Chapter 3 - Synthesized Signal Generators

3.1 - Operating in a Windows Environment

3.1.1 - Referencing the DLL Library

3.1.2 - Summary of DLL Functions

3.1.3 - Detailed Description of DLL Functions

3.1.3.1 - Connect to Signal Generator

3.1.3.2 - Connect to Signal Generator by Address

3.1.3.3 - Disconnect from Signal Generator

3.1.3.4 - Read Model Name of Signal Generator

3.1.3.5 - Read Serial Number of Signal Generator

3.1.3.6 - Set Address of Signal Generator

3.1.3.7 - Get Address of Signal Generator

3.1.3.8 - Get List of Connected Serial Numbers

3.1.3.9 - Get List of Available Addresses

3.1.3.10 - Turn On RF Output

3.1.3.11 - Turn Off RF Output

3.1.3.12 - Set Output Frequency and Power

3.1.3.13 - Set Output Frequency

3.1.3.14 - Set Output Power

3.1.3.15 - Get Generator Output Status

3.1.3.16 - Set Low Noise or Low Spur Mode

3.1.3.17 - Check External Reference

3.1.3.18 - Get Trigger In Status

3.1.3.19 - Set Pulse Mode

3.1.3.20 - Set Triggered Pulse Mode

3.1.3.21 - Get Generator Maximum Frequency Spec

3.1.3.22 - Get Generator Minimum Frequency Spec

3.1.3.23 - Get Generator Step Size Spec

3.1.3.24 - Get Generator Maximum Power Spec

3.1.3.25 - Get Generator Minimum Power Spec

3.1.3.26 - Get Temperature of Signal Generator

3.1.3.27 - Check Connection

3.1.3.28 - Get Firmware Version

3.1.3.29 - Frequency Sweep - Get Direction

3.1.3.30 - Frequency Sweep - Get Dwell Time

3.1.3.31 - Frequency Sweep - Get Maximum Dwell Time

3.1.3.32 - Frequency Sweep - Get Minimum Dwell Time

3.1.3.33 - Frequency Sweep - Get Power

3.1.3.34 - Frequency Sweep - Get Start Frequency

3.1.3.35 - Frequency Sweep - Get Stop Frequency

3.1.3.36 - Frequency Sweep - Get Step Size

3.1.3.37 - Frequency Sweep - Get Trigger In Mode

3.1.3.38 - Frequency Sweep - Get Trigger Out Mode

3.1.3.39 - Frequency Sweep - Set Direction

3.1.3.40 - Frequency Sweep - Set Dwell Time

3.1.3.41 - Frequency Sweep - Start/Stop Sweep

3.1.3.42 - Frequency Sweep - Set Power
3.1.3 (43) - Frequency Sweep – Set Start Frequency ........................................ 3-51
3.1.3 (44) - Frequency Sweep – Set Stop Frequency ....................................... 3-52
3.1.3 (45) - Frequency Sweep – Set Step Size .............................................. 3-53
3.1.3 (46) - Frequency Sweep – Set Trigger In Mode .................................. 3-54
3.1.3 (47) - Frequency Sweep – Set Trigger Out Mode ................................ 3-55
3.1.3 (48) - Frequency/Power Hop – Get Direction...................................... 3-56
3.1.3 (49) - Frequency/Power Hop – Get Dwell Time .................................... 3-57
3.1.3 (50) - Frequency/Power Hop – Get Maximum Dwell Time .................. 3-58
3.1.3 (51) - Frequency/Power Hop – Get Minimum Dwell Time ................... 3-59
3.1.3 (52) - Frequency/Power Hop – Get Maximum Number of Points ........... 3-60
3.1.3 (53) - Frequency/Power Hop – Get Hop Point ...................................... 3-61
3.1.3 (54) - Frequency/Power Hop – Get Trigger In Mode ............................. 3-62
3.1.3 (55) - Frequency/Power Hop – Get Trigger Out Mode ......................... 3-63
3.1.3 (56) - Frequency/Power Hop – Set Direction ....................................... 3-64
3.1.3 (57) - Frequency/Power Hop – Set Dwell Time ................................... 3-65
3.1.3 (58) - Frequency/Power Hop – Start/Stop Hop Sequence ....................... 3-66
3.1.3 (59) - Frequency/Power Hop – Set Number of Points ......................... 3-67
3.1.3 (60) - Frequency/Power Hop – Set Hop Point ..................................... 3-68
3.1.3 (61) - Frequency/Power Hop – Set Trigger In Mode ............................. 3-69
3.1.3 (62) - Frequency/Power Hop – Set Trigger Out Mode ......................... 3-70
3.1.3 (63) - Power Sweep – Get Direction ..................................................... 3-71
3.1.3 (64) - Power Sweep – Get Dwell Time .................................................. 3-72
3.1.3 (65) - Power Sweep – Get Maximum Dwell Time ................................. 3-73
3.1.3 (66) - Power Sweep – Get Minimum Dwell Time .................................. 3-74
3.1.3 (67) - Power Sweep – Get Frequency .................................................... 3-75
3.1.3 (68) - Power Sweep – Get Start Power .................................................. 3-76
3.1.3 (69) - Power Sweep – Get Stop Power ................................................... 3-77
3.1.3 (70) - Power Sweep – Get Step Size ..................................................... 3-78
3.1.3 (71) - Power Sweep – Get Trigger In Mode ......................................... 3-79
3.1.3 (72) - Power Sweep – Get Trigger Out Mode ...................................... 3-80
3.1.3 (73) - Power Sweep – Set Direction ..................................................... 3-81
3.1.3 (74) - Power Sweep – Set Dwell Time .................................................. 3-82
3.1.3 (75) - Power Sweep – Start/Stop Sweep .............................................. 3-83
3.1.3 (76) - Power Sweep – Set Frequency .................................................... 3-84
3.1.3 (77) - Power Sweep – Set Start Power .................................................. 3-85
3.1.3 (78) - Power Sweep – Set Stop Power ................................................... 3-86
3.1.3 (79) - Power Sweep – Set Step Size ...................................................... 3-87
3.1.3 (80) - Power Sweep – Set Trigger In Mode .......................................... 3-88
3.1.3 (81) - Power Sweep – Set Trigger Out Mode ...................................... 3-89

3.2 - Operating in a Linux Environment .............................................................. 3-90

3.2.1 - Summary of Commands ...................................................................... 3-91
3.2.2 - Detailed Description of Commands ..................................................... 3-92
  3.2.2 (1) - Get Device Model Name ............................................................. 3-92
  3.2.2 (2) - Get Device Serial Number ......................................................... 3-93
3.2.3 - Set Frequency and Power (SSG-4000 Series only) ............................ 3-94
  3.2.3 (1) - Set Frequency and Power (SSG-6000 only) ................................ 3-97
  3.2.3 (2) - Set Frequency (SSG-4000 Series Only) ..................................... 3-100
  3.2.3 (3) - Set Frequency (SSG-6000 Only) ................................................ 3-102
  3.2.3 (4) - Set Power .................................................................................. 3-104
  3.2.3 (5) - Set RF Power On/Off ................................................................. 3-106
<table>
<thead>
<tr>
<th>Process Number</th>
<th>Process Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2.3 (6)</td>
<td>Set Noise/Spur Mode</td>
<td>3-107</td>
</tr>
<tr>
<td>3.2.3 (7)</td>
<td>Get Generator Output Status (SSG-4000 Series)</td>
<td>3-108</td>
</tr>
<tr>
<td>3.2.3 (8)</td>
<td>Get Generator Output Status (SSG-6000 Only)</td>
<td>3-110</td>
</tr>
<tr>
<td>3.2.3 (9)</td>
<td>Get Generator Minimum Frequency</td>
<td>3-112</td>
</tr>
<tr>
<td>3.2.3 (10)</td>
<td>Get Generator Maximum Frequency (SSG-4000 Series Only)</td>
<td>3-114</td>
</tr>
<tr>
<td>3.2.3 (11)</td>
<td>Get Generator Maximum Frequency (SSG-6000 Only)</td>
<td>3-116</td>
</tr>
<tr>
<td>3.2.3 (12)</td>
<td>Get Generator Step Size</td>
<td>3-118</td>
</tr>
<tr>
<td>3.2.3 (13)</td>
<td>Get Generator Minimum Power</td>
<td>3-120</td>
</tr>
<tr>
<td>3.2.3 (14)</td>
<td>Get Generator Maximum Power</td>
<td>3-122</td>
</tr>
</tbody>
</table>
3.1 - Operating in a Windows Environment

3.1.1 - Referencing the DLL Library

The DLL file is installed in the host PC’s system folders using the steps outlined above. In order to use the DLL functionality, some programming environments will require the user to set a reference to the relevant file, usually through a built in GUI in the programming environment.

Once this is done, the user just needs to instantiate a new instance of the USB_Gen object in order to use the signal generator functions. The details of this vary greatly between programming environments and languages but Mini-Circuits can provide detailed support on request. A new signal generator object would need to be initialized for every USB signal generator that the user wishes to control. In the following examples, MyPTE1 and MyPTE2 will be used as names of 2 declared generator objects.

Examples

**Visual Basic**

```vbnet
Public MyPTE1 As New MCL_Gen.USB_Gen ' Initialize new generator object, assign to MyPTE1
Public MyPTE2 As New MCL_Gen.USB_Gen ' Initialize new generator object, assign to MyPTE2
```

**Visual C++**

```c++
usb_gen ^MyPTE1 = gcnew usb_gen;
  // Initialize new generator instance, assign to MyPTE1
usb_gen ^MyPTE2 = gcnew usb_gen;
  // Initialize new generator instance, assign to MyPTE2
```

**Visual C#**

```csharp
public MCL_Gen.USB_Gen MyPTE1 = new MCL_Gen.USB_Gen();
  // Initialize new generator instance, assign to MyPTE1
public MCL_Gen.USB_Gen MyPTE2 = new MCL_Gen.USB_Gen();
  // Initialize new generator instance, assign to MyPTE2
```

**Matlab**

```matlab
MyPTE1=actxserver('MCL_Gen.USB_Gen')
  % Initialize new generator instance, assign to MyPTE1
MyPTE2=actxserver('MCL_Gen.USB_Gen')
  % Initialize new generator instance, assign to MyPTE2
```
3.1.2 - Summary of DLL Functions

The following functions are defined in both of the DLL files. Please see the following sections for a full description of their structure and implementation.

1) Short Connect (Optional String SN)
2) Short ConnectByAddress (Optional Short Address)
3) Void Disconnect ()
4) Short Read_ModelName (String ModelName)
5) Short Read_SN (String SN)
6) Short Set_Address (Short Address)
7) Short Get_Address ()
8) Short Get_Available_SN_List (String SN_List)
9) Short Get_Available_Address_List (String Add_List)
10) Short SetPowerON ()
11) Short SetPowerOFF ()
12) Short SetFreqAndPower (Double Fr, Float Pr, Short TriggerOut)
13) Short SetFreq (Double Fr, Short TriggerOut)
14) Short SetPower (Float Pr, Short TriggerOut)
15) Short GetGenStatus(Byte Locked, Short PowerIsOn, Double Fr, Float pr, Short UNLEVELHigh, Short UNLEVELLow)
16) Short Set_Noise_Spur_Mode (Short nsm)
17) Short ExtRefDetected ()
18) Short GetTriggerln_Status ()
19) Short Set_PulseMode (Short T_OFF, Short T_ON, Short Tunit)
20) Short Set_PulseMode_Trigger (Short TriggerTyp, Short T_ON, Short Tunit)
21) Float GetGenMaxFreq ()
22) Float GetGenMinFreq ()
23) Float GetGenStepFreq ()
24) Float GetGenMaxPower ()
25) Float GetGenMinPower ()
26) Float GetDeviceTemperature ()
27) Short Check_Connection ()
28) Short GetFirmware ()
29) Short FSweep_GetDirection ()
30) Short FSweep_GetDwell ()
31) Short FSweep_GetMaxDwell ()
32) Short FSweep_GetMinDwell ()
33) Float FSweep_GetPower ()
34) Double FSweep_GetStartFreq ()
35) Double FSweep_GetStopFreq ()
36) Double FSweep_GetStepSize ()
37) Short FSweep_GetTriggerIn ()
38) Short FSweep_GetTriggerOut ()
39) Short FSweep_SetDirection (Short SweepDirection)
40) Short FSweep_SetDwell (Short dwell_msec)
41) Short FSweep_SetMode (Short onoff)
42) Float FSweep_SetPower (Float Pr)
43) Short FSweep_SetStartFreq (Double Fr)
44) Short FSweep_SetStopFreq (Double Fr)
45) Short FSweep_SetStepSize (Double Fr)
46) Short FSweep_SetTriggerIn (Short SweepTriggerIn)
47) Short FSweep_SetTriggerOut (Short SweepTriggerOut)
48) Short Hop_GetDirection ()
49) Short Hop_GetDwell ()
50) Short Hop_GetMaxDwell ()
51) Short Hop_GetMinDwell ()
52) Short Hop_GetMaxNoOfPoints ()
53) Short Hop_GetPoint (Short PointNo, Double HopFreq, float HopPower)
54) Short Hop_GetTriggerIn ()
55) Short Hop_GetTriggerOut ()
56) Short Hop_SetDirection (Short HopDirection)
57) Short Hop_SetDwell (Short dwell_msec)
58) Short Hop_SetMode (Short onoff)
59) Short Hop_SetNoOfPoints (Short HopNoOfPoints)
60) Short Hop_SetPoint (Short PointNo, Double HopFreq, Float HopPower)
61) Short Hop_SetTriggerIn (Short HopTriggerIn)
62) Short Hop_SetTriggerOut (Short HopTriggerOut)
63) Short PSweep_GetDirection ()
64) Short PSweep_GetDwell ()
65) Short PSweep_GetMaxDwell ()
66) Short PSweep_GetMinDwell ()
67) Double PSweep_GetFreq ()
68) Float PSweep_GetStartPower ()
69) Float PSweep_GetStopPower ()
70) Float PSweep_GetStepSize ()
71) Short PSweep_GetTriggerIn ()
72) Short PSweep_GetTriggerOut ()
73) Short PSweep_SetDirection (Short SweepDirection)
74) Short PSweep_SetDwell (Short dwell_msec)
75) Short Psweep_SetMode (Short onoff)
76) Double PSweep_SetFreq (Float Fr)
77) Short PSweep_SetStartPower (Float Pr)
78) Short PSweep_SetStopPower (Float Pr)
79) Short PSweep_SetStepSize (Float Pr)
80) Short PSweep_SetTriggerIn (Short SweepTriggerIn)
81) Short PSweep_SetTriggerOut (Short SweepTriggerOut)
3.1.3 - Detailed Description of DLL Functions

3.1.3 (1) - Connect to Signal Generator

Declaration

\texttt{Short \texttt{Connect}(Optional \texttt{String \ SN})}

Description

This function is called to initialize the connection to a USB signal generator. If multiple generators are connected to the same computer, then the serial number should be included, otherwise this can be omitted. The connection process can take a few milliseconds so it is recommended that the connection be made once at the beginning of the routine and left open until the generator is no longer needed. The generator should be disconnected on completion of the program using the \texttt{Disconnect} function.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>SN</td>
<td>Optional. A string containing the serial number of the USB signal generator. Can be omitted if only one generator is connected but must be included otherwise.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>0</td>
<td>No connection was possible</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Connection successfully established</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Device already connected</td>
</tr>
</tbody>
</table>

Examples

\begin{verbatim}
Visual Basic
status = MyPTE1.Connect(SN)

Visual C++
status = MyPTE1->Connect(SN);

