Keysight N7700 Photonic Application Suite

Insertion Loss Measurement Engine

User's Guide

KEYSIGHT TECHNOLOGIES

Notices

Manual Part Number

N7700-91002

Edition

Edition 13.0, August 2021

Keysight Technologies Deutschland GmbH
Herrenberger Strasse 130,
71034 Böblingen, Germany

Technology Licenses

U.S. Government Rights

Warranty

Safety Notices

CAUTION

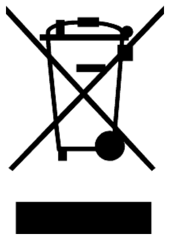
A CAUTION notice denotes a hazard. It calls attention to an operating procedure, practice, or the like that, if not correctly performed or adhered to, could result in damage to the product or loss of important data. Do not proceed beyond a CAUTION notice until the indicated conditions are fully understood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

Compliance and Environmental Information

Table 1 Compliance and Environmental Information

Safety Symbol	Description
	This product complies with WEEE Directive (2002/96/EC) marking requirements. The affixed label indicates that you must not discard this electrical/electronic product in domestic household waste. Product Category: With reference to the equipment types in WEEE Directive Annex I, this product is classed as a “Monitoring and Control instrumentation” product. Do not dispose in domestic household waste. To return unwanted products, contact your local Keysight office, or see http://about.keysight.com/en/companyinfo/environment/takeback.shtml for more information.

Contents

1 Quick Start Information

This section demonstrates how to connect your instruments optically and electrically for wavelength-swept insertion loss measurements using the Photonic Application Suite. It is intended as a quickstart guide, although it is strongly recommended that you read through Chapter , “Getting Started,” on page 13 and Chapter , “Insertion Loss Application,” on page 21 of this User's Guide and through the User's Guides of all instruments used in your setup prior to operating the instruments.

NOTE

You need to be logged in as an administrator to install the *Photonic Application Software Suite*.

Install the *Photonic Application Software Suite* before you connect any instrument(s) to the USB port of the PC. This ensures that the drivers are available when you connect the instrument(s). If you have connected the instrument(s) prior to software installation, you may have to delete the instrument(s) manually from the Windows Device Manager. This does not apply to instruments that are connected via GPIB.

After installation of the *Photonic Application Software Suite*, connect the instruments to the local area network or the USB or GPIB. You may use USB hubs in order to increase the number of USB ports on your PC. Several instruments can also be accessed through your local network. For the connected instruments to be found by Photonic Application Suite engines, they need to be configured in VISA, for example with the Connection Expert which is part of the IO Libraries Suite, if Keysight VISA is primary.

After the instruments have been connected and recognized, start the *Launch Pad* located in the start menu. Then click on “Insertion Loss”. On first start, the *Configuration Wizard* comes up and searches for all suitable instruments. Follow the instructions given by the *Configuration Wizard*. After finishing the wizard, the application comes up with all configured instruments. If you change your hardware configuration, run the wizard again by clicking on *File->Run Configuration Wizard*.

Multi-Channel Insertion Loss Setup

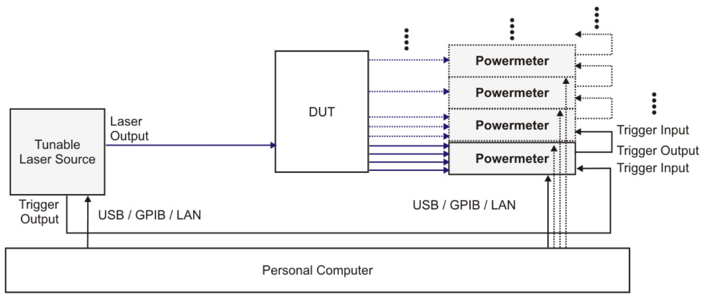
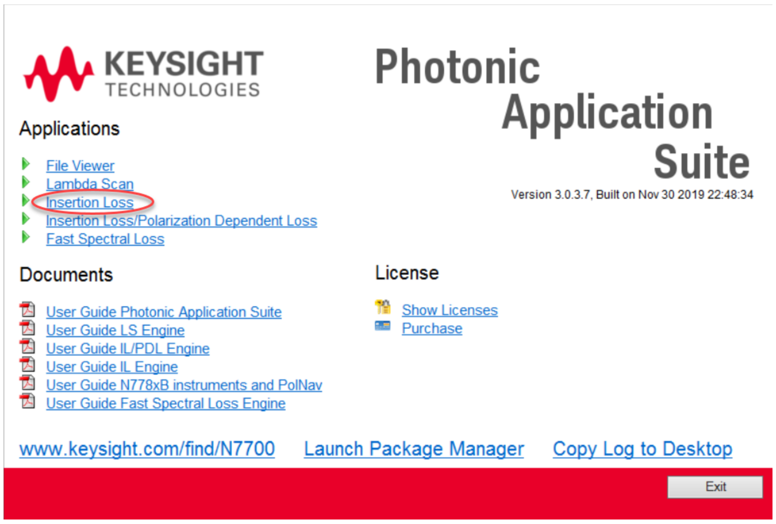


Figure 1 Insertion Loss Setup

NOTE Refer to Measurement Setup on page 40 for more information on measurement setup.

NOTE If there are multiple tunable laser sources inside one mainframe, only the one in the slot with the smallest slot number can be used in this application. This is a legacy restriction from the 816x driver MFlambdascan functions.

Software User Interface


Photonic Application Suite

Version 3.0.3.7, Built on Nov 30 2019 22:48:34

Applications

- File Viewer
- Lambda Scan
- Insertion Loss
- Insertion Loss/Polarization Dependent Loss
- Fast Spectral Loss

Documents

- User Guide Photonic Application Suite
- User Guide LS Engine
- User Guide IL/PDL Engine
- User Guide IL Engine
- User Guide N778xB instruments and PolNav
- User Guide Fast Spectral Loss Engine

License

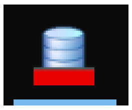
- Show Licenses
- Purchase

www.keysight.com/find/N7700 Launch Package Manager Copy Log to Desktop

Figure 2 The *Launch Pad* gives you access to the installed components. Click on "Insertion Loss" to start the Insertion Loss Engine

NOTE

The Insertion Loss Engine consists of a *Client Software*, which handles the graphical input/output, and a *Server Software,* which handles the communication with the hardware. For remote control or automation, the controlling program communicates directly with the *Server Software*. The server will be automatically started when you click on "Insertion Loss". The presence of the server is indicated in the task bar:



2 Getting Started

About this manual

This manual covers the swept-wavelength multi-channel Insertion loss Application that can be performed by combining one or more of Keysight’s optical power meters with one of Keysight’s tunable laser sources.

You might also refer to the *N7700 Keysight Photonic Application Suite User's Guide* to get information on the core Photonic Application Suite functionality including the File Viewer, Plugins, and the general COM API.

System Requirements

- One or more suitable mainframes (8163B, 8164B, or 8166B)
- Keysight tunable laser source with continuous-sweep capability
- One or more Keysight optical power meter(s) from the list below:
  - N7744A / N7745A
  - N7744C / N7745C
  - Any number of Lightwave Measurement System (LMS) power meter(s) supported by the 816x Plug & Play driver
- Optionally, one or more LMS mainframe(s) (8163B, 8164B, 8166B)
- BNC cables (50 Ohms)
- LAN, GPIB or USB cables
- Personal Computer:
  - Operating System: Microsoft Windows 10 (64 bit)
  - Depending on the instruments, PC interfaces to LAN, USB, and GPIB can be used.

NOTE **If there are multiple tunable laser sources inside one mainframe, only the one in the slot with the lowest slot number can be used in this application.**

Software Installation

The Insertion Loss engine communicates with the instruments over a VISA software layer that should be installed before connecting the instruments. Keysight VISA is provided by installing the IO Libraries Suite, which also provides drivers that may be required by some devices, like GPIB interface adapters and instruments connected by USB. Consult the instrument documentation for establishing connections with VISA.

All drivers and supplements necessary for using Keysight instruments with the Insertion Loss engine are included in the Photonic Application Suite installation package. The Package Manager will indicate if any required packages are missing and still need to be installed.

Refer to the *N7700 Photonic Application User's Guide* for information on how to install this software.

Instruments connected via GPIB and often USB will usually be recognized automatically for VISA, for example by Connection Expert when using Keysight VISA. Instruments connected via LAN might need to be added manually. Note that the N77xxC family of instrument will also be identified as LAN instruments in VISA and do not require extra USB drivers. For use with the IL engine, instruments should be connected with the primary VISA if more than one VISA installation is used.

NOTE You need to be logged in as an administrator to install the *Photonic Application Software Suite*.

Connecting the instruments

Install the software before you connect the hardware

Depending upon the type of TLS and power meter mainframes you are going to use for the setup, you have different options for connecting the instruments to the PC. Refer to the user guides of all instruments used, to check how to connect those instruments.

The Photonic Application Suite installs the Keysight IO Libraries, including the Keysight Connection Expert. Make sure that all instruments are shown in the Keysight Connection Expert, then start the Photonic Application Suite software.

NOTE

The swept-wavelength Insertion Loss application requires the trigger output of the tunable laser source to be connected to the trigger input ports of all power meters. The power meters and mainframes are automatically configured to pass through the trigger input signals, so you can connect the TLS trigger output with any of the power meter mainframe's trigger input and then cascade the trigger output of the power meter with the trigger input of the next power meter. Proceed like this for all remaining power meters.

