Application Programming Interface for the Keysight N5991 Test Automation Software Platform - User Guide

KEYSIGHT

USER GUIDE

Notices

Manual Part Number

N5991-91012

Edition

Edition 2.0, January 2024

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.

Contents

1 Introduction

Overview

This guide provides a detailed description of the Application Programming Interface (API) available for the ValiFrame Test Automation Software Platform, which is globally marketed and supported by Keysight Technologies as N5991.

NOTE The definitions of the acronyms and abbreviations used throughout this User Guide are given in Chapter 3, Appendix: Acronyms and Abbreviations.

Document History

First Edition (June 2022)

The first edition of this user guide describes the functionality of the Application Programming Interface for the Keysight N5991 Test Automation Software Platform.

Second Edition (January 2024)

The second edition of this user guide describes the functionality of the Application Programming Interface for the Keysight N5991 Test Automation Software Platform as of December 2023.

Support

The Keysight support team is happy to help you should you experience difficulties using the API or if you require further information.

For support options visit www.keysight.com/find/contactus.

2 Application Programming Interface

The Application Programming Interface enables an additional level of automation to be added, above that provided by the Keysight Technologies N5991 Test Automation Software Platform.

Overview

The Keysight Technologies N5991 Test Automation Software Platform (ValiFrame) provides an application programming interface (API). This may be of use when an extra level of automation is required above that supplied by the Keysight Technologies N5991 Test Automation Software Platform, for example, if a set of tests is to be repeated several times at different temperatures or with other different parameters.

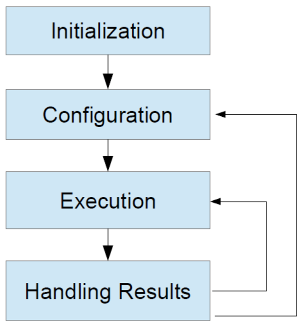


Figure 1 Typical ValiFrame flow diagram

This document describes the functions of the application interface and shows how to use them for a typical application. The flow of use is always the same. First, the application needs to be selected and initialized, then the system needs to be configured, then the calibrations and tests can be carried out, and finally, the results need to be processed, as shown in Figure 1.

In this chapter, the use of the application interface is explained and example code written in Python is given. This procedure is script based.

```
Example code is highlighted by a gray background.
```

Initialization

public ValiFrameRemote(ProductGroupE productGroup)

To get a reference to the ValiFrameRemote control, call the ValiFrameRemote( ) constructor.

```
#Set ValiFrameDllDirectory to the  path of your application
ValiFrameDllDirectory = r'C:\Program Files\BitifEye\ValiFrameK1\USB4\TestAutomation'

import sys
import platform
import string
import clr

if ValiFrameDllDirectory != None:
    sys.path.append(ValiFrameDllDirectory)
clr.AddReference(r'ValiFrameRemote')
clr.AddReference(r'VFBase')
import BitifEye.ValiFrame.ValiFrameRemote as vfRemote
import BitifEye.ValiFrame.Logging as vfLog
import System

try:

valiFrame = vfRemote.ValiFrameRemote(ProductGroupE.ValiFrameK1)

except Exception as e:
    print('EXCEPTION: %s' % str(e))
```

ValiFrameRemote is the class that provides the entire functionality to control ValiFrame. In the following, the functions are explained in the order they are called in a typical application.

First, the required application must connect to the instruments and initialize them.

public string[ ] GetApplications( )

```
applicationNames = valiFrame.GetApplications()
```

'GetApplications( )' gets a list of available applications.

public bool InitApplication(string application)

```
valiFrame.InitApplication(“USB4”)
```

One of the application names obtained with ‘GetApplications( )’ can then be used in ‘InitApplication( )’ to select the application (or “standard”, e.g. USB4, PCIe) to be used for the test.

public void SetConfigurationFileName(string filename)

```
valiFrame.SetConfigurationFileName(filename)
```

Then the configuration file of the instruments can be selected using ‘SetConfigurationFileName( )’. If no configuration file is selected, the same settings as last configured in the GUI will be used. These are stored in

ProgramData\BitifEye\[ProductGroup]\[Standard]\Settings\ValiFrame.vset

where [ProductGroup] could be ‘ValiFrameK1’ and [Standard] could be ‘USB4’, for example.