Visual C#
status = MyPTE1.Connect(SN);

Matlab
status = MyPTE1.Connect(SN)
\end{verbatim}

See Also

\begin{itemize}
  \item Connect to Signal Generator by Address
  \item Read Serial Number of Signal Generator
  \item Disconnect from Signal Generator
\end{itemize}
3.1.3 (2) - Connect to Signal Generator by Address

Declaration

Short ConnectByAddress(Optional Short Address)

Description

This function is called to initialize the connection to a USB signal generator by referring to a user defined address. The address is an integer number from 1 to 255 which can be assigned using the Set_Address function (the factory default is 255). The connection process can take a few milliseconds so it is recommended that the connection be made once at the beginning of the routine and left open until the generator is no longer needed. The generator should be disconnected on completion of the program using the Disconnect function.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>Address</td>
<td>Optional. A short containing the address of the USB signal generator. Can be omitted if only one signal generator is connected but must be included otherwise.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>0</td>
<td>No connection was possible</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Connection successfully established</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Device already connected</td>
</tr>
</tbody>
</table>

Examples

Visual Basic

```vbnet
status = MyPTE1.ConnectByAddress(5)
```

Visual C++

```cpp
status = MyPTE1->ConnectByAddress(5);
```

Visual C#

```csharp
status = MyPTE1.ConnectByAddress(5);
```

Matlab

```matlab
status = MyPTE1.connectByAddress(5)
```

See Also

Connect to Signal Generator
Get Address of Signal generator
Disconnect from Signal Generator
3.1.3 (3) - Disconnect from Signal Generator

Declaration

   Void Disconnect()

Description

This function is called to close the connection to the signal generator. It is strongly recommended that this function is used prior to ending the program. Failure to do so may result in a connection problem with the device. Should this occur, shut down the program and unplug the signal generator from the computer, then reconnect the signal generator before attempting to start again.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Examples

Visual Basic
   MyPTE1.Disconnect()

Visual C++
   MyPTE1->Disconnect();

Visual C#
   MyPTE1.Disconnect();

Matlab
   MyPTE1.Disconnect

See Also

Connect to Signal Generator
Connect to Signal Generator by Address
3.1.3 (4) - Read Model Name of Signal Generator

Declaration

\[ \text{Short Read\_ModelName(String ModelName)} \]

Description

This function is called to determine the Mini-Circuits part number of the connected signal generator. The user passes a string variable which is updated with the model name.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>ModelName</td>
<td>Required. A string variable that will be updated with the Mini-Circuits model name for the signal generator.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>0</td>
<td>Command failed</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

**Visual Basic**

```
If MyPTE1.Read_ModelName(ModelName) > 0 Then
    MsgBox ("The connected generator is " & ModelName) ' Display a message stating the model name
End If
```

**Visual C++**

```
if (MyPTE1->Read_ModelName(ModelName) > 0 )
{
    MessageBox::Show("The connected generator is " + ModelName);    // Display a message stating the model name
}
```

**Visual C#**

```
if (MyPTE1.Read_ModelName(ref(ModelName)) > 0 )
{
    MessageBox.Show("The connected generator is " + ModelName);    // Display a message stating the model name
}
```

**Matlab**

```
[status, ModelName]= MyPTE1.Read_ModelName(ModelName)
If status > 0 then
{    msgbox ('The connected generator is ', ModelName)
    % Display a message stating the model name
}
```

See Also

Read Serial Number of Signal Generator
3.1.3 (5) - Read Serial Number of Signal Generator

Declaration

\[ \text{Short Read_SN(String SN)} \]

Description

This function is called to determine the serial number of the connected signal generator. The user passes a string variable which is updated with the serial number.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>SN</td>
<td>Required. A string variable that will be updated with the Mini-Circuits serial number for the signal generator.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>0</td>
<td>Command failed</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

Visual Basic
\[
\text{If MyPTE1.Read_SN(SN) > 0 Then}
\text{\hspace{1em}MsgBox ("The connected generator is " \\ & SN) \hspace{1em} 'Display a message stating the serial number}
\text{\hspace{1em}End If}
\]

Visual C++
\[
\text{if (MyPTE1->Read_SN(SN) > 0 )}
\text{\hspace{1em}MessageBox::Show("The connected generator is " + SN); \hspace{1em} // Display a message stating the serial number}
\]

Visual C#
\[
\text{if (MyPTE1.Read_SN(ref(SN)) > 0 )}
\text{\hspace{1em}MessageBox.Show("The connected generator is " + SN); \hspace{1em} // Display a message stating the serial number}
\]

Matlab
\[
[\text{status, SN}] = \text{MyPTE1.Read_SN(SN)}
\]
\[
\text{if status > 0 then}
\text{\hspace{1em}msgbox('The connected generator is ', SN) \hspace{1em} \% Display a message stating the serial number}
\]

See Also

- Connect to Signal Generator
- Get List of Connected Serial Numbers
3.1.3 (6) - Set Address of Signal Generator

Declaration

\[
\text{Short } \text{Set Address}(\text{Short Address})
\]

Description

This function allows the internal address of the connected signal generator to be changed from the factory default of 255. This allows the user to connect by a short address rather than serial number in future.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>Address</td>
<td>Required. An integer value from 1 to 255</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>0</td>
<td>Command failed</td>
</tr>
<tr>
<td></td>
<td>Non zero</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

```visual_basic
status = MyPTE1.Set_Address(1)
```
```visual_c++
status = MyPTE1->Set_Address(1);
```
```visual_c#
status = MyPTE1.Set_Address(1);
```
```matlab
status = MyPTE1.Set_Address(1)
```

See Also

- Connect to Signal Generator by Address
- Get Address of Signal generator
- Get List of Available Addresses
3.1.3 (7) - Get Address of Signal generator

Declaration

    Short Get_Address ()

Description

This function returns the address of the connected signal generator.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1-255</td>
<td>Address of the signal generator</td>
</tr>
</tbody>
</table>

Examples

Visual Basic
    addr = MyPTE1.Get_Address()

Visual C++
    addr = MyPTE1->Get_Address();

Visual C#
    addr = MyPTE1.Get_Address();

Matlab
    addr = MyPTE1.Get_Address

See Also

Connect to Signal Generator by Address
Set Address of Signal Generator
Get List of Available Addresses
3.1.3 (8) - Get List of Connected Serial Numbers

Declaration

\[
\text{Short \textbf{Get\_Available\_SN\_List}(String \textbf{SN\_List})}
\]

Description

This function takes a user defined variable and updates it with a list of serial numbers for all available (currently connected) signal generators.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
</table>
| String    | SN\_List | Required. String variable which the function will update with a list of all available serial numbers, separated by a single space character, for example “11110001 11110002 11110003”.

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

Visual Basic

\[
\text{If MyPTE1.Get\_Available\_SN\_List(SN\_List) > 0 Then}
\]
\[
\quad \text{array\_SN() = Split(SN\_List, "}
\quad \text{\")}
\]
\[
\quad \text{\'} Split the list into an array of serial numbers
\]
\[
\quad \text{For i As Integer = 0 To array\_SN.Length - 1}
\quad \text{\'} Loop through the array and use each serial number
\]
\[
\quad \text{Next}
\]
\[
\text{End If}
\]

Visual C++

\[
\text{if (MyPTE1 \_\_Get\_Available\_SN\_List(SN\_List) > 0)}
\]
\[
\quad \text{\{} \text{ \text;heightasm}}
\quad \text{\'} \text{split the List into array of SN’s}
\]
\[
\text{\}}
\]

Visual C#

\[
\text{if (MyPTE1.Get\_Available\_SN\_List(ref(SN\_List)) > 0)}
\]
\[
\quad \text{\{} \text{ \text;heightasm}}
\quad \text{\'} \text{split the List into array of SN’s}
\]
\[
\text{\}}
\]

Matlab

\[
[\text{status, SN\_List}] = \text{MyPTE1.Get\_Available\_SN\_List(SN\_List)}
\]
\[
\text{if status > 0 then}
\]
\[
\quad \text{% split the List into array of SN’s}
\]
\[
\text{\}}
\]

See Also

Get List of Available Addresses
3.1.3 (9) - Get List of Available Addresses

**Declaration**

```short
Get_Available_Address_List(String Add_List)
```

**Description**

This function takes a user defined variable and updates it with a list of addresses of all connected generator matrices.

**Parameters**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>Add_List</td>
<td>Required. String variable which the function will update with a list of addresses separated by a single space character, for example, “5 101 254 255”</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>Non zero</td>
<td>The number of signal generators connected</td>
</tr>
</tbody>
</table>

**Examples**

**Visual Basic**

```vbnet
If MyPTE1.Get_Available_Address_List(Add_List) > 0 Then
    ' Get list of available addresses
    array_Ad() = Split(Add_List, " ")
    ' Split the list into an array of addresses
    For i As Integer = 0 To array_Ad.Length - 1
        ' Loop through the array and use each address
    Next
End If
```

**Visual C++**

```cpp
if (MyPTE1->Get_Available_Address_List(Add_List) > 0);
{
    // split the List into array of Addresses
}
```

**Visual C#**

```cshar
if (MyPTE1.Get_Available_Address_List(ref(Add_List)) > 0)
{
    // split the List into array of Addresses
}
```

**Matlab**

```matlab
[status, Add_List]= MyPTE1.Get_Available_Address_List(Add_List)
If status > 0 then
    % split the List into array of Addresses
End
```

**See Also**

- Connect to Signal Generator by Address
- Set Address of Signal Generator
- Get Address of Signal generator
3.1.3 (10) - Turn On RF Output

Declaration

`Short Set_Power_ON()`

Description

This function enables the RF output from the signal generator.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>Non zero</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

- **Visual Basic**
  ```vbnet```
  ```
  status = MyPTE1.SetPowerON
  ```

- **Visual C++**
  ```c++```
  ```
  status = MyPTE1->SetPowerON();
  ```

- **Visual C#**
  ```csharp```
  ```
  status = MyPTE1.SetPowerON();
  ```

- **Matlab**
  ```matlab```
  ```
  status = MyPTE1.SetPowerON
  ```

See Also

- Turn Off RF Output
- Set Output Frequency and Power
- Set Output Frequency
- Set Output Power
3.1.3 (11) - Turn Off RF Output

Declaration

\[ \text{Short Set\_Power\_OFF()} \]

Description

This function disables the RF output from the signal generator.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>Non zero</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

Visual Basic

\[
\text{status} = \text{MyPTE1.SetPowerOFF}
\]

Visual C++

\[
\text{status} = \text{MyPTE1->SetPowerOFF();}
\]

Visual C#

\[
\text{status} = \text{MyPTE1.SetPowerOFF();}
\]

Matlab

\[
\text{status} = \text{MyPTE1.SetPowerOFF}
\]

See Also

- Turn On RF Output
- Set Output Frequency and Power
- Set Output Frequency
- Set Output Power
3.1.3 (12) - Set Output Frequency and Power

Declaration

\[ \text{Short SetFreqAndPower(Double } \text{Fr, Float Pr, Short TriggerOut)} \]

Description

This function sets the RF output frequency and power level of the signal generator and enables or disables the “trigger out” function.

Note: For SSG-4000LH and SSG-4000HP models with serial numbers up to 11207100000, enabling the Trigger Out function will disable Trigger In since a common port is used. SSG-4000LH and SSG-4000HP models with serial numbers greater than 11207100000 (and all other SSG models) have separate Trigger ports so enabling Trigger Out has no affect on Trigger In.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double</td>
<td>Fr</td>
<td>Required. The frequency in MHz.</td>
</tr>
<tr>
<td>Float</td>
<td>Pr</td>
<td>Required. The power in dBm.</td>
</tr>
<tr>
<td>Short</td>
<td>TriggerOut</td>
<td>Required. An integer variable to determine whether the “trigger out” function should be enabled. 1 enables trigger out, 0 disables it.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>0</td>
<td>Command failed</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

Visual Basic

\[
\text{status = MyPTE1.SetFreqAndPower(Freq, Power, 0)}
\]

Visual C++

\[
\text{status = MyPTE1->SetFreqAndPower(Freq, Power, 0);}\]

Visual C#

\[
\text{status = MyPTE1.SetFreqAndPower(Freq, Power, (short)0);}\]

Matlab

\[
\text{status = MyPTE1.SetFreqAndPower(Freq, Power, 0)}
\]

See Also

- Turn On RF Output
- Turn Off RF Output
- Set Output Frequency
- Set Output Power
- Get Generator Output Status
3.1.3 (13) - Set Output Frequency

Declaration

Short SetFreq(Double Fr, Short TriggerOut)

Description

This function sets the RF output frequency of the signal generator and enables or disables the “trigger out” function. The output power of the signal generator will not be changed.

Note: For SSG-4000LH and SSG-4000HP models with serial numbers up to 11207100000, enabling the Trigger Out function will disable Trigger In since a common port is used. SSG-4000LH and SSG-4000HP models with serial numbers greater than 11207100000 (and all other SSG models) have separate Trigger ports so enabling Trigger Out has no affect on Trigger In.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double</td>
<td>Fr</td>
<td>Required. The frequency in MHz.</td>
</tr>
<tr>
<td>Short</td>
<td>TriggerOut</td>
<td>Required. An integer variable to determine whether the “trigger out” function should be enabled. 1 enables trigger out, 0 disables it.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>0</td>
<td>Command failed</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

Visual Basic
status = MyPTE1.SetFreq (Freq, 0)

Visual C++
status = MyPTE1->SetFreq (Freq, 0);

Visual C#
status = MyPTE1.SetFreq (Freq, (short)0);

Matlab
status = MyPTE1.SetFreq (Freq, 0)

See Also

Turn On RF Output
Turn Off RF Output
Set Output Frequency and Power
Set Output Power
Get Generator Output Status
3.1.3 (14) - Set Output Power

Declaration

```plaintext
Short SetPower(Float Pr, Short TriggerOut)
```

Description

This function sets the RF output power of the signal generator and enables or disables the “trigger out” function. The output frequency of the signal generator will not be changed.