Run the Configuration Wizard

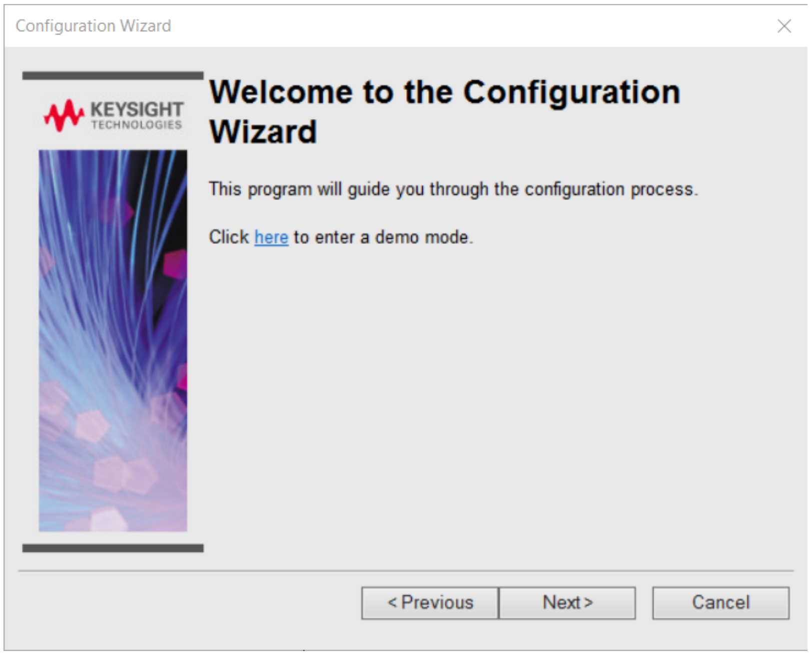
NOTE

If Keysight VISA is used, all the instruments need to be identified using the Keysight Connection Expert, before the Photonic Application Suite Configuration Wizard is used to configure the Insertion Loss application to use these instruments. The Connection Expert is part of the Keysight IO Libraries. Start the Connection Expert and click the Refresh All button. When all the connected instruments have been identified, proceed with the steps below.

This procedure is only required if a certain supported instrument is connected to the PC for the first time.

Once the software is installed and the instruments are connected and turned on, you can start the IL Engine and run the Configuration Wizard.

On first startup the Configuration Wizard will come up automatically.



Welcome to the Configuration Wizard

This program will guide you through the configuration process.

Click here to enter a demo mode.

Figure 3 Configuration Wizard

If you want to rerun the Configuration Wizard, you can select on **Run Configuration Wizard** in the File menu of the Insertion Loss Measurement Engine Client.

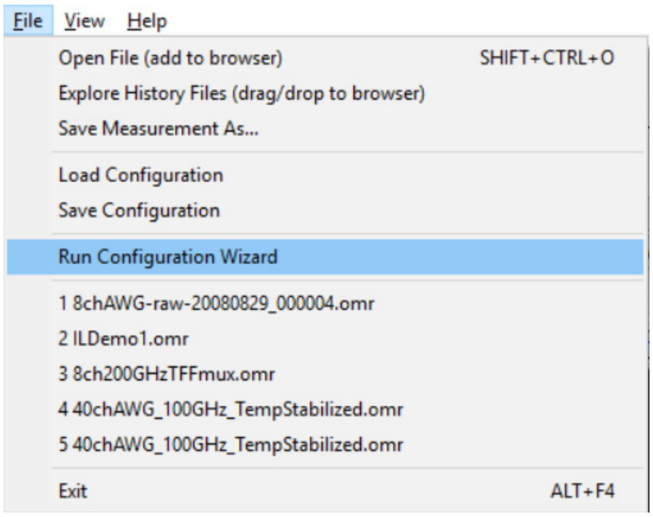

File
View
Help
Open File (add to browser)
SHIFT+CTRL+O
Explore History Files (drag/drop to browser)
Save Measurement As...
Load Configuration
Save Configuration
Run Configuration Wizard
1 8chAWG-raw-20080829_000004.omr
2 ILDemo1.omr
3 8ch200GHzTFFmux.omr
4 40chAWG_100GHz_TempStabilized.omr
5 40chAWG_100GHz_TempStabilized.omr
Exit
ALT+F4

Figure 4 Running Configuration Wizard from the File Menu

Please follow the instructions shown in the dialog box.

3 Insertion Loss Application

General

You can run the various applications of the Photonic Application Suite from the Launch pad that is found in the start menu.

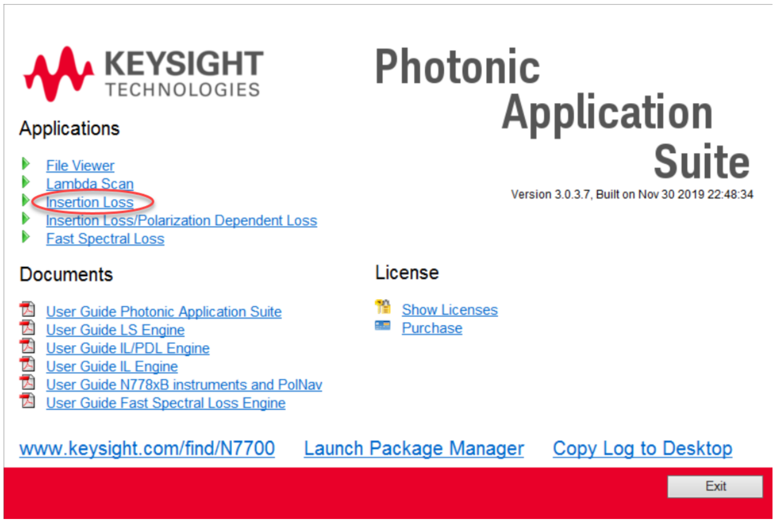

KEYSIGHT TECHNOLOGIES
Photonic Application Suite
Version 3.0.3.7, Built on Nov 30 2019 22:48:34
Applications
File Viewer
Lambda Scan
Insertion Loss
Insertion Loss/Polarization Dependent Loss
Fast Spectral Loss
Documents
User Guide Photonic Application Suite
User Guide LS Engine
User Guide IL/PDL Engine
User Guide IL Engine
User Guide N778xB instruments and PolNav
User Guide Fast Spectral Loss Engine
License
Show Licenses
Purchase
www.keysight.com/find/N7700
Launch Package Manager
Copy Log to Desktop
Exit

Figure 5 The *Launch Pad* gives you access to the installed components. Click on “Insertion Loss” to start the wavelength-swept Insertion Loss Engine.

The Insertion Loss application is based on the popular MFlambdascan functions of the 816x VXI Plug&Play driver, which can also be installed from the Package Manager. The Insertion Loss package provides a convenient graphical user interface for measurements using these functions as well as a server with COM interface for automation. This interface allows use of the MFlambdascan measurements from 64-bit programming environments that cannot directly use the 32-bit 816x driver.

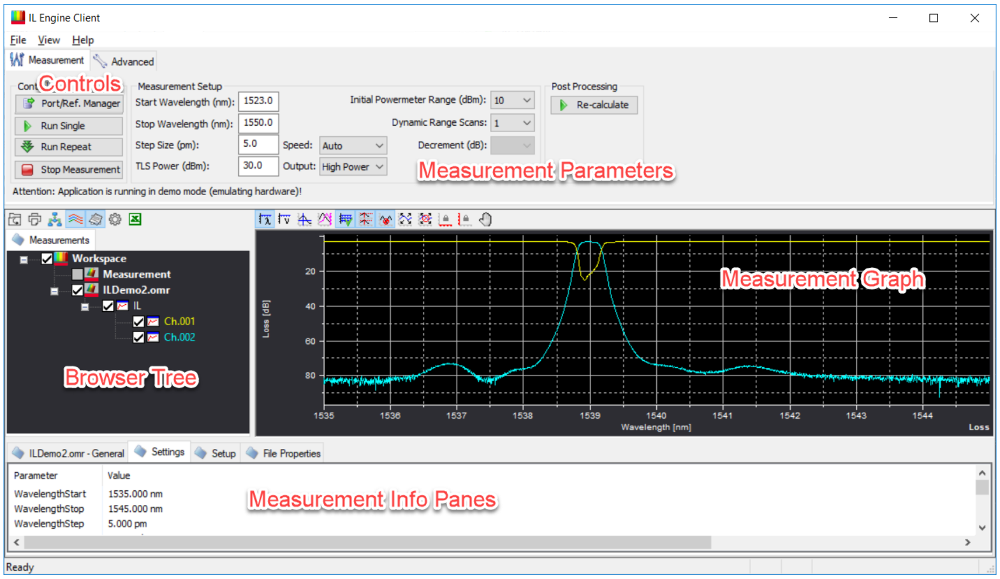
Wavelength-Swept Insertion Loss Application

Use this application for fast wavelength-resolved multi-channel insertion loss measurements. It uses a continuous sweep of the tunable laser source (TLS).

See Measurement Setup on page 40 for details on how to connect the DUT and the trigger cables.

Before starting a measurement you should make sure, that the measurement parameters suit your measurement task.

The user interface of the Insertion Loss engine looks like this:



Start Wavelength (nm): 1523.0
Stop Wavelength (nm): 1550.0
Step Size (pm): 5.0 Speed: Auto
TLS Power (dBm): 30.0 Output: High Power
Initial Powermeter Range (dBm): 10
Dynamic Range Scans: 1
Decrement (dB):

Attention: Application is running in demo mode (emulating hardware)!

Parameter	Value
WavelengthStart	1535.000 nm
WavelengthStop	1545.000 nm
WavelengthStep	5.000 pm

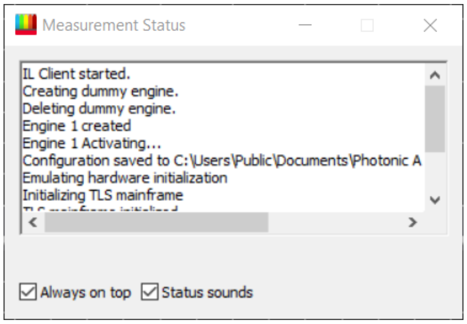
The measurement control buttons are displayed in the top left portion of the application window, the measurement parameters are shown to the right of these and the evaluation parameters are shown even further to the right. Details of those parameters are given in the tables in the next section.

The browser tree to the left of the application window lists the current measurement and all open files and allows to configure, which measurements, which traces (e.g. IL) and which channels are displayed in the measurement view to the right. The measurement information pane can be toggled by clicking the **Show/Hide Info Pane** icon above the browser tree.

For information on how to configure the Measurement File Viewer area, i.e. the browser tree and the display of the measurement traces, refer to the *Photonic Application Suite General User's Guide*.

