmission.

DirectionBoth

The measurement directions Transmission and Reflection.

Remarks: None.

See also: "IApPactMeasurementData::GetDUTLossCurve Method" on page 197

"IApPactMeasurementData::GetDUTPDLCurve Method" on page 198

"IApPactMeasurementData::GetDUTGDCurve Method" on page 199

"IApPactMeasurementData::GetDUTDGDCurve Method" on page 200

"IApPactMeasurementData::GetDUTCDCurve Method" on page 201

"ApPact Enumerations - Details" on page 242.

5
IApPact Interfaces

This chapter describes all the Agilent PFL COM interfaces that address All-Parameter Passive Component Test (ApPact) measurements.

IApPact Interfaces - Summary

This section provides a tabulated summary of the IApPact interfaces (collections of properties, methods and events) available.

Table 4 IApPact Interface Summary

Interface	Description
"IApPactMeasurementControl Interface" on page 167	Features all operations used to control the all-parameter measurement process. This interface inherits from IMeasurementControl.
"IApPactMeasurementData Interface" on page 188	Features access functions to the database that stores all relevant data for an all-parameter measurement in one place. This interface inherits from IMeasurementData.
"IApPactUserParams Interface" on page 202	Represents the parameters for an all-parameter PACT measurement set by the user.
"IApPactUserLossParams Interface" on page 205	Represents the user parameters for the loss part of an all-parameter measurement.
"IApPactUserPhaseParams Interface" on page 216	Represents the user parameters for the phase part of an all-parameter measurement.
"IApPactSystemParams Interface" on page 225	Represents the parameters for an all-parameter PACT measurement set by the system.
"IApPactSystemLossParams Interface" on page 227	Represents the system parameters for the loss part of an all-parameter measurement.
"IApPactSystemPhaseParams Interface" on page 233	Represents the system parameters for the phase part of an all-parameter measurement.
"IApPactReferenceParams Interface" on page 236	Represents the parameters used for the reference measurements in an all-parameter measurement.

IApPact Interfaces and Objects - Detail

The following sections list the properties, methods and events associated with each IApPact interface collection.

A short description of each property and method is provided, together with its syntax, parameters and return values.

IApPactMeasurementControl Interface

Features all operations to control the all-parameter measurement process. This interface inherits from the IMeasurementControl Interface.

Properties: None.

Methods:

IApPactMeasurementControl::ConfigureParameters Method	Defines the measurement parameters.
IApPactMeasurementControl::UseParameters Method	Reuses the measurement parameters of an ApPactMeasurementData object.
IApPactMeasurementControl::UseReference Method	Supplies a measurement with reference curves.
IApPactMeasurementControl::TidyUp Method	Cleans up a measurement control object.
IApPactMeasurementControl::AcquireResources Method	This method captures the resources; they now aren't accessible for others. Measurement settings are loaded into instruments.
IApPactMeasurementControl::ExecuteReference Method	Performs a reference measurement.
IApPactMeasurementControl::ExecuteMeasurement Method	Performs a DUT measurement.
IApPactMeasurementControl::ExecuteRealtimeMeasurement	Performs a realtime measurement.
IApPactMeasurementControl::StopRealtimeMeasurement	Closes a realtime measurement.
IApPactMeasurementControl::ClearData	Deletes all DUT measurement curves and Jones curves.
IApPactMeasurementControl::Abort Method	This method tries to prematurely terminate an ongoing operation.
IApPactMeasurementControl::ReleaseResources Method	This method terminates a series of reference and/or DUT measurements and releases all resources.
IApPactMeasurementControl::GetData Method	Returns the currently available measurement data.
IApPactMeasurementControl:: ZeroAllPowermeters Method	This method zeroes all powermeters.
IApPactMeasurementControl::CheckReference Method	This method checks the status of a particular reference measurement.

IApPactMeasurementControl::IsReadyForDUTMeasurement Method	This method checks if there are sufficient reference measurements available to perform a DUT measurement.

Events:

IApPactMeasurementControl::UpdateProgress Event	Delivers the progress of the current measurement process in percent.

See also: "IMeasurementControl Interface" on page 116.

IApPactMeasurementControl::ConfigureParameters Method

Defines the measurement parameters.

```
HRESULT ConfigureParameters(
    IApPactUserParams** params,
    ApPactParamCheckHintEnum* lossHint = 0,
    ApPactParamCheckHintEnum* phaseHint = 0,
    PflStatusEnum* status = 0,
);
```

Parameters:

params [in, out]

The user parameters for a measurement, potentially corrected.

lossHint [out, optional]

Hints about loss measurement parameters that were changed by this operation. Default is 0.

phaseHint [out, optional]

Hints about phase measurement parameters that were changed by this operation. Default is 0.

status [out, optional]

If the status is S_OK the parameters were taken unchanged; if it is S_PFL_PARAMS_CORRECTED, the parameters have been corrected. Default is 0.

Return value:

S_OK

The value returned if successful.

S_PFL_PARAM_CORRECTED

The parameter were invalid but it was possible to propose corrected values.

E_INVALIDARG

Wrong parameters.

E_PFL_PARAMS_CORRECTED

The parameters have been corrected

E_POINTER

The value returned if *params* is Null.

E_FAIL

An error has occurred. Refer to the event log file.

See also: "IApPactUserParams Interface" on page 202

"ApPactParamCheckHintEnum Enumeration" on page 246

"PflStatusEnum Enumeration" on page 156

"IApPactMeasurementControl Interface" on page 167.

IApPactMeasurementControl::UseParameters Method

Reuses the measurement parameters of an ApPactMeasurementData object.

HRESULT UseParameters(IApPactMeasurementData* measurementData);

Parameters: *measurementData* [in]

Only the measurement parameters of this object are read.

Return value: *S_OK*

The value returned if successful.

E_INVALIDARG

Wrong parameters.

E_POINTER

The value returned if *measurementData* is Null.

E_FAIL

An error has occurred. Refer to the event log file.

See also: "IApPactMeasurementData Interface" on page 188

"IApPactMeasurementControl Interface" on page 167.

IApPactMeasurementControl::UseReference Method

Supplies a measurement with reference curves.

HRESULT UseReference(IApPactMeasurementData* measurementData);

Parameters: *measurementData* [in]

Only the reference curves of this object are read that are compatible with the current user settings.

Return value: *S_OK*

The value returned if successful, even if no curves could be used.

E_PFL_CMD_ORDER_INVALID

The IPflManager::Initialize Method must be called before this function.

E_PFL_REFERENCE_INCOMPATIBLE

UseReference is called and the data in the passed measurement data object is incompatible to the data used for opening the measurement.

E_PFL_PARAMS_INCONSISTENT

Each provided parameter is valid but not the combination of the parameters.

E_PFL_PARAM_INVALID

At least one parameter is invalid.

E_PFL_DATA_INCONSISTENT

The content of the measurement data object is inconsistent.

E_POINTER

The value returned if *measurementData* is Null.

E_FAIL

An error has occurred. Refer to the event log file.

See also: "IApPactMeasurementData Interface" on page 188

"IApPactMeasurementControl Interface" on page 167.

IApPactMeasurementControl::TidyUp Method

Cleans up a measurement control object.

HRESULT TidyUp(void);

Parameters: None.

Return value: *S_OK*

Always successful.

See also: "IApPactMeasurementControl Interface" on page 167.

IApPactMeasurementControl::AcquireResources Method

This method captures the resources required; they are now not accessible by others. Measurement settings are loaded into instruments.

HRESULT AcquireResources(void);

Parameters: None.

Return value: *S_OK*

The value returned if successful.

S_PFL_ABORTED

The operation was aborted.

E_PFL_INSTR_SERVER_COMM

Communication with a shared Agent failed (For example, server has been shutdown, LAN disconnected, ...)

E_PFL_OPTICAL_CABLING

Incorrect optical cabling, such that the power level of light in the optical system is too low.

E_PFL_ELECTRICAL_CABLING

Incorrect electrical cabling, swapped data cables between OTH and DAQ.

E_PFL_LOCK_FAILED

Locking a necessary instrument failed - somebody else has locked this instrument

E_PFL_CMD_ORDER_INVALID

Command cannot be executed unless other commands are executed beforehand.

E_PFL_INSTR_SETUP

Instrument manipulated in some way, such as a module unplugged, optical head detached, ...

E_PFL_INSTR_COMM

Communication to an instrument connected via GPIB failed completely.

E_PFL_LAN_COMM

Communication to an instrument connected via LAN failed completely.

E_PFL_INSTR_TIMEOUT

Call to an instrument command timed out.

E_FAIL

An error has occurred. Refer to the event log file.

Remarks: The *Open* operation may be aborted with the *Abort* command.

See also: "IApPactMeasurementControl::ReleaseResources Method" on page 183

"IApPactMeasurementControl::Abort Method" on page 181

"IApPactMeasurementControl Interface" on page 167

IApPactMeasurementControl::ExecuteReference Method

Performs a reference measurement.

```
HRESULT ExecuteReference(
    ApPactReferenceTypeEnum measType,
    long timeout
);
```

Parameters:

measType [in]

Defines which reference device is currently connected.

timeout [in, optional]

Specifies an optional timeout in milliseconds. Default is 0, which means no timeout.

Return value:

S_OK

The value returned if successful.

S_PFL_ABORTED

The operation was aborted.

