Kaelus
Passive IM Analyzer System

Advanced Programming Interface
Programming and Usage Guide

April 2013 Edition

Kaelus Rev 2.4

Kaelus, Inc. 1997-2013
Centennial, Colorado 80111 USA

www.kaelus.com

Printed in the USA

Limited Warranty

The following document contains proprietary information intended solely for Kaelus authorized customers and representatives. By reading this documentation, you or the entity or company that you represent ("YOU") unconditionally consent and agree to be bound by and a party to the terms and conditions presented herein.

Kaelus believes that the information in this manual is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, Kaelus reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult Kaelus if errors are suspected. In no event shall Kaelus be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, KAELUS MAKES NO WARRANTIES, EXPRESSED OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. THE CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF KAELUS SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. KAELUS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of Kaelus will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against Kaelus must be brought within one year after the cause of action occurs. Kaelus shall not be liable for any delay in performance due to causes beyond its reasonable control The warranty provided herein does not cover damages, defects, malfunctions or service failures caused by owner's failure to follow Kaelus installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuses, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright

Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of Kaelus, Inc.

Trademarks

LabVIEW, NI-DAQ, DAQmx are trademarks of National Instruments Corporation.

Windows and Excel are registered trademarks of the Microsoft Corporation.

Product and company names listed are trademarks or trade names of their respective companies.

BPIM API DEVELOPER LICENSE AGREEMENT

IMPORTANT - READ CAREFULLY BEFORE INSTALLING OR USING THE ENCLOSED SOFTWARE INCLUDED WITH THIS UNIT. BY INSTALLING OR USING THIS SOFTWARE, YOU ARE AGREEING TO BE BOUND BY THE TERMS OF THIS AGREEMENT. THIS IS A LEGAL AGREEMENT BETWEEN YOU (EITHER AN INDIVIDUAL OR AN ENTITY) AND KAELUS, INC..

BY INTERACTING IN ANY WAY WITH THE KAELUS BPIM API, INCLUDING, BUT NOT LIMITED TO REQUESTING CREDENTIALS OR MAKING API CALLS, YOU OR THE ENTITY OR COMPANY THAT YOU REPRESENT ("YOU") UNCONDITIONALLY CONSENT AND AGREE TO BE BOUND BY AND A PARTY TO THESE TERMS AND CONDITIONS ("AGREEMENT").

DRIVER SOFTWARE LICENSE

1. **GRANT OF LICENSE.** Subject to your ("Licensee's") full compliance with all of the terms and conditions of this API Agreement ("Agreement") and upon written approval, Kaelus, Inc. ("Kaelus") grants Licensee a non-exclusive, revocable, nonsublicensable, nontransferable license to download and use the Kaelus PIM application program interface and other materials provided by Kaelus (collectively, "API") to develop, reproduce and distribute applications that interoperate with Kaelus' PIM Software Suite, or any other software or web property owned by Kaelus ("Kaelus Applications"). Licensee may not install or use the API for any other purpose without Kaelus' prior written consent, Licensee shall not use the API in connection with or to promote any products, services, or materials that constitute, promote or are used primarily for the purpose of dealing in: spyware, adware, or other malicious programs or code, counterfeit goods, items subject to U.S. embargo, unsolicited mass distribution of email ("spam"), multi-level marketing proposals, hate materials, hacking/surveillance/interception/descrambling equipment, libelous, defamatory, abusive or otherwise offensive content.
2. **COPYRIGHT.** The API is owned by Kaelus or its suppliers and is protected by United States copyright laws and international treaty provisions. Therefore, you must treat the API like any other copyrighted material except that you may either (a) make one copy of the API solely for backup or archival purposes, or (b) transfer the API to a single hard disk, provided you keep the original solely for backup or archival purposes. You may not copy the written materials accompanying the API.
3. **PROPRIETARY RIGHTS -** As between Kaelus and Licensee, the API and all intellectual property rights in and to the API are and shall at all times remain the sole and exclusive property of Kaelus and are protected by applicable intellectual property laws and treaties.
4. **OTHER RESTRICTIONS.** Except as expressly and unambiguously authorized under this Agreement, Licensee may not (i) copy, rent, lease, sell, transfer, assign, sublicense, disassemble, reverse engineer or decompile (except to the limited extent expressly authorized by applicable statutory law), modify or alter any part of the API; (ii) otherwise use the API on behalf of any third party. Kaelus expressly reserves the right to limit the number and/or frequency of API requests in its sole discretion.
5. **KAELUS TRADEMARKS -** This Agreement does not include any right for Licensee to use any trademark, service mark, trade name or any other mark of Kaelus or any other party or licensor. No rights or licenses are granted except as expressly and unambiguously set forth herein.

6. **MODIFICATIONS TO THIS AGREEMENT -** Kaelus reserves the right, in its sole discretion to modify this Agreement at any time by posting a notice to Kaelus.com. You shall be responsible for reviewing and becoming familiar with any such modification. Such modifications are effective upon first posting or notification and use of the Kaelus PIM API by Licensee following any such notification constitutes Licensee's acceptance of the terms and conditions of this Agreement as modified.

7. **WARRANTY DISCLAIMER -** THE API ISPROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, KAELUS AND ITS VENDORS EACH DISCLAIM ALL WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, REGARDING THE API, INCLUDING WITHOUT LIMITATION ANY AND ALL IMPLIED WARRANTIES OF MERCHANTABILITY, ACCURACY, RESULTS OF USE, RELIABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, INTERFERENCE WITH QUIET ENJOYMENT, AND NON-INFRINGEMENT OF THIRD-PARTY RIGHTS. FURTHER, KAELUS DISCLAIMS ANY WARRANTY THAT LICENSEE'S USE OF THE API WILL BE UNINTERRUPTED OR ERROR FREE.

8. **SUPPORT AND UPGRADES -** This Agreement does not entitle Licensee to any support for the APIs, unless Licensee makes separate arrangements with Kaelus and pays all fees associated with such support. Any such support provided by Kaelus shall be subject to the terms of this Agreement as modified by the associated support Agreement.

9. **LIABILITY LIMITATION -** REGARDLESS OF WHETHER ANY REMEDY SET FORTH HEREIN FAILS OF ITS ESSENTIAL PURPOSE OR OTHERWISE, AND EXCEPT FOR BODILY INJURY, IN NO EVENT WILL KAELUS OR ITS VENDORS, BE LIABLE TO LICENSEE OR TO ANY THIRD PARTY UNDER ANY TORT, CONTRACT, NEGLIGENCE, STRICT LIABILITY OR OTHER LEGAL OR EQUITABLE THEORY FOR ANY LOST PROFITS, LOST OR CORRUPTED DATA, COMPUTER FAILURE OR MALFUNCTION, INTERRUPTION OF BUSINESS, OR OTHER SPECIAL, INDIRECT, INCIDENTAL OR CONSEQUENTIAL DAMAGES OF ANY KIND ARISING OUT OF THE USE OR INABILITY TO USE THE API, EVEN IF KAELUS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH LOSS OR DAMAGES AND WHETHER OR NOT SUCH LOSS OR DAMAGES ARE FORESEEABLE. ANY CLAIM ARISING OUT OF OR RELATING TO THIS AGREEMENT MUST BE BROUGHT WITHIN ONE (1) YEAR AFTER THE OCCURRENCE OF THE EVENT GIVING RISE TO SUCH CLAIM. IN ADDITION, KAELUS DISCLAIMS ALL LIABILITY OF ANY KIND OF KAELUS' VENDORS.

10. **INDEMNITY -** Licensee agrees that Kaelus shall have no liability whatsoever for any use Licensee makes of the API. Licensee shall indemnify and hold harmless Kaelus from any and all claims, damages, liabilities, costs and fees (including reasonable attorneys' fees) arising from Licensee's use of the API.