Information on the current operation progress are displayed in the *Measurement Status* window. Sometimes user inputs are required, e.g. choosing whether to continue or to abort a certain operation. These user inputs are requested by and to be entered in the *Measurement Status* window as well.

Enable the *Status sounds* check box to receive auditory notifications (beeps) when the status of the engine operations change, for example, when a measurement completes or when an error is encountered.



```
IL Client started.
Creating dummy engine.
Deleting dummy engine.
Engine 1 created
Engine 1 Activating...
Configuration saved to C:\Users\Public\Documents\Photonic A
Emulating hardware initialization
Initializing TLS mainframe
```

☑ Always on top ☑ Status sounds

NOTE

If the *Measurement Status* window is closed, it will open automatically, once any user input is required. It can be opened manually by checking *Measurement Status* in the *View* menu.

If the *Measurement Status* window is minimized, it will **not** be restored automatically if a user request is required. In such case, the application may appear locked, although it is just waiting for a user input. If you are waiting for an operation to complete and are unsure about the current progress, make sure the *Measurement Status* window is visible, so you are aware of any user input requests.

If the Measurement Status window is not visible at all, it may have been moved to a screen region that is currently not visible. By choosing *Reset Status Window Position* from the *View* menu, the Measurement Status window will be put to the top left corner of the application window.

Application Setup Parameters

The **Measurement Setup** parameters are explained in the table below.

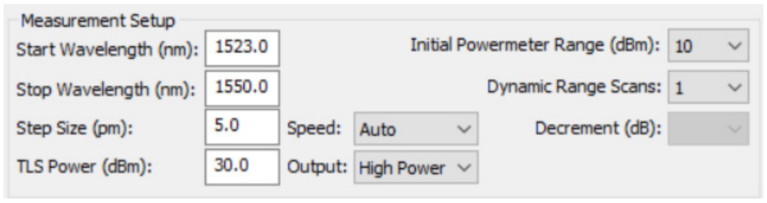
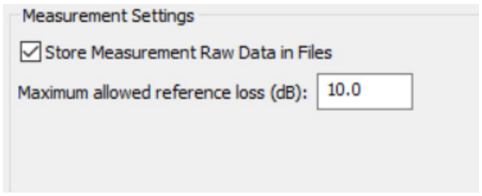

Measurement Setup
Start Wavelength (nm): 1523.0
Stop Wavelength (nm): 1550.0
Step Size (pm): 5.0
TLS Power (dBm): 30.0
Speed: Auto
Output: High Power
Initial Powermeter Range (dBm): 10
Dynamic Range Scans: 1
Decrement (dB):

Figure 6 Insertion Loss Application Setup Parameters

Parameter	Description
Start wavelength (nm):	Defines the start of the wavelength sweep.
Stop wavelength (nm):	Defines the stop of the wavelength sweep.
Step size (pm):	Defines the wavelength step size.

Parameter	Description
Speed:	Defines the sweep speed of the TLS. If set to AUTO, the software automatically chooses a sweep speed, depending upon the chosen step size, the available sweep speeds of the TLS and the averaging times of the incorporated power meters.
TLS Power (dBm):	Defines the optical output power of the laser source.
Output:	If the TLS supports more than one optical output, the output is chosen here. The choices are “High Power”, “Low SSE”, “Both HR” and “Both LR”. When selecting “Both HR”, both outputs are active with the high power output being regulated. When selecting “Both LR”, both outputs are active with the Low SSE output being regulated.
Initial Power meter Range (dBm):	Sets up the power dynamic range of the measurement. The value defines the maximum allowed power. Larger powers will result in clipped results. In case of low power at the power meter (e.g. DUTs with significant loss) the Initial Power meter Range should be reduced to allow for maximum dynamic range.
Dynamic Range Scans:	To achieve high measurement dynamics, up to three wavelength sweeps can be performed at different power ranges. This parameter defines the number of sweeps that are performed. If the Initial Power meter Range is set to a very low value, the number of allowed Dynamic Range Scans is reduced automatically.
Decrement:	To achieve high measurement dynamics, up to three wavelength sweeps can be performed at different power ranges. This parameter defines the amount by which the Initial Power meter Range is decreased during each Dynamic Range Scan. Depending upon the Initial Power meter Range and the number of Dynamic Range Scans, the possible choices for Decrement are reduced automatically.

Some additional parameters are located on the “Advanced” tab of the user interface.



Measurement Settings

☑ Store Measurement Raw Data in Files

Maximum allowed reference loss (dB): 10.0

These parameters are explained in Table 2 on page -27.

Table 2 Insertion Loss Application Advanced Setup Parameters

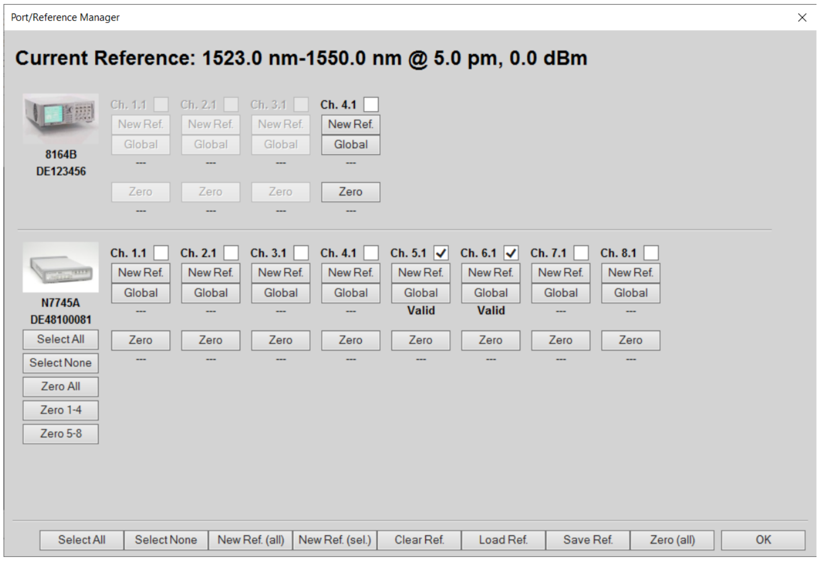
Parameter	Description
Store Measurement Raw Data in Files:	If checked, measurement raw data is stored in saved files, in addition to evaluated data. This allows for later re-evaluation of the file with changed parameters, e.g. a different smoothing bandwidth.
Maximum allowed reference loss (dB):	When performing reference measurements, the average power is compared with the TLS power set in the GUI. If this difference is greater than the maximum allowed reference loss, a warning is shown to the user.

Using the Port/Reference Manager

The Port/Reference Manager is used to configure:

- which channels should be incorporated into subsequent measurements,
- to perform zeroing on individual channels, and
- to handle power references.

You can access the Port/Reference Manager by clicking the **Port/Ref. Manager** button to the top-left.



The Port/Reference Manager lists all mainframes and power meters that are configured to be used by the application. There is a check box for each power meter channel, which is used to tell the application whether to use that channel or not.

NOTE

Keep in mind that different power meters have different capabilities regarding, e.g., averaging time. Therefore, when using different kind of power meters, the application may be unable to perform a measurement using the desired parameters.

Power Meter Zeroing

Each channel has a set of buttons for controlling zeroing and referencing. By clicking the **Zero** button, the corresponding channel is zeroed. By clicking the **Zero (all)** button, all power meter channels are zeroed subsequently. In case of N774xA/C multiport power meters, there are buttons to zero sets of four channels each. The time of the last zeroing operation is displayed below the **Zero** button.

NOTE

Make sure that the power meter detectors are covered during zeroing. The TLS used by the application is automatically turned off during zeroing. For zeroing the multi-port power meters N7744A and N7745A, we recommend using at least one zeroing adapter N7740ZI.

Taking References

You can perform a reference measurement on a certain channel by clicking the corresponding **New Ref.** button. You can do this for any number of channels, one after another. Each channel that has been processed in this way will display the string “Valid”. You can use a single channel as a reference for all channels by clicking the corresponding **Global** button. This channel will then display the string “Valid”, while all others will display “Shared”. You can perform a **Global** reference and afterward perform a number of **New Ref.** measurements on certain channels that are of special interest. You can also perform a reference simultaneously on all channels by clicking the **New ref. (all)** button or only on selected channels by clicking the **New Ref. (sel.)** button.

Information about the current measurement parameters are displayed at the top of the Port/Reference Manager window. You can clear, load and save references by clicking the **Clear Ref**., **Load Ref.** and **Save Ref.** buttons. By saving the reference, not only the last performed channel reference is saved, but all channels that show either a “Shared” or a “Valid” string.

Use the **Select All** and **Select None** buttons to set or remove all check marks simultaneously.

NOTE

Optimum performance is achieved if a reference measurement is performed for each channel subsequently or simultaneously.

NOTE

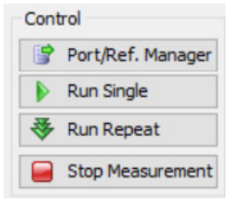
If you change the measurement parameters, e.g. step size, wavelength range, TLS power, the reference is no longer valid. If you do not clear the reference manually, it is done automatically the next time you take a new reference.

NOTE You can only load references that match the current hardware setup.

NOTE When exiting the application, the current reference is stored automatically and loaded automatically, the next time the application is started with the same hardware setup.

NOTE Channels that neither show a “Valid” string nor a “Shared” string are referenced to the TLS power set in the Measurement Parameters section.

Performing a Measurement



Before starting a measurement, perform the following steps:

- Ensure that the Measurement Setup parameters suit your measurement task (see Application Setup Parameters on page 25).
- Check the **Port/Reference Manager** to ensure that all the power meter channels you want to incorporate into the measurement are selected.
- In the Port/Reference Manager you can also perform reference measurements, that subsequent measurements are referenced to. Any channel that has no reference data stored will be referenced to the TLS power set in the measurement parameters. Maximum performance will be achieved if reference data has been stored for each channel.

NOTE **When using reference measurements, any subsequent measurements have to be performed with the same settings that the measurement has been performed with, e.g. wavelength range and step size.**

If you chose Measurement parameters that do not match the current reference parameters, the current parameters are replaced by those of the reference measurement automatically.