E_PFL_INSTR_SERVER_COMM

Communication with a shared Agent failed (For example, server has been shutdown, LAN disconnected, ...)

E_PFL_INSTR_LASER_LOCK

Attempt to turn on the laser failed due to an active 'RemoteInterlock' or'Softlock'.

E_PFL_OPTICAL_CABLING

Incorrect optical cabling, such that the power level of light in the optical system is too low.

E_PFL_ELECTRICAL_CABLING

Incorrect electrical cabling, swapped data cables between OTH and DAQ.

E_PFL_DUT_OUT_OF_SPEC

For example, DUT to long.

E_PFL_LOCK_FAILED

Locking a necessary instrument failed - somebody else has locked this instrument.

E_PFL_MATH

A computational error occurred.

E_PFL_PWM_ZERO_FAILED

Attempt to Zero the Powermeters failed - usually due to light on the sensor.

E_PFL_TRIGGER_LOST

Init of a PFL component failed.

E_PFL_CMD_DISALLOWED

Command is not allowed in the current context.

E_PFL_CMD_ORDER_INVALID

Command cannot be executed unless other commands are executed beforehand.

E_OUTOFMEMORY

Not enough memory to perform the operation.

E_PFL_INSTR_SETUP

Instrument manipulated in some way, such as a module unplugged, optical head detached, ...

E_PFL_GPIB_COMM

Communication to an instrument connected via GPIB failed completely.

E_PFL_LAN_COMM

Communication to an instrument connected via LAN failed completely.

E_PFL_INSTR_TIMEOUT

Call to an instrument command timed out.

E_FAIL

An error has occurred. Refer to the event log file.

Remarks: The *ExecuteReference* operation may be aborted with the *Abort* command.

See also: "ApPactReferenceTypeEnum Enumeration" on page 244

"IApPactMeasurementControl::ExecuteMeasurement Method" on page 176

"IApPactMeasurementControl::Abort Method" on page 181

"IApPactMeasurementControl Interface" on page 167

IApPactMeasurementControl::ExecuteMeasurement Method

Performs a DUT measurement. Call for each channel for which the measurement is to be performed.

```
HRESULT ExecuteMeasurement(
    long channel = 1,
    long timeout = 0
);
```

Parameters:

channel [in, optional]

The channel for which the measurement is to be performed. Defaults to 1.

timeout [in, optional]

Specifies an optional timeout in milliseconds. Default is 0, which means no timeout.

Return value:

S_OK

The value returned if successful.

S_PFL_ABORTED

The operation was aborted.

E_PFL_INSTR_SERVER_COMM

Communication with a shared agent failed (For example, server has been shutdown, LAN disconnected, ...)

E_PFL_INSTR_LASER_LOCK

Attempt to turn on the laser failed due to an active 'RemoteInterlock' or'Softlock'.

E_PFL_OPTICAL_CABLING

Incorrect optical cabling, such that the power level of light in the optical system is too low.

E_PFL_ELECTRICAL_CABLING

Incorrect electrical cabling, swapped data cables between OTH and DAQ.

E_PFL_DUT_OUT_OF_SPEC

For example, DUT to long.

E_PFL_LOCK_FAILED

Locking a necessary instrument failed - somebody else has locked this instrument

E_PFL_MATH

A computational error occurred.

E_PFL_TRIGGER_LOST

Init of a PFL component failed.

E_PFL_CMD_DISALLOWED

Command is not allowed in the current context.

E_PFL_CMD_ORDER_INVALID

Command cannot be executed unless other commands are executed beforehand.

E_OUTOFMEMORY

Not enough memory to perform the operation.

E_PFL_REFERENCE_MISSING

No valid set of references is available

E_PFL_INSTR_SETUP

Instrument manipulated in some way, such as a module unplugged, optical head detached, ...

E_PFL_GPIB_COMM

Communication to an instrument connected via GPIB failed completely

E_PFL_LAN_COMM

Communication to an instrument connected via LAN failed completely

E_PFL_INSTR_TIMEOUT

Call to an instrument command timed out

E_FAIL

An error has occurred. Refer to the event log file.

Remarks: The *ExecuteMeasurement* operation may be aborted with the *Abort* command.

See also: "IApPactMeasurementControl::ExecuteReference Method" on page 174

"IApPactMeasurementControl::Abort Method" on page 181

"IApPactMeasurementControl Interface" on page 167

IApPactMeasurementControl::ExecuteRealtimeMeasurement

Performs a realtime measurement.

```
HRESULT ExecuteRealtimeMeasurement(
    long timeout = 0,
    ICurves** curves
);
```

Parameters:

timeout [in, optional]

Specifies an optional timeout in milliseconds. Default is 0, which means no timeout.

curves [out, retval]

The result curves.

Return value:

S_OK

The value returned if successful.

S_PFL_ABORTED

The operation was aborted.

E_PFL_INSTR_SERVER_COMM

Communication with a shared agent failed (For example, server has been shutdown, LAN disconnected, ...)

E_PFL_INSTR_LASER_LOCK

Attempt to turn on the laser failed due to an active 'RemoteInterlock' or'Softlock'.

E_PFL_OPTICAL_CABLING

Incorrect optical cabling, such that the power level of light in the optical system is too low.

E_PFL_ELECTRICAL_CABLING

Incorrect electrical cabling, swapped data cables between OTH and DAQ.

E_PFL_DUT_OUT_OF_SPEC

For example, DUT to long.

E_PFL_LOCK_FAILED

Locking a necessary instrument failed - somebody else has locked this instrument.

E_PFL_MATH

A computational error occurred.

E_PFL_TRIGGER_LOST

Init of a PFL component failed.

E_PFL_CMD_DISALLOWED

Command is not allowed in the current context.

E_PFL_CMD_ORDER_INVALID

Command cannot be executed unless other commands are executed beforehand.

E_OUTOFMEMORY

Not enough memory to perform the operation.

E_PFL_REFERENCE_MISSING

No valid set of references is available.

E_PFL_INSTR_SETUP

Instrument manipulated in some way, such as a module unplugged, optical head detached, ...

E_PFL_GPIB_COMM

Communication to an instrument connected via GPIB failed completely.

E_PFL_LAN_COMM

Communication to an instrument connected via LAN failed completely.

E_PFL_INSTR_TIMEOUT

Call to an instrument command timed out.

E_POINTER

The value returned if *curves* is Null.

E_FAIL

An error has occurred. Refer to the event log file.

See also: "IApPactMeasurementControl::StopRealtimeMeasurement" on page 180

"IApPactMeasurementControl Interface" on page 167

"ICurves Interface" on page 127.

IApPactMeasurementControl::StopRealtimeMeasurement

HRESULT StopRealtimeMeasurement(void);

Parameters: None.

Return value: *S_OK*

The value returned if successful.

E_PFL_INSTR_SERVER_COMM

Communication with a shared agent failed (For example, server has been shutdown, LAN disconnected, ...).

E_PFL_INSTR_LASER_LOCK

Attempt to turn on the laser failed due to an active 'RemoteInterlock' or'Softlock'.

E_PFL_CMD_DISALLOWED

Command is not allowed in the current context.

E_PFL_CMD_ORDER_INVALID

Command cannot be executed unless other commands are executed beforehand.

E_OUTOFMEMORY

Not enough memory to perform the operation.

E_PFL_INSTR_SETUP

Instrument manipulated in some way, such as a module unplugged, optical head detached, ...

E_PFL_GPIB_COMM

Communication to an instrument connected via GPIB failed completely.

E_PFL_LAN_COMM

Communication to an instrument connected via LAN failed completely.

E_PFL_INSTR_TIMEOUT

Call to an instrument command timed out.

E_FAIL

An error has occurred. Refer to the event log file.

See also: "IApPactMeasurementControl::ExecuteRealtimeMeasurement" on page 178

"IApPactMeasurementControl Interface" on page 167.

IApPactMeasurementControl::ClearData

Deletes all DUT measurement curves and Jones curves.
HRESULT ClearData(void);

Parameters: None.

Return value: *S_OK*

The value returned if successful.

See also: "IApPactMeasurementControl::ExecuteMeasurement Method" on page 176

"IApPactMeasurementControl Interface" on page 167

IApPactMeasurementControl::Abort Method

This method tries to prematurely terminate an ongoing operation.
HRESULT Abort(void);

Parameters: None.

Return value: *S_OK*

The value returned if successful.

E_PFL_CMD_ORDER_INVALID

Command cannot be executed unless other commands are executed beforehand.

E_FAIL

An error has occurred. Refer to the event log file.

E_PFL_INSTR_SETUP

Instrument manipulated in some way, such as a module unplugged, optical head detached, ...

E_PFL_GPIB_COMM

Communication to an instrument connected via GPIB failed completely.

E_PFL_LAN_COMM

Communication to an instrument connected via LAN failed completely.

E_PFL_INSTR_TIMEOUT

Call to an instrument command timed out.

E_PFL_GENERAL_INSTR

An unspecified instrument error occured.

Remarks: This method may be used to abort the *AcquireResources*, *ZeroAllPowermeters*, *ExecuteReference* and *ExecuteMeasurement* operations.

These operations will either return *Stat_Aborted* or another status if the abort request could not be handled. If successfully aborted, the operations leave the MeasurementControl object in a state equivalent to the one it was in before the call.

The *Abort* operation must be called from a separate thread.