Configuration

public bool ConfigureProduct( )

```
valiFrame.ConfigureProduct()
```

The configuration of an application is available through ‘ConfigureProduct( )’.

public void LoadProject(string filename)

```
valiFrame.LoadProject('myProjectFile.vfp')
```

Alternatively, you can load a project using ‘LoadProject( )’.

public List<Tuple<Int,string>> GetProcedures( )

After the configuration, the test and calibration procedures are available. Each procedure has a unique number – the procedure ID – and name. Access to all procedure parameters and the execution of the procedures require the procedure ID.

The function ‘GetProcedures( )’ returns a list of tuples (procedure IDs and procedure names).

The example Python code shows how to access the procedure IDs and names.

```
procedureIdsAndNames = valiFrame.GetProcedures()
for procedureIdAndName in procedureIdsAndNames:
     print(‘ID: %d, Name: %s’ % (procedureIdAndName.Item1, procedureIdAndName.Item2))
```

public Tuple<string, System.Drawing.Bitmap> GetConnectionInfo(int procedureId)

The function GetConnectionInfo(int procedureId) returns a tuple containing the connection description and the connection diagram (or picture) of the specific procedure with the given procedure ID.

public VFObject[ ] GetProcedureProperties(string procedureName)

public VFObject[ ] GetProcedureProperties(int procedureId)

public Dictionary <string, string> GetProcedurePropertiesList(int procedureId)

public VFObject[ ] GetProcedureRelatedProperties(int procedureId)

Several functions are available to access the procedure properties (e.g., voltages, jitter amplitudes).

The overloaded functions 'GetProcedureProperties(string procedureName)' and 'GetProcedureProperties(int procedureId)' can be used to get a list of VFObjects. The VFObjects have names that can be used to access the properties' values.

An alternative function to access the procedure properties is 'GetProcedurePropertiesList(int procedureId)'. It returns a System.Collections.Generic.Dictionary.

The procedures are structured hierarchically. In addition to the procedure properties, the higher-level procedure groups and the application (top level) can have properties. These properties can impact the test and calibration procedures. To access properties higher up in the hierarchy, use the method 'GetProcedureRelatedProperties( )'. The usage is the same as for the 'GetProcedureProperties( )' method.

public bool SetProcedureProperty(int procedureId, string propertyName, string propertyValue)

public void SetProcedurePropertiesToDefault(int procedureId)

All properties can be modified via the application interface.

Via the properties' names, the values can be changed by 'SetProcedureProperty(, ,)'. Example:

```
valiFrame.SetProcedureProperty(1050010, 'Min Level', 500 mV')
```

This function is overloaded and allows the properties' values to be modified via several pathways.

The function 'SetProcedurePropertiesToDefault( )' is used to reset all values to their default (i.e., starting) values.

Execution of Procedures and Result Handling

There are several ways to run calibration and test procedures depending on what you want to achieve.

```
public void SelectProcedures(int[ ] procedureIds)
public bool Run( )
public bool StartRun( )
```

The ‘Run( )’ and ‘StartRun( )’ methods execute whatever was selected with ‘SelectProcedures(int[ ] procedureIds)’.

The ‘StartRun( )’ method is specifically intended to build responsive GUIs on top of the API and should be avoided in any other scenario.

```
public string[ ] RunProcedures(int[ ] procedureIds)
```

The ‘RunProcedures(int[] procedureIds)’ method executes the procedures handed in as the procedureIds parameter.

```
public string RunProcedure(int procedureID)
```

‘RunProcedure(int procedureID)’ executes a single procedure identified by the procedureID parameter and returns the procedure result.

```
public delegate void StatusChangedEventHandler(object sender, string description);
public event StatusChangedEventHandler StatusChanged
```

The StatusChanged event is fired whenever the ValiFrame sequencer changes to a new status. It can be used as a hook to react to things happening on the inside of the ValiFrame sequencer, for example, logging progress to an external system. The description parameter contains a compact description of the new status in the format '{target} [{station name}] {change type} [{index}]'.

- {target} is one of the following:
  - Station
  - Family
  - Product
  - ProcedureSubset
  - Procedure
- {station name} appears only if {target} is 'Station'.
- {change type} is one of the following:
  - Start
  - Abort
  - Init
  - Start Iteration
  - Conclude Iteration
  - Step
  - Pause
  - Continue
  - CleanUp
  - Complete
  - OfflineStatusChanged
- {index} appears only if {change type} is 'Start Iteration' or 'Step'. It is a zero-based counter of the current iteration or step within a procedure.

public void SetViewerSuppressState(bool state)

If you do not want results to be displayed in the HTML viewer, this can be suppressed using the ‘SetViewerSuppressState(bool)’ method.