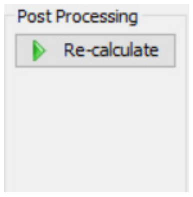
Use the measurement controls as described below:

- **Run Single:** You can perform single measurements by clicking Run Single.
- **Run Repeat:** By clicking Run Repeat the software will perform single measurements in a repeat mode.
- **Stop Measurement:** If measurements are being performed, you can abort the current measurement sweep and stop the repeat operation by clicking Stop Measurement. The operation is stopped, as soon as the current sweep data has been obtained and evaluated.

Re-evaluating Measurements

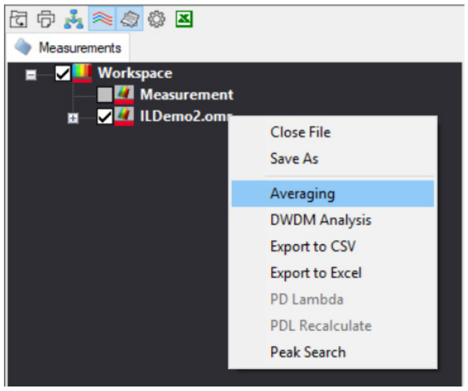
The current measurement can be re-evaluated, after taking a new reference. This is useful, if the DUT measurement has to be performed first and the reference measurement afterwards, e.g. by removing the DUT and splicing together the pigtails.

Click the Re-calculate button to perform a re-evaluation.



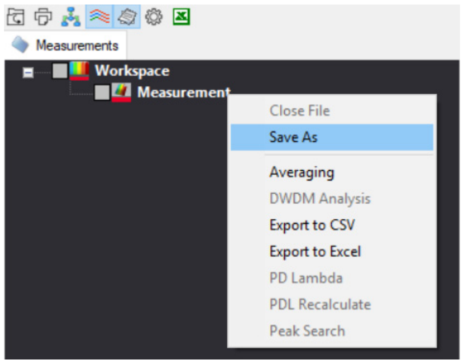
Applying Plugins

You can apply evaluation plugins to the current measurement, e.g. averaging, peak-search or DWDM channel analysis filters. The plugins are listed in the measurement context menu in the browser tree. Which plugins are available depends on the license(s) that are installed on your system. For more information about the available feature packages and licenses, refer to Keysight N7700 Photonic Application Suite - Brochure. Some plugins directly modify the traces of the measurement, some add more traces, some just add information to the table on the bottom of the user interface.



Saving Measurements

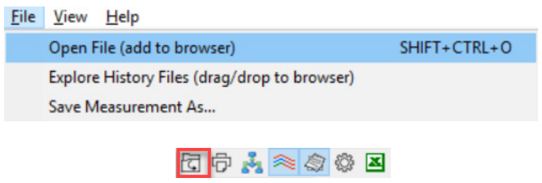
You can save measurement data using the **Save as** option from the measurement context menu in the browser tree and providing a filename.



Alternatively, choose File > Save Measurement As.

Loading Measurements

Saved files can be accessed through the **Open File** dialog from the **File** menu, by clicking the **Open** icon above the browser tree. Open files will be listed in the browser tree to the left of the user interface.


File
View
Help
Open File (add to browser) SHIFT+CTRL+O
Explore History Files (drag/drop to browser)
Save Measurement As...

Creating .agconfig Files from Measurement Files

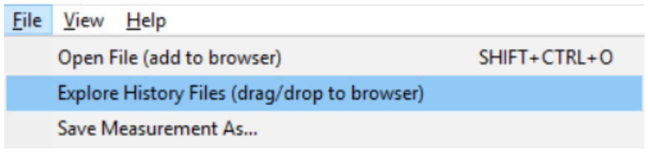
You might require the .agconfig file for a particular measurement that you have performed in the past. In case you did not explicitly save the .agconfig file, you can extract it from a measurement (.omr) file by using ExtractAgconfigFromOmr, a PowerShell utility, available in the Examples folder (C:\Program Files\Keysight\Photonic Application Suite\Examples\DemoPowerShell). This utility takes an OMR file as an input and creates an agconfig file that reflects the settings with which the specified measurement had been performed. The extracted .agconfig file has the same name as the specified OMR file and is created in the same directory as the OMR file. For easy access and execution of this PowerShell utility, use the batch file ExtractAgconfigFromOmr.bat specifying the OMR file as a parameter (either specify the full path of the OMR file or place it in the current working directory).

This utility can also be used for troubleshooting purposes.

Automatic History Saves

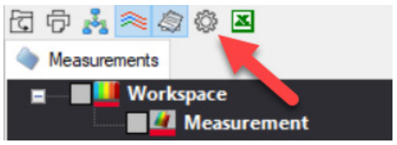
By default, the measurement engine automatically saves a number of recent measurements to a specific folder. This is a very helpful feature, since one often performs a number of measurements that are not intended for saving, then realizes, after a certain, subsequent measurement, that comparison with one of the previous measurements would indeed reveal certain insights.

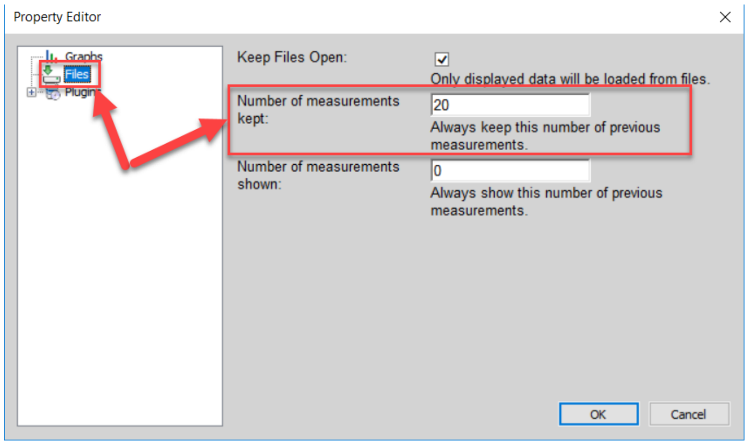
Selecting **Explore History Files** from the **File** menu opens this engine's history file save folder. The files are named using the engine acronym followed by a timestamp. Use drag and drop to add any file(s) to the engine's browser tree or double-click any file(s) for display in the File Viewer application.


File
View
Help
Open File (add to browser) SHIFT+CTRL+O
Explore History Files (drag/drop to browser)
Save Measurement As...

Especially when working with large amounts of measurement data (many ports, broad range, high resolution), you may want to reduce the number of automatically stored files or even turn off the automatic saving of history files. On the other hand, you may want to increase the number for obtaining many measurement files without having to perform manual saves, or having to use measurement automation.

You can do so by modifying the corresponding value in the options:





Keep Files Open: ✓
Only displayed data will be loaded from files.

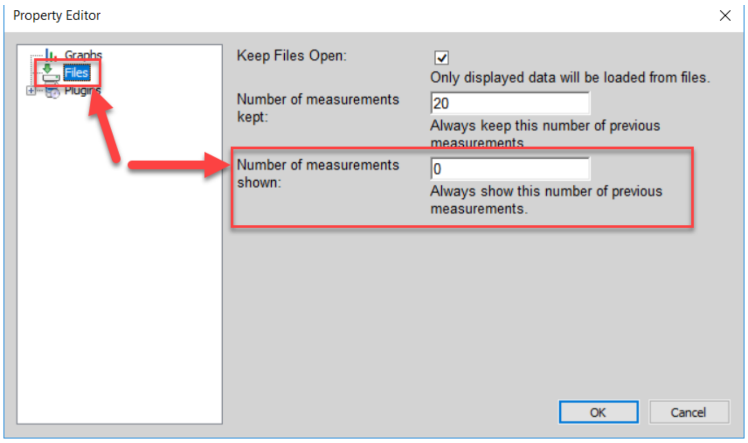
Number of measurements kept: 20
Always keep this number of previous measurements.

Number of measurements shown: 0
Always show this number of previous measurements.

Set the **Number of measurements kept** value to 0 to turn off automatic saves.

When tuning a device or a component, or monitoring stability, it might be desirable to automatically have a certain number of most recent measurements displayed simultaneously.

To do so, set the **Number of measurements shown** value in the options to the desired value or to 0 to disable this feature.



Keep Files Open: ✓
Only displayed data will be loaded from files.

Number of measurements kept: 20
Always keep this number of previous measurements

Number of measurements shown: 0
Always show this number of previous measurements.

NOTE

Number of measurements kept must be larger than or equal to **Number of measurements shown**, for this feature to work as intended.

NOTE

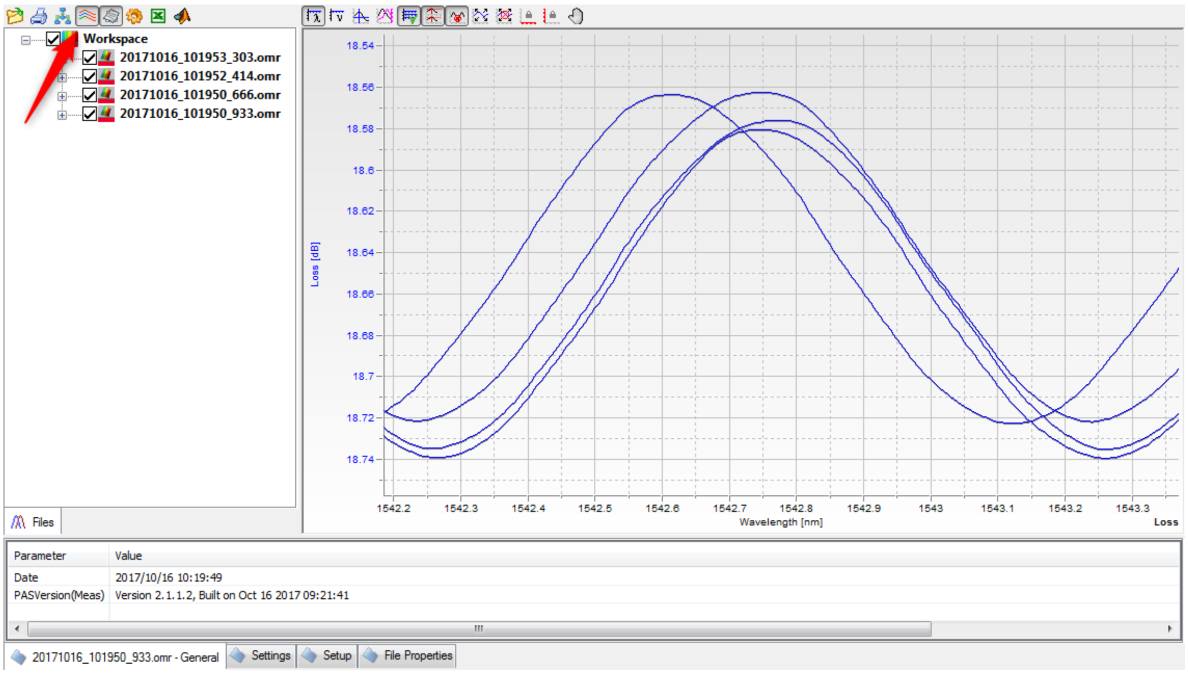
Number of measurements kept and **Number of measurements shown** are global settings used for all Photonic Application Suite engines.