See also: "IApPactMeasurementControl::AcquireResources Method" on page 172

"IApPactMeasurementControl:: ZeroAllPowermeters Method" on page 185

"IApPactMeasurementControl::ExecuteReference Method" on page 174

"IApPactMeasurementControl::ExecuteMeasurement Method" on page 176

"IApPactMeasurementControl Interface" on page 167.

IApPactMeasurementControl::ReleaseResources Method

This method terminates a series of reference and/or DUT measurements, and releases all resources.

HRESULT ReleaseResources(void);

Parameters: None.

Return value: *S_OK*

The value returned if successful.

E_PFL_CMD_ORDER_INVALID

Command cannot be executed unless other commands are executed beforehand.

E_FAIL

An error has occurred. Refer to the event log file.

E_PFL_INSTR_SETUP

Instrument manipulated in some way, such as a module unplugged, optical head detached, ...

E_PFL_GPIB_COMM

Communication to an instrument connected via GPIB failed completely.

E_PFL_LAN_COMM

Communication to an instrument connected via LAN failed completely.

E_PFL_INSTR_TIMEOUT

Call to an instrument command timed out.

E_PFL_GENERAL_INSTR

An unspecified instrument error occured.

See also: "IApPactMeasurementControl::AcquireResources Method" on page 172

"IApPactMeasurementControl Interface" on page 167

IApPactMeasurementControl::GetData Method

Returns the currently available measurement data.

HRESULT GetData(IApPactMeasurementData data);**

Parameters: *data* [out, retval]

A measurement data object filled with the currently available data.

Return value: *S_OK*

The value returned if successful.

E_PFL_CMD_ORDER_INVALID

Command cannot be executed unless other commands are executed beforehand.

E_POINTER

The value returned if *data* is Null.

E_FAIL

An error has occurred. Refer to the event log file.

Remarks: All data supplied to or by the measurement will be found in the ApPactMeasurementData object. The amount of the data present depends on the ApPactMeasurementControl object's history.

All curves measured between an *Open* and *Close* call will be stored in this single object.

See also: "IApPactMeasurementData Interface" on page 188

"IApPactMeasurementControl Interface" on page 167.

IApPactMeasurementControl:: ZeroAllPowermeters Method

This method zeroes all powermeters.

HRESULT ZeroAllPowermeters(void);

Parameters: None.

Return value: *S_OK*

The value returned if successful.

E_PFL_CMD_ORDER_INVALID

Command cannot be executed unless other commands are executed beforehand.

E_FAIL

An error has occurred. Refer to the event log file.

See also: "IApPactMeasurementControl Interface" on page 167

IApPactMeasurementControl::CheckReference Method

This method checks the status of a particular reference measurement.

```
HRESULT CheckReference(
    ApPactReferenceTypeEnum referenceType,
    ApPactReferenceStatusEnum* referenceStatus
);
```

Parameters: *referenceType* [in]

The type of the reference measurement for which the status is to be queried.

referenceStatus [out, retval]

The status of the reference measurement.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *referenceStatus* is Null.

See also: "ApPactReferenceTypeEnum Enumeration" on page 244

"ApPactReferenceStatusEnum Enumeration" on page 245

"IApPactMeasurementControl::IsReadyForDUTMeasurement Method" on page 186

"IApPactMeasurementControl Interface" on page 167

IApPactMeasurementControl::IsReadyForDUTMeasurement Method

This method checks if there are sufficient reference measurements available to perform a DUT measurement.

HRESULT IsReadyForDUTMeasurement(VARIANT_BOOL* readyForDutMeasurement);

Parameters: *readyForDUTMeasurement* [out, retval]

Tells whether the available reference measurements are sufficient for a DUT measurement.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *readyForDUTMeasurement* is Null.

See also: "IApPactMeasurementControl::CheckReference Method" on page 185

"IApPactMeasurementControl Interface" on page 167.

IApPactMeasurementControl::UpdateProgress Event

Delivers the progress of the current measurement process in percent.

HRESULT UpdateProgress(long percent);

Parameters: *percent* [in]

The percentage of the current measuement process that has been completed.

Return value: The return value is not evaluated by PFL. Any value is allowed.

See also: "IApPactMeasurementControl Interface" on page 167.

IApPactMeasurementData Interface

Features access functions to the data base that stores all relevant data for an all-parameter measurement in one single place. This interface inherits from the IMeasurementData Interface.

Properties:

IApPactMeasurementData::UserParams Property	Returns the measurement parameters set by the user. Read only.
IApPactMeasurementData::SystemParams Property	Returns the measurement parameters set by the system. Read only.
IApPactMeasurementData::ReferenceParams Property	Returns the parameters used for the reference measurements. Read only.
IApPactMeasurementData::MaxCDAveraging WindowWidth Property	Returns the maximum averaging window width for the chromatic dispersion calculation. Read.

Methods:

IApPactMeasurementData::CalculateLoss Method	Calculates loss from the raw data.
IApPactMeasurementData::CalculatePDLMueller Method	Calculates polarization dependent loss from the raw data using the Mueller-Matrix method.
IApPactMeasurementData::CalculateCD Method	Calculates chromatic dispersion from group delay data.
IApPactMeasurementData::Export Method	Performs ASCII export to a file.
IApPactMeasurementData::ExportJonesMatrix Method	Performs Jones Matrix export to a file.
IApPactMeasurementData::ExportFirstRowMuellerMatrix Method	Performs Mueller Matrix export of first row to a file
IApPactMeasurementData::ExportCurves Method	Exports all Curves into a text file.
IApPactMeasurementData::GetDUTLossCurve Method	Returns the measured Loss curve in RX or TX.
IApPactMeasurementData::GetDUTPDLCurve Method	Returns the measured Polarization Dependent Loss curve in RX or TX.
IApPactMeasurementData::GetDUTGDCurve Method	Returns the measured Group Delay curve in RX or TX.
IApPactMeasurementData::GetDUTDGDCurve Method	Returns the measured Differential Group Delay curve in RX or TX.
IApPactMeasurementData::GetDUTCDCurve Method	Returns the measured Chromatic Dispersion curve in RX or TX.

Events: None.

IApPactMeasurementData::UserParams Property

Returns the measurement parameters set by the user.
HRESULT get_UserParams(IApPactUserParams params);**

Parameters: *params* [out, retval]

The measurement parameters set by the user.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *params* is Null.

Remarks: This is a read-only property.

See also: "IApPactUserParams Interface" on page 202

"IApPactMeasurementData::SystemParams Property" on page 190

"IApPactMeasurementData::ReferenceParams Property" on page 190

"IApPactMeasurementData Interface" on page 188.

IApPactMeasurementData::SystemParams Property

Returns the measurement parameters set by the system.

HRESULT get_SystemParams(IApPactSystemParams params);**

Parameters: *params* [out, retval]

The measurement parameters set by the system.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *params* is Null.

Remarks: This is a read-only property.

See also: "IApPactSystemParams Interface" on page 225

"IApPactMeasurementData::UserParams Property" on page 189

"IApPactMeasurementData::ReferenceParams Property" on page 190

"IApPactMeasurementData Interface" on page 188

IApPactMeasurementData::ReferenceParams Property

Returns the parameters used for the reference measurements.

HRESULT get_ReferenceParams(IApPactReferenceParams params);**

Parameters: *params* [out, retval]

The reference parameters.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *params* is Null.

Remarks: This is a read-only property.

See also: "IApPactReferenceParams Interface" on page 236

"IApPactMeasurementData::UserParams Property" on page 189

"IApPactMeasurementData::SystemParams Property" on page 190

"IApPactMeasurementData Interface" on page 188

IApPactMeasurementData::MaxCDAveragingWindowWidth Property

Returns the maximum averaging window width for the chromatic dispersion calculation.

HRESULT put_MaxCDAveragingWindowWidth(LONG* averagingWindowWidth);

Parameters: *averagingWindowWidth* [out, retval]

The max averaging window.

Return value: *S_OK*

The value returned if successful.

E_PFL_CMD_ORDER_INVALID

Command cannot be executed unless other commands are executed beforehand.

E_POINTER

The value returned if *averagingWindowWidth* is Null.

Remarks: This is a write-only property.

See also: "IApPactReferenceParams Interface" on page 236

"IApPactMeasurementData::UserParams Property" on page 189

"IApPactMeasurementData::SystemParams Property" on page 190

"IApPactMeasurementData Interface" on page 188

IApPactMeasurementData::CalculateLoss Method

Calculates loss from the raw data.

HRESULT CalculateLoss(void);

Return value:

S_OK

The value returned if successful.

E_PFL_DATA_INCOMPLETE

The value returned if either the measurement parameters or the reference parameters are missing.

E_PFL_DATA_INCONSISTENT

The value returned if an invalid curve is detected or no matching pairs of reference and DUT power curves could found.

E_PFL_MATH

The value returned if a mathematical error occurred.

E_FAIL

The value returned if the calculation failed for some other reason.

See also: "IApPactMeasurementData::CalculatePDLMueller Method" on page 193

"IApPactMeasurementData::CalculateCD Method" on page 194

"IApPactMeasurementData Interface" on page 188.

IApPactMeasurementData::CalculatePDLMueller Method

Calculates polarization dependent loss from the raw data using the Mueller-Matrix method.

HRESULT CalculatePDLMueller(void);

Return value:

S_OK

The value returned if successful.

E_PFL_DATA_INCOMPLETE

The value returned if either the measurement parameters or the reference parameters are missing.