Note: For SSG-4000LH and SSG-4000HP models with serial numbers up to 11207100000, enabling the Trigger Out function will disable Trigger In since a common port is used. SSG-4000LH and SSG-4000HP models with serial numbers greater than 11207100000 (and all other SSG models) have separate Trigger ports so enabling Trigger Out has no affect on Trigger In.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Float</td>
<td>Pr</td>
<td>Required. The power in dBm.</td>
</tr>
<tr>
<td>Short</td>
<td>TriggerOut</td>
<td>Required. An integer variable to determine whether the “trigger out” function should be enabled. 1 enables trigger out, 0 disables it.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>0</td>
<td>Command failed</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

```plaintext
Visual Basic
status = MyPTE1.SetPower (Power, 0)

Visual C++
status = MyPTE1->SetPower (Power, 0);

Visual C#
status = MyPTE1.SetPower (Power, (short)0);

Matlab
status = MyPTE1.SetPower (Power, 0)
```

See Also

- Turn On RF Output
- Turn Off RF Output
- Set Output Frequency and Power
- Set Output Frequency
- Get Generator Output Status
3.1.3 (15) - Get Generator Output Status

Declaration

```c
Short GetGenStatus(Byte Locked, Short PowerIsOn, Double Fr, Float Pr,
                   _ Short UnLevelHigh, Short UnLevelLow)
```

Description

This function returns the current status of the signal generator RF output in a series of user defined variables. The following parameters are checked:

- Generator lock status (locked/unlocked)
- RF output status (on/off)
- Current output frequency
- Current output power
- Current output power relative to user requested level

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte</td>
<td>Locked</td>
<td>Required. User defined variable which will be set to 1 if the frequency is locked or 0 otherwise.</td>
</tr>
<tr>
<td>Short</td>
<td>PowerIsOn</td>
<td>Required. User defined variable which will be set to 1 if the RF output power is enabled or 0 otherwise.</td>
</tr>
<tr>
<td>Double</td>
<td>Fr</td>
<td>Required. User defined variable which will be updated with the generator output frequency in MHz.</td>
</tr>
<tr>
<td>Float</td>
<td>Pr</td>
<td>Required. User defined variable which will be updated with the generator output power in dBm.</td>
</tr>
<tr>
<td>Short</td>
<td>UnLevelHigh</td>
<td>Required. User defined variable that will be set to 1 if the user requested a higher power level than the generator can achieve. The variable is set to 0 if the output power is at the correct level. See model datasheets for output power specifications.</td>
</tr>
<tr>
<td>Short</td>
<td>UnLevelLow</td>
<td>Required. User defined variable that will be set to 1 if the user requested a lower power level than the generator can achieve. The variable is set to 0 if the output power is at the correct level. See model datasheets for output power specifications.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>0</td>
<td>Command failed</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>
Examples

Visual Basic

\[
\text{status} = \text{MyPTE1.GetGenStatus(lock, PowerIsOn, freq, power, UNLEVELHigh, UNLEVELLow)}
\]

Visual C++

\[
\text{status} = \text{MyPTE1->GetGenStatus(lock, PowerIsOn, freq, power, UNLEVELHigh, UNLEVELLow)};
\]

Visual C#

\[
\text{status} = \text{MyPTE1.GetGenStatus(ref(lock), ref(PowerIsOn), ref(freq), ref(power), ref(UNLEVELHigh), ref(UNLEVELLow))};
\]

Matlab

\[
[\text{status, lock, PowerIsOn, freq, power, UNLEVELHigh, UNLEVELLow}] = \text{MyPTE1.GetGenStatus(lock, PowerIsOn, freq, power, UNLEVELHigh, UNLEVELLow)}
\]

See Also

- Turn On RF Output
- Turn Off RF Output
- Set Output Frequency and Power
- Set Output Frequency
- Set Output Power
3.1.3 (16) - Set Low Noise or Low Spur Mode

Declaration

\[ \text{Short Set\_Noise\_Spur\_Mode (Short nsm)} \]

Description

This function applies to SSG-4000LH and SSG-4000HP only. It sets the generator in either “low noise” mode (for best phase noise performance) or “low spur” mode (for best spurious performance). The generator defaults to “low noise” mode.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>nsm</td>
<td>Required. Integer value 1 or 0 to set the generator noise/spur mode; 1 = low spur mode, 0 = low noise mode.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>0</td>
<td>Command failed</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

**Visual Basic**

\[ \text{status = MyPTE1.Set\_Noise\_Spur\_Mode(nsm)} \]

**Visual C++**

\[ \text{status = MyPTE1->Set\_Noise\_Spur\_Mode(nsm);} \]

**Visual C#**

\[ \text{status = MyPTE1.Set\_Noise\_Spur\_Mode(nsm);} \]

**Matlab**

\[ \text{status = MyPTE1.Set\_Noise\_Spur\_Mode(nsm)} \]

See Also

- Get Trigger In Status
- Check External Reference
- Get Temperature of Signal Generator
3.1.3 (17) - Check External Reference

**Declaration**

```plaintext
Short ExtRefDetected()
```

**Description**

This function checks whether an external 10MHz reference is connected to the generator. The generator automatically uses the external reference if it is present, otherwise the internal reference is used.

**Parameters**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>0</td>
<td>External reference is not connected</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>External reference is connected</td>
</tr>
</tbody>
</table>

**Examples**

Visual Basic

```plaintext
status = MyPTE1.ExtRefDetected()
```

Visual C++

```plaintext
status = MyPTE1->ExtRefDetected();
```

Visual C#

```plaintext
status = MyPTE1.ExtRefDetected();
```

Matlab

```plaintext
status = MyPTE1.ExtRefDetected()
```

**See Also**

- Set Low Noise or Low Spur Mode
- Get Trigger In Status
- Get Temperature of Signal Generator
3.1.3 (18) - Get Trigger In Status

Declaration

    Short  GetTriggerIn_Status()

Description

This function indicates whether the generator’s trigger input is at logic level low or high.

For SSG-4000LH and SSG-4000HP models with serial numbers up to 11207100000, the Trigger In function is only active when Trigger Out is disabled (see Set Output Frequency and Power). SSG-4000LH and SSG-4000HP models with serial numbers greater than 11207100000 (and all other SSG models) have separate Trigger In and Trigger Out ports so both can be used independently.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>0</td>
<td>Trigger input is at logic low</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Trigger input is at logic high</td>
</tr>
</tbody>
</table>

Examples

**Visual Basic**

    Status = MyPTE1.GetTriggerIn_Status()

**Visual C++**

    status = MyPTE1->GetTriggerIn_Status();

**Visual C#**

    status = MyPTE1.GetTriggerIn_Status();

**Matlab**

    status = MyPTE1.GetTriggerIn_Status()

See Also

- Set Low Noise or Low Spur Mode
- Check External Reference
- Get Temperature of Signal Generator
3.1.3 (19) - Set Pulse Mode

Declaration

\[
\text{Short \ Set\_PulseMode(Short T\_OFF, Short T\_ON, Short Tunit)}
\]

Description

This function creates a pulsed output with a user specified pulse duration and time period. The output during the pulse “on” period is a CW signal with a frequency and power level which should be set by the user in advance (see Set Output Frequency and Power). The pulse period will repeat indefinitely until any other function is called by the user’s program.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>T_OFF</td>
<td>Required. The off period between pulses.</td>
</tr>
<tr>
<td>Short</td>
<td>T_ON</td>
<td>Required. The pulse “on” duration.</td>
</tr>
<tr>
<td>Short</td>
<td>Tunit</td>
<td>Required. The units for the T_OFF and T_ON time periods; 0 for microseconds or 1 for milliseconds.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>0</td>
<td>Command failed</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

**Visual Basic**

```vbnet
status = MyPTE1.SetFreqAndPower(1000, 10, 0)
status = MyPTE1.Set_PulseMode(10, 2, 1)
' Set 2ms pulses of 1000MHz, 10dBm CW, separated by 10ms
```

**Visual C++**

```c
status = MyPTE1->SetFreqAndPower(1000, 10, 0);
status = MyPTE1->Set_PulseMode(10, 2, 1);
// Set 2ms pulses of 1000MHz, 10dBm CW, separated by 10ms
```

**Visual C#**

```csharp
status = MyPTE1.SetFreqAndPower(1000, 10, 0);
status = MyPTE1.Set_PulseMode(10, 2, 1);
// Set 2ms pulses of 1000MHz, 10dBm CW, separated by 10ms
```

**Matlab**

```matlab
status = MyPTE1.SetFreqAndPower(1000, 10, 0);
status = MyPTE1.Set_PulseMode(10, 2, 1);
% Set 2ms pulses of 1000MHz, 10dBm CW, separated by 10ms
```

See Also

- Set Output Frequency and Power
- Set Triggered Pulse Mode
3.1.3 (20) - Set Triggered Pulse Mode

Declaration

```
Short Set_PulseMode_Trigger(Short TriggerType, Short T_ON, 
  _ Short Tunit)
```

Description

This function creates a pulsed output with a user specified pulse duration that will start when an external trigger is received at the “Trigger In” input. The output during the pulse “on” period is a CW signal with a frequency and power level which should be set by the user in advance (see Set Output Frequency and Power).

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>TriggerType</td>
<td>Required. The trigger input sequence that will trigger the pulsed output; 0 for Trigger In = on then off, or 1 for Trigger In = off then on.</td>
</tr>
<tr>
<td>Short</td>
<td>T_ON</td>
<td>Required. The pulse “on” duration.</td>
</tr>
<tr>
<td>Short</td>
<td>Tunit</td>
<td>Required. The units for the T_ON time period; 0 for microseconds or 1 for milliseconds.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>0</td>
<td>Command failed</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>
Examples

Visual Basic

    status = MyPTE1.SetFreqAndPower(1000, 10, 0)
    status = MyPTE1.Set_PulseMode_Trigger(0, 2, 1)
    ' Set 2ms pulses of 1000MHz, 10dBm CW
    ' Start the pulse when a "on, off" is received at Trigger In

Visual C++

    status = MyPTE1->SetFreqAndPower(1000, 10, 0);
    status = MyPTE1->Set_PulseMode_Trigger(0, 2, 1);
    // Set 2ms pulses of 1000MHz, 10dBm CW
    // Start the pulse when a "on, off" is received at Trigger In

Visual C#

    status = MyPTE1.SetFreqAndPower(1000, 10, 0);
    status = MyPTE1.Set_PulseMode_Trigger(0, 2, 1);
    // Set 2ms pulses of 1000MHz, 10dBm CW
    // Start the pulse when a "on, off" is received at Trigger In

Matlab

    status = MyPTE1.SetFreqAndPower(1000, 10, 0)
    status = MyPTE1.Set_PulseMode_Trigger(0, 2, 1)
    % Set 2ms pulses of 1000MHz, 10dBm CW
    % Start the pulse when a "on, off" is received at Trigger In

See Also

Set Output Frequency and Power
Set Pulse Mode
3.1.3 (21) - Get Generator Maximum Frequency Spec

Declaration

```
Float GetGenMaxFreq()
```

Description

This function reports the maximum output frequency in MHz that the generator is capable of providing.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Float</td>
<td>Frequency</td>
<td>Maximum output frequency in MHz</td>
</tr>
</tbody>
</table>

Examples

**Visual Basic**

```
Freq = MyPTE1.GetGenMaxFreq
```

**Visual C++**

```
Freq = MyPTE1->GetGenMaxFreq();
```

**Visual C#**

```
Freq = MyPTE1.GetGenMaxFreq();
```

**Matlab**

```
Freq = MyPTE1.GetGenMaxFreq
```

See Also

- Get Generator Minimum Frequency Spec
- Get Generator Step Size Spec
- Get Generator Maximum Power Spec
- Get Generator Minimum Power Spec
3.1.3 (22) - Get Generator Minimum Frequency Spec

Declaration

Float GetGenMinFreq()

Description

This function reports the minimum output frequency in MHz that the generator is capable of providing.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Float</td>
<td>Frequency</td>
<td>Minimum output frequency in MHz</td>
</tr>
</tbody>
</table>

Examples

Visual Basic
Freq = MyPTE1.GetGenMinFreq

Visual C++
Freq = MyPTE1->GetGenMinFreq();

Visual C#
Freq = MyPTE1.GetGenMinFreq();

Matlab
Freq = MyPTE1.GetGenMinFreq

See Also

Get Generator Maximum Frequency Spec
Get Generator Step Size Spec
Get Generator Maximum Power Spec
Get Generator Minimum Power Spec
3.1.3 (23) - Get Generator Step Size Spec

Declaration

Float GetGenStepFreq()

Description

This function reports the generator’s step size in KHz.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Float</td>
<td>Frequency</td>
<td>Generator step size in KHz</td>
</tr>
</tbody>
</table>

Examples

Visual Basic
Freq = MyPTE1.GetGenStepFreq

Visual C++
Freq = MyPTE1->GetGenStepFreq();

Visual C#
Freq = MyPTE1.GetGenStepFreq();

Matlab
Freq = MyPTE1.GetGenStepFreq

See Also

Get Generator Maximum Frequency Spec
Get Generator Minimum Frequency Spec
Get Generator Maximum Power Spec
Get Generator Minimum Power Spec
3.1.3 (24) - Get Generator Maximum Power Spec

Declaration

    Float GetGenMaxPower()

Description

This function reports the maximum output power specification in dBm for the active generator. Requesting a higher output level than this value will result in the generator setting the UnLevelHigh flag in order to report an inaccurate power level (see Get Generator Output Status). Actual minimum power will vary between units.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Float</td>
<td>Power</td>
<td>Maximum output power in dBm</td>
</tr>
</tbody>
</table>

Example

Visual Basic

    Power = MyPTE1.GetGenMaxPower

Visual C++

    Power = MyPTE1->GetGenMaxPower();

Visual C#

    Power = MyPTE1.GetGenMaxPower();

Matlab

    Power = MyPTE1.GetGenMaxPower

See Also

Get Generator Maximum Frequency Spec
Get Generator Minimum Frequency Spec
Get Generator Step Size Spec
Get Generator Minimum Power Spec
3.1.3 (25) - Get Generator Minimum Power Spec

Declaration

    Float GetGenMinPower()

Description

This function reports the minimum output power specification in dBm for the active generator. Requesting a lower output level than this value will result in the generator setting the UnLevelLow flag in order to report an inaccurate power level (see Get Generator Output Status). Actual minimum power will vary between units.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Float</td>
<td>Power</td>
<td>Minimum output power in dBm</td>
</tr>
</tbody>
</table>

Example

Visual Basic

    Power = MyPTE1.GetGenMinPower

Visual C++

    Power = MyPTE1->GetGenMinPower();

Visual C#

    Power = MyPTE1.GetGenMinPower();

Matlab

    Power = MyPTE1.GetGenMinPower

See Also

Get Generator Maximum Frequency Spec
Get Generator Minimum Frequency Spec
Get Generator Step Size Spec
Get Generator Maximum Power Spec
3.1.3 (26) - Get Temperature of Signal Generator

Declaration

\[ \text{Float GetDeviceTemperature()} \]

Description

This function returns the internal temperature of the signal generator.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Float</td>
<td>Temperature</td>
<td>The device internal temperature in degrees Celsius</td>
</tr>
</tbody>
</table>

Example

**Visual Basic**
```
MsgBox("Temperature is " & MyPTE1.GetDeviceTemperature(2))
' Display a message box with the device temperature
```

**Visual C++**
```
MessageBox::Show("Temperature is " + MyPTE1->GetDeviceTemperature(2));
// Display a message box with the device temperature
```

**Visual C#**
```
MessageBox.Show("Temperature is " + MyPTE1.GetDeviceTemperature(2));
// Display a message box with the device temperature
```

**Matlab**
```
[temp, status]=MyPTE1.GetDeviceTemperature(2)
Msgbox('Temperature is ', temp)
% Display a message box with the device temperature
```
### 3.1.3 (27) - Check Connection

#### Declaration

```ini
Short Check_Connection()
```

#### Description

This function checks whether the USB connection to the signal generator is still active.

#### Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>0</td>
<td>No connection</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>USB connection to signal generator is active</td>
</tr>
</tbody>
</table>

#### Example

- **Visual Basic**
  ```vbnet
  Status = MyPTE1.Check_Connection()
  ```
- **Visual C++**
  ```cpp
  Status = MyPTE1->Check_Connection();
  ```
- **Visual C#**
  ```csharp
  Status = MyPTE1.Check_Connection();
  ```
- **Matlab**
  ```matlab
  Status = MyPTE1.Check_Connection()
  ```

#### See Also

- Connect to Signal Generator
- Connect to Signal Generator by Address
- Disconnect from Signal Generator
3.1.3 (28) - Get Firmware Version

Declaration

    Short GetFirmware()

Description

This function returns a numeric value which indicates the internal firmware version of the signal generator.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td></td>
<td>Version number of the internal signal generator firmware</td>
</tr>
</tbody>
</table>

Example

**Visual Basic**

    FW = MyPTE1.GetFirmware()

**Visual C++**

    FW = MyPTE1->GetFirmware();

**Visual C#**

    FW = MyPTE1.GetFirmware();

**Matlab**

    FW = MyPTE1.GetFirmware()
3.1.3 (29) - Frequency Sweep – Get Direction

Declaration

    Short FSweep_GetDirection()

Description

This function returns the current frequency sweep direction. The possible settings are:
0 – Increasing from start to stop frequency
1 – Decreasing from stop to start frequency
2 – Increasing from start to stop, before decreasing from stop to start frequency

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>0</td>
<td>Increasing frequency sweep from start to stop frequency</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Decreasing frequency sweep from stop to start frequency</td>
</tr>
<tr>
<td>Short</td>
<td>2</td>
<td>Increasing then decreasing frequency sweep (from start to stop to start frequency)</td>
</tr>
</tbody>
</table>

Examples

Visual Basic

    Sweep = MyPTE1.FSweep_GetDirection

Visual C++

    Sweep = MyPTE1->FSweep_GetDirection();

Visual C#

    Sweep = MyPTE1.FSweep_GetDirection();

Matlab

    Sweep = MyPTE1.FSweep_GetDirection

See Also

    Frequency Sweep – Set Direction
3.1.3 (30) - Frequency Sweep – Get Dwell Time

Declaration

```
Short FSweep_GetDwell()
```

Description

This function returns the current dwell time setting in milliseconds; this is the length of time that the generator will pause at each frequency point.

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>Time</td>
<td>Dwell time in milliseconds</td>
</tr>
</tbody>
</table>

Examples

```
Visual Basic
Dwell = MyPTE1.FSweep_GetDwell

Visual C++
Dwell = MyPTE1->FSweep_GetDwell();

Visual C#
Dwell = MyPTE1.FSweep_GetDwell();

Matlab
Dwell = MyPTE1.FSweep_GetDwell
```

See Also

Frequency Sweep – Get Maximum Dwell Time
Frequency Sweep – Get Minimum Dwell Time
Frequency Sweep – Set Dwell Time
3.1.3 (31) - Frequency Sweep – Get Maximum Dwell Time

Declaration

    Short FSweep_GetMaxDwell()

Description

This function returns the maximum allowed dwell time in milliseconds; this is the length of time that the generator can pause at each frequency point.

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>Time</td>
<td>Maximum dwell time in milliseconds</td>
</tr>
</tbody>
</table>

Examples

Visual Basic

    Dwell = MyPTE1.FSweep_GetMaxDwell

Visual C++

    Dwell = MyPTE1->FSweep_GetMaxDwell();

Visual C#

    Dwell = MyPTE1.FSweep_GetMaxDwell();

Matlab

    Dwell = MyPTE1.FSweep_GetMaxDwell

See Also

    Frequency Sweep – Get Dwell Time
    Frequency Sweep – Get Minimum Dwell Time
    Frequency Sweep – Set Dwell Time
3.1.3 (32) - Frequency Sweep – Get Minimum Dwell Time

Declaration

    Short FSweep_GetMinDwell()

Description

This function returns the minimum allowed dwell time in milliseconds; this is the length of
time that the generator can pause at each frequency point.

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>Time</td>
<td>Minimum dwell time in milliseconds</td>
</tr>
</tbody>
</table>

Examples

Visual Basic

    Dwell = MyPTE1.FSweep_GetMinDwell

Visual C++

    Dwell = MyPTE1->FSweep_GetMinDwell();

Visual C#

    Dwell = MyPTE1.FSweep_GetMinDwell();

Matlab

    Dwell = MyPTE1.FSweep_GetMinDwell

See Also

    Frequency Sweep – Get Dwell Time
    Frequency Sweep – Get Maximum Dwell Time
    Frequency Sweep – Set Dwell Time
3.1.3 (33) - Frequency Sweep – Get Power

Declaration

```c
Float FSweep_GetPower()
```

Description

This function returns the current output power setting of the frequency sweep in dBm.

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Float</td>
<td>-999</td>
<td>Command failed</td>
</tr>
<tr>
<td>Float</td>
<td>Power</td>
<td>Output power in dBm</td>
</tr>
</tbody>
</table>

Examples

**Visual Basic**
```vbnet
Power = MyPTE1.FSweep_GetPower
```

**Visual C++**
```cpp
Power = MyPTE1->FSweep_GetPower();
```

**Visual C#**
```csharp
Power = MyPTE1.FSweep_GetPower();
```

**Matlab**
```matlab
Power = MyPTE1.FSweep_GetPower
```

See Also

- Get Generator Maximum Power Spec
- Get Generator Minimum Power Spec
- Frequency Sweep – Set Power
3.1.3 (34) - Frequency Sweep – Get Start Frequency

Declaration

```
Double FSweep_GetStartFreq()
```

Description

This function returns the start frequency in MHz of the current frequency sweep.

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Double</td>
<td>Freq</td>
<td>Start frequency in MHz</td>
</tr>
</tbody>
</table>

Examples

- **Visual Basic**
  ```
  Freq = MyPTE1.FSweep_GetStartFreq
  ```
- **Visual C++**
  ```
  Freq = MyPTE1->FSweep_GetStartFreq();
  ```
- **Visual C#**
  ```
  Freq = MyPTE1.FSweep_GetStartFreq();
  ```
- **Matlab**
  ```
  Freq = MyPTE1.FSweep_GetStartFreq
  ```

See Also

- Get Generator Maximum Frequency Spec
- Get Generator Minimum Frequency Spec
- Frequency Sweep – Get Stop Frequency
- Frequency Sweep – Set Start Frequency
- Frequency Sweep – Set Stop Frequency
3.1.3 (35) - Frequency Sweep – Get Stop Frequency

Declaration

```
Double FSweep_GetStopFreq()
```

Description

This function returns the stop frequency in MHz of the current frequency sweep.

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Double</td>
<td>Freq</td>
<td>Stop frequency in MHz</td>
</tr>
</tbody>
</table>

Examples

```
Visual Basic
Freq = MyPTE1.FSweep_GetStopFreq

Visual C++
Freq = MyPTE1->FSweep_GetStopFreq();

Visual C#
Freq = MyPTE1.FSweep_GetStopFreq();

Matlab
Freq = MyPTE1.FSweep_GetStopFreq
```

See Also

- Get Generator Maximum Frequency Spec
- Get Generator Minimum Frequency Spec
- Frequency Sweep – Get Start Frequency
- Frequency Sweep – Set Start Frequency
- Frequency Sweep – Set Stop Frequency
3.1.3 (36) - Frequency Sweep – Get Step Size

Declaration

    Double FSweep_GetStepSize()

Description

This function returns the step size in MHz of the current frequency sweep.

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Double</td>
<td>Freq</td>
<td>Step size in MHz</td>
</tr>
</tbody>
</table>

Examples

Visual Basic

    Freq = MyPTE1.FSweep_GetStepSize

Visual C++

    Freq = MyPTE1->FSweep_GetStepSize();

Visual C#

    Freq = MyPTE1.FSweep_GetStepSize();

Matlab

    Freq = MyPTE1.FSweep_GetStepSize

See Also

Get Generator Step Size Spec
Frequency Sweep – Get Start Frequency
Frequency Sweep – Get Stop Frequency
Frequency Sweep – Set Start Frequency
Frequency Sweep – Set Stop Frequency
Frequency Sweep – Set Step Size
3.1.3 (37) - Frequency Sweep – Get Trigger In Mode

Declaration

\texttt{Short FSweep\_GetTriggerIn()}

Description

This function returns the Trigger Input mode for the frequency sweep, this dictates how the generator will respond to an external trigger:

0 – Ignore trigger input
1 – Wait for external trigger (Trigger In = logic 1) before setting each frequency point
2 – Wait for external trigger (Trigger In = logic 1) before starting the frequency sweep

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{Short}</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>\texttt{Short}</td>
<td>0</td>
<td>Ignore Trigger In</td>
</tr>
<tr>
<td>\texttt{Short}</td>
<td>1</td>
<td>Wait for Trigger In for each frequency point</td>
</tr>
<tr>
<td>\texttt{Short}</td>
<td>2</td>
<td>Wait for Trigger In before commencing sweep</td>
</tr>
</tbody>
</table>

Examples

\textbf{Visual Basic}

\begin{verbatim}
Status = MyPTE1.FSweep_GetTriggerIn
\end{verbatim}

\textbf{Visual C++}

\begin{verbatim}
Status = MyPTE1->FSweep_GetTriggerIn();
\end{verbatim}

\textbf{Visual C#}

\begin{verbatim}
Status = MyPTE1.FSweep_GetTriggerIn();
\end{verbatim}

\textbf{Matlab}

\begin{verbatim}
Status = MyPTE1.FSweep_GetTriggerIn
\end{verbatim}

See Also

Frequency Sweep – Get Trigger Out Mode
Frequency Sweep – Set Trigger In Mode
Frequency/Power Hop – Set Trigger Out Mode
3.1.3 (38) - Frequency Sweep – Get Trigger Out Mode

Declaration

```
Short FSweep_GetTriggerOut()
```

Description

This function returns Trigger Output mode for the frequency sweep, this dictates how the Trigger Out port will be used during the frequency sweep:

0 – Disable trigger output

1 – Provide a trigger output (Trigger Out = logic 1) as each frequency point is set

2 – Provide a trigger output (Trigger Out = logic 1) as the frequency sweep is initiated

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>0</td>
<td>Trigger Out disabled</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Trigger Out set at each frequency point</td>
</tr>
<tr>
<td>Short</td>
<td>2</td>
<td>Trigger Out set as the sweep is initialized</td>
</tr>
</tbody>
</table>

Examples

```
Visual Basic
Status = MyPTE1.FSweep_GetTriggerOut

Visual C++
Status = MyPTE1->FSweep_GetTriggerOut();

Visual C#
Status = MyPTE1.FSweep_GetTriggerOut();

Matlab
Status = MyPTE1.FSweep_GetTriggerOut
```

See Also

- Frequency Sweep – Get Trigger In Mode
- Frequency Sweep – Set Trigger In Mode
- Frequency/Power Hop – Set Trigger Out Mode
3.1.3 (39) - Frequency Sweep – Set Direction

Declaration

    Short FSweep_SetDirection(Short SweepDirection)

Description

This function sets the direction of the frequency sweep. The 3 options are:
0 – Increasing from start to stop frequency
1 – Decreasing from stop to start frequency
2 – Increasing from start to stop, before decreasing from stop to start frequency

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>Sweep_Direction</td>
<td>Required. Numeric value corresponding to the sweep direction mode: 0 - Increasing frequency sweep from start to stop frequency 1 - Decreasing frequency sweep from stop to start frequency 2 - Increasing then decreasing frequency sweep (from start to stop to start frequency)</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

Visual Basic

    Status = MyPTE1.FSweep_SetDirection(0)

Visual C++

    Status = MyPTE1->FSweep_SetDirection(0);

Visual C#

    Status = MyPTE1.FSweep_SetDirection(0);

Matlab

    Status = MyPTE1.FSweep_SetDirection(0)

See Also

    Frequency Sweep – Get Direction
3.1.3 (40) - Frequency Sweep – Set Dwell Time

Declaration

\[ \text{Short } \text{FSweep\_SetDwell}(\text{Short } \text{dwell\_msec}) \]

Description

This function sets the dwell time in milliseconds; this is the length of time that the generator will pause at each frequency point.

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>dwell_msec</td>
<td>Required. The dwell time in milliseconds.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Command completed successfullly</td>
</tr>
</tbody>
</table>

Examples

**Visual Basic**

\[ \text{Status } = \text{MyPTE1.\text{FSweep\_SetDwell}(15)} \]

**Visual C++**

\[ \text{Status } = \text{MyPTE1->FSweep\_SetDwell(15);} \]

**Visual C#**

\[ \text{Status } = \text{MyPTE1.FSweep\_SetDwell(15);} \]

**Matlab**

\[ \text{Status } = \text{MyPTE1.FSweep\_SetDwell(15)} \]

See Also

- Frequency Sweep – Get Dwell Time
- Frequency Sweep – Get Maximum Dwell Time
- Frequency Sweep – Get Minimum Dwell Time
3.1.3 (41) - Frequency Sweep – Start/Stop Sweep

Declaration

\[
\text{Short } \text{FSweep\_SetMode}(\text{Short onoff})
\]

Description

This function starts or stops the frequency sweep using the previously defined parameters.

Note: The frequency sweep will stop automatically if any other command is sent.

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
</table>
| Short     | onoff    | Required. Integer value to enable/disable the sweep:
|           |          | 1 – Start frequency sweep
|           |          | 0 – Stop frequency sweep |

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

**Visual Basic**

\[
\begin{align*}
\text{Status} &= \text{MyPTE1.FSweep\_SetMode}(1) \quad \text{'Start} \\
\text{Status} &= \text{MyPTE1.FSweep\_SetMode}(0) \quad \text{'Stop}
\end{align*}
\]

**Visual C++**

\[
\begin{align*}
\text{Status} &= \text{MyPTE1->FSweep\_SetMode}(1); \quad \text{// Start} \\
\text{Status} &= \text{MyPTE1->FSweep\_SetMode}(0); \quad \text{// Stop}
\end{align*}
\]

**Visual C#**

\[
\begin{align*}
\text{Status} &= \text{MyPTE1.FSweep\_SetMode}(1); \quad \text{// Start} \\
\text{Status} &= \text{MyPTE1.FSweep\_SetMode}(0); \quad \text{// Stop}
\end{align*}
\]

**Matlab**

\[
\begin{align*}
\text{Status} &= \text{MyPTE1.FSweep\_SetMode}(1) \quad \% \text{Start} \\
\text{Status} &= \text{MyPTE1.FSweep\_SetMode}(0) \quad \% \text{Stop}
\end{align*}
\]

See Also

- Frequency/Power Hop – Start/Stop Hop Sequence
- Power Sweep – Start/Stop Sweep
3.1.3 (42) - Frequency Sweep – Set Power

Declaration

Short FSweep_SetPower(Float Pr)

Description

This function sets the output power level in dBm to be used for the frequency sweep in.

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Float</td>
<td>Pr</td>
<td>Required. The fixed power level in dBm to be used for the frequency sweep.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

Visual Basic

Status = MyPTE1.FSweep_SetPower(-10.5)

Visual C++

Status = MyPTE1->FSweep_SetPower(-10.5);

Visual C#

Status = MyPTE1.FSweep_SetPower(-10.5);

Matlab

Status = MyPTE1.FSweep_SetPower(-10.5)

See Also

Frequency Sweep – Get Power
3.1.3 (43) - Frequency Sweep – Set Start Frequency

Declaration

\[ \text{Short FSweep\_SetStartFreq(Double Fr)} \]

Description

This function sets the frequency in MHz at which the sweep will start.

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double</td>
<td>Fr</td>
<td>Required. The start frequency in MHz.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

Visual Basic

\[ \text{Status = MyPTE1\_FSweep\_SetStartFreq(250)} \]

Visual C++

\[ \text{Status = MyPTE1\_FSweep\_SetStartFreq(250);} \]

Visual C#

\[ \text{Status = MyPTE1\_FSweep\_SetStartFreq(250);} \]

Matlab

\[ \text{Status = MyPTE1\_FSweep\_SetStartFreq(250);} \]

See Also

- Get Generator Maximum Frequency Spec
- Get Generator Minimum Frequency Spec
- Frequency Sweep – Get Start Frequency
- Frequency Sweep – Get Stop Frequency
- Frequency Sweep – Set Stop Frequency
3.1.3 (44) - Frequency Sweep – Set Stop Frequency

Declaration

\[ \text{Short } \text{FSweep\_SetStopFreq(Double Fr)} \]

Description

This function sets the frequency in MHz at which the sweep will stop.