NOTE

The **Number of measurements shown** setting affects measurements performed after starting the engine, after changing this setting, or after closing all the open files. It does not open any older files, so after any of the mentioned operations, the number of automatically shown history files will grow with each new measurement until the total number reaches the **Number of measurements shown** value. For each new measurement after that, the oldest history file will be closed automatically.

By default, trace colors indicate different ports, but by toggling the **Color by Channel/File** icon, this can be changed to indicate different measurements instead. This is usually desired, when using automatic history file display with single-port devices, or devices, where individual ports are clearly distinguishable.

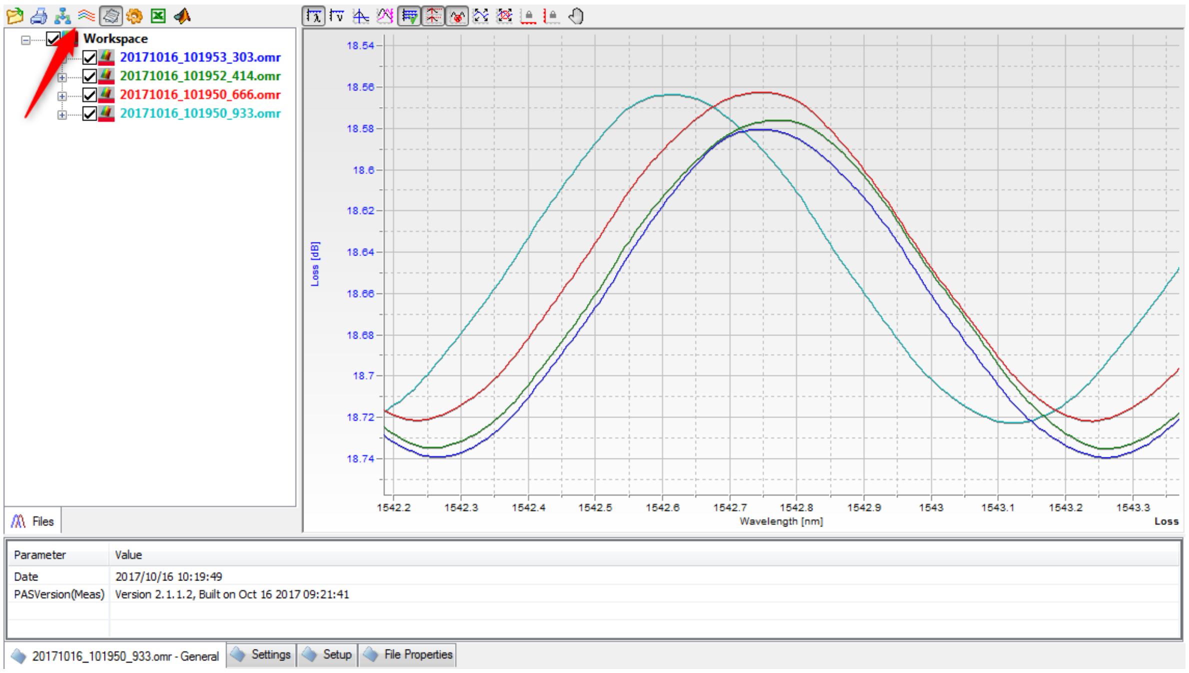
See examples for both modes below.



20171016_101953_303.omr
20171016_101952_414.omr
20171016_101950_666.omr
20171016_101950_933.omr

Parameter	Value
Date	2017/10/16 10:19:49
PASVersion(Meas)	Version 2.1.1.2, Built on Oct 16 2017 09:21:41

20171016_101950_933.omr - General



Parameter	Value
Date	2017/10/16 10:19:49
PASVersion(Meas)	Version 2.1.1.2, Built on Oct 16 2017 09:21:41

When either replacing the DUT, introducing a significant change to DUT tuning parameters, or running a specific path de-embedding measurement, corresponding measurement results may vary significantly from previous history files. To remove all the old measurements from the Workspace at once, click **Close All** from the context menu of the Workspace node in the tree on the left of the GUI.

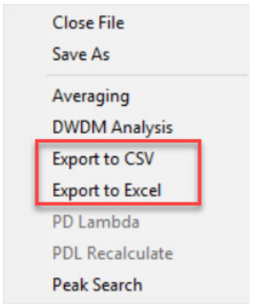
NOTE

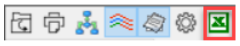
History files are stored in an engine specific subfolder of

C:\Users\<username>\Documents\My Photonic Application Suite History\

Exporting Measurement Data

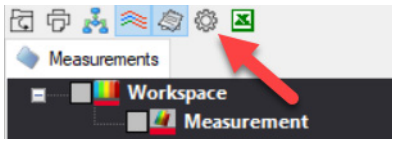
Measurement data can be exported to different applications or file types (e.g. Excel, CSV). You can export the current measurement by selecting the corresponding entry of the measurement context menu in the browser tree or the corresponding icon above the browser tree.

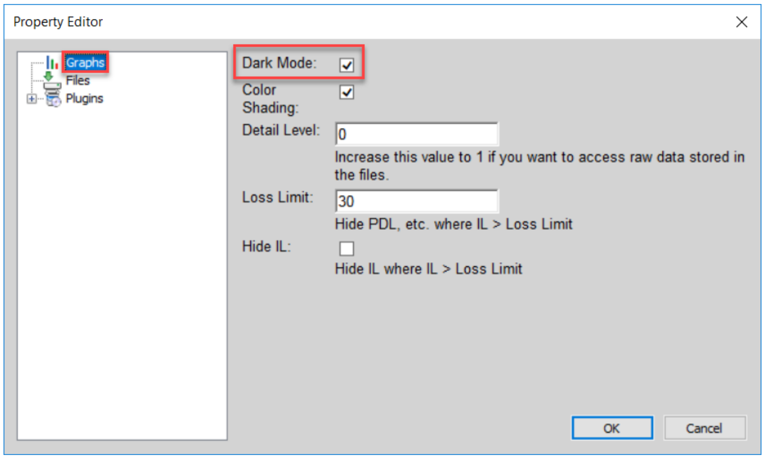




Changing the Color Theme

You can switch to a lighter color theme for the File Viewer within the user interface of Insertion Loss Engine. Click the Settings icon and disable the Dark Mode check box in the Graphs section.

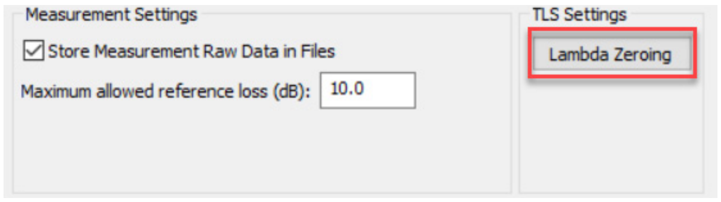



Property Editor

Dark Mode: ✓
Color Shading: ✓
Detail Level: 0
Increase this value to 1 if you want to access raw data stored in the files.
Loss Limit: 30
Hide PDL, etc. where IL > Loss Limit
Hide IL: ☐
Hide IL where IL > Loss Limit

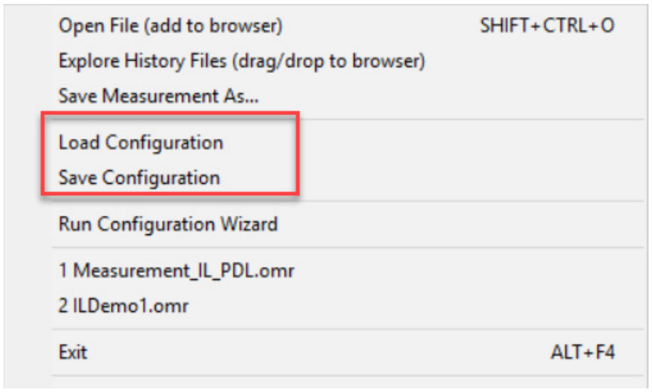
TLS Lambda Zeroing

In order to ensure optimum wavelength sweeping results through varying ambient conditions, a lambda zeroing function is available. Use the **Lambda Zeroing** button on the Advanced tab of the user interface to perform a lambda zeroing operation on the TLS. This helps to avoid missing points at the beginning or end of sweeps, due to an offset starting point, and helps evenly lubricate the full mechanical range.


Measurement Settings
☑ Store Measurement Raw Data in Files
Maximum allowed reference loss (dB): 10.0

Configuration Handling

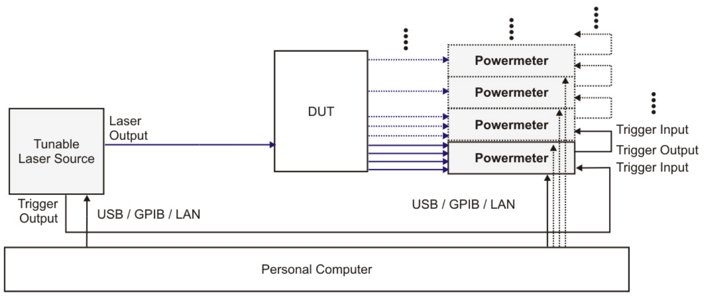
The current configuration, i.e. attached hardware as well as current parameters like wavelength range, can be saved and loaded from the **File** menu by selecting **Save Configuration** and **Load Configuration**. A saved configuration file (*.agconfig) can also be started from the Windows Explorer and will launch the IL engine if it’s not running.