11. **TERM AND TERMINATION -** This Agreement shall continue until terminated as set forth in this Section. Either party may terminate this Agreement at any time, for any reason, or for no reason including, but not limited to, if Licensee violates any provision of this Agreement. Any termination of this Agreement shall also terminate the license granted hereunder. Upon termination of this Agreement for any reason, Licensee shall destroy and remove from all computers, hard drives, networks, and other storage media all copies of the API, and shall so certify to Kaelus that such

actions have occurred. Kaelus shall have the right to inspect and audit Licensee's facilities to confirm the foregoing. Sections 6 through 11 and all accrued rights to payment shall survive termination of this Agreement.

12. **U.S. GOVERNMENT RESTRICTED RIGHTS -** The API SOFTWARE and DOCUMENTATION are provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the Government is subject to restrictions as set forth in the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013(c)(1)(ii) or the Commercial Computer Software - Restricted Rights clause at 48 CFR 52.227-19(c)(2), or clause 18-52.227-86(d) of the NASA Supplement, as applicable. Contractor/Manufacturer is Kaelus, Inc., 12503 E. Euclid Drive, Suite 7, Centennial, Colorado, 80111.

13. **EXPORT CONTROLS -** Licensee shall comply with all export laws and restrictions and regulations of the Department of Commerce, the United States Department of Treasury Office of Foreign Assets Control ("OFAC"), or other United States or foreign agency or authority, and Licensee shall not export, or allow the export or re-export of the API in violation of any such restrictions, laws or regulations. By downloading or using the API, Licensee agrees to the foregoing and represents and warrants that Licensee is not located in, under the control of, or a national or resident of any restricted country.

14. **MISCELLANEOUS -** Unless the parties have entered into a written amendment to this agreement that is signed by both parties regarding the Kaelus PIM API, this Agreement constitutes the entire agreement between Licensee and Kaelus pertaining to the subject matter hereof, and supersedes any and all written or oral agreements with respect to such subject matter. This Agreement, and any disputes arising from or relating to the interpretation thereof, shall be governed by and construed under Colorado law as such law applies to agreements between Colorado residents entered into and to be performed within Colorado by two residents thereof and without reference to its conflict of laws principles or the United Nations Conventions for the International Sale of Goods. Except to the extent otherwise determined by Kaelus, any action or proceeding arising from or relating to this Agreement must be brought in a federal court in Colorado or in state court in Arapahoe County, Colorado, and each party irrevocably submits to the jurisdiction and venue of any such court in any such action or proceeding. The prevailing party in any action arising out of this Agreement shall be entitled to an award of its costs and attorneys' fees. This Agreement may be amended only by a writing executed by Kaelus. If any provision of this Agreement is held to be unenforceable for any reason, such provision shall be reformed only to the extent necessary to make it enforceable. The failure of Kaelus to act with respect to a breach of this Agreement by Licensee or others does not constitute a waiver and shall not limit Kaelus' rights with respect to such breach or any subsequent breaches. This Agreement is personal to Licensee and may not be assigned or transferred for any reason whatsoever (including, without limitation, by operation of law, merger, reorganization, or as a result of an acquisition or change of control involving Licensee) without Kaelus' prior written consent and any action or conduct in violation of the foregoing shall be void and without effect. Kaelus expressly reserves the right to assign this Agreement and to delegate any of its obligations hereunder.

Contents

1. Overview

The PIM Advanced Programmer Interface is a fully documented command interface language used to control and perform measurements using a Kaelus PIM analyzer. The API has been designed using a common language interface approach, making it easily adoptable to most modern programming platforms. The purpose of publishing this API is to allow PIM customers the flexibility to integrate PIM testing within their own custom designed software.

This programmer interface will be available on all PIM controller devices using PIM software version 9.0 or newer. Each PIM controller will utilize a new software component called the PIM Web Engine, which will launch by default at Windows startup. The PIM Web Engine is a hardware instrument driver platform which hosts an API web service. This service is your direct command interface which can be used to control the PIM analyzer instruments connected to the client machine.

Commands are sent over a standard HTTP connection (supporting HTTP/1.0), this interface lends itself to a majority of popular programming languages which are capable of utilizing TCP/IP socket connections. This allows the web service to be accessible from any machine supporting the HTTP protocol, though the PIM Web Engine is restricted for use only on a Windows-based system (part of the system requirements of the BPIM Software Suite).

2. Installation

The PIM API is hosted as a web service by the BPIM Web Engine application, which is installed by default with PIM controller software version 9.0 and newer (Kaelus recommends the latest released version for API development. As of this writing, the latest release is version 11.0.0). The API will be accessible following the successful launch of the BPIM Web Engine application.

An easy mechanism to test the deployment of the PIM software is to launch the included Virtual Front Panel application, which takes full advantage of the PIM API interface language to communicate with the BPIM Web Engine and a valid set of calibration files.

3. Configuration

3.1. E|D-Series BPIM Analyzer Control

The PIM API web service is deployed automatically to PIM controller systems by simply installing the BPIM Software Suite. The suite includes the BPIM Web Engine application (which hosts the API interface), and the Virtual Front Panel application (which connects through the Web Engine to control the BPIM analyzer).

A requirement of the BPIM Web Engine is that the web server port (TCP port 80) be accessible by the local machine, but if your custom application requires remote control of the PIM controller, you must ensure that the web server port is visible to external devices as well by ensuring the TCP port is not blocked by any gateways, proxies, or firewalls.

3.2. A|B-Series BPIM Analyzer Control

The latest version of the PIM software suite does not offer support for older BPIM analyzers. However, the PIM API has been designed to allow control of older BPIM analyzers through the legacy PIM Hardware Engine application (pimhw.exe). To control legacy PIM analyzers, you must first install the PIM Distribution Software version 7.3 or newer to your PIM controller. Be sure to copy your calibration files to the appropriate program directory (typically "C:\Program Files\Summitek\PIM"). Verify that your legacy software is functional by running the VFP software (vfp.exe). Legacy PIM analyzers will require the installation of NI-DAQ Traditional software version 7.1 or newer (available for download from http://www.ni.com).

Next, install the BPIM Software Suite to the same PIM controller. This software suite will install the BPIM Web Engine (BPIM Web Engine.exe) and the new Virtual Front Panel (Virtual Front Panel.exe) software to your machine. The API will communicate to your PIM Hardware Engine over TCP port 3388 on the local machine (ensure this port is not blocked by any firewall or proxy interfaces). The BPIM Web Engine will serve the API on the traditional web server port 80 (this port should remain unblocked as well).

Please Note: The new Virtual Front Panel will be unable to communicate with your legacy PIM hardware. Use the legacy VFP.exe to control your older PIM hardware instead.

After both PIM software packages are installed on the target machine, you can simply make API calls setting the Legacy attribute to TRUE, and all API calls will automatically be relayed to the PIM Hardware Engine to control your legacy PIM hardware.

3.3. Configuring your custom Application

Custom applications do not have to reside on the PIM controller machine (though they can coexist perfectly fine). They simply require visibility of the target PIM controller machines' TCP port 80. Opening a TCP socket connection to this port will trigger the ability to communicate to the BPIM Web Engine using the standard HTTP Request/Response model, allowing your application to send and receive XML commands and responses following the PIM XML Schema Definition (referenced below).

4. Operation

Communication with the PIM API web service is done using the HTTP request/response model, and must follow the protocol standards for sending requests and receiving responses. Further information about the HTTP protocol can be obtained through its RFC documentation (RFC 2616 - http://www.w3.org/Protocols/rfc2616/rfc2616.html). It is ideal to use software libraries which include the capability of sending HTTP requests and receiving HTTP responses.

Also, to ensure proper command/response syntax in commands and responses sent to and from the PIM API, we have developed a customized XML Schema Definition (http://www.w3.org/TR/xmlschema-1) which defines the command and response syntax. This schema definition can be taken advantage of by many existing XML toolkits by performing a validation of your commands or the received responses against the schema definition. More information about XML Schema Validation can be obtained through W3.org.