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double</td>
<td>Fr</td>
<td>Required. The stop frequency in MHz.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

**Visual Basic**

Status = MyPTE1.FSweep_SetStopFreq(5500)

**Visual C++**

Status = MyPTE1->FSweep_SetStopFreq(5500);

**Visual C#**

Status = MyPTE1.FSweep_SetStopFreq(5500);

**Matlab**

Status = MyPTE1.FSweep_SetStopFreq(5500)

See Also

Get Generator Maximum Frequency Spec
Get Generator Minimum Frequency Spec
Frequency Sweep – Get Start Frequency
Frequency Sweep – Get Stop Frequency
Frequency Sweep – Set Start Frequency
3.1.3 (45) - Frequency Sweep – Set Step Size

Declaration

```
Short FSweep_SetStepSize(Double Fr)
```

Description

This function sets the step size in MHz to be used in the frequency sweep.

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double</td>
<td>Fr</td>
<td>Required. The step size in MHz.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

Visual Basic

```
Status = MyPTE1.FSweep_SetStepSize(0.1)
```

Visual C++

```
Status = MyPTE1->FSweep_SetStepSize(0.1);
```

Visual C#

```
Status = MyPTE1.FSweep_SetStepSize(0.1);
```

Matlab

```
Status = MyPTE1.FSweep_SetStepSize(0.1)
```

See Also

- Get Generator Step Size Spec
- Frequency Sweep – Get Start Frequency
- Frequency Sweep – Get Stop Frequency
- Frequency Sweep – Get Step Size
- Frequency Sweep – Set Start Frequency
- Frequency Sweep – Set Stop Frequency
3.1.3 (46) - Frequency Sweep – Set Trigger In Mode

Declaration

Short FSweep_SetTriggerIn(Short SweepTriggerIn)

Description

This function specifies how the frequency sweep should respond to an external trigger. The modes are:
0 – Ignore trigger input
1 – Wait for external trigger (Trigger In = logic 1) before setting each frequency point
2 – Wait for external trigger (Trigger In = logic 1) before commencing the frequency sweep

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>Sweep_TriggerIn</td>
<td>Required. Integer value to specify the Trigger In mode:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 – Ignore Trigger In</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 - Wait for Trigger In before each frequency point</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 - Wait for Trigger In before commencing sweep</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

Visual Basic

Status = MyPTE1.FSweep_SetTriggerIn(1)

Visual C++

Status = MyPTE1->FSweep_SetTriggerIn(1);

Visual C#

Status = MyPTE1.FSweep_SetTriggerIn(1);

Matlab

Status = MyPTE1.FSweep_SetTriggerIn(1)

See Also

Frequency Sweep – Get Trigger In Mode
Frequency Sweep – Get Trigger Out Mode
Frequency Sweep – Set Trigger Out Mode
3.1.3 (47) - Frequency Sweep – Set Trigger Out Mode

Declaration

    Short FSweep_SetTriggerOut(Short SweepTriggerOut)

Description

This function specified how the Trigger Out port will be used during the frequency sweep. The modes are:
0 – Disable trigger output
1 – Provide a trigger output (Trigger Out = logic 1) as each frequency point is set
2 – Provide a trigger output (Trigger Out = logic 1) as the frequency sweep is initiated

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>Sweep_TriggerOut</td>
<td>Required. Integer value to specify the Trigger Out mode:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 – Trigger Out disabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 – Set Trigger Out at each frequency point</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 – Set Trigger Out on commencing the sweep</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

Visual Basic

    Status = MyPTE1.FSweep_SetTriggerOut(1)

Visual C++

    Status = MyPTE1->FSweep_SetTriggerOut(1);

Visual C#

    Status = MyPTE1.FSweep_SetTriggerOut(1);

Matlab

    Status = MyPTE1.FSweep_SetTriggerOut(1)

See Also

    Frequency Sweep – Get Trigger In Mode
    Frequency Sweep – Get Trigger Out Mode
    Frequency Sweep – Set Trigger In Mode
3.1.3 (48) - Frequency/Power Hop – Get Direction

Declaration

```
Short Hop_GetDirection()
```

Description

This function returns the direction setting for the current hop sequence:
0 – Ascending from first to last
1 – Descending from last to first
2 – Ascending from first to last, then descending from last to first

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>0</td>
<td>Ascending frequency from lowest to highest</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Descending frequency from highest to lowest</td>
</tr>
<tr>
<td>Short</td>
<td>2</td>
<td>Ascending frequency from lowest to highest, then descending to lowest</td>
</tr>
</tbody>
</table>

Examples

```
Visual Basic
    Direction = MyPTE1.Hop_GetDirection

Visual C++
    Direction = MyPTE1->Hop_GetDirection();

Visual C#
    Direction = MyPTE1.Hop_GetDirection();

Matlab
    Direction = MyPTE1.Hop_GetDirection
```

See Also

```
Frequency/Power Hop – Set Direction
```
3.1.3 (49) - Frequency/Power Hop – Get Dwell Time

Declaration

    Short Hop_GetDwell()

Description

This function returns the dwell time setting in milliseconds for the current hop sequence.

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>Dwell Time</td>
<td>Dwell time setting in milliseconds</td>
</tr>
</tbody>
</table>

Examples

Visual Basic

    DwellTime = MyPTE1.Hop_GetDwell

Visual C++

    DwellTime = MyPTE1->Hop_GetDwell();

Visual C#

    DwellTime = MyPTE1.Hop_GetDwell();

Matlab

    DwellTime = MyPTE1.Hop_GetDwell

See Also

    Frequency/Power Hop – Get Maximum Dwell Time
    Frequency/Power Hop – Get Minimum Dwell Time
    Power Sweep – Set Dwell Time
3.1.3 (50) - Frequency/Power Hop – Get Maximum Dwell Time

Declaration

    Short Hop_GetMaxDwell()

Description

This function returns the maximum allowed dwell time in milliseconds for any point in a hop sequence.

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>Dwell Time</td>
<td>Maximum allowed dwell time in milliseconds</td>
</tr>
</tbody>
</table>

Examples

<table>
<thead>
<tr>
<th>Visual Basic</th>
</tr>
</thead>
<tbody>
<tr>
<td>DwellTime = MyPTE1.Hop_GetMaxDwell</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Visual C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>DwellTime = MyPTE1-&gt;Hop_GetMaxDwell();</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Visual C#</th>
</tr>
</thead>
<tbody>
<tr>
<td>DwellTime = MyPTE1.Hop_GetMaxDwell();</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Matlab</th>
</tr>
</thead>
<tbody>
<tr>
<td>DwellTime = MyPTE1.Hop_GetMaxDwell</td>
</tr>
</tbody>
</table>

See Also

- Frequency/Power Hop – Get Dwell Time
- Frequency/Power Hop – Get Minimum Dwell Time
- Power Sweep – Set Dwell Time
3.1.3 (51) - Frequency/Power Hop – Get Minimum Dwell Time

Declaration

```
Short Hop_GetMinDwell()
```

Description

This function returns the minimum allowed dwell time in milliseconds for any point in a hop sequence.

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>Dwell Time</td>
<td>Minimum allowed dwell time in milliseconds</td>
</tr>
</tbody>
</table>

Examples

Visual Basic

```
DwellTime = MyPTE1.Hop_GetMinDwell
```

Visual C++

```
DwellTime = MyPTE1->Hop_GetMinDwell();
```

Visual C#

```
DwellTime = MyPTE1.Hop_GetMinDwell();
```

Matlab

```
DwellTime = MyPTE1.Hop_GetMinDwell
```

See Also

- Frequency/Power Hop – Get Dwell Time
- Frequency/Power Hop – Get Maximum Dwell Time
- Power Sweep – Set Dwell Time
3.1.3 (52) - Frequency/Power Hop – Get Maximum Number of Points

Declaration

    Short Hop_GetMaxPoints()

Description

This function returns the maximum allowed number of points in a hop sequence.

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>Max Hops</td>
<td>Maximum number of frequency hop points</td>
</tr>
</tbody>
</table>

Examples

Visual Basic

    Hops = MyPTE1.Hop_GetMaxPoints

Visual C++

    Hops = MyPTE1->Hop_GetMaxPoints();

Visual C#

    Hops = MyPTE1.Hop_GetMaxPoints();

Matlab

    Hops = MyPTE1.Hop_GetMaxPoints

See Also

    Frequency/Power Hop – Get Hop Point
    Frequency/Power Hop – Set Number of Points
    Frequency/Power Hop – Set Hop Point
3.1.3 (53) - Frequency/Power Hop – Get Hop Point

Declaration

\[ \text{Short Hop\_GetPoint(Short PointNo, Double HopFreq, Float HopPower)} \]

Description

This function returns the frequency and power settings for a specific point in a hop sequence, from 1 to the maximum allowed number of points (device specific, see Frequency/Power Hop – Get Maximum Number of Points).

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>PointNo</td>
<td>Required. The point number; for example, 1 for the first point in the sequence, 2 for the second.</td>
</tr>
<tr>
<td>Double</td>
<td>HopFreq</td>
<td>Required. User defined variable which will be overwritten with the frequency in MHz of the specified hop point.</td>
</tr>
<tr>
<td>Float</td>
<td>HopPower</td>
<td>Required. User defined variable which will be overwritten with the power in dBm of the specified hop point.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

- **Visual Basic**
  
  \[ \text{status = MyPTE1.Hop\_GetPoint(PointNo, HopFreq, HopPower)} \]

- **Visual C++**
  
  \[ \text{status = MyPTE1->Hop\_GetPoint(PointNo, HopFreq, HopPower);} \]

- **Visual C#**
  
  \[ \text{status = MyPTE1.Hop\_GetPoint(PointNo, HopFreq, HopPower);} \]

- **Matlab**
  
  \[ \text{[PointNo, HopFreq, HopPower] = MyPTE1.Hop\_GetPoint(PointNo, HopFreq, HopPower);} \]

See Also

- Frequency/Power Hop – Get Maximum Number of Points
- Frequency/Power Hop – Set Number of Points
- Frequency/Power Hop – Set Hop Point
3.1.3 (54) - Frequency/Power Hop – Get Trigger In Mode

Declaration

    Short Hop_GetTriggerIn()

Description

This function returns the Trigger Input mode for the hop sequence, this dictates how the generator will respond to an external trigger:

0 – Ignore trigger input
1 – Wait for external trigger (Trigger In = logic 1) before hopping to the next point
2 – Wait for external trigger (Trigger In = logic 1) before starting the hop sequence

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>0</td>
<td>Ignore Trigger In</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Wait for Trigger In before hopping to next point</td>
</tr>
<tr>
<td>Short</td>
<td>2</td>
<td>Wait for Trigger In before starting hop sequence</td>
</tr>
</tbody>
</table>

Examples

Visual Basic

    Mode = MyPTE1.Hop_GetTriggerIn

Visual C++

    Mode = MyPTE1->Hop_GetTriggerIn();

Visual C#

    Mode = MyPTE1.Hop_GetTriggerIn();

Matlab

    Mode = MyPTE1.Hop_GetTriggerIn

See Also

Frequency/Power Hop – Get Trigger Out Mode
Frequency/Power Hop – Set Trigger In Mode
Frequency/Power Hop – Set Trigger Out Mode
3.1.3 (55) - Frequency/Power Hop – Get Trigger Out Mode

Declaration

Short Hop_GetTriggerOut()

Description

This function returns the Trigger Output mode for the hop sequence, this dictates how the Trigger Out port will be used during the frequency sweep:

0 – Disable trigger output
1 – Provide a trigger output (Trigger Out = logic 1) as each point is set
2 – Provide a trigger output (Trigger Out = logic 1) as hop is initiated

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>0</td>
<td>Trigger Out disabled</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Trigger Out set at each point</td>
</tr>
<tr>
<td>Short</td>
<td>2</td>
<td>Trigger Out set as the hop is initiated</td>
</tr>
</tbody>
</table>

Examples

**Visual Basic**

Mode = MyPTE1.Hop_GetTriggerOut

**Visual C++**

Mode = MyPTE1->Hop_GetTriggerOut();

**Visual C#**

Mode = MyPTE1.Hop_GetTriggerOut();

**Matlab**

Mode = MyPTE1.Hop_GetTriggerOut

See Also

Frequency/Power Hop – Get Trigger In Mode
Frequency/Power Hop – Set Trigger In Mode
Frequency/Power Hop – Set Trigger Out Mode
3.1.3 (56) - Frequency/Power Hop – Set Direction

Declaration

    Short Hop_SetDirection(Short HopDirection)

Description

This function sets the direction of the hop sequence:
0 – Ascending from first to last
1 – Descending from last to first
2 – Ascending from first to last, then descending from last to first

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>Hop_Direction</td>
<td>Required. Numeric value corresponding to the sweep direction mode: 0 – Ascending from first to last 1 – Descending from last to first 2 – Ascending from first to last, then descending from last to first</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

Visual Basic

    Status = MyPTE1.Hop_SetDirection(0)

Visual C++

    Status = MyPTE1->Hop_SetDirection(0);

Visual C#

    Status = MyPTE1.Hop_SetDirection(0);

Matlab

    Status = MyPTE1.Hop_SetDirection(0)

See Also

    Frequency/Power Hop – Get Direction
3.1.3 (57) - Frequency/Power Hop – Set Dwell Time

Declaration

```plaintext
Short Hop_SetDwell(Short dwell_msec)
```

Description

This function sets the dwell time in milliseconds; this is the length of time that the generator will pause at each point in the hop sequence.

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>dwell_msec</td>
<td>Required. The dwell time in milliseconds.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

```plaintext
Visual Basic
Status = MyPTE1.FSweep_SetDwell(15)
Visual C++
Status = MyPTE1->FSweep_SetDwell(15);
Visual C#
Status = MyPTE1.FSweep_SetDwell(15);
Matlab
Status = MyPTE1.FSweep_SetDwell(15)
```

See Also

- Frequency/Power Hop – Get Dwell Time
- Frequency/Power Hop – Get Maximum Dwell Time
- Frequency/Power Hop – Get Minimum Dwell Time
3.1.3 (58) - Frequency/Power Hop – Start/Stop Hop Sequence

Declaration

`Short Hop_SetMode(Short onoff)`

Description

This function starts or stops the hop sequence using the previously defined parameters.

Note: The hop sequence will stop automatically if any other command is sent.

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>onoff</td>
<td>Required. Integer value to enable/disable the hop sequence: 1 – Start 0 – Stop</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

**Visual Basic**

```
Status = MyPTE1.Hop_SetMode(1)   ' Start
Status = MyPTE1.Hop_SetMode(0)   ' Stop
```

**Visual C++**

```
Status = MyPTE1->Hop_SetMode(1);  // Start
Status = MyPTE1->Hop_SetMode(0);  // Stop
```

**Visual C#**

```
Status = MyPTE1.Hop_SetMode(1);   // Start
Status = MyPTE1.Hop_SetMode(0);   // Stop
```

**Matlab**

```
Status = MyPTE1.Hop_SetMode(1)    % Start
Status = MyPTE1.Hop_SetMode(0)    % Stop
```

See Also

- Frequency Sweep – Start/Stop Sweep
- Power Sweep – Start/Stop Sweep
3.1.3 (59) - Frequency/Power Hop – Set Number of Points

Declaration

\[ \text{Short} \ Hop\_SetNoOfPoints(\text{Short} \ NoOfPoints) \]

Description

This function sets the number of points to be used in the hop sequence.