Once a procedure has been run, xml-formatted output is available. Even if several procedures are selected to be run, the results are always created per procedure, not only at the end.

The out parameters ‘string xmlResult’ and ‘out string[] xmlResults’ from all methods contain the results of the previous executions. For all methods you can access the xml result string(s) after the run has finished via the ‘Results’ property.

public delegate void ProcedureCompletedEventHandler(int procedureId, string xmlResult);
public event ProcedureCompletedEventHandler ProcedureCompleted

If you want to use one of the methods that execute multiple procedures with a single call but still handle xml results per procedure as soon as they are available, you can do that in a handler method registered to the ‘ProcedureCompleted’ event.

The event ‘ProcedureCompleted’ is fired as soon as one of the selected procedures is completed. The handler method will be called with the procedure ID as the first input parameter and the xml result string as the second whenever a method is finished.

public void SaveResultsAsWorkbook(string path)

The test results can be saved in a Workbook by ‘SaveResultsAsWorkbook( )’.

By default the formatting is HTML. If the filename ends in .xls, the format will be Microsoft Excel.

If a list of tests is executed, the result sheet will contain one or more sheets per test. In the case of ‘StartRun( )’ and ‘RunProcedures( )’, the result sheet contains the results of all selected tests.

void SetXmlResultFormat(ResultFormatE format)
string[] Results { get; }

Two choices are available for the result format in the xml result string, ‘Version 1’ and ‘Version 2’. The version can be selected with ‘SetXmlResultFormat( )’.

After the version has been set, the xml result string, which can be obtained by the Results property, which is part of the ValiFrameRemote class, is formatted according to this setting. Version 1 is more compact, but some information, e.g., the result unit and the column width, is missing.

The following two examples are for Version 1 and Version 2 formatting of the same results.

xml Version 1 result example

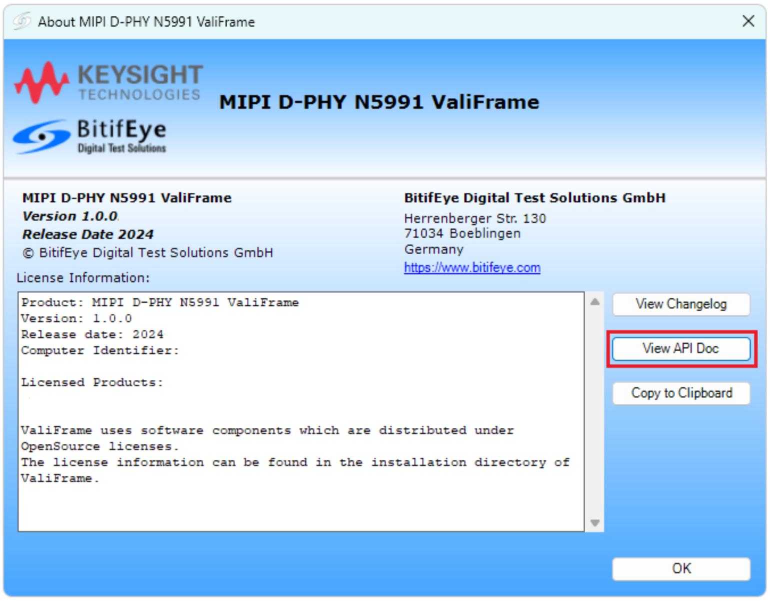
```
<?xml version="1.0" encoding="utf-16"?>
<TestResults>
   <Summary>
      <ProcedureName>LP High Level Calibration Clock; 03/22/2012 13:18:59</ProcedureName>
      <ProcedureID>1050010</ProcedureID>
      <Result>Passed</Result>
      <DateTime>3/26/2012 3:07:40 PM</DateTime>
   </Summary>
   <DocumentElement>
      <Parameters>
         <Name>MIPI BER Reader</Name>
         <Value>PPI LogicAnalyzer</Value>
      </Parameters>
      <Parameters>
         <Name>LA Setting Path</Name>
         <Value>c:\MIPI</Value>
      </Parameters>
      <Parameters>
         <Name>LA Wait Time</Name>
         <Value>5 s</Value>
      </Parameters>
      <Parameters>
         <Name>LP Frequency</Name>
         <Value>10 MBit/s</Value>
      </Parameters>
      <Parameters>
         <Name>HS frequencies</Name>
         <Value>800 MBit/s</Value>
      </Parameters>
      <Parameters>
         <Name>Manual Clock Data Skew Alignment</Name>
         <Value>False</Value>
      </Parameters>
      <Parameters>
         <Name>Min Level</Name>
         <Value>400 mV</Value>
      </Parameters>
      <Parameters>
         <Name>Max Level</Name>
         <Value>1.3 V</Value>
      </Parameters>
      <Parameters>
         <Name>Step Size</Name>
         <Value>50 mV</Value>
      </Parameters>
      </DocumentElement>
   <Data>
      <ColumnHeader>|Result|Set Level|Measured Normal LP High Level|Measured Complement
LP High Level|</ColumnHeader>
      <Values>|pass|1.3|1.185|1.185|</Values>
      <Values>|pass|1.25|1.1375|1.1375|</Values>
      <Values>|pass|1.2|1.09|1.09|</Values>
      <Values>|pass|1.15|1.0425|1.0425|</Values>
      <Values>|pass|1.1|0.995|0.995|</Values>
      <Values>|pass|1.05|0.9475|0.9475|</Values>
      <Values>|pass|1|0.9|0.9|</Values>
      <Values>|pass|0.95|0.8525|0.8525|</Values>
      <Values>|pass|0.9|0.805|0.805|</Values>
      <Values>|pass|0.85|0.7575|0.7575|</Values>
      <Values>|pass|0.8|0.71|0.71|</Values>
      <Values>|pass|0.75|0.6625|0.6625|</Values>
      <Values>|pass|0.7|0.614999999999999|0.614999999999999|</Values>
      <Values>|pass|0.649999999999999|0.567499999999999|0.567499999999999|</Values>
      <Values>|pass|0.599999999999999|0.519999999999999|0.519999999999999|</Values>
      <Values>|pass|0.549999999999999|0.472499999999999|0.472499999999999|</Values>
      <Values>|pass|0.499999999999999|0.424999999999999|0.424999999999999|</Values>
      <Values>|pass|0.449999999999999|0.377499999999999|0.377499999999999|</Values>
      <Values>|pass|0.4|0.33|0.33|</Values>
   </Data>
</TestResults>
```