4.1. Initializing the Connection (Obtaining Authorization)

Before any commands will be accepted by the API web service, your application must first open a connection to the application and retrieve the authorization token. This token is dynamically created at each PIM Web Engine launch, and so must be captured dynamically within your application in order to allow authorized commands to the PIM API web service. Retrieve the token by sending an HTTP request (either GET or POST) to the following URL:

```
http://<controller>/api/connect/<Legacy?>
```

The response received from this command will contain the current API authentication token, as well as a list of any detected PIM analyzer instruments visible to the PIM API. A response with a single instrument would look like the following code segment:

```
<?xml version="1.0"?>
<PIM xmlns="http://www.kaelus.com"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.kaelus.com PIMXMLSchema.xsd">

  <Connection>
    <Token>C83AE89F</Token>
    <Analyzer Name="SI-1900" Legacy="FALSE" BandIndex="0"/>
  </Connection>
</PIM>
```

From this response the API token can be captured to use in the remainder of your application interactions with the PIM API. Additionally, you can get an idea of which instruments are available for communication with your application through this interface. Please note that multi-band PIM analyzers configured using Kaelus switch (SW) modules will each show up as their own individual analyzer. Analyzer selection can then occur by issuing a **SetMeasBand** command with the appropriate band index (also captured here).

4.2. Sending Single Commands (HTTP GET)

For the purposes of convenience, commands may be issued individually using an HTTP GET request method. This method is useful to test communications between your application and the PIM analyzer, as these commands can easily be issued manually through a web browser by simply entering a valid API token in your request string:

```
http://[PIM Controller Address]/api/[API Token]/[API Command]
```

Commanding the API in this single command format is less efficient than performing batch commands through an HTTP POST operation, however it may be easier to implement these simplified commands for initial testing, as no special command formatting is necessary. This is also an effective way to easily test your response analysis methods, as the response can simply be captured and parsed as necessary.

An example of a single command and its associated response is shown here:

Command

```
http://localhost/api/C83AE89F/GetError
```

Response

```
<?xml version="1.0"?>
<PIM xmlns="http://www.kaelus.com"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.kaelus.com PIMXMLSchema.xsd">

   <Responses Legacy="FALSE">
      <Response Action="GetError" Units="">
     <Status Error="FALSE" Code="5010">The PIM hardware engine could not load a
valid calibration file for the test set. Default values will be used for calibration
coefficients, however the resulting measurements will not be accurate.</Status>
    </Response>
  </Responses>
</PIM>
```

The above example issues a **GetError** command which highlights the Status data-type in its response. This data-type is defined to provide error flow between the PIM API and your custom application. When failures occur, the API will return errors in the form of a Status data-type within its associated response.

The following example demonstrates issuing a command with parameters to a legacy PIM Hardware Engine:

Command

```
http://localhost/api/C83AE89F/TRUE/setTxFreqs?Double=1930.0&Double=1990.0
```

Response

```
<?xml version="1.0"?>
<PIM xmlns="http://www.kaelus.com"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.kaelus.com PIMXMLSchema.xsd">

   <Responses Legacy="TRUE">
      <Response Action="SetTxFreqs" Units="" />
  </Responses>
</PIM>
```

The above example issues a **SetTxFreqs** command to a Legacy PIM hardware engine. The command includes two parameters issued as part of the HTTP Query String to set the two carrier frequencies. Notice that the query string is identified by the **?** symbol, and parameters take the form of ***[DataType]=[DataValue]*** and are delimited with **&** symbols.

4.3. Sending Batch Commands (HTTP POST)

A much more efficient mechanism to communicate to the PIM API is through batch command sequences. There are many common operations which may be executed as batch commands to the PIM API, in order to save request/response time between each command. To take advantage of these advanced functionalities, commands must be formatted as specified in the PIM XML Schema definition, as they are submitted to the application through an HTTP POST request method. While issuing commands through the POST method are slightly more cumbersome than issuing a simple GET command, they are in fact quite simple, and can increase your application throughput significantly.

As previously mentioned, the HTTP POST methodology requires that your commands follow the PIM XML Schema Definition format for commands, which means that your commands will look similar to the following format:

```
<?xml version="1.0"?>
<PIM xmlns="http://www.kaelus.com"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.kaelus.com PIMXMLSchema.xsd">

 <Commands Token="[API Token]" Legacy="[TRUE/FALSE]" >
  <Command Action="[API Command]" />
  <Command Action="[API Command]" >
   <[DataType]>
    [DataValue]
   </[DataType]>
  </Command>
 </Commands>
</PIM>
```

Following the above format, commands can be issued in batches as a time saving mechanism. In fact, if the command response is not immediately necessary, large measurement operations can be performed in this manner, capturing only intermittent response data (rather than attempting to stream response data).

The following example illustrates the configuration and measurement of a fixed frequency IM measurement in the 1900 band. Following this example, a collection of similar commands could be issued to simulate a frequency sweep behavior (modifying the carrier frequencies slightly in each iteration).

Command

```
<?xml version="1.0"?>
<PIM xmlns="http://www.kaelus.com"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.kaelus.com PIMXMLSchema.xsd">

 <Commands Token="C83AE89F" Legacy="FALSE" >
  <Command Action="SetTxFreqs" >
   <Double>1930.00</Double>
   <Double>1990.00</Double>
  </Command>
  <Command Action="GetImFreqs" />
  <Command Action="GetImPower" />
 </Commands>
</PIM>
```

Response

```
<?xml version="1.0"?>
<PIM xmlns="http://www.kaelus.com"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.kaelus.com PIMXMLSchema.xsd">

 <Responses Legacy="FALSE"
  <Response Action="SetTxFreqs" Units="" />
  <Response Action="GetImFreqs" Units="MHz">
   <Double>1870.00</Double>
   <Double>1810.00</Double>
   <Double>1750.00</Double>
   <Double>1690.00</Double>
  </Response>
  <Response Action="GetImPower" Units="dBm">
   <Double>-115.85</Double>
   <Double>-200.00</Double>
   <Double>-200.00</Double>
   <Double>-200.00</Double>
  </Response>
 </Responses>
</PIM>
```

The above example illustrates the method for issuing multiple commands simultaneously. Commands will be executed in the order in which they are received. When sent as batch processes, commands will be executed top to bottom, ensuring commands are executed in their desired order. This example also illustrates the use of the schema-defined data-types to send and receive data to and from the PIM API. In the command, we set the carrier frequencies by defining them as defined in the **SetTxFreqs** command. Then in the response

for both the **GetImFreqs** and the **GetImPower**commands we see the returned data take the same format.