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>NoOfPoints</td>
<td>Required. The number of points to hop</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

**Visual Basic**

\[ \text{Status} = \text{MyPTE1.Hop\_SetNoOfPoints}(10) \]

**Visual C++**

\[ \text{Status} = \text{MyPTE1->Hop\_SetNoOfPoints}(10); \]

**Visual C#**

\[ \text{Status} = \text{MyPTE1.Hop\_SetNoOfPoints}(10); \]

**Matlab**

\[ \text{Status} = \text{MyPTE1.Hop\_SetNoOfPoints}(10) \]

See Also

- Frequency/Power Hop – Get Maximum Number of Points
- Frequency/Power Hop – Get Hop Point
- Frequency/Power Hop – Set Number of Points
- Frequency/Power Hop – Set Hop Point
3.1.3 (60) - Frequency/Power Hop – Set Hop Point

Declaration

\[ \text{Short Hop\_SetPoint(Short PointNo, Double HopFreq, Float HopPower)} \]

Description

This function sets the frequency and power for a specific point in the hop sequence.

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>PointNo</td>
<td>Required. The point number; 1 for the first point in the sequence, 2 for the second, up to the maximum number of points.</td>
</tr>
<tr>
<td>Double</td>
<td>HopFreq</td>
<td>Required. The frequency in MHz.</td>
</tr>
<tr>
<td>Float</td>
<td>HopPower</td>
<td>Required. The power in dBm.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

**Visual Basic**

\[
\text{Status} = \text{MyPTE1.Hop\_SetPoint(3, 1000, 10)}
\]

'Set point 3 in the sequence to 1000MHz @ 10dBm

**Visual C++**

\[
\text{Status} = \text{MyPTE1->Hop\_SetPoint(3, 1000, 10);} \\
// Set point 3 in the sequence to 1000MHz @ 10dBm
\]

**Visual C#**

\[
\text{Status} = \text{MyPTE1.Hop\_SetPoint(3, 1000, 10);} \\
// Set point 3 in the sequence to 1000MHz @ 10dBm
\]

**Matlab**

\[
\text{Status} = \text{MyPTE1.Hop\_SetPoint(3, 1000, 10)}
\]

% Set point 3 in the sequence to 1000MHz @ 10dBm

See Also

- Frequency/Power Hop – Get Maximum Number of Points
- Frequency/Power Hop – Get Hop Point
- Frequency/Power Hop – Set Number of Points
- Frequency/Power Hop – Set Hop Point
3.1.3 (61) - Frequency/Power Hop – Set Trigger In Mode

Declaration

\[
\text{Short Hop\_SetTriggerIn(Short HopTriggerIn)}
\]

Description

This function specifies how the hop sequence should respond to an external trigger. The modes are:
0 – Ignore trigger input
1 – Wait for external trigger (Trigger In = logic 1) before hopping to the next point
2 – Wait for external trigger (Trigger In = logic 1) before starting the hop sequence

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>HopTriggerIn</td>
<td>Required. Integer value to specify the Trigger In mode:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 – Ignore Trigger In</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 - Wait for Trigger In before each hop</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 - Wait for Trigger In before commencing hop sequence</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

Visual Basic

\[
\text{Status} = \text{MyPTE1.\_Hop\_SetTriggerIn(1)}
\]

Visual C++

\[
\text{Status} = \text{MyPTE1->Hop\_SetTriggerIn(1)};
\]

Visual C#

\[
\text{Status} = \text{MyPTE1.Hop\_SetTriggerIn(1)};
\]

Matlab

\[
\text{Status} = \text{MyPTE1.Hop\_SetTriggerIn(1)}
\]

See Also

Frequency/Power Hop – Get Trigger In Mode
Frequency/Power Hop – Get Trigger Out Mode
Frequency/Power Hop – Set Trigger Out Mode
3.1.3 (62) - Frequency/Power Hop – Set Trigger Out Mode

Declaration

\[
\text{Short Hop\_SetTriggerOut(Short HopTriggerOut)}
\]

Description

This function specified how the Trigger Out port will be used during the hop sequence. The
modes are:
0 – Disable trigger output
1 – Provide a trigger output (Trigger Out = logic 1) on setting each point
2 – Provide a trigger output (Trigger Out = logic 1) on commencing the sequence

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>HopTrigger_Out</td>
<td>Required. Integer value to specify the Trigger Out mode:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 – Trigger Out disabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 – Set Trigger Out at each point</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 – Set Trigger Out on commencing the hop sequence</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

Visual Basic

\[
\text{Status} = \text{MyPTE1.Hop\_SetTriggerOut(1)}
\]

Visual C++

\[
\text{Status} = \text{MyPTE1->Hop\_SetTriggerOut(1)};
\]

Visual C#

\[
\text{Status} = \text{MyPTE1.Hop\_SetTriggerOut(1)};
\]

Matlab

\[
\text{Status} = \text{MyPTE1.Hop\_SetTriggerOut(1)}
\]

See Also

Frequency/Power Hop – Get Trigger In Mode
Frequency/Power Hop – Get Trigger Out Mode
Frequency/Power Hop – Set Trigger In Mode
3.1.3 (63) - Power Sweep – Get Direction

Declaration

    Short PSweep_GetDirection()  

Description

This function returns the current power sweep direction. The possible settings are:
0 – Increasing from start to stop power
1 – Decreasing from stop to start power
2 – Increasing from start to stop, before decreasing from stop to start power

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>0</td>
<td>Ascending power sweep</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Descending power sweep</td>
</tr>
<tr>
<td>Short</td>
<td>2</td>
<td>Ascending then descending power sweep</td>
</tr>
</tbody>
</table>

Examples

Visual Basic

    Sweep = MyPTE1.PSweep_GetDirection

Visual C++

    Sweep = MyPTE1->PSweep_GetDirection();

Visual C#

    Sweep = MyPTE1.PSweep_GetDirection();

Matlab

    Sweep = MyPTE1.PSweep_GetDirection

See Also

Power Sweep – Set Direction
### 3.1.3 (64) - Power Sweep – Get Dwell Time

**Declaration**

```c
Short PSweep_GetDwell()
```

**Description**

This function returns the current dwell time setting in milliseconds; this is the length of time that the generator will pause at each power setting.

Applies to SSG-6000 only.

**Parameters**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>Time</td>
<td>Dwell time in milliseconds</td>
</tr>
</tbody>
</table>

**Examples**

**Visual Basic**

```vbnet
Dwell = MyPTE1.PSweep_GetDwell
```

**Visual C++**

```c
Dwell = MyPTE1->PSweep_GetDwell();
```

**Visual C#**

```csharp
Dwell = MyPTE1.PSweep_GetDwell();
```

**Matlab**

```matlab
Dwell = MyPTE1.PSweep_GetDwell
```

**See Also**

- Power Sweep – Get Maximum Dwell Time
- Power Sweep – Get Minimum Dwell Time
- Power Sweep – Set Dwell Time
3.1.3 (65) - Power Sweep – Get Maximum Dwell Time

Declaration

```c
Short PSweep_GetMaxDwell()
```

Description

This function returns the maximum allowed dwell time in milliseconds; this is the length of time that the generator can pause at each power setting.

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>Time</td>
<td>Maximum dwell time in milliseconds</td>
</tr>
</tbody>
</table>

Examples

**Visual Basic**

```vbnet
Dwell = MyPTE1.PSweep_GetMaxDwell
```

**Visual C++**

```c
Dwell = MyPTE1->PSweep_GetMaxDwell();
```

**Visual C#**

```c
Dwell = MyPTE1.PSweep_GetMaxDwell();
```

**Matlab**

```matlab
Dwell = MyPTE1.PSweep_GetMaxDwell
```

See Also

- Power Sweep – Get Dwell Time
- Power Sweep – Get Minimum Dwell Time
- Power Sweep – Set Dwell Time
3.1.3 (66) - Power Sweep – Get Minimum Dwell Time

Declaration

```
Short PSweep_GetMinDwell()
```

Description

This function returns the minimum allowed dwell time in milliseconds; this is the length of time that the generator can pause at each power setting.

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>Time</td>
<td>Minimum dwell time in milliseconds</td>
</tr>
</tbody>
</table>

Examples

```
Visual Basic
Dwell = MyPTE1.PSweep_GetMinDwell

Visual C++
Dwell = MyPTE1->PSweep_GetMinDwell();

Visual C#
Dwell = MyPTE1.PSweep_GetMinDwell();

Matlab
Dwell = MyPTE1.PSweep_GetMinDwell
```

See Also

- Power Sweep – Get Dwell Time
- Power Sweep – Get Maximum Dwell Time
- Power Sweep – Set Dwell Time
3.1.3 (67) - Power Sweep – Get Frequency

Declaration

    Float PSweep_GetFreq()

Description

This function returns the current frequency setting of the power sweep in MHz.

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Float</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Float</td>
<td>Frequency</td>
<td>Frequency in MHz</td>
</tr>
</tbody>
</table>

Examples

```
Visual Basic
Freq = MyPTE1.PSweep_GetFreq

Visual C++
Freq = MyPTE1->PSweep_GetFreq();

Visual C#
Freq = MyPTE1.PSweep_GetFreq();

Matlab
Freq = MyPTE1.PSweep_GetFreq
```

See Also

Power Sweep – Set Frequency
3.1.3 (68) - Power Sweep – Get Start Power

Declaration

\[
\text{Double } \text{PSweep\_GetStartPower}()
\]

Description

This function returns the start power of the current power sweep in dBm.

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Double</td>
<td>Power</td>
<td>Start power in dBm</td>
</tr>
</tbody>
</table>

Examples

Visual Basic

\[
\text{Power } = \text{MyPTE1.PSweep\_GetStartPower}
\]

Visual C++

\[
\text{Power } = \text{MyPTE1->PSweep\_GetStartPower();}
\]

Visual C#

\[
\text{Power } = \text{MyPTE1.PSweep\_GetStartPower();}
\]

Matlab

\[
\text{Power } = \text{MyPTE1.PSweep\_GetStartPower}
\]

See Also

- Power Sweep – Get Stop Power
- Power Sweep – Get Step Size
- Power Sweep – Set Start Power
- Power Sweep – Set Stop Power
- Power Sweep – Set Step Size
3.1.3 (69) - Power Sweep – Get Stop Power

Declaration

    Double PSweep_GetStopPower()

Description

This function returns the stop power of the current power sweep in dBm.

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Double</td>
<td>Power</td>
<td>Stop power in dBm</td>
</tr>
</tbody>
</table>

Examples

Visual Basic

    Power = MyPTE1.PSweep_GetStopPower

Visual C++

    Power = MyPTE1->PSweep_GetStopPower();

Visual C#

    Power = MyPTE1.PSweep_GetStopPower();

Matlab

    Power = MyPTE1.PSweep_GetStopPower

See Also

    Power Sweep – Get Start Power
    Power Sweep – Get Step Size
    Power Sweep – Set Start Power
    Power Sweep – Set Stop Power
    Power Sweep – Set Step Size
3.1.3 (70) - Power Sweep – Get Step Size

Declaration

Double PSweep_GetStepSize()

Description

This function returns the step size in dBm of the current power sweep.

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Double</td>
<td>Power</td>
<td>Step size in dBm</td>
</tr>
</tbody>
</table>

Examples

Visual Basic

   Power = MyPTE1.PSweep_GetStepSize

Visual C++

   Power = MyPTE1->PSweep_GetStepSize();

Visual C#

   Power = MyPTE1.PSweep_GetStepSize();

Matlab

   Power = MyPTE1.PSweep_GetStepSize

See Also

   Power Sweep – Get Start Power
   Power Sweep – Get Stop Power
   Power Sweep – Set Start Power
   Power Sweep – Set Stop Power
   Power Sweep – Set Step Size
3.1.3 (71) - Power Sweep – Get Trigger In Mode

Declaration

    Short PSweep_GetTriggerIn()

Description

This function returns the Trigger Input mode for the power sweep, this dictates how the generator will respond to an external trigger:
0 – Ignore trigger input
1 – Wait for external trigger (Trigger In = logic 1) before setting each power
2 – Wait for external trigger (Trigger In = logic 1) before starting the power sweep

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>0</td>
<td>Ignore Trigger In</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Wait for Trigger In for each power setting</td>
</tr>
<tr>
<td>Short</td>
<td>2</td>
<td>Wait for Trigger In before commencing sweep</td>
</tr>
</tbody>
</table>

Examples

Visual Basic
    Status = MyPTE1.PSweep_GetTriggerIn

Visual C++
    Status = MyPTE1->PSweep_GetTriggerIn();

Visual C#
    Status = MyPTE1.PSweep_GetTriggerIn();

Matlab
    Status = MyPTE1.PSweep_GetTriggerIn

See Also

    Power Sweep – Get Trigger Out Mode
    Power Sweep – Set Trigger In Mode
    Power Sweep – Set Trigger Out Mode
3.1.3 (72) - Power Sweep – Get Trigger Out Mode

Declaration

```
Short PSweep_GetTriggerOut()
```

Description

This function returns Trigger Output mode for the power sweep, this dictates how the Trigger Out port will be used during the power sweep:

- 0 – Disable trigger output
- 1 – Provide a trigger output (Trigger Out = logic 1) as each power is set
- 2 – Provide a trigger output (Trigger Out = logic 1) as the power sweep is initiated

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>0</td>
<td>Trigger Out disabled</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Trigger Out set at each power</td>
</tr>
<tr>
<td>Short</td>
<td>2</td>
<td>Trigger Out set as the sweep is initialized</td>
</tr>
</tbody>
</table>

Examples

```
Visual Basic
Status = MyPTE1.PSweep_GetTriggerOut

Visual C++
Status = MyPTE1->PSweep_GetTriggerOut();

Visual C#
Status = MyPTE1.PSweep_GetTriggerOut();

Matlab
Status = MyPTE1.PSweep_GetTriggerOut
```

See Also

- Power Sweep – Get Trigger In Mode
- Power Sweep – Set Trigger In Mode
- Power Sweep – Set Trigger Out Mode
3.1.3 (73) - Power Sweep – Set Direction

Declaration

```c
Short PSweep_SetDirection(Short SweepDirection)
```

Description

This function sets the direction of the power sweep:
0 – Ascending from start to stop power
1 – Descending from stop to start power
2 – Ascending, then descending power

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>SWsweep_Direction</td>
<td>Required. Numeric value corresponding to the sweep direction mode:</td>
</tr>
</tbody>
</table>

```
0 – Ascending power
1 – Descending power
2 – Ascending, then descending power
```

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

```c
Visual Basic
Status = MyPTE1.PSweep_SetDirection(0)
```

```c
Visual C++
Status = MyPTE1->PSweep_SetDirection(0);
```

```c
Visual C#
Status = MyPTE1.PSweep_SetDirection(0);
```

Matlab

```
Status = MyPTE1.PSweep_SetDirection(0)
```

See Also

Power Sweep – Get Direction
3.1.3 (74) - Power Sweep – Set Dwell Time

Declaration

\[ \text{Short \ P Sweep\_SetDwell(Short \ dwell\_msec)} \]

Description

This function sets the dwell time in milliseconds; this is the length of time that the generator will pause at each power setting.

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>dwell_msec</td>
<td>Required. The dwell time in milliseconds.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Visual Basic</strong></td>
<td></td>
</tr>
<tr>
<td>Status = MyPTE1.PSweep_SetDwell(15)</td>
<td></td>
</tr>
<tr>
<td><strong>Visual C++</strong></td>
<td></td>
</tr>
<tr>
<td>Status = MyPTE1-&gt;PSweep_SetDwell(15);</td>
<td></td>
</tr>
<tr>
<td><strong>Visual C#</strong></td>
<td></td>
</tr>
<tr>
<td>Status = MyPTE1.PSweep_SetDwell(15);</td>
<td></td>
</tr>
<tr>
<td><strong>Matlab</strong></td>
<td></td>
</tr>
<tr>
<td>Status = MyPTE1.PSweep_SetDwell(15)</td>
<td></td>
</tr>
</tbody>
</table>

See Also

- Power Sweep – Get Dwell Time
- Power Sweep – Get Maximum Dwell Time
- Power Sweep – Get Minimum Dwell Time
3.1.3 (75) - Power Sweep – Start/Stop Sweep

Declaration

Short PSweep_SetMode(Short onoff)

Description

This function starts or stops the power sweep using the previously defined parameters.