E_PFL_DATA_INCONSISTENT

The value returned if an invalid curve is detected or no matching pairs of reference and DUT power curves could found.

E_PFL_MATH

The value returned if a mathematical error occurred.

E_FAIL

The value returned if the calculation failed for some other reason.

See also: "IApPactMeasurementData::CalculateLoss Method" on page 193

"IApPactMeasurementData::CalculateCD Method" on page 194

"IApPactMeasurementData Interface" on page 188.

IApPactMeasurementData::CalculateCD Method

Calculates chromatic dispersion from group delay data.

HRESULT CalculateCD([in] long averagingWindowWidth);

Parameters: *averagingWindowWidth* [in]

The averaging window width.

Return value: *S_OK*

The value returned if successful.

E_PFL_DATA_INCOMPLETE

The value returned if either the measurement parameters or the reference parameters are missing.

E_PFL_DATA_INCONSISTENT

The value returned if:

- An invalid curve is detected, or
- The group delay curve is missing, or
- The measurement parameters are inconsistent.

E_PFL_MATH

The value returned if a mathematical error occurred.

E_FAIL

The value returned if the calculation failed for some other reason.

See also: "IApPactMeasurementData::CalculateLoss Method" on page 193

"IApPactMeasurementData::CalculatePDLMueller Method" on page 193

"IApPactMeasurementData Interface" on page 188.

IApPactMeasurementData::Export Method

Performs ASCII export to a file.

```
HRESULT Export(
    BSTR filename,
    ApPactExportOptionsEnum options = ApPactExport_All,
    ExportSampleCountEnum sampleCount = ExportSampleCount_All,
    BSTR headline = “”
);
```

Parameters: *filename* [in]

The name of the file to save the data to.

options [in, optional]

Optionally specifies which parts of the data get exported. Defaults to all data.

sampleCount [in, optional]

Optionally specifies the number of data points that exported curves have. Default is that exported curves have the same number of samples as the curves stored in the data object.

headline [in, optional]

An optional headline that is written to the exported file.

Return value: *S_OK:*

The value returned if successful.

E_OUTOFMEMORY:

Not enough memory to perform the operation.

E_PFL_FILE_HANDLING:

The file could not be created; Insufficient disk space

See also: "ApPactExportOptionsEnum Enumeration" on page 243

"ExportSampleCountEnum Enumeration" on page 155

"IApPactMeasurementData Interface" on page 188.

IApPactMeasurementData::ExportJonesMatrix Method

Performs Jones Matrix export to an ascii file.
HRESULT ExportJonesMatrix(BSTR filename);

Parameters: *filename* [in]

The name of the file to save the data to.

Return value: *S_OK:*

The value returned if successful.

E_OUTOFMEMORY:

Not enough memory to perform the operation.

E_PFL_FILE_HANDLING:

The file could not be created; Insufficient disk space

See also: "IApPactMeasurementData Interface" on page 188.

IApPactMeasurementData::ExportFirstRowMuellerMatrix Method

Performs export of the first row of the Mueller Matrix to an ascii file.
HRESULT ExportFirstRowMuellerMatrix (BSTR filename);

Parameters: *filename* [in]

The name of the file to save the data to.

Return value: *S_OK:*

The value returned if successful.

E_OUTOFMEMORY:

Not enough memory to perform the operation.

E_PFL_FILE_HANDLING:

The file could not be created; Insufficient disk space

See also: "IApPactMeasurementData Interface" on page 188.

IApPactMeasurementData::ExportCurves Method

Exports all Curves into a text file.

HRESULT ExportCurves(BSTR filename);

Parameters: *filename* [in]

The name of the file to save the data to.

Return value: *S_OK:*

The value returned if successful.

E_OUTOFMEMORY:

Not enough memory to perform the operation.

E_PFL_FILE_HANDLING:

The file could not be created; Insufficient disk space

See also: "IApPactMeasurementData Interface" on page 188.

IApPactMeasurementData::GetDUTLossCurve Method

Returns the measured Loss curve in reflection or transmission.

```
HRESULT GetDUTLossCurve(
    DirectionEnum direction,
    LONG channel,
    ICurve** curve
);
```

Parameters: *direction* [in]

The direction of the measurement RX/TX.

channel [in]

The channel of the curve to be queried.

curve [out, retval]

The first curve matching the query expression, or Null if no match is found.

Return value: *S_OK:*

The value returned if successful.

E_FALSE

Not returned if there is no match.

E_POINTER:

The value returned if pLightPath or ppCurve is Null.

See also: "DirectionEnum Enumeration" on page 161

"IApPactMeasurementData Interface" on page 188.

IApPactMeasurementData::GetDUTPDLCurve Method

Returns the measured Polarization Dependent Loss curve in reflection or transmission.

```
HRESULT GetPDLLossCurve(
    DirectionEnum direction,
    LONG channel,
    ICurve** curve
);
```

Parameters: *direction* [in]

The direction of the measurement RX/TX.

channel [in]

The channel of the curve to be queried.

curve [out, retval]

The first curve matching the query expression, or Null if no match is found.

Return value: *S_OK:*

The value returned if successful.

E_FALSE

Not returned if there is no match.

E_POINTER:

The value returned if pLightPath or ppCurve is Null.

See also: "DirectionEnum Enumeration" on page 161

"IApPactMeasurementData Interface" on page 188.

IApPactMeasurementData::GetDUTGDCurve Method

Returns the measured Group Delay curve in reflection or transmission.

```
HRESULT GetGDCurve(
    DirectionEnum direction,
    LONG channel,
    ICurve** curve
);
```

Parameters:

direction [in]

The direction of the measurement RX/TX.

channel [in]

The channel of the curve to be queried.

curve [out, retval]

The first curve matching the query expression, or Null if no match is found.

Return value:

S_OK:

The value returned if successful.

E_FALSE

Not returned if there is no match.

E_POINTER:

The value returned if pLightPath or ppCurve is Null.

See also:

"DirectionEnum Enumeration" on page 161

"IApPactMeasurementData Interface" on page 188.

IApPactMeasurementData::GetDUTDGDCurve Method

Returns the measured Differential Group Delay curve in reflection or transmission.

```
HRESULT GetDGDCurve(
    DirectionEnum direction,
    LONG channel,
    ICurve** curve
);
```

Parameters:

direction [in]

The direction of the measurement RX/TX.

channel [in]

The channel of the curve to be queried.

curve [out, retval]

The first curve matching the query expression, or Null if no match is found.

Return value:

S_OK:

The value returned if successful.

E_FALSE

Not returned if there is no match.

E_POINTER:

The value returned if pLightPath or ppCurve is Null.

See also: "DirectionEnum Enumeration" on page 161

"IApPactMeasurementData Interface" on page 188.

IApPactMeasurementData::GetDUTCDCurve Method

Returns the measured Chromatic Dispersion curve in reflection or transmission.

```
HRESULT GetCDCurve(
    DirectionEnum direction,
    LONG channel,
    ICurve** curve
);
```

Parameters:

direction [in]

The direction of the measurement RX/TX.

channel [in]

The channel of the curve to be queried.

curve [out, retval]

The first curve matching the query expression, or Null if no match is found.

Return value:

S_OK:

The value returned if successful.

E_FALSE

Not returned if there is no match.

E_POINTER:

The value returned if pLightPath or ppCurve is Null.

See also: "DirectionEnum Enumeration" on page 161

"IApPactMeasurementData Interface" on page 188.

IApPactUserParams Interface

Represents the parameters for an all-parameter PACT measurement set by the user.

Properties:

IApPactUserParams::LossParams Property	Contains the user parameters for the loss part of an all-parameter measurement. Read/write.
IApPactUserParams::PhaseParams Property	Contains the user parameters for the phase part of an all-parameter measurement. Read/write.
IApPactUserParams::EnableRealtime Measurements Property	Determines whether realtime measurements are possible. Read/write.

Methods: None.

Events: None.

See also: "IApPactMeasurementControl::ConfigureParameters Method" on page 169.

IApPactUserParams::LossParams Property

Contains the user parameters for the loss part of an all-parameter measurement.

HRESULT get_LossParams(IApPactUserLossParams params);**
HRESULT put_LossParams(IApPactUserLossParams* params);

Parameters: *params* [out, retval] [in]

The user parameters for the loss part of an all-parameter measurement.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *params* is Null.

Remarks: This is a read/write property.

See also: "IApPactUserLossParams Interface" on page 205

"IApPactUserParams Interface" on page 202.

IApPactUserParams::PhaseParams Property

Contains the user parameters for the phase part of an all-parameter measurement.

HRESULT get_PhaseParams(IApPactUserPhaseParams params);**
HRESULT put_PhaseParams(IApPactUserPhaseParams* params);

Parameters: *params* [out, retval] [in]

The user parameters for the phase part of an all-parameter measurement.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *params* is Null.

Remarks: This is a read/write property.

See also: "IApPactUserPhaseParams Interface" on page 216

"IApPactUserParams Interface" on page 202.

IApPactUserParams::EnableRealtimeMeasurements Property

Determines whether realtime measurements are possible.

HRESULT get_EnableRealtimeMeasurements(VARIANT_BOOL* enabled);
HRESULT put_EnableRealtimeMeasurements(VARIANT_BOOL enabled);

Parameters: *enabled* [out, retval] [in]

True if realtime measurements are possible, false otherwise.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *enabled* is Null.