5. Example Code

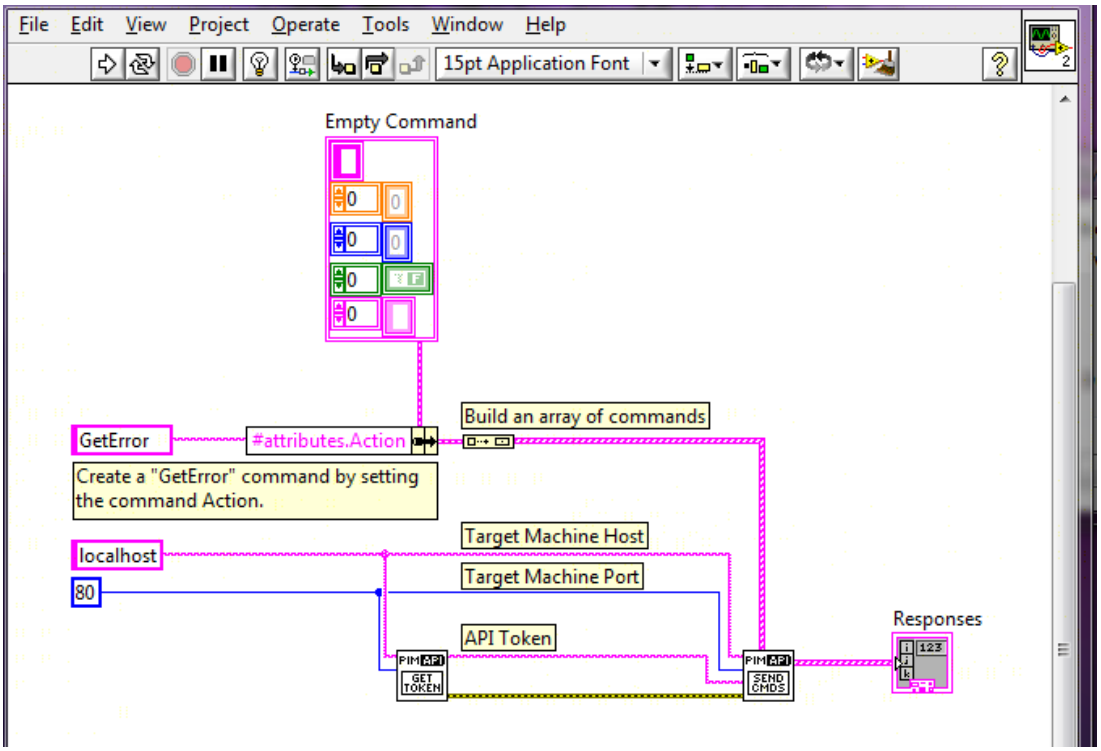
5.1. LabVIEW Based API Control

As of this writing, LabVIEW 2010 had introduced panel functions designed to support HTTP GET and POST behaviors, however it has been discovered that these functions perform the HTTP operations slower than desired for PIM API interaction (approximately 1.3 seconds per command issued). Older versions of LabVIEW do not provide native support for HTTP communications.

To support high-speed communications between your custom LabVIEW software and the BPIM Web Engine, we have designed an optimized I/O interface library for use within your source code. This library is designed for optimized communications with the BPIM Web Engine only, and should not be utilized as a standard HTTP request library without additional optimizations (this modification is based on the simplified communications protocol which does not take full advantage of the HTTP protocol features).

See the "BPIM API Command LIB" source in the API EXAMPLES folder.

As a demonstration of the library, the below pictured VI sends a GetError command through the API, then awaits the response.



5.2. Visual Studio 2010 - C# Using WinForms

As an example of how the API can be taken advantage of from text-based languages, we have written a simple example of how the API might be accessed through .NET languages using the System.Net WebClient libraries. The following example uses a WinForm user interface to present connection options, and perform a sweep operation on command. The code does not focus on XML parsing actions, but rather on the communication protocol between the API and Web Engine, thereby providing a good example of how a developer can interact with PIM analyzers using the BPIM API language.

See the "BPIM API Using Winforms" source in the API EXAMPLES folder.

5.3. Taking Advantage of Measurement Modes

Measurement modes were introduced to the API in release version 11.0 of the BPIM Software Suite as a means to improve communications speeds and increase the amount of information which is communicated in a single response. Using the GetSamples command, all relevant measurement data can be retrieved in a single response instead of polling individual properties. Data such as Frequencies, Power levels, status indicators, and some peak measurement statistics can be retrieved as a single sample. In SweepTx mode, an entire sweep can be returned as a collection of Samples. The following examples explain how these measurement modes can be used.

Fixed Tone Measurement using "Standard" Measurement mode

```
<Command Action="setPreset" />

<Command Action="setModeStandard" /> <!-- Sets a standard "fixed tone" measurement
mode where frequencies don't change rapidly -->

<Command Action="setTxFreqs">

        <Double>1930.0</Double>

        <Double>1990.0</Double>

</Command>

<Command Action="setTxOn">

        <Boolean>True</Boolean>

        <Boolean>True</Boolean>

</Command>
```

```
<!-- Now we can trigger a measurement and collect sample data.  Repeat these steps for as many samples as required. -->

<Command Action=”setTrigger” />

<Command Action=”getSamples” />
```

Swept Frequency Measurement using “SweepTx” Measurement Mode

```
<Command Action=”setPreset” />

<Command Action=”setModeSweepTx”>

    <Double>0.2</Double> <!-- Frequency Step Size -->

    <Double>1930.0</Double> <!-- Carrier 1 Sweep-Band Edge -->

    <Double>1990.0</Double> <!-- Carrier 2 Sweep-Band Edge -->

    <Boolean>True</Boolean> <!-- Return Entire Sweep (False here indicates that the sweep should be returned one point at a time (i.e. getSamples returns only one point instead of all points) -->

</Command>

<Command Action=”setTxOn”>

    <Boolean>True</Boolean>

    <Boolean>True</Boolean>

</Command>

<Command Action=”setTrigger” />

<Command Action=”getSamples” /> <!-- Since we indicated that the sweep should return all points, this command will return an array of points -->
```

6. Reference

6.1. XML Schema Definition (PIMXMLSchema.xsd)

PIM API commands and responses are formatted using a customized XML Schema Definition, following the standards defined by in the W3C Recommendation document located at the url http://www.w3.org/TR/xmlschema-1. This document offers "facilities for describing the structure and constraining the contents of XML 1.0 documents, including those which exploit the XML Namespace facility" among other guidelines. Following the procedures and guidelines defined within the XML Schema Language restrictions, Kaelus has developed a customized schema definition document which defines the syntax of PIM commands and responses, allowing any generated XML to be syntactically validated both before execution and before response interpretation, helping to ensure a more robust communication model between your custom application and the PIM API.

For these purposes, we have developed a PIM XML Schema Definition document which reads as follows:

6.1.1. PIMXMLSchema.xsd

```
<?xml version='1.0' ?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.kaelus.com"
        xmlns="http://www.kaelus.com"
        elementFormDefault="qualified">

    <!-- ROOT ELEMENT - Contains All supported elements -->
    <xs:element name="PIM" type="PIMComm" />

    <xs:complexType name="PIMComm">
      <xs:choice minOccurs="1" maxOccurs="1">
        <xs:element name="Commands" type="PIMCommands" />
        <xs:element name="Responses" type="PIMResponses" />
        <xs:element name="Connection" type="PIMConnection" />
      </xs:choice>
    </xs:complexType>

    <!-- Custom PIM Data-Types -->

    <!-- PIM Commands: The Root collection of all commands being sent to the PIMHW -->
    <xs:complexType name="PIMCommands">
      <xs:sequence>
        <!-- Command(s) -->
        <xs:element name="Command" type="PIMcommand" minOccurs="1"
maxOccurs="unbounded" />
      </xs:sequence>

      <!-- Attributes: Token (Authorization purposes), Version (Compatibility Purposes) -->
      <xs:attribute name="Token" type="xs:string" />
      <xs:attribute name="Legacy" type="xs:boolean" default="false" />
```

```
    </xs:complexType>

    <!-- Individual Command -->

      <!-- PIM Command: An individual command (with optional parameters) to be sent to the
PIMHW -->
      <xs:complexType name="PIMcommand">
        <!-- Parameters: Specific to specified action, parameters must all be of the same data-type --
>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
          <xs:element name="Double" type="xs:decimal" />
          <xs:element name="Integer" type="xs:integer" />
          <xs:element name="Boolean" type="xs:boolean" />
          <xs:element name="String"  type="xs:string" />
        </xs:choice>
        <xs:attribute name="Action" type="xs:string" />
      </xs:complexType>


    <!-- PIM Response: The Root collection of all response data returned by the PIMHW -->
    <xs:complexType name="PIMResponses">
      <xs:sequence>
        <!-- CommandResponse(s) -->
        <xs:element name="Response" type="PIMResponse" minOccurs="1"
maxOccurs="unbounded" />
      </xs:sequence>
      <!-- Attributes: Version (Responses compatible with this version) -->
      <xs:attribute name="Legacy" type="xs:boolean" default="false" />

    </xs:complexType>

    <!-- Individual Response -->


      <!-- PIM CommandResponse: An individual response with optional parameters and error
clusters -->
      <xs:complexType name="PIMResponse">
        <!-- Response Data: Data specific to the specified action being returned, and error or
warning information -->
        <xs:sequence>
          <xs:choice minOccurs="0" maxOccurs="unbounded">
            <xs:element name="Double" type="xs:decimal" />
            <xs:element name="Integer" type="xs:integer" />
            <xs:element name="Boolean" type="xs:boolean" />
            <xs:element name="String"  type="xs:string" />
            <xs:element name="Sample" type="PIMSample" />
            <xs:element name="Status" type="PIMError" />
          </xs:choice>
        </xs:sequence>

        <!-- Attributes: Action (Action Performed), Units (Apply to Data returned) -->
        <xs:attribute name="Action" type="xs:string" />
        <xs:attribute name="Units" type="xs:string" />
      </xs:complexType>

    <!-- PIMSample: Contains all measurement data relevant to a particular measurement -->
    <xs:complexType name="PIMSample">
      <xs:sequence>
        <xs:element name="TX1" type="PIMTX" />
```

```
            <xs:element name="TX2" type="PIMTX" />
            <xs:element name="IM" type="PIMRX" minOccurrs="1" maxOccurs="unbounded" />
        </xs:sequence>
        <xs:attribute name="Count" type="xs:integer" default="1" />
        <xs:attribute name="Total" type="xs:integer" default="1" />
        <xs:attribute name="Finished" type="xs:boolean" default="true" />
        <xs:attribute name="F1" type="xs:decimal" />
        <xs:attribute name="F2" type="xs:decimal" />
        <xs:attribute name="FStep" type="xs:decimal" />
        <xs:attribute name="SweepLeg" type="xs:string" />
    </xs:complexType>

    <!-- PIMTransmit: Contains all relevant data for a PIM transmitter including frequency, power,
and temperature -->
    <xs:complexType name="PIMTX">
        <xs:attribute name="Frequency" type="xs:decimal" />
        <xs:attribute name="Power_dBm" type="xs:decimal" />
        <xs:attribute name="Power_Watts" type="xs:decimal" />
        <xs:attribute name="Squelched" type="xs:boolean" default="false" />
        <xs:attribute name="Unleveled" type="xs:boolean" default="false" />
        <xs:attribute name="CheckTx" type="xs:boolean" default="false" />
        <xs:attribute name="ToneOn" type="xs:boolean" />
        <xs:attribute name="TempC" type="xs:decimal" />
    </xs:complexType>

    <!-- PIMReceive: Contains all relevant data for a PIM receiver including frequency, measured
power, and peak power -->
    <xs:complexType name="PIMRX">
        <xs:attribute name="Frequency" type="xs:decimal" />
        <xs:attribute name="Power_dBm" type="xs:decimal" />
        <xs:attribute name="Power_dBc" type="xs:decimal" />
        <xs:attribute name="PeakPower_dBm" type="xs:decimal" />
        <xs:attribute name="PeakPower_dBc" type="xs:decimal" />
        <xs:attribute name="IM_Order" type="xs:integer" />
    </xs:complexType>

    <!-- PIMError: Contains source and error code information for both errors and warnings -->
    <xs:complexType name="PIMError">
        <xs:sequence>
            <xs:element name="Message" type="xs:string" />
        </xs:sequence>
        <xs:attribute name="Error" type="xs:boolean" default="false" />
        <xs:attribute name="Code" type="xs:integer" default="0" />
    </xs:complexType>

    <!-- PIMConnection: This response is used to present a user with an authorized command Token,
as well as a list of devices visible to the API -->
    <xs:complexType name="PIMConnection">
        <xs:sequence>
            <xs:element name="Token" type="xs:string" />
            <xs:element name="System" type="PIMSystem" minOccurs="0" maxOccurs="unbounded"
/>
        </xs:sequence>
    </xs:complexType>

    <!-- PIMSystem: Contains the properties of a system connected to the API engine -->
    <xs:complexType name="PIMSystem">
        <xs:attribute name="Name" type="xs:string" />
```

```
            <xs:attribute name="Legacy" type="xs:boolean" />
            <xs:attribute name="BandIndex" type="xs:integer" />
        </xs:complexType>

    </xs:schema>
```

The above schema definition clearly defines the PIM command/response model, by identifying which attributes and elements are legally allowed to be sent to and from the PIM API web service. As each communication is required to live within an XML document, each command/response must be formatted with an appropriate XML body. An example of a properly formatted PIM XML document is shown here:

```
<?xml version="1.0"?>
<PIM xmlns="http://www.kaelus.com"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.kaelus.com PIMXMLSchema.xsd">

 ...

</PIM>
```

The commands sent and responses received to and from the PIM API service will also follow the formatting defined.

6.2. XML Command Format

In order to be authenticated, commands must be issued using a valid API token key, which can be obtained using the methods described earlier in this document. The other optional attribute which can be sent with a set of commands is the Legacy attribute. If TRUE, the Legacy attribute tells the API web service to issue these commands to an older PIM hardware engine (use this method to communicate to older A/B series PIM analyzers). The Legacy attribute is completely optional, and if left out of the command string, the API assumes communication with modern PIM analyzers through the PIM Web Engine application.

Commands may be issued with or without parameters, however some commands will require certain parameter settings to be properly executed (see command definitions below). An example of a properly formatted command is shown here:

```
...
  <Commands Token="[API Token]" Legacy="[TRUE/FALSE]">

    <!-- Comments may appear within a command string if formatted as a valid XML
```

```
comment -->

   <!-- The following command has no parameters -->
   <Command Action="[NoParamCommand]" />

   <!-- The following command requires 4 parameters of type [DataType] to be issued --
>
   <Command Action="[ParamsCommand]">
    <[DataType]>[Data]</[DataType]>
    <[DataType]>[Data]</[DataType]>
    <[DataType]>[Data]</[DataType]>
    <[DataType]>[Data]</[DataType]>
   </Command>

  </Commands>

...
```

Please note that commands must be issued within a valid PIM XML document (as described above) in order to be properly parsed and executed. Commands which are not properly formatted will be strictly ignored by the PIM API web service, resulting in an empty response being returned.

Following the flow of the XML document from top to bottom, commands will be executed in the order in which they are received (following traditional batch command standards).

6.3. XML Response Format

Responses received from the API web service will follow the format described in the XML schema definition. Thus ensuring that data received from the API web service can easily be interpreted and managed within custom applications. Response data can take two forms, either a command response, or a connection response.

Command Responses represent the outcome of issued commands, and are returned in the order in which the commands were received. Each response-set will contain a single Legacy attribute, indicating whether the commands were executed on a modern or a legacy PIM analyzer. While several responses will return only confirmation of their execution, some may contain response parameters (which is how the API will return data to custom applications). Each response also contains a Units field for convenient identification of data-format (for example: "MHz", "dBm", "dBc", etc.). The PIM response data will be returned in the following format:

```
...

   <Responses Legacy="[TRUE/FALSE]">

     <!-- The following response has no parameters -->
    <Response Action="[NoParamCommand]" Units="" />

    <!-- The following response includes 4 response parameters of type [DataType] -->
    <!-- NOTE: These are the data returned based on the issued command -->
```

```
    <Response Action="[ParamsCommand]" Units="[DataFormat]">
      <[DataType]>[Data]</[DataType]>
      <[DataType]>[Data]</[DataType]>
      <[DataType]>[Data]</[DataType]>
      <[DataType]>[Data]</[DataType]>
    </Response>

  </Responses>

...
```

Connection Responses follow a similar format, though they contain completely different data. The connection-response will return the API token, which is required for command authentication, as well as a list of available analyzers which are visible to the target PIM Web Engine. The PIM connection response will be returned in the following format:

```
...

<Connection>

  <Token>[HexidecimalToken]</Token>

  <!-- The following analyzers were detected on your system -->
  <Analyzer Name="[AnalyzerModel]" Legacy="[TRUE/FALSE]" BandIndex="[index]" />
  <Analyzer Name="[AnalyzerModel]" Legacy="[TRUE/FALSE]" BandIndex="[index]" />



  ...

</Connection>



...
```

6.4. XML Data Types

In order to allow common data types to be transmitted between the PIM Web Engine and the various other programming languages used by the PIM development community, the PIM XML Schema Definition (PIMXMLSchema.xsd) defines various data types which are used with each command and response sent to and from the PIM API. The following data types are used by PIM API commands and responses to communicate data values between various programming languages.

> The following data-types are based on XML Schema defined data types, therefore performing validation of a command-set against the PIMXMLSchema.xsd will ensure that data entered in

the corresponding data fields is valid according to their data-type definition.