Note: The power sweep will stop automatically if any other command is sent.

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>onoff</td>
<td>Required. Integer value to enable/disable the sweep:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 – Start power sweep</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 – Stop power sweep</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

Visual Basic

Status = MyPTE1.PSweep_SetMode(1) ' Start
Status = MyPTE1.PSweep_SetMode(0) ' Stop

Visual C++

Status = MyPTE1->PSweep_SetMode(1); // Start
Status = MyPTE1->PSweep_SetMode(0); // Stop

Visual C#

Status = MyPTE1.PSweep_SetMode(1); // Start
Status = MyPTE1.PSweep_SetMode(0); // Stop

Matlab

Status = MyPTE1.PSweep_SetMode(1) % Start
Status = MyPTE1.PSweep_SetMode(0) % Stop

See Also

- Frequency Sweep – Start/Stop Sweep
- Frequency/Power Hop – Start/Stop Hop Sequence
3.1.3 (76) - Power Sweep – Set Frequency

Declaration

\[ \text{Short} \text{ PSweep\_SetFreq(Float Fr)} \]

Description

This function sets the output power level in dBm to be used for the frequency sweep in.

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Float</td>
<td>Fr</td>
<td>Required. The fixed frequency in MHz to be used for the power sweep.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

**Visual Basic**

\[
\text{Status} = \text{MyPTE1\_PSweep\_SetFreq}(1000)
\]

**Visual C++**

\[
\text{Status} = \text{MyPTE1\_PSweep\_SetFreq}(1000);
\]

**Visual C#**

\[
\text{Status} = \text{MyPTE1\_PSweep\_SetFreq}(1000);
\]

**Matlab**

\[
\text{Status} = \text{MyPTE1\_PSweep\_SetFreq}(1000);
\]

See Also

Power Sweep – Get Frequency
3.1.3 (77) - Power Sweep – Set Start Power

Declaration

    Short PSweep_SetStartPower(Float Pr)

Description

This function sets the power in dBm at which the sweep will start.

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Float</td>
<td>Pr</td>
<td>Required. The start power in dBm.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

**Visual Basic**

    Status = MyPTE1.PSweep_SetStartPower(-10)

**Visual C++**

    Status = MyPTE1->PSweep_SetStartPower(-10);

**Visual C#**

    Status = MyPTE1.PSweep_SetStartPower(-10);

**Matlab**

    Status = MyPTE1.PSweep_SetStartPower(-10)

See Also

- Power Sweep – Get Start Power
- Power Sweep – Get Stop Power
- Power Sweep – Set Step Size
- Power Sweep – Set Stop Power
- Power Sweep – Set Step Size
### 3.1.3 (78) - Power Sweep – Set Stop Power

**Declaration**

```c
Short PSweep_SetStopPower(Float Pr)
```

**Description**

This function sets the power in dBm at which the sweep will stop.

Applies to SSG-6000 only.

**Parameters**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double</td>
<td>Fr</td>
<td>Required. The stop power in dBm.</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

**Examples**

- **Visual Basic**
  ```vb
  Status = MyPTE1.PSweep_SetStopPower(5500)
  ```

- **Visual C++**
  ```cpp
  Status = MyPTE1->PSweep_SetStopPower(5500);
  ```

- **Visual C#**
  ```cs
  Status = MyPTE1.PSweep_SetStopPower(5500);
  ```

- **Matlab**
  ```matlab
  Status = MyPTE1.PSweep_SetStopPower(5500)
  ```

**See Also**

- Power Sweep – Get Start Power
- Power Sweep – Get Stop Power
- Power Sweep – Set Step Size
- Power Sweep – Set Start Power
- Power Sweep – Set Step Size
3.1.3 (79) - Power Sweep – Set Step Size

Declaration

```
Short PSweep_SetStepSize(Float Pr)
```

Description

This function sets the step size in dBm to be used in the power sweep.

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double</td>
<td>Pr</td>
<td>Required. The step size in dBm.</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

```
Visual Basic
Status = MyPTE1.PSweep_SetStepSize(0.5)

Visual C++
Status = MyPTE1->PSweep_SetStepSize(0.5);

Visual C#
Status = MyPTE1.PSweep_SetStepSize(0.5);

Matlab
Status = MyPTE1.PSweep_SetStepSize(0.5)
```

See Also

- Power Sweep – Get Start Power
- Power Sweep – Get Stop Power
- Power Sweep – Set Start Power
- Power Sweep – Set Stop Power
- Power Sweep – Set Step Size
### 3.1.3 (80) - Power Sweep – Set Trigger In Mode

**Declaration**

```c
Short PSweep_SetTriggerIn(Short SweepTriggerIn)
```

**Description**

This function specifies how the power sweep should respond to an external trigger. The modes are:
- 0 – Ignore trigger input
- 1 – Wait for external trigger (Trigger In = logic 1) before setting each power
- 2 – Wait for external trigger (Trigger In = logic 1) before starting the power sweep

Applies to SSG-6000 only.

**Parameters**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>Sweep_TriggerIn</td>
<td>Required. Integer value to specify the Trigger In mode:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 – Ignore Trigger In</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 - Wait for Trigger In before each power</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 - Wait for Trigger In before commencing sweep</td>
</tr>
</tbody>
</table>

**Return Values**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

**Examples**

**Visual Basic**

```vbnet
Status = MyPTE1.PSweep_SetTriggerIn(1)
```

**Visual C++**

```c
Status = MyPTE1->PSweep_SetTriggerIn(1);
```

**Visual C#**

```c
Status = MyPTE1.PSweep_SetTriggerIn(1);
```

**Matlab**

```matlab
Status = MyPTE1.PSweep_SetTriggerIn(1)
```

**See Also**

- Power Sweep – Get Trigger In Mode
- Power Sweep – Get Trigger Out Mode
- Power Sweep – Set Trigger Out Mode
3.1.3 (81) - Power Sweep – Set Trigger Out Mode

Declaration

```c
Short PSweep_SetTriggerOut(Short SweepTriggerOut)
```

Description

This function specified how the Trigger Out port will be used during the power sweep. The modes are:
- 0 – Disable trigger output
- 1 – Provide a trigger output (Trigger Out = logic 1) as each power is set
- 2 – Provide a trigger output (Trigger Out = logic 1) as the power sweep is initiated

Applies to SSG-6000 only.

Parameters

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>Sweep_TriggerOut</td>
<td>Required. Integer value to specify the Trigger Out mode:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 – Trigger Out disabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 – Set Trigger Out at each power</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 – Set Trigger Out on commencing the sweep</td>
</tr>
</tbody>
</table>

Return Values

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short</td>
<td>&lt;0</td>
<td>Command failed</td>
</tr>
<tr>
<td>Short</td>
<td>1</td>
<td>Command completed successfully</td>
</tr>
</tbody>
</table>

Examples

- **Visual Basic**
  ```vbnet
  Status = MyPTE1.PSweep_SetTriggerOut(1)
  ```
- **Visual C++**
  ```cpp
  Status = MyPTE1->PSweep_SetTriggerOut(1);
  ```
- **Visual C#**
  ```csharp
  Status = MyPTE1.PSweep_SetTriggerOut(1);
  ```
- **Matlab**
  ```matlab
  Status = MyPTE1.PSweep_SetTriggerOut(1)
  ```

See Also

- Power Sweep – Get Trigger In Mode
- Power Sweep – Get Trigger Out Mode
- Power Sweep – Set Trigger In Mode
3.2 - Operating in a Linux Environment

To open a connection to Mini-Circuits Signal Generators (SSG Series), the Vendor ID and Product ID are required:

- Mini-Circuits Vendor ID: 0x20CE
- Signal generator Product ID: 0x12

Communication with the signal generator is carried out by way of USB Interrupt. The transmitted and received buffer sizes are 64 Bytes each:

- Transmit Array = [Byte 0][Byte1][Byte2]...[Byte 63]
- Returned Array = [Byte 0][Byte1][Byte2]...[Byte 63]

In most cases, the full 64 byte buffer size is not needed so any unused bytes become “don’t care” bytes; they can take on any value without affecting the operation of the signal generator.

Following a successful operation, the first byte of the returned array will mirror the code sent in the first byte of the transmit array.

A worked example is included in Appendix C of this document. The example uses the libhid and libusb libraries to interface with the signal generator as a USB HID (Human Interface Device).
3.2.1 - Summary of Commands

The commands that can be sent to the signal generator are summarized in the table below and detailed on the following pages.

<table>
<thead>
<tr>
<th>#</th>
<th>Description</th>
<th>Command Code (Byte 0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Get Device Model Name</td>
<td>40</td>
</tr>
<tr>
<td>b</td>
<td>Get Device Serial Number</td>
<td>41</td>
</tr>
<tr>
<td>c</td>
<td>Set Frequency &amp; Power (SSG-4000 Series) Set Frequency &amp; Power (SSG-6000)</td>
<td>103</td>
</tr>
<tr>
<td>d</td>
<td>Set Frequency (SSG-4000 Series) Set Frequency (SSG-6000)</td>
<td>101</td>
</tr>
<tr>
<td>e</td>
<td>Set Power</td>
<td>102</td>
</tr>
<tr>
<td>f</td>
<td>Set RF Power On/Off</td>
<td>104</td>
</tr>
<tr>
<td>g</td>
<td>Set Noise/Spur Mode</td>
<td>106</td>
</tr>
<tr>
<td>h</td>
<td>Get Generator Output Status (SSG-4000 Series) Get Generator Output Status (SSG-6000)</td>
<td>105</td>
</tr>
<tr>
<td>i</td>
<td>Get Generator Minimum Frequency</td>
<td>42</td>
</tr>
<tr>
<td>j</td>
<td>Get Generator Maximum Frequency (SSG-4000 Series) Get Generator Maximum Frequency (SSG-6000)</td>
<td>43</td>
</tr>
<tr>
<td>k</td>
<td>Get Generator Step Size Spec</td>
<td>44</td>
</tr>
<tr>
<td>l</td>
<td>Get Generator Minimum Power Spec</td>
<td>45</td>
</tr>
<tr>
<td>m</td>
<td>Get Generator Maximum Power Spec</td>
<td>46</td>
</tr>
</tbody>
</table>
### 3.2.2 - Detailed Description of Commands

#### 3.2.2 (1) - Get Device Model Name

**Description**

This function determines the Mini-Circuits part number of the connected signal generator.

Send code 40 in BYTE0 of the transmit array. BYTE1 through to BYTE63 are don’t care bytes and can be any value.

The model name is represented as a series of ASCII characters in the returned array, starting from BYTE1. The final ASCII character is contained in the byte immediately proceeding the first zero value byte. All subsequent bytes up to BYTE63 are “don’t care” bytes and could be any value.

**Transmit Array**

<table>
<thead>
<tr>
<th>Byte</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Code</td>
<td>40</td>
</tr>
</tbody>
</table>

**Returned Array**

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>...</th>
<th>Byte (N-1)</th>
<th>Byte N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Code</td>
<td>First Char</td>
<td>Second Char</td>
<td>...</td>
<td>Last Char</td>
<td>End Marker</td>
</tr>
<tr>
<td>Value</td>
<td>40</td>
<td>ASCII</td>
<td>ASCII</td>
<td>...</td>
<td>ASCII</td>
<td>0</td>
</tr>
</tbody>
</table>

**Example**

The following array would be returned for Mini-Circuits’ SSG-4000HP signal generator. See Appendix A for conversions between decimal, binary and ASCII characters.

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
<th>Byte 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Code</td>
<td>Char 1</td>
<td>Char 2</td>
<td>Char 3</td>
<td>Char 4</td>
<td>Char 5</td>
</tr>
<tr>
<td>Value</td>
<td>40</td>
<td>83</td>
<td>83</td>
<td>71</td>
<td>45</td>
<td>52</td>
</tr>
<tr>
<td>ASCII Character</td>
<td>N/A</td>
<td>S</td>
<td>S</td>
<td>G</td>
<td>-</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 6</th>
<th>Byte 7</th>
<th>Byte 8</th>
<th>Byte 9</th>
<th>Byte 10</th>
<th>Byte 11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Char 6</td>
<td>Char 7</td>
<td>Char 8</td>
<td>Char 9</td>
<td>Char 10</td>
<td>End Marker</td>
</tr>
<tr>
<td>Value</td>
<td>48</td>
<td>48</td>
<td>48</td>
<td>72</td>
<td>80</td>
<td>0</td>
</tr>
<tr>
<td>ASCII Character</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>H</td>
<td>P</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**See Also**

*Get Device Serial Number*
3.2.2 (2) - Get Device Serial Number

Description

This function determines the serial number of the connected signal generator.

Send code 41 in BYTE0 of the transmit array. BYTE1 through to BYTE63 are “don’t care” bytes and can be any value.

The serial number is represented as a series of ASCII characters in the returned array, starting from BYTE1. The final ASCII character is contained in the byte immediately proceeding the first zero value byte. All subsequent bytes up to BYTE63 are “don’t care” bytes and could be any value.

Transmit Array

<table>
<thead>
<tr>
<th>Byte</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Code</td>
</tr>
<tr>
<td>Value</td>
<td>41</td>
</tr>
</tbody>
</table>

Returned Array

<table>
<thead>
<tr>
<th>Byte</th>
<th>BYTE0</th>
<th>BYTE1</th>
<th>BYTE2</th>
<th>...</th>
<th>BYTE(N-1)</th>
<th>BYTE N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Code</td>
<td>First Char</td>
<td>Second Char</td>
<td>...</td>
<td>Last Char</td>
<td>End Marker</td>
</tr>
<tr>
<td>Value</td>
<td>41</td>
<td>ASCII</td>
<td>ASCII</td>
<td>...</td>
<td>ASCII</td>
<td>0</td>
</tr>
</tbody>
</table>

Example

The following example indicates that the current signal generator has serial number 1100040023. See Appendix A for conversions between decimal, binary and ASCII characters.

<table>
<thead>
<tr>
<th>Byte</th>
<th>BYTE0</th>
<th>BYTE1</th>
<th>BYTE2</th>
<th>BYTE3</th>
<th>BYTE4</th>
<th>BYTE5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Code</td>
<td>Char 1</td>
<td>Char 2</td>
<td>Char 3</td>
<td>Char 4</td>
<td>Char 5</td>
</tr>
<tr>
<td>Value</td>
<td>41</td>
<td>49</td>
<td>49</td>
<td>48</td>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td>ASCII Character</td>
<td>N/A</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Byte</th>
<th>BYTE6</th>
<th>BYTE7</th>
<th>BYTE8</th>
<th>BYTE9</th>
<th>BYTE10</th>
<th>BYTE11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Char 6</td>
<td>Char 7</td>
<td>Char 8</td>
<td>Char 9</td>
<td>Char 10</td>
<td>End Marker</td>
</tr>
<tr>
<td>Value</td>
<td>52</td>
<td>48</td>
<td>48</td>
<td>50</td>
<td>51</td>
<td>0</td>
</tr>
<tr>
<td>ASCII Character</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>N/A</td>
</tr>
</tbody>
</table>

See Also

Get Device Model Name
3.2.3 - Set Frequency and Power (SSG-4000 Series only)

Description

This function sets the RF output frequency and power level of the SSG-4000 Series signal generators and enables or disables the “trigger out” function.