The configuration wizard can be launched from the File menu as well.

Measurement Setup

The general setup is illustrated in the following diagram:



NOTE **Make sure that to perform the software installation steps described in** Software Installation **on page 16 before connecting the instruments.**

- Connect the TLS and all power meters / mainframes to the PC using the GPIB, USB or LAN interface.
- Connect the Trigger Output of the TLS with the Trigger Input of the first power meter (using a 50 Ohms BNC cable).

- Connect the Trigger Output of the first power meter / mainframe with the Trigger Input of the next power meter / mainframe (using a 50 Ohms BNC cable).
- Repeat the previous step for every power meter / mainframe in your setup (using 50 Ohms BNC cables).
- Connect the optical output of the TLS with the optical input of the DUT with a patchcord with appropriate connectors (angled / straight).
- Connect any optical output of the DUT to the optical inputs of the power meters using the corresponding connector interfaces.

4 Automation

COM Components

Automation is implemented using a mechanism called “COM”. COM has been introduced by Microsoft on the Windows platform to allow a unified way to communicate between different software components. Today, almost every programming language such as C#, C++, LabView, Keysight Vee as well as MATLAB offers ways to use these so-called COM components. Once familiar with handling COM components, programming can be done using function browsers or auto-completion that show you the available properties and functions/methods including their parameter names. It is thus often possible to use the available functions without consulting the documentation.

Since the syntax of calling COM components is a little different in every programming language, we focus in our examples on MATLAB code. MATLAB has a very generic syntax and it should be easy to adapt the code to any other language. Further examples are included in the software distribution. These examples get installed with the Photonic Application Suite and can usually be found at the following location:

C:\Program Files\Keysight\Photonic Application Suite\Examples

Here is a simple example which starts a measurement (similar to clicking the “Run Single” button):

```
% Connect to Engine Manager
EngineMgr=actxserver('AgServerIL.EngineMgr');

% List all Engines currently running
EngineIDs=EngineMgr.EngineIDs;

% Always connect to first engine
Engine=EngineMgr.OpenEngine(EngineIDs(1));

% Start measurement
Engine.StartMeasurement;

% Release the engine and the engine manager
Engine.release;
EngineMgr.release;
```

This example assumes that you have already started the Client Software and thus a server is already running. Therefore, this example connects to an existing engine.

The next section explains how to create a new instance of the engine, in case the server has not yet been started.

Creating a New Engine

When writing your own software, it is very useful to start the Client Software first. Starting the Client Software will start the Server Software and creates an instance of the “Engine” COM component which handles the hardware communication. If your self-written software connects to this engine, you can see changes immediately in the client user interface. This is extremely useful in the debugging phase.

Finalizing your software, you probably don’t need the user interface anymore. In that case, you can start the COM server yourself and create an instance of the engine:

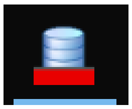
```
% Connect to Engine Manager
EngineMgr=actxserver('AgServerIL.EngineMgr');

% Create a new engine
Engine=EngineMgr.NewEngine;

% Load configuration file
Engine.LoadConfiguration(’c:\test.agconfig’);

% Activate engine
Engine.Activate;

% Start the measurement
Engine.StartMeasurement;

% Deactivate engine
Engine.DeActivate;

% Release the engine and the engine manager
Engine.release;
EngineMgr.release;
```

Note that you have to activate/deactivate the engine explicitly. If connecting to an existing engine, this is done by the client. When activating the engine, communication with the hardware is started.

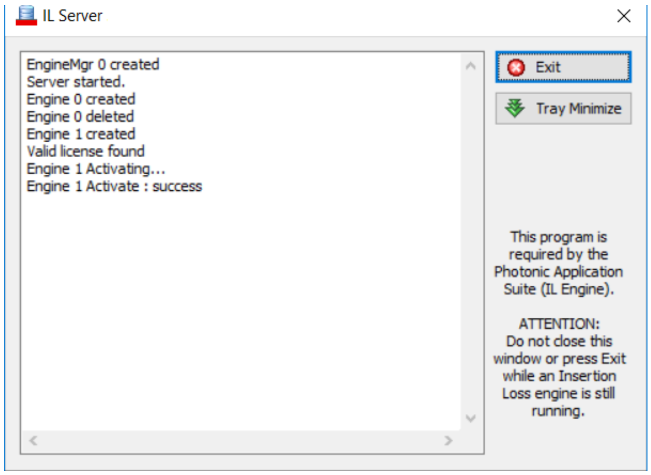
We recommend to configure your engine initially using the Client Software and store the configuration using the “Save Configuration” menu. This example uses the “LoadConfiguration” command to load an .agconfig-file which has been stored before by the client.

NOTE

An icon appears in the Windows icon tray (on the bottom right of your screen) when the server has started (i.e. the engine manager has been invoked):



Left-click on the icon to bring the server window to the front. The server will display a protocol line when an engine has been created or deleted:



```
EngineMgr 0 created
Server started.
Engine 0 created
Engine 0 deleted
Engine 1 created
Valid license found
Engine 1 Activating...
Engine 1 Activate : success
```

This program is required by the Photonic Application Suite (IL Engine).

ATTENTION:
Do not close this window or press Exit while an Insertion Loss engine is still running.

Connecting to an Existing Engine

Use the following code to connect to an existing engine:

```
% Connect to Engine Manager
EngineMgr=actxserver('AgServerIL.EngineMgr');

% List all Engines currently running
EngineIDs=EngineMgr.EngineIDs;

% Always connect to first engine
Engine=EngineMgr.OpenEngine(EngineIDs(1));

% Start the measurement
Engine.StartMeasurement;

% Release the engine and the engine manager
Engine.release;
EngineMgr.release;
```

NOTE In most of the following MATLAB examples, we assume that the engine has been created and activated before.

Performing a Measurement

The following code starts a measurement and waits for the measurement to be finished. The result is stored as an OMR file.

```
% Start
Engine.StartMeasurement;

% Wait for measurement to be finished
while Engine.Busy; pause(1); end;

% Get result object
MeasurementResult = Engine.MeasurementResult;

% Save as OMR file
MeasurementResult.Write('c:\test.omr');

% Release measurement object
MeasurementResult.release;
```

Accessing the Measurement Result

The measurement result is returned as a reference to an OMR file object. This file COM object represents an OMR file and wraps all methods to read and write these files (Interface: IOMRFile).

NOTE **Refer to the *N7700 Photonic Application Suite General User's Guide* for a reference of the common interfaces, e.g. IOMRFile, IOMRGraph and IOMRProperty.**

Once the measurement has finished, you can request a reference to this object and manipulate it. Typically you want to store the file somewhere on your hard disk or you want to access the graphs.

Saving the file can be done using the Write-method:

```
% Save as OMR file
MeasurementResult.Write('c:\test.omr');
```

Accessing the graph data of the spectral loss measurement is done like this:

```
% Access IL-graph
Graph = MeasurementResult.Graph('RXTXAvgIL');
noChannels      = Graph.noChannels;
dataPerCurve    = Graph.dataPerCurve;
YData = reshape(Graph.YData,dataPerCurve,noChannels);
xStart = Graph.xStart;
xStep = Graph.xStep;
```

The graphs in the file have names. The name “RXTXAvgIL” corresponds to the Loss measurement data.

The data are returned as a one-dimensional array, even if it contains more than one channel. The data for each channel are concatenated. The array starts with the first channel and continues with the next channels. The MATLAB command “reshape” converts this to a 2-dimensional array, a matrix. You can use the properties “noChannels” to get the total number of contained channels and the property “dataPerCurve” to find out how many values belong to a single channel.

NOTE

Using automation commands it is possible to start a new measurement engine as well as connect to an existing one, for example, one started from the Keysight Client GUI.

Certain automation operations, such as the Write method of the OMRFile interface or the FileSave method of the IEngine interface, lock the measurement object while accessing it. This might cause temporary failures in automation operations that are triggered from other instances connected to the same N7700 engine server.

This might especially be the case when using engine automation in addition to running the Keysight Client, which locks the OMRFile object in memory while updating measurement data in the GUI right after a measurement and when performing any automatic history save (see Automatic History Saves on page 33 for information on how to disable automatic saves).

To deal with this behavior, either implement retries or delays on affected operations or avoid connecting multiple instances to the same measurement engine server.

Reference: Interface “IEngineMgr”

The IEngineMgr interface is invoked using the following PROGID to identify the COM server:

```
AgServerIL.EngineMgr
```

MATLAB:

```
EngineMgr=actxserver('AgServerIL.EngineMgr');
```

property Version

Type-Library:

```
get: HRESULT Version([out, retval] BSTR* pVal)
```

MATLAB:

```
Version=EngineMgr.Version;
```

Shows information about revision and build date of the engine.

method IsVersionGreaterOrEqual

Type-Library:

```
HRESULT IsVersionGreaterOrEqual([in] BSTR
VersionNumber,[out,retval] BOOL* pVal);
```

MATLAB:

```
Engine=EngineMgr.IsVersionGreaterOrEqual(’1.1.0.1’);
```

Performs a check, whether the engine revision is greater than or equal to the revision number provided as a parameter.

method NewEngine

Type-Library:

```
HRESULT NewEngine([out,retval] IEngine** aEngine);
```

MATLAB:

```
Engine=EngineMgr.NewEngine;
```

Tells the engine manager to create a new engine and returns the corresponding handle.

method OpenEngine

Type-Library:

```
HRESULT OpenEngine([in] LONG EngineID,[out,retval] IEngine**
aEngine);
```

MATLAB:

```
Engine=EngineMgr.OpenEngine(EngineID);
```

Returns the handle to an existing engine. The engine ID is a unique identifier which can be obtained by requesting the property “EngineIDs”.

property EngineIDs

Type-Library:

```
get: HRESULT EngineIDs([out, retval] SAFEARRAY(LONG)* pVal)
```

MATLAB:

```
EngineIDs=EngineMgr.EngineIDs;
```

Returns the IDs of all created engines.

method DeleteEngine

Type-Library:

```
HRESULT DeleteEngine([in] IEngine* aEngine);
```

MATLAB:

```
EngineMgr.DeleteEngine(Engine);
```

To ensure that all memory cleanup operations are executed properly, it is recommended to call the EngineMgr's DeleteEngine method with the current engine object as an argument after running the engine's DeActivate method and before releasing the engine object. After that, either create a new engine or release the EngineMgr object.