Double
Double-precision floating point number. This data type is used to set and retrieve floating point values used throughout the PIM API (i.e. frequencies, powers, etc.)

Integer
Signed 32-bit integer. This data type is used to set and retrieve integer and enumeration values used throughout the PIM API (i.e. indices, enumeration values, etc.)

Boolean
true/false OR 0/1. This data type is used to set and retrieve boolean values used throughout the PIM API (i.e. TX on, ALC on, VAhold on, etc.)

String
UTF-8. This data type is used to set and retrieve string values used throughout the PIM API. String values may represent single values (i.e. Messages, Enumeration Descriptions, etc.), though some String responses will contain XML encoded data (i.e. getTestSetDef, or getAnalyzers).

Sample
A complex data-type consisting of data and attributes related to a measurement sample. Attributes of a Sample include **Count** (sample # in a set), **Total** (total samples in this set), **Finished** (final sample in this set), **F1** (Base carrier 1 frequency for this sample), **F2** (Base carrier 2 frequency for this sample), **FStep**(Transmitter frequency step size for this sample set), **SweepLeg** (F1Up, or F2Down). A sample also contains the following data elements:

TX1 (single value only)
Attributes include **Frequency** (Frequency of this carrier), **Power_dBm**(Indicated power in dBm), **Power_Watts** (Indicated power in Watts),**Squelched** (Carrier exceeded squelch warning condition?), **Unleveled**(Problems leveling the requested carrier power within 0.35 dB spec),**CheckTx** (indicated carrier power unresponsive), **ToneOn** (RF Power present on this carrier?), and **TempC** (indicated amplifier temperature in degrees Celsius).

TX2 (single value only)
Attributes include **Frequency** (Frequency of this carrier), **Power_dBm**(Indicated power in dBm), **Power_Watts** (Indicated power in Watts),**Squelched** (Carrier exceeded squelch warning condition?), **Unleveled**(Problems leveling the requested carrier power within 0.35 dB spec),**CheckTx** (indicated carrier power unresponsive), **ToneOn** (RF Power present on this carrier?), and **TempC** (indicated amplifier temperature in degrees Celsius).

IM (one or more values)
Attributes include **Frequency** (IM frequency measured), **Power_dBm**(power in dBm received at this frequency), **Power_dBc** (power in dBc received at this frequency), **PeakPower_dBm** (peak dBm measured to this sample), **PeakPower_dBc** (peak dBc measured to this sample), **IM_Order**(Base IM product analyzed)

Status
A complex data-type consisting of two attributes (Error, and Code) and a Message.

The **Error** attribute is a Boolean value indicating whether the status returned represents an error state (TRUE) or just a warning (FALSE). The **Code** attribute is an Integer value which holds the error code representative of the error or warning that has occurred.
The **Message** returned is simply a string value which provides a human readable description of the error which has occurred. The Status data-type is returned with each response which has encountered an error or warning in performing the desired action.

7. Commands and Interfaces

7.1. API Methods

7.1.1. GetAnalyzers

Description: Get a list of Available analyzers and their indices, which can be selected for operation using the **SetMeasBand** command. The Analyzers returned include an analyzer name, whether it is a Legacy analyzer, and its Band index.

COMMAND	RESPONSE
<Command Action="GetAnalyzers" />	<Response Action="GetAnalyzers" Units=""> <String> <Analyzer Name="[*Analyzer Model*]" Legacy="[*TRUE/FALSE*]" BandIndex="[*integer*]" /> ... </String> </Response>

7.1.2. ResetPeaks

Description: Reset the Peak IM values being tracked by the PIM Hardware Component. These values can be retrieved by performing either a **GetImPeakPower** or**GetImPeakPowerDbc** command.

COMMAND	RESPONSE
<Command Action="ResetPeaks" />	<Response Action="ResetPeaks" Units="" />

7.1.3. SetExit

Description: Close the connection with the PIM controller and cause the hardware component(s) to exit. Note that the carriers are turned off automatically when the hardware component is closed.

COMMAND	RESPONSE
<Command Action="SetExit" />	<Response Action="SetExit" Units="" />

7.1.4. SetInit

Description: Perform the hardware initialization process, connecting to each PIM device module, and loading the appropriate PIM analyzer calibration file(s).

COMMAND	RESPONSE
<Command Action="SetInit" />	<Response Action="SetInit" Units="" />

7.1.5. SetMeasBand

Description: Select a PIM analyzer to connect to by providing its BandIndex (provided during the api/connect operation).

COMMAND	RESPONSE
<Command Action="SetMeasBand"> <Integer>[*index*]</Integer> </Command>	<Response Action="SetMeasBand" Units="" />

7.1.6. SetModeStandard

Description: Activate the standard hardware measurement mode. This command optimizes hardware for fixed frequency pair measurements, limiting synthesizer tuning during measurement to optimize measurement speed. Use this measurement mode to simulate spectral or strip-chart measurements as shown in the VFP.

COMMAND	RESPONSE

<Command Action="SetModeStandard" />	<Response Action="SetModeStandard" Units="" />

7.1.7. SetModeSweepTx

Description: Activate the swept TX frequency hardware measurement mode. Use this command to configure a swept frequency operation by setting F1, F2, FStep, and FullSweep variables. In this measurement mode, the hardware will automatically sweep carrier 2 down toward carrier 1 (fixed at F1) starting at F2 and decrementing in FStep intervals, and will then sweep carrier 1 up toward carrier 2 (fixed at F2) starting at F1 and incrementing in FStep intervals. Passing a "TRUE" value for FullSweep will command the hardware to complete both F2Down and F1Up sweeps automatically and return all sweep data as a single response with multiple Samples.

COMMAND	RESPONSE
<Command Action="SetModeStandard"> <Double>[*FStep*]</Double> <Double>[*F1*]</Double> <Double>[*F2*]</Double> <Boolean>[*FullSweep*]</Boolean> </Command>	<Response Action="SetModeSweepTx" Units="" />

7.1.8. SetPreset

Description: Perform an initial configuration of the analyzer to preset it to its default operation settings (Carrier frequencies to band edges (for this analyzer), Carrier power set at 43.0 dBm, Carrier Power OFF, ALC ON, Normal Averaging, Default Settling Time, Port 1 Reverse (P11)).

COMMAND	RESPONSE
<Command Action="SetPreset" />	<Response Action="SetPreset" Units="" />

7.1.9. SetTimeout

Description: Set the PIM Driver timeout period (default is 60 seconds) which will determine the amount of time the PIM analyzer may sit idle without receiving commands before it closes itself from memory (NOTE: once the driver has closed, it must be reinitialized again before it can be of any use).

COMMAND	RESPONSE
<Command Action="SetTimeout"> <Integer>[*Timeout in Seconds*]</Integer> </Command>	<Response Action="SetTimeout" Units="seconds"> <Integer>[*Timeout in Seconds*]</Integer> </Response>

7.1.10. SetTrigger

Description: Force the hardware to perform a single measurement with the current configuration. The following optional parameters may be set for this command as of release version 11.2.2 of the BPIM Web Engine: [ForceMeasure] is a boolean variable which when set to TRUE will force the hardware to tune all synthesizers even if settings have not changed. [hwTimeout] represents the amount of time (in whole seconds ranging between 10 and 300 seconds) that the hardware will wait before timing out during a measurement (NOTE: This is useful when measuring full sweep responses (see setModeSweepTx above)).

COMMAND	RESPONSE
<Command Action="SetTrigger"> <Boolean>[*ForceMeasure*]</Boolean> <Integer>[*hwTimeout*]</Integer> </Command>	<Response Action="SetTrigger" Units="" />

7.2. Accessors

7.2.1. GetCalInfo

Description: Request the calibration information for the selected analyzer. This command will return the calibration version, date, and serial number(s) for the connected analyzer.

COMMAND	RESPONSE
<Command Action="GetCalInfo" />	<Response Action="GetCalInfo" Units=""> <Integer>[*version*]</Integer> <String>[*calibration date*]</String> <String>[*serial number(s)*]</String> </Response>

7.2.2. GetError

Description: Request the error state of the selected analyzer. If the PIM Web Engine detects any errors communicating with the selected analyzer, they can be retrieved through this interface. Errors such as hardware failure conditions, invalid calibration files, and missing devices can be detected using this command.