Remarks: This is a read/write property.

See also: "IApPactMeasurementControl::ExecuteRealtimeMeasurement" on page 178

"IApPactMeasurementControl::StopRealtimeMeasurement" on page 180

"IApPactUserParams Interface" on page 202.

IApPactUserLossParams Interface

Represents the user parameters for the loss part of an all-parameter measurement.

Properties:

IApPactUserLossParams::StartWavelength Property	The start wavelength for the loss measurements. Read/write.
IApPactUserLossParams::StepWavelength Property	The step wavelength for the loss measurements. Read/write.
IApPactUserLossParams::StopWavelength Property	The stop wavelength for the loss measurements. Read/write.
IApPactUserLossParams::MeasurementType Property	The type of the loss measurement to perform. Read/write.
IApPactUserLossParams::MeasureTransmission Property	Determines whether the transmission path should be loss measured. Read/write.
IApPactUserLossParams::MeasureReflection Property	Determines whether the reflection path should be loss measured. Read/write.
IApPactUserLossParams::EnableCoherenceControl Property	Determines whether the coherence control of the laser is enabled. Read/write.
IApPactUserLossParams::MaxSweepSpeed Property	Determines the maximum sweep speed of the laser. Read/write.
IApPactUserLossParams::MaxAveragingTime Property	Determines the maximum averaging time of the power meters. Read/write.
IApPactUserLossParams::SweepCount Property	Determines the sweep count. Read/write.
IApPactUserLossParams::TransmissionStartRange Property	Determines the start range of the power meters for the transmission path. Read/write.
IApPactUserLossParams::ReflectionStartRange Property	Determines the start range of the power meters for the reflection path. Read/write.
IApPactUserLossParams::TransmissionRangeDecrement Property	Determines the range decrement of the power meters for the transmission path between two sweeps. Read/write.
IApPactUserLossParams::ReflectionRangeDecrement Property	Determines the range decrement of the power meters for the relection path between two sweeps. Read/write.

Methods: None.

Events: None.

See also: "IApPactUserParams Interface" on page 202

"IApPactSystemLossParams Interface" on page 227.

IApPactUserLossParams::StartWavelength Property

The start wavelength for the loss measurements.

HRESULT get_StartWavelength(double* startWavelength);
HRESULT put_StartWavelength(double startWavelength);

Parameters: *startWavelength* [out, retval] [in]

The start wavelength.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *startWavelength* is Null.

Remarks: This is a read/write property.

See also: "IApPactUserLossParams Interface" on page 205

"IApPactUserPhaseParams::StepWavelength Property" on page 218

"IApPactUserPhaseParams::StopWavelength Property" on page 219

"IApPactUserPhaseParams::StartWavelength Property" on page 217.

IApPactUserLossParams::StepWavelength Property

The step wavelength for the loss measurements.

HRESULT get_StepWavelength(double* stepWavelength);
HRESULT put_StepWavelength(double stepWavelength);

Parameters: *stepWavelength* [out, retval] [in]

The step wavelength.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *stepWavelength* is Null.

Remarks: This is a read/write property.

See also: "IApPactUserLossParams Interface" on page 205

"IApPactUserLossParams::StartWavelength Property" on page 206

"IApPactUserLossParams::StopWavelength Property" on page 207

"IApPactUserLossParams::StepWavelength Property" on page 206.

IApPactUserLossParams::StopWavelength Property

The stop wavelength for the loss measurements.

HRESULT get_StopWavelength(double* stopWavelength);
HRESULT put_StopWavelength(double stopWavelength);

Parameters: *stopWavelength* [out, retval] [in]

The stop wavelength.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *stopWavelength* is Null.

Remarks: This is a read/write property.

See also: "IApPactUserLossParams Interface" on page 205

"IApPactUserLossParams::StartWavelength Property" on page 206

"IApPactUserLossParams::StepWavelength Property" on page 206

"IApPactUserPhaseParams::StopWavelength Property" on page 219.

IApPactUserLossParams::MeasurementType Property

The type of the loss measurement to perform.

HRESULT get_MeasurementType(
ApPactLossMeasurementTypeEnum* measurementType
);
HRESULT put_MeasurementType(
ApPactLossMeasurementTypeEnum measurementType
);

Parameters: *measurementType* [out, retval] [in]

The measurement type.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *measurementType* is Null.

Remarks: This is a read/write property.

See also: "ApPactLossMeasurementTypeEnum Enumeration" on page 242

"IApPactUserPhaseParams::MeasurementType Property" on page 219

"IApPactUserLossParams Interface" on page 205.

IApPactUserLossParams::MeasureTransmission Property

Determines whether loss in the transmission path should be measured.

HRESULT get_MeasureTransmission(VARIANT_BOOL* measureTransmission);
HRESULT put_MeasureTransmission(VARIANT_BOOL measureTransmission);

Parameters: *measureTransmission* [out, retval] [in]

True if the transmission path is measured; False if not.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *measureTransmission* is Null.

Remarks: Boolean. The property is read/write.

See also: "IApPactUserLossParams::MeasureReflection Property" on page 209

"IApPactUserLossParams Interface" on page 205

"IApPactUserLossParams::MeasureTransmission Property" on page 208.

IApPactUserLossParams::MeasureReflection Property

Determines whether loss in the reflection path should be measured.

HRESULT get_MeasureReflection(VARIANT_BOOL* measureReflection);
HRESULT put_MeasureReflection(VARIANT_BOOL measureReflection);

Parameters: *measureReflection* [out, retval] [in]

True if the reflection path is measured; False if not.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *measureReflection* is Null.

Remarks: Boolean. The property is read/write.

See also: "IApPactUserLossParams::MeasureTransmission Property" on page 208

"IApPactUserLossParams Interface" on page 205

"IApPactUserLossParams::MeasureReflection Property" on page 209.

IApPactUserLossParams::EnableCoherenceControl Property

Determines whether the coherence control of the laser is enabled.

HRESULT get_EnableCoherenceControl(VARIANT_BOOL* enableCoherenceControl);
HRESULT put_EnableCoherenceControl(VARIANT_BOOL enableCoherenceControl);

Parameters: *enableCoherenceControl* [out, retval] [in]

True if the coherence control is enabled; False if not.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *enableCoherenceControl* is Null.

Remarks: Boolean. The property is read/write.

See also: "ApPactParamCheckHintEnum Enumeration" on page 246

"IApPactUserLossParams Interface" on page 205.

IApPactUserLossParams::MaxSweepSpeed Property

Determines the maximum sweep speed of the laser.

HRESULT get_MaxSweepSpeed(SweepSpeedEnum* sweepSpeed);
HRESULT put_MaxSweepSpeed(SweepSpeedEnum sweepSpeed);

Parameters: *sweepSpeed* [out, retval] [in]

The maximum sweep speed of the laser.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *sweepSpeed* is Null.

Remarks: The property is read/write.

See also: "SweepSpeedEnum Enumeration" on page 145

"ApPactParamCheckHintEnum Enumeration" on page 246

"IApPactUserLossParams Interface" on page 205

"IApPactSystemLossParams::SweepSpeed Property" on page 228.

IApPactUserLossParams::MaxAveragingTime Property

Determines the maximum averaging time of the power meters.

HRESULT get_MaxAveragingTime(AveragingTimeEnum* averagingTime);
HRESULT put_MaxAveragingTime(AveragingTimeEnum averagingTime);

Parameters: *averagingTime* [out, retval] [in]

The maximum averaging time of the power meters.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *averagingTime* is Null.

Remarks: The property is read/write.

See also: "AveragingTimeEnum Enumeration" on page 146

"ApPactParamCheckHintEnum Enumeration" on page 246

"IApPactUserLossParams Interface" on page 205

"IApPactSystemLossParams::AveragingTime Property" on page 229.

IApPactUserLossParams::SweepCountProperty

Determines the sweep count.

HRESULT get_SweepCount(SweepCountEnum* sweepCount);
HRESULT put_SweepCount(SweepCountEnum sweepCount);

Parameters: *sweepCount* [out, retval] [in]

The sweep count.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *sweepCount* is Null.

Remarks: The property is read/write.

See also: "SweepCountEnum Enumeration" on page 147

"IApPactUserLossParams Interface" on page 205.

IApPactUserLossParams::TransmissionStartRange Property

Determines the start range of the power meters for the transmission path.

HRESULT get_TransmissionStartRange(PowerRangeEnum* startRange);
HRESULT put_TransmissionStartRange(PowerRangeEnum startRange);

Parameters: *startRange* [out, retval] [in]

The start range for the transmission path.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *startRange* is Null.

Remarks: The property is read/write.

See also: "PowerRangeEnum Enumeration" on page 147

"IApPactUserLossParams Interface" on page 205

"IApPactSystemLossParams::TransmissionRanges Property" on page 230.

IApPactUserLossParams::ReflectionStartRange Property

Determines the range decrement of the power meters for the transmission path between two sweeps.

HRESULT get_ReflectionStartRange(PowerRangeEnum* startRange);
HRESULT put_ReflectionStartRange(PowerRangeEnum startRange);

Parameters: *startRange* [out, retval] [in]

The start range for the reflection path.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *startRange* is Null.

Remarks: The property is read/write.

See also: "PowerRangeEnum Enumeration" on page 147

"IApPactUserLossParams Interface" on page 205

"IApPactSystemLossParams::ReflectionRanges Property" on page 231.