Reference: Interface “IEngine”

The IEngine interface is invoked either using the method “NewEngine” or “OpenEngine” of the IEngineMgr interface:

MATLAB:

```
Engine=EngineMgr.NewEngine;
```

property Version

Type-Library:

```
get: HRESULT Version([out, retval] BSTR* pVal)
```

MATLAB:

```
Version=Engine.Version;
```

Shows information about revision and build date of the engine.

method IsVersionGreaterOrEqual

Type-Library:

```
HRESULT IsVersionGreaterOrEqual([in] BSTR
VersionNumber,[out,retval] BOOL* pVal);
```

MATLAB:

```
Engine=Engine.IsVersionGreaterOrEqual(’1.1.0.1’);
```

Performs a check, whether the engine revision is greater than or equal to the revision number provided as a parameter.

method Activate

Type-Library:

```
HRESULT Activate(void);
```

MATLAB:

```
Engine.Activate;
```

Activates the engine. Usually, a configuration file, created using the engine’s client software, should be loaded prior to activating the engine. This is done using the “LoadConfiguration” method. Activating an engine will cause the server to communicate with the instruments.

method DeActivate

Type-Library:

```
HRESULT DeActivate(void);
```

MATLAB:

```
Engine.DeActivate;
```

Deactivates the engine. You should deactivate the engine before loading a different configuration.

NOTE After deactivating an engine and before releasing the engine object and its corresponding EngineMgr object, the EngineMgr's DeleteEngine method should be called. Refer to method DeleteEngine on page 53 for further details.

property Active

Type-Library:

```
get: HRESULT Active([out, retval] BOOL* pVal);
```

MATLAB:

```
x = Engine.Active;
```

Returns 0/1 if the engine is inactive/active.

property Busy

Type-Library:

```
get: HRESULT Busy([out, retval] BOOL* pVal);
```

MATLAB:

```
x = Engine.Busy;
```

Returns 0/1 if the engine is not-busy/busy. The engine might be busy when transferring large amounts of data to or from the instruments or when it's waiting for a measurement operation to complete.

property EmulationMode

Type-Library:

```
get: HRESULT EmulationMode([out, retval] BOOL* pVal);

put: HRESULT EmulationMode([in] BOOL Val);
```

MATLAB:

```
b = Engine.EmulationMode;
```

```
Engine.EmulationMode = b;
```

Sets or gets the emulation (demo) mode setting. If operated in demo mode, no actual communication with instruments will be performed and simulated measurement results will be shown instead.

method LoadConfiguration

Type-Library:

```
HRESULT LoadConfiguration([in] BSTR Filename);
```

MATLAB:

```
Engine.LoadConfiguration('c:\\test.agconfig');
```

Loads a configuration file (extension: .agconfig). The engine should be inactive while loading a new config file, i.e., either before calling Activate for the first time after the engine has been created or after calling Deactivate.

property Configuration

Type-Library:

```
get: HRESULT Configuration([out, retval] BSTR* ConfigXML);
```

```
put: HRESULT Configuration([in] BSTR ConfigXML);
```

MATLAB:

```
xml = Engine.Configuration;
```

```
Engine.Configuration = xml;
```

Transfers the contents of an .agconfig-file to or from the engine. An .agconfig-file contains XML formatted text.

property SoftwareConfiguration

Type-Library:

```
put: HRESULT SoftwareConfiguration([in] BSTR ConfigXML);
```

MATLAB:

```
Engine.SoftwareConfiguration = xml;
```

Transfers only the software section contents of an .agconfig-file to the engine. An .agconfig-file contains XML formatted text. This can be used to change all measurements parameters at once, without changing any hardware settings or having to reactivate the engine.

property WavelengthStart

Type-Library:

```
get: HRESULT WavelengthStart([out, retval] DOUBLE* pVal);
```

```
put: HRESULT WavelengthStart([in] DOUBLE Val);
```

MATLAB:

```
d = Engine.WavelengthStart;
```

```
Engine.WavelengthStart = d;
```

Sets or gets the start wavelength in m.

property WavelengthStop

Type-Library:

```
get: HRESULT WavelengthStop([out, retval] DOUBLE* pVal);
```

```
put: HRESULT WavelengthStop([in] DOUBLE Val);
```

MATLAB:

```
d = Engine.WavelengthStop;
```

```
Engine.WavelengthStop = d;
```

Sets or gets the stop wavelength in m.

property WavelengthStep

Type-Library:

```
get: HRESULT WavelengthStep([out, retval] DOUBLE* pVal);
```

```
put: HRESULT WavelengthStep([in] DOUBLE Val);
```

MATLAB:

```
d = Engine.WavelengthStep;
```

```
Engine.WavelengthStep = d;
```

Sets or gets the wavelength step in m. The TLS will perform continuous sweeps, but the measurement results will be returned at wavelengths defined by WavelengthStart, WavelengthStop and WavelengthStep.

property SweepRates

Type-Library:

```
get: HRESULT SweepRates([out, retval] SAFEARRAY(double)*
pVal);
```

MATLAB:

```
dArray = Engine.SweepRates;
```

Returns a double array containing the sweep rates in nm/s that the TLS can sweep at.

property SweepRate

Type-Library:

```
get: HRESULT SweepRate([out, retval] DOUBLE* pVal);
```

```
put: HRESULT SweepRate([in] DOUBLE Val);
```

MATLAB:

```
d = Engine.SweepRate;
```

```
Engine.SweepRate = d;
```

Sets or gets the sweep rate in nm/s. When set to -1, the engine will automatically choose an appropriate sweep rate.

property PWMRanges

Type-Library:

```
get: HRESULT PWMRanges([out, retval] SAFEARRAY(double)*
pVal);
```

MATLAB:

```
dArray = Engine.PWMRanges;
```

Returns a double array containing the available power meter ranges.

property PWMSensitivity

Type-Library:

```
get: HRESULT PWMSensitivity([out, retval] DOUBLE* pVal);
```

```
put: HRESULT PWMSensitivity([in] DOUBLE Val);
```

MATLAB:

```
d = Engine.PWMSensitivity;
```

```
Engine.PWMSensitivity = d;
```

Sets or gets the power meter sensitivity for the DUT sweep in dBm. This property has to be set in multiples of 10dBm. Use property PWMRanges on page 58 to get a list of available ranges for the current setup.

The parameter set “property PWMSensitivity, “property NumberOfScans and “property RangeDecrement must match the supported maximum and minimum power ranges of all attached power meters.

method ZeroPWMChannels

Type-Library:

```
HRESULT ZeroPWMChannels([in] SAFEARRAY(LONG) Ports);
```

MATLAB:

```
Engine.ZeroPWMChannels(int32([3;6]));
```

Performs a zeroing operation on the selected power meter ports. As a parameter an integer array of port numbers (starting at 0 for port 1) must be provided. There will be a subsequent user input request to make sure that the corresponding ports are covered.

NOTE

When using this method from MATLAB, you need to pass a column vector.

Also, due to the way the parameter is passed from MATLAB to the engine, it is not possible to pass single-element arrays. Pass a line array instead, with a dummy value as a second element, e.g., Engine.MethodName(int32([3 3])).

method ZeroTLS

Type-Library:

```
HRESULT ZeroTLS(void);
```

MATLAB:

```
Engine.ZeroTLS;
```

Performs a lambda zeroing operation on the tunable laser source in use.

property TLSOutputPort

Type-Library:

```
get: HRESULT TLSOutputPort([out, retval] LONG* pVal);
```

```
put: HRESULT TLSOutputPort([in] LONG Val);
```

MATLAB:

```
i = Engine.TLSOutputPort;
```

```
Engine.TLSOutputPort = i;
```

Sets or gets the output port to be used for the TLS. This integer value refers to the available Output port configuration list obtained with `OutputPorts`.

property TLSPower

Type-Library:

```
get: HRESULT TLSPower([out, retval] DOUBLE* pVal);
```

```
put: HRESULT TLSPower([in] DOUBLE Val);
```

MATLAB:

```
d = Engine.TLSPower;
```

```
Engine.TLSPower = d;
```

Sets or gets the laser power in W.

property NumberOfScans

Type-Library:

```
get: HRESULT NumberOfScans([out, retval] LONG* pVal);
```

```
put: HRESULT NumberOfScans([in] LONG Val);
```

MATLAB:

```
i = Engine.NumberOfScans;
```

```
Engine.NumberOfScans = i;
```

Sets or gets the number of TLS scans used for the (stitched) measurement. This parameter is zero-based, so set 0/1/2 for 1/2/3 scans. Each measurement will be performed with a reduced range (see property RangeDecrement on page 61)

The parameter set “property PWMSensitivity, “property NumberOfScans and “property RangeDecrement must match the supported maximum and minimum power ranges of all attached power meters.

property RangeDecrement

Type-Library:

```
get: HRESULT RangeDecrement([out, retval] LONG* pVal);
```

```
put: HRESULT RangeDecrement([in] LONG Val);
```

MATLAB:

```
i = Engine.RangeDecrement;
```

```
Engine.RangeDecrement = i;
```

Sets or gets the range decrement in dB used for the (stitched) measurement (see property NumberOfScans on page 60). This parameter needs to be multiples of 10dB.