COMMAND	RESPONSE
<Command Action="GetError" />	<Response Action="GetError" Units=""> <Status Error="[*TRUE/FALSE*]" Code="[*ErrorCode*]"> [*Error Message*] </Status> </Response>

7.2.3. GetImFreqs

Description: This command will return all IM frequencies currently being measured. Frequencies will be returned by ascending IM order, starting with the base IM Order (see command **SetImOrder**). If measuring only Single IM then only a single IM frequency will be returned, otherwise the function will return four IM frequencies.

COMMAND	RESPONSE
<Command Action="GetImFreqs" />	<Response Action="GetImFreqs" Units="MHz"> <Double>[*frequency*]</Double>

	<Double>[*frequency*]</Double> <Double>[*frequency*]</Double> <Double>[*frequency*]</Double> </Response>

7.2.4. GetImOrder

Description: This command will return the current base IM Order (i.e. 3rd, 5th, 7th, etc.). All measurements are performed against frequencies based on this IM Order.

COMMAND	RESPONSE
<Command Action="GetImOrder" />	<Response Action="GetImOrder" Units=""> <Integer>[*IMOrder {3, 5, 7, ...}*]</Integer> <String>[*IMOrder String {3rd, 5th, 7th, ...}*]</String> </Response>

7.2.5. GetImPower

Description: This command will return the measured IM power in dBm units for each IM frequency being measured. Powers will be returned by ascending IM order, starting with the base IM order (see command **SetImOrder**). If measuring only Single IM then only one IM power will be returned, otherwise the function will return four IM powers.

COMMAND	RESPONSE
<Command Action="GetImPower" />	<Response Action="GetImPower" Units="dBm"> <Double>[*power*]</Double> <Double>[*power*]</Double> <Double>[*power*]</Double> <Double>[*power*]</Double> </Response>

7.2.6. GetImPowerDbc

Description: This command will return the measured IM power in dBc units for each IM frequency being measured. Powers will be returned by ascending IM order, starting with the base IM order (see command **SetImOrder**). If measuring only Single IM then only one IM power will be returned, otherwise the function will return four IM powers.

COMMAND	RESPONSE

<Command Action="GetImPowerDbc" />	<Response Action="GetImPowerDbc" Units="dBc"> <Double>[*power*]</Double> <Double>[*power*]</Double> <Double>[*power*]</Double> <Double>[*power*]</Double> </Response>

7.2.7. GetImPeakPower

Description: This command will return the peak measured IM power in dBm units for each IM frequency being measured. Powers will be returned by ascending IM order, starting with the base IM order (see command **SetImOrder**). If measuring only Single IM then only one IM power will be returned, otherwise the function will return four IM powers.

COMMAND	RESPONSE
<Command Action="GetImPeakPower" />	<Response Action="GetImPeakPower" Units="dBm"> <Double>[*power*]</Double> <Double>[*power*]</Double> <Double>[*power*]</Double> <Double>[*power*]</Double> </Response>

7.2.8. GetImPeakPowerDbc

Description: This command will return the peak measured IM power in dBc units for each IM frequency being measured. Powers will be returned by ascending IM order, starting with the base IM order (see command **SetImOrder**). If measuring only Single IM then only one IM power will be returned, otherwise the function will return four IM powers.

COMMAND	RESPONSE
<Command Action="GetImPeakPowerDbc" />	<Response Action="GetImPeakPowerDbc" Units="dBc"> <Double>[*power*]</Double> <Double>[*power*]</Double> <Double>[*power*]</Double> <Double>[*power*]</Double> </Response>

7.2.9. GetImStatus

Description: This command will return TRUE if the receiver is experiencing an RX Overload error.

COMMAND	RESPONSE
<Command Action="GetImStatus" />	<Response Action="GetImStatus" Units=""> <Boolean>[*IMOverload TRUE/FALSE*]</Boolean> </Response>

7.2.10. GetSamples

Description: Whether measuring using Standard sample mode (**SetModeStandard**), or Swept Frequency Mode (**SetModeSweepTx**), the GetSamples command returns the latest collection of sample data from the BPIM Web Engine. This command typically follows a **SetTrigger** command which triggers the measurement operation to be performed. Running this command multiple times will return the same data if a trigger operation has not occurred.

COMMAND	RESPONSE
<Command Action="GetSamples" />	<Response Action="GetSamples" Units=""> <Sample Count="1" Total="1" Finished="TRUE" F1="[double]" F2="[double]"> <TX1 Frequency="[double]" Power_dBm="[double]" Power_Watts="[double]" Squelched="[boolean]" Unleveled="[boolean]" CheckTx="[boolean]" ToneOn="[boolean]" TempC="[double]" /> <TX2 Frequency="[double]" Power_dBm="[double]" Power_Watts="[double]" Squelched="[boolean]" Unleveled="[boolean]" CheckTx="[boolean]" ToneOn="[boolean]" TempC="[double]" /> <IM Frequency="[double]" Power_dBm="[double]" Power_dBc="[double]" PeakPower_dBm="[double]" PeakPower_dBc="[double]" IM_Order="[integer]" /> ... <!-- A sample may contain multiple IM products when SingleIM is disabled --> </Sample> ... <!-- A response may contain multiple samples if performing a FullSweep operation --> </Response>

7.2.11. GetSweepStep

Description: This command will return the current Transmitter FStep used in sweep mode operations.

COMMAND	RESPONSE
<Command Action="GetSweepStep" />	<Response Action="GetSweepStep" Units=""> <Double>[*Transmitter Step Size*]</Double> </Response>

7.2.12. GetTestSetDef

Description: This command will return the PIM TestSet Definition following the customized data-type format shown below.

COMMAND	RESPONSE
<Command Action="GetTestSetDef" />	<Response Action="GetTestSetDef" Units=""> <String> <TestSetDef> <General> <ID>[*TestSetId*]</ID> <Model>[*TestSetModel*]</Model> <Dual_Port>[*TRUE/FALSE*]</Dual_Port> <Options_String>[*Options String*]</Options_String> <Options> <E>[*TRUE/FALSE*]</E> <K>[*TRUE/FALSE*]</K> <H>[*TRUE/FALSE*]</H> <W>[*TRUE/FALSE*]</W> <F>[*TRUE/FALSE*]</F> </Options> <Cal_File>[*Calibration File Path*]</Cal_File> </General> <Receiver> <Fstart_MHz>[*double*]</Fstart_MHz> <Fstop_MHz>[*double*]</Fstop_MHz> <Fstep_MHz>[*double*]</Fstep_MHz> <IF_MHz>[*double*]</IF_MHz> </Receiver> <RX_Band> <Fstart_MHz>[*double*]</Fstart_MHz> <Fstop_MHz>[*double*]</Fstop_MHz> </RX_Band> <TX_1> <Fstart_MHz>[*double*]</Fstart_MHz> <Fstop_MHz>[*double*]</Fstop_MHz> <Pmax_dBm>[*double*]</Pmax_dBm> <Pstep_dB>[*double*]</Pstep_dB> </TX_1> <TX_2>

	<Fstart_MHz>[*double*]</Fstart_MHz> <Fstop_MHz>[*double*]</Fstop_MHz> <Pmax_dBm>[*double*]</Pmax_dBm> <Pstep_dB>[*double*]</Pstep_dB> </TX_2> <Low-side>[*TRUE/FALSE*]</Low-side> </TestSetDef> </String> </Response>

7.2.13. GetTxFreqs

Description: This command will return the current Carrier frequencies in MHz. Carrier frequencies can be set using either the **SetPreset** or the **SetTxFreqs** commands.

COMMAND	RESPONSE
<Command Action="GetTxFreqs" />	<Response Action="GetTxFreqs" Units="MHz"> <Double>[*Carrier1 Frequency*]</Double> <Double>[*Carrier2 Frequency*]</Double> </Response>

7.2.14. GetTxOn

Description: This command will return the current Carrier transmit status (On/Off). Carrier power can be applied using the **SetTxOn** command.