IApPactUserLossParams::TransmissionRangeDecrement Property

Determines the start range of the power meters for the transmission path.

```
HRESULT get_TransmissionRangeDecrement(
    RangeDecrementEnum* rangeDecrement
);
HRESULT put_TransmissionRangeDecrement(
    RangeDecrementEnum rangeDecrement
);
```

Parameters: *rangeDecrement* [out, retval] [in]

The range decrement for the transmission path.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *rangeDecrement* is Null.

Remarks: The property is read/write.

See also: "RangeDecrementEnum Enumeration" on page 149

"IApPactUserLossParams Interface" on page 205

"IApPactSystemLossParams::TransmissionRanges Property" on page 230.

IApPactUserLossParams::ReflectionRangeDecrement Property

Determines the start range of the power meters for the reflection path.

```
HRESULT get_ReflectionRangeDecrement(
    RangeDecrementEnum* rangeDecrement
);
HRESULT put_ReflectionRangeDecrement(
    RangeDecrementEnum rangeDecrement
);
```

Parameters: *rangeDecrement* [out, retval] [in]

The range decrement for the reflection path.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *rangeDecrement* is Null.

Remarks: The property is read/write.

See also: "RangeDecrementEnum Enumeration" on page 149

"IApPactUserLossParams Interface" on page 205

"IApPactSystemLossParams::ReflectionRanges Property" on page 231.

IApPactUserPhaseParams Interface

Represents the user parameters for the phase part of an all-parameter measurement.

Properties:

IApPactUserPhaseParams::StartWavelength Property	The start wavelength for the phase measurements. Read/write.
IApPactUserPhaseParams::StepWavelength Property	The step wavelength for the phase measurements. Read/write.
IApPactUserPhaseParams::StopWavelength Property	The stop wavelength for the phase measurements. Read/write.
IApPactUserPhaseParams::MeasurementType Property	The type of the phase measurement performed. Read/write.
IApPactUserPhaseParams::MeasureTransmission Property	Determines whether the transmission path should be measured for phase. Read/write.
IApPactUserPhaseParams::MeasureReflection Property	Determines whether the reflection path should be measured for phase. Read/write.
IApPactUserPhaseParams::AveragingCount Property	Determines the number of measurements for averaging. Read/write.
IApPactUserPhaseParams::AveragingWindowWidth Property	Determines the width of the averaging window. Read/write.
IApPactUserPhaseParams::CDAveragingWindowWidth Property	Determines the width of the averaging window for CD. Read/write.
IApPactUserPhaseParams::SupplyJonesMatrix Property	Deternines whether to supply the Jones matrix or not. Read/write.
IApPactUserPhaseParams::SweepSpeed Property	Deternines the sweep speed for the laser. Valid sweep speeds are: 80nm/s, 40nm/s, 20nm/s and 10nm/s. Read/write.

Methods: None.

Events: None.

See also: "IApPactUserLossParams Interface" on page 205.

IApPactUserPhaseParams::StartWavelength Property

The start wavelength for the phase measurements.

HRESULT get_StartWavelength(double* startWavelength);
HRESULT put_StartWavelength(double startWavelength);

Parameters: *startWavelength* [out, retval] [in]

The start wavelength.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *startWavelength* is Null.

Remarks: This is a read/write property.

See also: "IApPactUserPhaseParams Interface" on page 216

"IApPactUserPhaseParams::StepWavelength Property" on page 218

"IApPactUserPhaseParams::StopWavelength Property" on page 219

"IApPactUserLossParams::StartWavelength Property" on page 206.

IApPactUserPhaseParams::StepWavelength Property

The step wavelength for the phase measurements.

HRESULT get_StepWavelength(double* stepWavelength);
HRESULT put_StepWavelength(double stepWavelength);

Parameters: *stepWavelength* [out, retval] [in]

The step wavelength.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *stepWavelength* is Null.

Remarks: This is a read/write property.

See also: "IApPactUserPhaseParams Interface" on page 216

"IApPactUserPhaseParams::StartWavelength Property" on page 217

"IApPactUserPhaseParams::StopWavelength Property" on page 219

"IApPactUserPhaseParams::StepWavelength Property" on page 218.

IApPactUserPhaseParams::StopWavelength Property

The stop wavelength for the phase measurements.

HRESULT get_StopWavelength(double* stopWavelength);
HRESULT put_StopWavelength(double stopWavelength);

Parameters: *stopWavelength* [out, retval] [in]

The stop wavelength.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *stopWavelength* is Null.

Remarks: This is a read/write property.

See also: "IApPactUserPhaseParams Interface" on page 216

"IApPactUserPhaseParams::StartWavelength Property" on page 217

"IApPactUserPhaseParams::StepWavelength Property" on page 218

"IApPactUserPhaseParams::StopWavelength Property" on page 219.

IApPactUserPhaseParams::MeasurementType Property

The type of the loss measurement to perform.

```
HRESULT get_MeasurementType(
    ApPactPhaseMeasurementTypeEnum* measurementType
);
HRESULT put_MeasurementType(
    ApPactPhaseMeasurementTypeEnum measurementType
);
```

Parameters: *measurementType* [out, retval] [in]

The measurement type.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *measurementType* is Null.

Remarks: This is a read/write property.

See also: "ApPactPhaseMeasurementTypeEnum Enumeration" on page 243

"IApPactUserLossParams::MeasurementType Property" on page 208

"IApPactUserPhaseParams Interface" on page 216.

IApPactUserPhaseParams::MeasureTransmission Property

Determines whether the transmission path should be measured for phase.

HRESULT get_MeasureTransmission(VARIANT_BOOL* measureTransmission);
HRESULT put_MeasureTransmission(VARIANT_BOOL measureTransmission);

Parameters: *measureTransmission* [out, retval] [in]

True if the transmission path is measured; False if not.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *measureTransmission* is Null.

Remarks: Boolean. The property is read/write.

See also: "IApPactUserPhaseParams::MeasureReflection Property" on page 221

"IApPactUserPhaseParams Interface" on page 216

"IApPactUserPhaseParams::MeasureTransmission Property" on page 220.

IApPactUserPhaseParams::MeasureReflection Property

Determines whether the reflection path should be measured for phase.

HRESULT get_MeasureReflection(VARIANT_BOOL* measureReflection);
HRESULT put_MeasureReflection(VARIANT_BOOL measureReflection);

Parameters: *measureReflection* [out, retval] [in]

True if the reflection path is measured; False if not.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *measureReflection* is Null.

Remarks: Boolean. The property is read/write.

See also: "IApPactUserPhaseParams::MeasureTransmission Property" on page 220

"IApPactUserPhaseParams Interface" on page 216

"IApPactUserPhaseParams::MeasureReflection Property" on page 221.

IApPactUserPhaseParams::AveragingCount Property

Determines the number of measurements for averaging.

HRESULT get_AveragingCount(long* averagingCount);
HRESULT put_AveragingCount(long averagingCount);

Parameters: *averagingCount* [out, retval] [in]

The averaging count.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *averagingCount* is Null.

Remarks: The property is read/write.

See also: "ApPactParamCheckHintEnum Enumeration" on page 246

"IApPactUserPhaseParams Interface" on page 216.

IApPactUserPhaseParams::AveragingWindowWidth Property

Determines the width of the averaging window.

HRESULT get_AveragingWindowWidth(long* averagingWindowWidth);
HRESULT put_AveragingWindowWidth(long averagingWindowWidth);

Parameters: *averagingWindowWidth* [out, retval] [in]

The averaging window width.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *averagingWindowWidth* is Null.

Remarks: The property is read/write.

See also: "ApPactParamCheckHintEnum Enumeration" on page 246

"IApPactUserPhaseParams Interface" on page 216.

IApPactUserPhaseParams::CDAveragingWindowWidth Property

Determines the width of the averaging window for CD.

HRESULT get_CDAveragingWindowWidth(long* averagingWindowWidth);
HRESULT put_CDAveragingWindowWidth(long averagingWindowWidth);

Parameters: *averagingWindowWidth* [out, retval] [in]

The averaging window width.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *averagingWindowWidth* is Null.

Remarks: The property is read/write.

See also: "ApPactParamCheckHintEnum Enumeration" on page 246

"IApPactUserPhaseParams Interface" on page 216.

IApPactUserPhaseParams::SupplyJonesMatrix Property

Deternines whether to supply the Jones matrix or not. Read/write.

HRESULT get_SupplyJonesMatrix(VARIANT_BOOL* supplyJonesMatrix);
HRESULT put_SupplyJonesMatrix(VARIANT_BOOL supplyJonesMatrix);

Parameters: *supplyJonesMatrix* [out, retval]

True if the jones matrix is supplied, false otherwise.

supplyJonesMatrix [in]

True if the jones matrix is to be supplied, false otherwise.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *supplyJonesMatrix* is Null.

Remarks: Boolean. This is a read/write property.

See also: "IApPactUserPhaseParams Interface" on page 216.

IApPactUserPhaseParams::SweepSpeed Property

Determines the sweep speed of the laser.

HRESULT get_SweepSpeed(SweepSpeedEnum* sweepSpeed);
HRESULT put_SweepSpeed(SweepSpeedEnum sweepSpeed);

Parameters: *sweepSpeed* [out, retval] [in]

The sweep speed of the laser.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *sweepSpeed* is Null.

Remarks: The property is read/write.