The parameter set “property PWMSensitivity, “property NumberOfScans and “property RangeDecrement must match the supported maximum and minimum power ranges of all attached power meters.

method StartReference

Type-Library:

```
HRESULT StartReference([in] SAFEARRAY(LONG) Ports);
```

MATLAB:

```
Engine.StartReference(int32([3;6]));
```

Starts a reference measurement. As a parameter an integer array of port numbers (starting at 0 for port 1) must be provided. A reference sweep will be performed and reference data for all provided ports will be stored.

NOTE

When using this method from MATLAB, you need to pass a column vector.

Also, due to the way the parameter is passed from MATLAB to the engine, it is not possible to pass single-element arrays. Pass a line array instead, with a dummy value as a second element, e.g., Engine.MethodName(int32([3 3])).

method StartReferenceGlobal

Type-Library:

```
HRESULT StartReferenceGlobal([in] LONG Port);
```

MATLAB:

```
Engine.StartReferenceGlobal(0);
```

Starts a reference measurement on a specific port, using that reference as a global reference for all ports.

method ClearReference

Type-Library:

```
HRESULT ClearReference(void);
```

MATLAB:

```
Engine.ClearReference();
```

Clears reference data of the currently loaded reference file. The saved file will not be changed by this operation. To update the saved reference file, the modified reference needs to be saved with the same filename.

method LoadReference

Type-Library:

```
HRESULT LoadReference([in] BSTR Filename);
```

MATLAB:

```
Engine.LoadReference('Reference.omr');
```

Loads a reference file in OMR format for the current TLS.

method SaveReference

Type-Library:

```
HRESULT SaveReference([in] BSTR Filename);
```

MATLAB:

```
Engine.SaveReference('Reference.omr');
```

Saves the current reference to an omr file.

Note that the file will be stored on the computer running the engine server, not the client.

property MaxAllowedRefLoss

Type-Library:

```
get: HRESULT MaxAllowedRefLoss([out, retval] DOUBLE* pVal);
```

```
put: HRESULT MaxAllowedRefLoss([in] DOUBLE Val);
```

MATLAB:

```
d = Engine.MaxAllowedRefLoss;
```

```
Engine.MaxAllowedRefLoss = d;
```

Sets or gets the maximum allowed reference loss in dB. If the minimum difference between the set TLS power and the measured power at the used power / current meter channel(s) during a reference sweep is larger than this value, the engine will display a warning. To avoid warnings, increase this value, if you are using additional components in the reference measurement path that contribute to the overall loss.

method ValidateSettings

Type-Library:

```
HRESULT ValidateSettings(void);
```

MATLAB:

```
Engine.ValidateSettings;
```

Checks measurement parameters for validity and prompts for adjustments, if required, e.g. wavelength range exceeding the sweep range of the TLS used. Will also check whether the parameters match the currently selected reference.

method ValidateSettingsNoRefCheck

Type-Library:

```
HRESULT ValidateSettingsnoRefCheck(void);
```

MATLAB:

```
Engine.ValidateSettingsnoRefCheck;
```

Checks measurement parameters for validity and prompts for adjustments, if required, e.g. wavelength range exceeding the sweep range of the TLS used. Will not check whether the parameters match the current reference. Can be used before taking a new reference with changed parameters.

property ChannelSetup

Type-Library:

```
get: HRESULT ChannelSetup([out, retval] BSTR* ChannelXML);
```

MATLAB:

```
xml = Engine.ChannelSetup;
```

Contains additional information for internal use.

method StartMeasurement

Type-Library:

```
HRESULT StartMeasurement(void);
```

MATLAB:

```
Engine.StartMeasurement;
```

Starts a measurement.

method StartMeasurementRepeat

Type-Library:

```
HRESULT StartMeasurementRepeat(void);
```

MATLAB:

```
Engine.StartMeasurementRepeat;
```

Starts a series of measurements. Will stop after StopMeasurement method has been called. Will not save any measurement data automatically, except for any automatic history saves that might have been configured in an engine client (IL Engine GUI), connected to the same engine server.

method StopMeasurement

Type-Library:

```
HRESULT StopMeasurement(void);
```

MATLAB:

```
Engine.StopMeasurement;
```

Stops a measurement at the next possible time. This will, most of the times, mean between measurements, such as in repeat measurement mode.

method FileSave

Type-Library:

```
HRESULT FileSave([in] BSTR Filename);
```

MATLAB:

```
Engine.FileSave('test.omr');
```

Saves the measurement result to an OMR-file at the location provided as parameter.

See the last Note in Accessing the Measurement Result on page 50.

property MeasurementResult

Type-Library:

```
HRESULT MeasurementResult([out, retval] IOMRFile** pVal);
```

MATLAB:

```
Result = Engine.MeasurementResult;
```

Returns a reference to the underlying IOMRFile object which contains the measurement result.

NOTE Refer to the *N7700 Photonic Application Suite General User's Guide* for a reference of the common interfaces, e.g. IOMRFile, IOMRGraph and IOMRProperty.

See the last Note in Accessing the Measurement Result on page 50.

property EventPropertiesChanged

Type-Library:

```
get: HRESULT EventPropertiesChanged([out, retval] LONG*
pVal);
```

MATLAB:

```
nEvent = Engine.EventPropertiesChanged;
```

This property is a 32 bit integer value. The value will be increased whenever the state of the engine has changed. By polling this value and observing the changes, a client can be notified of any change of the engine state.

property EventMeasurementFinished

Type-Library:

```
get: HRESULT EventMeasurementFinished([out, retval] LONG*
pVal);
```

MATLAB:

```
nEvent = Engine.EventMeasurementFinished;
```

This property is a 32 bit integer value. The value will be increased whenever the engine has finished a measurement. By polling this value and observing the changes, a client can be notified of any change of the engine state.

property EventReferenceZeroFinished

Type-Library:

```
get: HRESULT EventRefZeroFinished([out, retval] LONG* pVal);
```

MATLAB:

```
nEvent = Engine.EventRefZeroFinished;
```

This property is a 32 bit integer value. The value will be increased whenever the engine has finished a reference or zeroing operation, e.g. performing a reference sweep, deleting the reference data for certain ports, etc. By polling this value and observing the changes, a client can be notified of any change of the engine state.

property ProtocolText

Type-Library:

```
get: HRESULT ProtocolText([out, retval] BSTR* pVal)
```

MATLAB:

```
ProtocolText = Engine.ProtocolText;
```

Returns a string containing all engine messages separated by a '|' character. If the number of messages exceeds 1000, the oldest message is deleted from the list for each new message.

property ProtocolMin

Type-Library:

```
get: HRESULT ProtocolMin([out, retval] LONG* pVal)
```

MATLAB:

```
First = Engine.ProtocolMin;
```

Returns the index to the first entry in the message list.

property ProtocolMax

Type-Library:

```
get: HRESULT ProtocolMax([out, retval] LONG* pVal)
```

MATLAB:

```
Last = Engine.ProtocolMax;
```

Returns the index to the most recent entry in the message list.

method GetProtocolTextAt

Type-Library:

```
HRESULT GetProtocolTextAt([in] LONG nMin,[in] LONG nMax,
[out,retval] BSTR* pVal);
```

MATLAB:

```
s = Engine.GetProtocolTextAt(Engine.ProtocolMin,
Engine.ProtocolMax);
```

Returns a portion of the message history defined by nMin und nMax.

property UserInputWaiting

Type-Library:

```
get: HRESULT UserInputWaiting([out, retval] BOOL* pVal);
```

```
put: HRESULT UserInputWaiting([in] BOOL Val);
```

MATLAB:

```
b = Engine.UserInputWaiting;
```

```
Engine.UserInputWaiting = b;
```

Returns 1 if the engine is waiting for a user input, e.g. to choose between different options or to acknowledge an error message. Returns 0 otherwise. Set to 0 after you have handled the user input (see method UserInputResponse on page 68)

property UserInputPrompt

Type-Library:

```
get: HRESULT UserInputPrompt([out, retval] BSTR* pVal)
```

MATLAB:

```
Prompt = Engine.UserInputPrompt;
```

Returns a string containing the prompt of the current user input request, if any.

property UserInputChoice

Type-Library:

```
get: HRESULT UserInputChoice([out, retval] BSTR* pVal)
```

MATLAB:

```
Choices = Engine.UserInputChoice;
```

Returns a string containing all possible responses to the user input requests separated by a '|' character. Each response contains a corresponding number to be used by `Engine.UserInputResponse` and the description of the response, e.g. for labelling a button. Number and description are separated by a ',' character.

method UserInputResponse

Type-Library:

```
HRESULT UserInputResponse([in] LONG nResponse);
```

MATLAB:

```
Engine.UserInputResponse(1);
```

Set the response to a user input request. Must be a number from the list obtained by `Engine.UserInputChoice`. Set `Engine.UserInputWaiting` to 0 after setting `Engine.UserInputResponse` for the engine to continue operation.

Alphabetical Automation Index

5 Troubleshooting

Symptoms and Solutions

The list of values for a drop-down control appears invisible or displaced

Possible Reason	Solution
Different scaling factors for the different screens in some multi-screen setups.	Ensure that the scaling factor for all the screens is consistent.
	Consider using a single screen setup, define another screen as the default screen, or start the engine from an Explorer window on the screen that the engine GUI is intended to show on.
The engine window is maximized.	To make the invisible list of values appear on the screen, restore the window to its normal size ensuring that it is not too close to the top left edge of the screen. Then set the required value in the drop-down control. Note that the list of values will still appear displaced.
	If the list of values is visible (although displaced), select the drop-down list and use the cursor (arrow) and Enter keys to navigate to and select the required value.

COM API registration fails

Possible Reason	Solution
MATLAB Compiler Runtime (MCR) has not been installed prior to measurement engine installation.	Register the COM API after later MCR installation or re-register for any other reasons, by running the **Re-register all N7700 COM APIs** program from the Start menu.

NOTE

A troubleshooting utility called **Get N7700 System Information** has been introduced to help Keysight technical support gather some vital details required to resolve problems with the software at the user end, engine startup problems, for instance. This utility is accessible from the Start menu and should be run to capture information that can be shared with Keysight technical support.

N7700-91002