COMMAND	RESPONSE
<Command Action="GetTxOn" />	<Response Action="GetTxOn" Units=""> <Boolean>[*Carrier1 Power Applied*]</Boolean> <Boolean>[*Carrier2 Power Applied*]</Boolean> </Response>

7.2.15. GetTxPower

Description: This command will return the current Carrier power levels (in dBm). Carrier powers are requested using the **SetTxPower** command.

COMMAND	RESPONSE
<Command Action="GetTxPower" />	<Response Action="GetTxPower" Units="dBm"> <Double>[*Carrier1 Power*]</Double> <Double>[*Carrier2 Power*]</Double> </Response>

7.2.16. GetTxStatus

Description: This command will return the Carrier Power Status alarms to identify when a carrier is experiencing problems. The command returns boolean status for each carrier when power is applied, for the following conditions: Check TX (Measured power is more than 2 dB different than requested power), Unleveled (ALC is on, and Measured power is greater than 0.35 dB different than requested power), and Squelched (Both Requested and Measured Powers are less than the Squelch Limit (20 dBm)).

COMMAND	RESPONSE
<Command Action="GetTxStatus" />	<Response Action="GetTxStatus" Units=""> <Boolean>[*Carrier1 CheckTx*]</Boolean> <Boolean>[*Carrier1 Unleveled*]</Boolean> <Boolean>[*Carrier1 Squelched*]</Boolean> <Boolean>[*Carrier2 CheckTx*]</Boolean> <Boolean>[*Carrier2 Unleveled*]</Boolean> <Boolean>[*Carrier2 Squelched*]</Boolean> </Response>

7.2.17. GetTxRange

Description: This command will return the Carrier transmit range for both Carrier 1 and Carrier 2 of the connected analyzer.

COMMAND	RESPONSE
<Command Action="GetTxRange" />	<Response Action="GetTxRange" Units="MHz"> <Double>[*Carrier1 Start Frequency*]</Double> <Double>[*Carrier1 Stop Frequency*]</Double> <Double>[*Carrier2 Start Frequency*]</Double> <Double>[*Carrier2 Stop Frequency*]</Double> </Response>

7.2.18. GetRxRange

Description: This command will return the Receive Band for the connected analyzer.

COMMAND	RESPONSE
<Command Action="GetRxRange" />	<Response Action="GetRxRange" Units="MHz"> <Double>[*Receiver Band Start Frequency*]</Double> <Double>[*Receiver Band Stop Frequency*]</Double> </Response>

7.2.19. SetAlc

Description: This command is used to enable or disable the Automatic Leveling Control. By Default, ALC is turned ON (TRUE). It is highly recommended that the ALC remain ON during measurement operations, as this feature enhances carrier transmit power accuracies.

COMMAND	RESPONSE
<Command Action="SetAlc"> <Boolean>[*TRUE/FALSE*]</Boolean> </Command>	<Response Action="SetAlc" Units="" />

7.2.20. SetDutPort

Description: This command is used with multi-port enabled analyzers (option D) to select which port should be used for measurement. Note that port selection cannot occur when power is being applied, and so power will automatically be shut off if a port switch is commanded during measurement. Valid values for DUT Port are as follows: **0** - Port 1, **1** - Port 2.

COMMAND	RESPONSE
<Command Action="SetDutPort"> <Integer>[*0,1*]</Integer> </Command>	<Response Action="SetDutPort" Units="" />

7.2.21. SetImAvg

Description: Set the IM Averaging to be applied during measurements. IM averaging levels effect the number of measurements which are averaged together for an IM power measurement. Valid values for IM Averaging are as follows: **0** - None, **1** - Minimal, **2** - Normal, **3** - High, **4** - Maximum.

COMMAND	RESPONSE
<Command Action="SetImAvg"> <Integer>[*0..4*]</Integer> </Command>	<Response Action="SetImAvg" Units="" />

7.2.22. SetImMode

Description: Set the IM Measurement Direction (Reverse/Forward). Valid values for IM Mode are as follows: **0** - Reverse, **1** - Forward.

COMMAND	RESPONSE
<Command Action="SetImMode"> <Integer>[*0,1*]</Integer> </Command>	<Response Action="SetImMode" Units="" />

7.2.23. SetImOrder

Description: Set the base IM Order to be measured against. Valid IM Orders are odd-ordered IM products ranging from 3 to 33 (i.e. **3** - IM3, **5** -IM5, etc.).

COMMAND	RESPONSE
<Command Action="SetImOrder"> <Integer>[*3..33 (odds only)*]</Integer> </Command>	<Response Action="SetImOrder" Units="" />

7.2.24. SetPxx

Description: Set the PIM Parameter to be measured. This command will set both the IM Mode and the DUT Port simultaneously. Valid values are as follows: **0** - P11 (Port 1 Reverse), **1** - P12 (Port 2 Forward), **2 -** P21 (Port 1 Forward), **3** - P22 (Port 2 Reverse).

COMMAND	RESPONSE
<Command Action="SetPxx"> <Integer>[*0..3*]</Integer> </Command>	<Response Action="SetPxx" Units="" />

7.2.25. SetSettlingTime

Description: Set the receiver time to settle (in miliseconds) before taking measurements. Default is **0**, which forces the receiver to measure as quickly as possible.

COMMAND	RESPONSE
<Command Action="SetSettlingTime"> <Integer>[*0..*]</Integer>	<Response Action="SetSettlingTime" Units="" />

</Command>	

7.2.26. SetSingleIm

Description: Set the selected analyzer to take measurements at either a single IM frequency (Single IM = **TRUE**) or four sequential IM frequencies (Single IM = **FALSE**). Measurements are based on the IM order which is set by issuing a **SetImOrder**command.

COMMAND	RESPONSE
<Command Action="SetSingleIm"> <Boolean>[*TRUE/FALSE*]</Boolean> </Command>	<Response Action="SetSingleIm" Units="" />

7.2.27. SetSweepStep

Description: Set the TX Step size (in MHz) for SweepTx measurement operations. This value can also be set as a variable in the **SetModeSweepTx** command.

COMMAND	RESPONSE
<Command Action="SetSweepStep"> <Double>[*Sweep Step Size*]</Double> </Command>	<Response Action="SetSweepStep" Units="" />

7.2.28. SetTxFreqs

Description: Set the carrier frequencies for the selected analyzer. Note that frequencies are limited based on the defined TX Range for each carrier (TX Range can be obtained by executing the **GetTxRange** command).

COMMAND	RESPONSE
<Command Action="SetTxFreqs"> <Double>[*Carrier1 Frequency*]</Double> <Double>[*Carrier2 Frequency*]</Double> </Command>	<Response Action="SetTxFreqs" Units="" />

7.2.29. SetTxOn

Description: Set the carrier amplifier powers to **TRUE** - On, or **FALSE** - Off. Call this command with no parameters to force carrier power OFF.

COMMAND	RESPONSE
<Command Action="SetTxOn"> <Boolean>[*Carrier1 Power On*]</Boolean> <Boolean>[*Carrier2 Power On*]</Boolean> </Command>	<Command Action="SetTxOn" Units="" />

7.2.30. SetTxPower

Description: Set the requested carrier power levels which should be achieved when carrier power is applied.

COMMAND	RESPONSE
<Command Action="SetTxPower"> <Double>[*Carrier1 Power*]</Double> <Double>[*Carrier2 Power*]</Double> </Command>	<Response Action="SetTxPower" Units="" />

7.2.31. SetVaHold

Description: Set the Variable Attenuators to hold at their current settings. Setting this value to **TRUE** forces the Variable Attenuators to hold their current power output levels while frequency parameters are changed. Please note that this setting may produce inaccurate results at individual frequencies as power output varies by frequency.

COMMAND	RESPONSE
<Command Action="SetVaHold"> <Boolean>[*TRUE/FALSE*]</Boolean> </Command>	<Response Action="SetVaHold" Units="" />

This Page Intentionally Left Blank