See also: "SweepSpeedEnum Enumeration" on page 145

"ApPactParamCheckHintEnum Enumeration" on page 246

"IApPactUserPhaseParams Interface" on page 216

"IApPactSystemPhaseParams::SweepSpeed Property" on page 234.

IApPactSystemParams Interface

Represents the parameters for an all-parameter PACT measurement set by the system.

Properties:

IApPactSystemParams::LossParams Property	Contains the system parameters for the loss part of an all-parameter measurement. Read-only.
IApPactSystemParams::PhaseParams Property	Contains the system parameters for the phase part of an all-parameter measurement. Read-only.

Methods: None.

Events: None.

IApPactSystemParams::LossParams Property

Contains the system parameters for the loss part of an all-parameter measurement.

HRESULT get_LossParams(IApPactSystemLossParams params);**

Parameters: *params* [out, retval]

The system parameters for the loss part of an all-parameter measurement.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned *params* is Null.

Remarks: This is a read-only property.

See also: "IApPactSystemLossParams Interface" on page 227

"IApPactSystemParams Interface" on page 225.

IApPactSystemParams::PhaseParams Property

Contains the system parameters for the phase part of an all-parameter measurement.

HRESULT get_PhaseParams(IApPactSystemPhaseParams params);**

Parameters: *params* [out, retval]

The system parameters for the phase part of an all-parameter measurement.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned *params* is Null.

Remarks: This is a read-only property.

See also: "IApPactSystemPhaseParams Interface" on page 233

"IApPactSystemParams Interface" on page 225.

IApPactSystemLossParams Interface

Represents the system parameters for the loss part of an all- parameter measurement.

Properties:

IApPactSystemLossParams::SampleCount Property	The number of trace samples acquired during a sweep as calculated by the system. Read-only.
IApPactSystemLossParams::SweepSpeed Property	The sweep speed actually used in the loss measurements. Read-only.
IApPactSystemLossParams::AveragingTime Property	The averaging time actually used in the loss measurements. Read-only.
IApPactSystemLossParams::LaserPowerDbm Property	The laser power actually used in the loss measurements. Unit is dBm. Read-only.
IApPactSystemLossParams::TransmissionRanges Property	The power meter ranges actually used in the loss measurements for the transmission path. Read-only.
IApPactSystemLossParams::ReflectionRanges Property	The power meter ranges actually used in the loss measurements for the reflection path. Read-only.
IApPactSystemLossParams::PolStates Property	The polarization states for which the loss measurements are performed. Read-only.

Methods: None.

Events: None.

See also: "IApPactUserLossParams Interface" on page 205.

IApPactSystemLossParams::SampleCount Property

The number of trace samples acquired during a sweep as calculated by the system.

HRESULT get_SampleCount(long* sampleCount);

Parameters: *sampleCount* [out, retval]

The sample count.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *sampleCount* is Null.

Remarks: The property is read-only.

See also: "IApPactSystemPhaseParams::SampleCount Property" on page 234

"IApPactSystemLossParams Interface" on page 227.

IApPactSystemLossParams::SweepSpeed Property

The sweep speed actually used in the loss measurements.

HRESULT get_SweepSpeed(SweepSpeedEnum* sweepSpeed);

Parameters: *sweepSpeed* [out, retval]

The sweep speed of the tunable laser.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *sweepSpeed* is Null.

Remarks: The property is read-only.

See also: "SweepSpeedEnum Enumeration" on page 145

"IApPactSystemLossParams Interface" on page 227

"IApPactUserLossParams::MaxSweepSpeed Property" on page 210

"IApPactSystemPhaseParams::SweepSpeed Property" on page 234.

IApPactSystemLossParams::AveragingTime Property

The averaging time actually used in the loss measurements.
HRESULT get_AveragingTime(AveragingTimeEnum* averagingTime);

Parameters: *averagingTime* [out, retval]

The averaging time of the power meters.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *averagingTime* is Null.

Remarks: The property is read-only.

See also: "AveragingTimeEnum Enumeration" on page 146

"IApPactSystemLossParams Interface" on page 227

"IApPactUserLossParams::MaxAveragingTime Property" on page 211.

IApPactSystemLossParams::LaserPowerDbm Property

The laser power actually used in the loss measurements. Unit is dBm.
HRESULT get_LaserPowerDbm(double* laserPowerDbm);

Parameters: *laserPowerDbm* [out, retval]

The laser power in dBm.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *laserPowerDbm* is Null.

Remarks: The property is read-only.

See also: "IApPactSystemLossParams Interface" on page 227.

IApPactSystemLossParams::TransmissionRanges Property

The power meter ranges actually used in the loss measurements for the transmission path.

HRESULT get_TransmissionRanges(IPowerRanges ranges);**

Parameters: *ranges* [out, retval]

The power meter ranges used for the transmission path.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *ranges* is Null.

Remarks: The property is read-only.

See also: "IPowerRanges Interface" on page 100

"PowerRangeEnum Enumeration" on page 147

"IApPactSystemLossParams::ReflectionRanges Property" on page 231

"IApPactSystemLossParams Interface" on page 227

"IApPactUserLossParams::TransmissionStartRange Property" on page 212

"IApPactUserLossParams::TransmissionRangeDecrement Property" on page 214.

IApPactSystemLossParams::ReflectionRanges Property

The power meter ranges actually used in the loss measurements for the reflection path.

HRESULT get_ReflectionRanges(IPowerRanges ranges);**

Parameters: *ranges* [out, retval]

The power meter ranges used for the reflection path.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *ranges* is Null.

Remarks: The property is read-only.

See also: "IPowerRanges Interface" on page 100

"PowerRangeEnum Enumeration" on page 147

"IApPactSystemLossParams::TransmissionRanges Property" on page 230

"IApPactSystemLossParams Interface" on page 227

"IApPactUserLossParams::ReflectionStartRange Property" on page 213

"IApPactUserLossParams::ReflectionRangeDecrement Property" on page 215.

IApPactSystemLossParams::PolStates Property

The polarization states for which the loss measurements are performed.

HRESULT get_PolStates(IPolStates polStates);**

Parameters: *polStates* [out, retval]

The list of polarization states.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *polStates* is Null.

Remarks: The property is read-only.

See also: "IPolStates Interface" on page 103

"PolStateEnum Enumeration" on page 149

"IApPactSystemLossParams Interface" on page 227.

IApPactSystemPhaseParams Interface

Represents the system parameters for the phase part of an all-parameter measurement.

Properties:

IApPactSystemPhaseParams::SampleCount Property	The number of trace samples acquired during a sweep as calculated by the system. Read-only.
IApPactSystemPhaseParams::SweepSpeed Property	The sweep speed actually used in the phase measurements. Read-only.
IApPactSystemPhaseParams::LaserPowerDbm Property	The laser power actually used in the phase measurements. Unit is dBm. Read-only.

Methods: None.

Events: None.

See also: "IApPactUserPhaseParams Interface" on page 216

IApPactSystemPhaseParams::SampleCount Property

The number of trace samples acquired during a sweep as calculated by the system.

HRESULT get_SampleCount(long* sampleCount);

Parameters: *sampleCount* [out, retval]

The number of trace samples.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *sampleCount* is Null.

Remarks: The property is read-only.

See also: "IApPactSystemPhaseParams Interface" on page 233.

IApPactSystemPhaseParams::SweepSpeed Property

The sweep speed actually used in the phase measurements.

HRESULT get_SweepSpeed(SweepSpeedEnum* sweepSpeed);

Parameters: *sweepSpeed* [out, retval]

The sweep speed of the laser.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *sweepSpeed* is Null.

Remarks: The property is read-only.

See also: "SweepSpeedEnum Enumeration" on page 145

"IApPactSystemPhaseParams Interface" on page 233.

IApPactSystemPhaseParams::LaserPowerDbm Property

The laser power actually used in the phase measurements.

HRESULT get_LaserPowerDbm(double* laserPowerDbm);

Parameters: *laserPowerDbm* [out, retval]

The laser power.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *laserPowerDbm* is Null.

Remarks: Unit is dBm. The property is read-only.

See also: "IApPactSystemPhaseParams Interface" on page 233.

IApPactReferenceParams Interface

Represents the parameters used for the reference measurements in an all-parameter measurement.

Properties:

IApPactReferenceParams::PolarizerAngle Property	Polarizer Angle. Read-only.
IApPactReferenceParams::WavelengthOffset Property	Wavelength Offset. Read-only.

Methods: None.

Events: None.

IApPactReferenceParams::PolarizerAngle Property

Polarizer Angle.
HRESULT get_PolarizerAngle(double* polarizerAngle);

Parameters: *polarizerAngle* [out, retval]

The polarizer angle.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *polarizerAngle* is Null.

Remarks: This is a read-only property.

See also: "IApPactReferenceParams Interface" on page 236.

IApPactReferenceParams::WavelengthOffset Property

Wavelength offset.
HRESULT get_WavelengthOffset(double* wavelengthOffset);

Parameters: *wavelengthOffset* [out, retval]

Wavelength offset.

Return value: *S_OK*

The value returned if successful.

E_POINTER

The value returned if *wavelengthOffset* is Null.

Remarks: This is a read-only property.

See also: "IApPactReferenceParams Interface" on page 236.

6
ApPact Enumerations

Enumerations (enumerated data types) are used to define and list all possible values that can be placed in particular variables. No other values are valid.