xml Version 2 result example

```
<?xml version="1.0" encoding="utf-16"?>
<TestResults version="2.0">
   <TestResult>
      <Summary>
         <ProcedureName>LP High Level Calibration Clock; 03/22/2012
13:18:59</ProcedureName>
         <ProcedureID>1050010</ProcedureID>
         <Result>Passed</Result>
         <DateTime>3/26/2012 3:07:40 PM</DateTime>
         <Duration>00:00:02.8945788</Duration>
         <Description>This procedure does the calibration of levels for the data and clock
generators.</Description>
      </Summary>
      <ResultTables>
         <ResultTable>
            <Title>LP V_OH Calibration Clock</Title>
            <Subtitle>Calibrates the Generator LP Level</Subtitle>
            <Name>LP V_OH Calibration Clock</Name>
            <Parameters>
               <Parameter name="MIPI BER Reader" value="PPI LogicAnalyzer" />
               <Parameter name="LA Setting Path" value="c:\MIPI" />
               <Parameter name="LA Wait Time" value="5 s" />
               <Parameter name="LP Frequency" value="10 MBit/s" />
               <Parameter name="HS frequencies" value="800 MBit/s" />
               <Parameter name="Manual Clock Data Skew Alignment" value="False" />
               <Parameter name="Min Level" value="400 mV" />
               <Parameter name="Max Level" value="1.3 V" />
               <Parameter name="Step Size" value="50 mV" />
            </Parameters>
            <Columns>
               <Column name="Result" unit="" recommendedExponent="0" width="13">
                  <Value>pass</Value>
                  <Value>pass</Value>
                  <Value>pass</Value>
                  <Value>pass</Value>
                  <Value>pass</Value>
                  <Value>pass</Value>
                  <Value>pass</Value>
                  <Value>pass</Value>
                  <Value>pass</Value>
                  <Value>pass</Value>
                  <Value>pass</Value>
                  <Value>pass</Value>
                  <Value>pass</Value>
                  <Value>pass</Value>
                  <Value>pass</Value>
                  <Value>pass</Value>
                  <Value>pass</Value>
                  <Value>pass</Value>
                  <Value>pass</Value>
               </Column>
               <Column name="Set Level" unit="V" precision="0" recommendedExponent="-3" 
width="8">
                  <Value>1.3</Value>
                  <Value>1.25</Value>
                  <Value>1.2</Value>
                  <Value>1.15</Value>
                  <Value>1.1</Value>
                  <Value>1.05</Value>
                  <Value>1</Value>
                  <Value>0.95</Value>
                  <Value>0.9</Value>
                  <Value>0.85</Value>
                  <Value>0.8</Value>
                  <Value>0.75</Value>
                  <Value>0.7</Value>
                  <Value>0.649999999999999</Value>
                  <Value>0.599999999999999</Value>
                  <Value>0.549999999999999</Value>
                  <Value>0.499999999999999</Value>
                  <Value>0.449999999999999</Value>
                  <Value>0.4</Value>
               </Column>
               <Column name="Measured Normal LP High Level" unit="V" precision="0" 
recommendedExponent="-3" width="8">
                  <Value>1.185</Value>
                  <Value>1.1375</Value>
                  <Value>1.09</Value>
                  <Value>1.0425</Value>
                  <Value>0.995</Value>
                  <Value>0.9475</Value>
                  <Value>0.9</Value>
                  <Value>0.8525</Value>
                  <Value>0.805</Value>
                  <Value>0.7575</Value>
                  <Value>0.71</Value>
                  <Value>0.6625</Value>
                  <Value>0.614999999999999</Value>
                  <Value>0.567499999999999</Value>
                  <Value>0.519999999999999</Value>
                  <Value>0.472499999999999</Value>
                  <Value>0.424999999999999</Value>
                  <Value>0.377499999999999</Value>
                  <Value>0.33</Value>
               </Column>
               <Column name="Measured Complement LP High Level" unit="V" precision="0"
recommendedExponent="-3" width="8">
                  <Value>1.185</Value>
                  <Value>1.1375</Value>
                  <Value>1.09</Value>
                  <Value>1.0425</Value>
                  <Value>0.995</Value>
                  <Value>0.9475</Value>
                  <Value>0.9</Value>
                  <Value>0.8525</Value>
                  <Value>0.805</Value>
                  <Value>0.7575</Value>
                  <Value>0.71</Value>
                  <Value>0.6625</Value>
                  <Value>0.614999999999999</Value>
                  <Value>0.567499999999999</Value>
                  <Value>0.519999999999999</Value>
                  <Value>0.472499999999999</Value>
                  <Value>0.424999999999999</Value>
                  <Value>0.377499999999999</Value>
                  <Value>0.33</Value>
               </Column>
            </Columns>
            <Images />
         </ResultTable>
      </ResultTables>
   </TestResult>
</TestResults>
```

Further Documentation

You can find more details about the ValiFrameRemote API in the API Documentation, which can be accessed under About in the ValiFrame applications (Figure 2). Click “View API Doc”.


About MIPI D-PHY N5991 ValiFrame
KEYSIGHT TECHNOLOGIES
BitifEye Digital Test Solutions
MIPI D-PHY N5991 ValiFrame
View Changelog
View API Doc
Copy to Clipboard
OK

MIPI D-PHY N5991 ValiFrame
Version 1.0.0
Release Date 2024
© BitifEye Digital Test Solutions GmbH

BitifEye Digital Test Solutions GmbH
Herrenberger Str. 130
71034 Boeblingen
Germany
https://www.bitifeye.com

License Information:

```
Product: MIPI D-PHY N5991 ValiFrame
Version: 1.0.0
Release date: 2024
Computer Identifier:

Licensed Products:


ValiFrame uses software components which are distributed under
OpenSource licenses.
The license information can be found in the installation directory of
ValiFrame.
```

Figure 2 About window of N5991 ValiFrame (example for MIPI D-PHY^{SM})

3 Appendix: Acronyms and Abbreviations

This Appendix contains a list of acronyms and abbreviations used in the Application Programming Interface for the Keysight N5991 Test Automation Software Platform User Guide.

List of Acronyms

Acronym	Definition
API	Application Programming Interface
GUI	Graphical User Interface
HTML	HyperText Markup Language
ID	Identifier
VF	ValiFrame
xml	Extensible Markup Language

N5991-91012

www.keysight.com