This chapter describes all the enumerations associated with All-Parameter Passive Component Test (ApPact) measurements.

ApPact Enumerations - Summary

This section provides a tabulated summary of the All-Parameter Passive Component Test enumerations.

Table 5 ApPact Enumeration Summary

Enumeration	Description
"ApPactLossMeasurementTypeEnum Enumeration" on page 242	This enumeration defines the types of loss measurement to perform as part of an all-parameter measurement.
"ApPactPhaseMeasurementTypeEnum Enumeration" on page 243	This enumeration defines the types of phase measurement to perform as part of an all-parameter measurement.
"ApPactExportOptionsEnum Enumeration" on page 243	This enumeration defines the parts of the all-parameter measurement data that are exported to an ASCII file.
"ApPactReferenceTypeEnum Enumeration" on page 244	This enumeration defines all reference measurement types that occur in all-parameter measurements.
"ApPactReferenceStatusEnum Enumeration" on page 245	This enumeration defines the possible status of a reference measurement.
"ApPactParamCheckHintEnum Enumeration" on page 246	Returned by the IApPactMeasurementControl::ConfigureParameters Method.

ApPact Enumerations - Details

This section provides a short description of each All-Parameter Passive Component Test enumeration, its syntax, and lists any defined constants.

ApPactLossMeasurementTypeEnum Enumeration

This enumeration defines the loss measurement types to perform as part of an all-parameter measurement.

```
enum ApPactLossMeasurementTypeEnum
{
    ApPactLossMeasurementTypeNone = 0,
    ApPactLossMeasurementTypeLoss = 1,
    ApPactLossMeasurementTypePDL = 2
};
```

Constants: *ApPactLossMeasurementTypeNone*

Specifies that no loss measurements are performed.

ApPactLossMeasurementTypeLoss

Specifies that loss is measured, but not Polarization Dependent Loss.

ApPactLossMeasurementTypePDL

Specifies that Polarization Dependent Loss is measured, as well as loss.

Remarks: None.

See also: "ApPactPhaseMeasurementTypeEnum Enumeration" on page 243

"IApPactUserLossParams::MeasurementType Property" on page 208.

ApPactPhaseMeasurementTypeEnum Enumeration

This enumeration defines the phase measurement types to perform as part of an all-parameter measurement.

```
enum ApPactPhaseMeasurementTypeEnum
{
    ApPactPhaseMeasurementTypeNone = 0,
    ApPactPhaseMeasurementTypeGD = 1,
    ApPactPhaseMeasurementTypeDGD = 2
};
```

Constants: *ApPactPhaseMeasurementTypeNone*

Specifies that no phase measurements are performed.

ApPactPhaseMeasurementTypeGD

Specifies that group delay is measured.

ApPactPhaseMeasurementTypeDGD

Specifies that differential group delay is measured, as well as group delay.

Remarks: None.

See also: "ApPactPhaseMeasurementTypeEnum Enumeration" on page 243

"IApPactUserPhaseParams::MeasurementType Property" on page 219.

ApPactExportOptionsEnum Enumeration

This enumeration defines the parts of the all-parameter measurement data that is exported to an ASCII file.

```
enum ApPactExportOptionsEnum
{
    ApPactExportDUTSettings = 1,
    ApPactExportMeasurementParams = 2,
    ApPactExportAnalysisData = 4,
    ApPactExportTraces = 8,
    ApPactExportAll = 15
};
```

Constants: *ApPactExportDUTSettings*

Specifies that the DUT settings are exported.

ApPactExportMeasurementParams

Specifies that the measurement parameters are exported.

ApPactExportAnalysisData

Specifies that the analysis data are exported.

ApPactExportTraces

Specifies that the curves are exported.

ApPactExportAll

Specifies that all data are exported.

Remarks: The constants can be linked using logical or.

See also: "IApPactMeasurementData::Export Method" on page 195.

ApPactReferenceTypeEnum Enumeration

This enumeration defines all reference measurement types that occur in all-parameter measurements.

```
enum ApPactReferenceTypeEnum
{
    ApPactReferenceTypeTransmission = 0,
    ApPactReferenceTypeReflection = 1,
    ApPactReferenceTypeTermination = 2,
    ApPactReferenceTypeInitialization = 3
};
```

Constants: *ApPactReferenceTypeTransmission*

Specifies a transmission reference measurement.

ApPactReferenceTypeReflection

Specifies a reflection reference measurement.

ApPactReferenceTypeTermination

Specifies a termination reference measurement.

ApPactReferenceTypeInitialization

Specifies an initialization reference measurement.

Remarks: None.

See also: IApPactMeasurementControl::ExecuteReference Method.

ApPactReferenceStatusEnum Enumeration

This enumeration defines the possible status of a reference measurement.

```
enum ApPactReferenceStatusEnum
{
    ApPactReferenceStatusRequiredMissing = 0,
    ApPactReferenceStatusRequiredPresent = 1,
    ApPactReferenceStatusOptionalMissing = 2,
    ApPactReferenceStatusOptionalPresent = 3,
    ApPactReferenceStatusNotApplicable = 4,
    ApPactReferenceStatusRequiredInContextMissing = 5
};
```

Constants:

ApPactReferenceStatusRequiredMissing

The reference measurement is required for a DUT measurement, but is missing.

ApPactReferenceStatusRequiredPresent

The reference measurement is required for a DUT measurement and is present.

ApPactReferenceStatusOptionalMissing

The reference measurement is optional for a DUT measurement, but is missing.

ApPactReferenceStatusOptionalPresent

The reference measurement is optional for a DUT measurement and is present.

ApPactReferenceStatusNotApplicable

The reference neasurement is not used for DUT measurement.

ApPactReferenceStatusRequiredInContextMissing

Another reference measurement is present, so a previously optional reference measurement is now required. For example, reflection and transmission references are optional for loss measurements. But if one is specified, so must the other.

Remarks: None.

See also: "IApPactMeasurementControl::CheckReference Method" on page 185.

ApPactParamCheckHintEnum Enumeration

Returned by the IApPactMeasurementControl::ConfigureParameters Method.

```
enum ApParamCheckHintEnum
{
    ApParamCheckHintOK =0,
    ApParamCheckHintCoherenceCtrlNotAvailable =1,
    ApParamCheckHintSweepSpeedNotAvailable = 2,
    ApParamCheckHintAveragingTimeNotAvailable = 3,
    ApParamCheckHintWavelengthStartStop = 4,
    ApParamCheckHintWavelengthStart = 5,
    ApParamCheckHintWavelengthStop = 6,
    ApParamCheckHintWavelengthStep = 7,
    ApParamCheckHintSampleCount = 8,
    ApParamCheckHintScanRange = 9,
    ApParamCheckHintWavelengthStepRange = 10,
    ApParamCheckHintSweepCount = 11,
    ApParamCheckHintPowermeterRange = 12,
    ApParamCheckHintAveragingWindowWidth = 13,
    ApParamCheckHintAveragingWindowWvl = 14,
    ApParamCheckHintAveragingCount = 15,
    ApParamCheckHintRealtime = 16,
    ApParamCheckHintSupplyJonesMatrix = 17,
    ApParamCheckHintCDAvgWndWidth =18,
};
```

Constants:

ApParamCheckHintOK

All parameters are valid.

ApParamCheckHintCoherenceCtrlNotAvailable

The IApPactUserLossParams::EnableCoherenceControl Property is set to true, but the laser has no coherence control.

ApParamCheckHintSweepSpeedNotAvailable

The IApPactUserLossParams::MaxSweepSpeed Property is set to a value which is not supported by the tunable laser.

ApParamCheckHintAveragingTimeNotAvailable

The IApPactUserLossParams::MaxAveragingTime Property is set to a value which is not supported by the power meters.

ApParamCheckHintWavelengthStartStop

The stop wavelength is larger than the start wavelength.

ApParamCheckHintWavelengthStart

The start wavelength has been changed or, more probably, is out of range.

ApParamCheckHintWavelengthStop

The stop wavelength has been changed or, more probably, is out of range.

ApParamCheckHintWavelengthStep

The wavelength step has been changed or, more probably, is out of range.

ApParamCheckHintSampleCount

The number of samples, resulting from start, stop and step wavelength, is out of range.

ApParamCheckHintScanRange

The defined scan range is out of range.

ApParamCheckHintWavelengthStepRange

The scan range is not a multiple of the step size.

ApParamCheckHintSweepCount

Not enough ranges found for the specified number of sweeps.

ApParamCheckHintPowermeterRange

The power range is not supported by the power meter.

ApParamCheckHintAveragingWindowWidth

The averaging window width for GD or DGD measurements is either out of range or not an even number.

ApParamCheckHintAveragingWindowWvl

The product of average window width and step wavelength is out of range.

ApParamCheckHintAveragingCount

The number of averages is invalid.

ApParamCheckRealTime

The parameter set does not fulfill the conditions required to support realtime measurements.

ApParamCheckHintSupplyJonesMatrix

The selected measurement type does not support the export of a Jones matrix.

ApParamCheckHintCDAvgWndWidth

The averaging window width for the calculation of Chromatic Dispersion is not compatible with the size of the the Group Delay curve.

Remarks: None.

See also: IApPactMeasurementControl::ConfigureParameters Method.

Index

www.agilent.com

Printed in Germany September 2004
Second edition, September 2004

81910-90A61

Agilent Technologies