The transmit array is made up of the following bytes:

- **BYTE0**
  - 103 (code for Set Frequency and Power)

- **BYTE1 to BYTE4**
  - Frequency in Hz broken up into 4 bytes, with MSB in BYTE1 and LSB in BYTE4
  - The value for each byte is calculated as:
    - BYTE1 = INTEGER VALUE (FREQUENCY / 256 ^ 3)
    - REMAINDER1 = FREQUENCY - BYTE1 * (256 ^ 3)
    - BYTE2 = INTEGER VALUE (REMAINDER1 / 256 ^ 2)
    - REMAINDER2 = REMAINDER1 - BYTE2 * (256 ^ 2)
    - BYTE3 = INTEGER VALUE (REMAINDER2 / 256)
    - BYTE4 = REMAINDER2 - BYTE3 * 256

- **BYTE5**
  - 1 (to set a negative power value) or 0 (to set a positive power value)

- **BYTE6 to BYTE7**
  - Absolute power in dBm multiplied by 100
  - The value is split into MSB (BYTE6) and LSB (BYTE7)
  - BYTE6 = INTEGER VALUE ((ABSOLUTE POWER * 100) / 256)
  - BYTE7 = (ABSOLUTE POWER * 100) - (BYTE6 * 256)

- **BYTE8**
  - 1 (to enable Trigger Out) or 0 (to disable Trigger Out)

- **BYTE9 to BYTE63**
  - Can be any value (“don’t care” bytes)

The returned array contains 103 in BYTE0 (the code for “Set Frequency and Power”), BYTE1 to BYTE63 are “don’t care” bytes and could be any value.

Transmit Array

<table>
<thead>
<tr>
<th>Byte</th>
<th>BYTE0</th>
<th>BYTE1</th>
<th>BYTE2</th>
<th>BYTE3</th>
<th>BYTE4</th>
<th>BYTE5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>103</td>
<td>Byte</td>
<td>Byte</td>
<td>Byte</td>
<td>Byte</td>
<td>Power</td>
</tr>
<tr>
<td></td>
<td>Code</td>
<td>Freq 0 (MSB)</td>
<td>Freq 1</td>
<td>Freq 2 (LSB)</td>
<td>0 or 1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Byte</th>
<th>BYTE6</th>
<th>BYTE7</th>
<th>BYTE8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Byte</td>
<td>Byte</td>
<td>Trigger Out</td>
</tr>
<tr>
<td></td>
<td>Power (MSB)</td>
<td>Power (LSB)</td>
<td>0 or 1</td>
</tr>
</tbody>
</table>
 Returned Array

<table>
<thead>
<tr>
<th>Byte</th>
<th>Description</th>
<th>Code</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Returned Array</td>
<td></td>
<td>103</td>
</tr>
</tbody>
</table>

Example

To set the generator output to 3501.56MHz with a power level of -5.5dBm and enable the Trigger Out:

1) Frequency = 3,501,560,000 Hz

   BYTE1 = INTEGER (3,501,560,000 / 256 ^ 3)
   = INTEGER (208.71)
   = 208

   REMAINDER1 = FREQUENCY - BYTE1 * (256 ^ 3)
   = 3,501,560,000 - 208 * (256 ^ 3)
   = 11,899,072

   BYTE2 = INTEGER (REMAINDER1 / 256 ^ 2)
   = INTEGER (11,899,072 / 256 ^ 2)
   = 181

   REMAINDER2 = REMAINDER1 - BYTE2 * (256 ^ 2)
   = 11,899,072 - 181 * (256 ^ 2)
   = 37,056

   BYTE3 = INTEGER (REMAINDER2 / 256)
   = INTEGER (37,056 / 256)
   = 144

   BYTE4 = REMAINDER2 - BYTE3 * 256
   = 37,056 - 144 * 256
   = 192

2) Power

   BYTE5 = 1 (ignore the sign in the below calculations)

   BYTE6 = INTEGER ((5.5 * 100) / 256)
   = INTEGER (2.15)
   = 2

   BYTE7 = (5.5 *100) - (2 * 256)
   = 38

3) Trigger Out

   BYTE8 = 1 to enable the trigger out
The complete transmit array is therefore:

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
<th>Byte 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Code (MSB)</td>
<td>Freq 0</td>
<td>Freq 1</td>
<td>Freq 2</td>
<td>Freq (LSB)</td>
<td>Power Sign</td>
</tr>
<tr>
<td>Value</td>
<td>103</td>
<td>208</td>
<td>181</td>
<td>144</td>
<td>192</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 6</th>
<th>Byte 7</th>
<th>Byte 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Power (MSB)</td>
<td>Power (LSB)</td>
<td>Trigger Out</td>
</tr>
<tr>
<td>Value</td>
<td>2</td>
<td>38</td>
<td>1</td>
</tr>
</tbody>
</table>

See Also

Set Frequency and Power (SSG-6000 only)
Set Frequency (SSG-4000 Series Only)
Set Power
Set RF Power On/Off
Get Generator Output Status
3.2.3 (1) - Set Frequency and Power (SSG-6000 only)

Description

This function sets the RF output frequency and power level of the SSG-6000 signal generator and enables or disables the “trigger out” function.

The transmit array is made up of the following bytes:

- **BYTE0**
  - 103 (code for Set Frequency and Power)

- **BYTE1 to BYTE5**
  - Frequency in Hz broken up into 5 bytes, with MSB in BYTE1 and LSB in BYTE5
  - The value for each byte is calculated as:
    - BYTE1 = INTEGER VALUE (FREQUENCY / 256 ^ 4)
    - REMAINDER1 = FREQUENCY - BYTE1 * (256 ^ 4)
    - BYTE2 = INTEGER VALUE (REMAINDER1 / 256 ^ 3)
    - REMAINDER2 = REMAINDER1 - BYTE2 * (256 ^ 3)
    - BYTE3 = INTEGER VALUE (REMAINDER2 / 256 ^ 2)
    - REMAINDER3 = REMAINDER2 - BYTE3 * (256 ^ 2)
    - BYTE4 = INTEGER VALUE (REMAINDER3 / 256)
    - BYTE5 = INTEGER VALUE (REMAINDER3 - BYTE4 * 256)

- **BYTE6**
  - 1 (to set a negative power value) or 0 (to set a positive power value)

- **BYTE7 to BYTE8**
  - Absolute power in dBm multiplied by 100
  - The value is split into MSB (BYTE7) and LSB (BYTE8)
    - BYTE7 = INTEGER VALUE ((ABSOLUTE POWER * 100) / 256)
    - BYTE8 = (ABSOLUTE POWER * 100) - (BYTE6 * 256)

- **BYTE9**
  - 1 (to enable Trigger Out) or (0 to disable Trigger Out)

- **BYTE10 to BYTE63**
  - Can be any value (“don’t care” bytes)

The returned array contains 103 in BYTE0 (the code for “Set Frequency and Power”), BYTE1 to BYTE63 are “don’t care” bytes and could be any value.

Transmit Array

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
<th>Byte 5</th>
<th>Byte 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Code</td>
<td>Freq 0 (MSB)</td>
<td>Freq 1</td>
<td>Freq 2</td>
<td>Freq 3</td>
<td>Freq (LSB)</td>
<td>Power Sign</td>
</tr>
<tr>
<td>Value</td>
<td>103</td>
<td>Byte</td>
<td>Byte</td>
<td>Byte</td>
<td>Byte</td>
<td>Byte</td>
<td>0 or 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 7</th>
<th>Byte 8</th>
<th>Byte 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Power (MSB)</td>
<td>Power (LSB)</td>
<td>Trigger Out</td>
</tr>
<tr>
<td>Value</td>
<td>Byte</td>
<td>Byte</td>
<td>0 or 1</td>
</tr>
</tbody>
</table>
Returned Array

<table>
<thead>
<tr>
<th>Byte</th>
<th>Code</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>103</td>
<td></td>
</tr>
</tbody>
</table>

Example

To set the generator output to 5501.56MHz with a power level of +4.5dBm and enable the Trigger Out:

4) Frequency = 5,501,560,000 Hz

\[
\text{BYTE1} = \text{INTEGER} \left( \frac{5,501,560,000}{256} \right)^4 \\
= \text{INTEGER} (1.2809) \\
= 1
\]

\[
\text{REMAINDER1} = 5,501,560,000 - 1 \times (256)^4 \\
= 1,206,592,704
\]

\[
\text{BYTE2} = \text{INTEGER} \left( \frac{1,206,592,704}{256} \right)^3 \\
= 71
\]

\[
\text{REMAINDER2} = 1,206,592,704 - 71 \times (256)^3 \\
= 60,196
\]

\[
\text{BYTE3} = \text{INTEGER} \left( \frac{60,196}{256} \right)^2 \\
= 235
\]

\[
\text{REMAINDER3} = 1,206,592,704 - 71 \times (256)^2 \\
= 9,408
\]

\[
\text{BYTE4} = \text{INTEGER} \left( \frac{9,408}{256} \right) \\
= 36
\]

\[
\text{BYTE5} = \text{INTEGER} \left( \frac{9,408 - (36 \times 256)}{256} \right) \\
= 192
\]

5) Power

\[
\text{BYTE6} = 0 \text{ (ignore the sign in the below calculations)}
\]

\[
\text{BYTE7} = \text{INTEGER} \left( \frac{(\text{POWER} \times 100)}{256} \right) \\
= \text{INTEGER} \left( \frac{4.5 \times 100}{256} \right) \\
= \text{INTEGER} (1.76) \\
= 1
\]

\[
\text{BYTE8} = \left( \frac{\text{POWER} \times 100}{256} \right) - \left( \frac{\text{BYTE7} \times 256}{256} \right) \\
= \left( \frac{4.5 \times 100}{256} \right) - (1 \times 256) \\
= 194
\]
6) Trigger Out

\[
\text{BYTE9} = 1 \text{ to enable the trigger out}
\]

The complete transmit array is therefore:

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
<th>Byte 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Code</td>
<td>Freq 0 (MSB)</td>
<td>Freq 1</td>
<td>Freq 2</td>
<td>Freq 3</td>
<td>Freq (LSB)</td>
</tr>
<tr>
<td>Value</td>
<td>103</td>
<td>1</td>
<td>71</td>
<td>235</td>
<td>36</td>
<td>192</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 6</th>
<th>Byte 7</th>
<th>Byte 8</th>
<th>Byte 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Power</td>
<td>Power</td>
<td>Power</td>
<td>Trigger</td>
</tr>
<tr>
<td>Sign (MSB)</td>
<td>(LSB)</td>
<td>Out</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>0</td>
<td>1</td>
<td>194</td>
<td>1</td>
</tr>
</tbody>
</table>

See Also

- Set Frequency and Power (SSG-4000 Series only)
- Set Frequency (SSG-6000 Only)
- Set Power
- Set RF Power On/Off
- Get Generator Output Status
3.2.3 (2) - Set Frequency (SSG-4000 Series Only)

Description

This function sets the RF output frequency of the SSG-4000 series signal generators and enables or disables the “trigger out” function. It does not affect the current power setting.

The transmit array is made up of the following bytes:
- BYTE0
  - 101 (code for Set Frequency)
- BYTE1 to BYTE4
  - Frequency in Hz broken up into 4 bytes, with MSB in BYTE1 and LSB in BYTE4
  - The value for each byte is calculated as:
    - BYTE1 = INTEGER VALUE (FREQUENCY / 256^3)
    - REMAINDER1 = FREQUENCY - BYTE1 * (256^3)
    - BYTE2 = INTEGER VALUE (REMAINDER1 / 256^2)
    - REMAINDER2 = REMAINDER1 - BYTE2 * (256^2)
    - BYTE3 = INTEGER VALUE (REMAINDER2 / 256)
    - BYTE4 = REMAINDER2 - BYTE3 * 256
- BYTE5
  - 1 (to enable Trigger Out) or (0 to disable Trigger Out)
- BYTE6 to BYTE63
  - Can be any value ("don’t care" bytes)

The returned array contains 101 in BYTE0 (the code for “Set Frequency”), BYTE1 to BYTE63 are “don’t care” bytes and could be any value.

Transmit Array

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
<th>Byte 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Code</td>
<td>Freq 0 (MSB)</td>
<td>Freq 1</td>
<td>Freq 2</td>
<td>Freq (LSB)</td>
<td>Trigger Out</td>
</tr>
<tr>
<td>Value</td>
<td>101</td>
<td>Byte</td>
<td>Byte</td>
<td>Byte</td>
<td>Byte</td>
<td>0 or 1</td>
</tr>
</tbody>
</table>

Returned Array

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Code</td>
</tr>
<tr>
<td>Value</td>
<td>101</td>
</tr>
</tbody>
</table>
Example

To set the generator output to 1000.55MHz and enable the Trigger Out:

1) Frequency = 1,000,550,000 Hz

\[
\begin{align*}
\text{BYTE1} &= \text{INTEGER} \left( \frac{1,000,550,000}{256^3} \right) \\
&= \text{INTEGER} (59.64) \\
&= 59 \\

\text{REMAINDER1} &= \text{FREQUENCY} - \text{BYTE1} \times (256^3) \\
&= 1,000,550,000 - 59 \times (256^3) \\
&= 10,694,256 \\

\text{BYTE2} &= \text{INTEGER} \left( \frac{\text{REMAINDER1}}{256^2} \right) \\
&= \text{INTEGER} \left( \frac{10,694,256}{256^2} \right) \\
&= 163 \\

\text{REMAINDER2} &= \text{REMAINDER1} - \text{BYTE2} \times (256^2) \\
&= 10,694,256 - 163 \times (256^2) \\
&= 11,888 \\

\text{BYTE3} &= \text{INTEGER} \left( \frac{\text{REMAINDER2}}{256} \right) \\
&= \text{INTEGER} \left( \frac{11,888}{256} \right) \\
&= 46 \\

\text{BYTE4} &= \text{REMAINDER2} - \text{BYTE3} \times 256 \\
&= 11,888 - 46 \times 256 \\
&= 112 \\

2) Trigger Out

\text{BYTE5} = 1 \text{ to enable the trigger out}

3) The complete transmit array is therefore:

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
<th>Byte 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Code (MSB)</td>
<td>Freq 1</td>
<td>Freq 0 (MSB)</td>
<td>Freq 2</td>
<td>Freq (LSB)</td>
<td>Trigger Out</td>
</tr>
<tr>
<td>Value</td>
<td>101</td>
<td>59</td>
<td>163</td>
<td>46</td>
<td>112</td>
<td>1</td>
</tr>
</tbody>
</table>

See Also

- Set Frequency and Power (SSG-4000 Series only)
- Set Frequency (SSG-6000 Only)
- Set Power
- Set RF Power On/Off
- Get Generator Output Status
3.2.3 (3) - Set Frequency (SSG-6000 Only)

Description

This function sets the output frequency of the SSG-6000 signal generator and enables or disables the “trigger out” function. It does not affect the current power setting.

The transmit array is made up of the following bytes:

- **BYTE0**
  - 101 (code for Set Frequency)

- **BYTE1 to BYTE5**
  - Frequency in Hz broken up into 5 bytes, with MSB in BYTE1 and LSB in BYTE5
  - The value for each byte is calculated as:
    - BYTE1 = INTEGER VALUE (FREQUENCY / 256 ^ 4)
    - REMAINDER1 = FREQUENCY - BYTE1 * (256 ^ 4)
    - BYTE2 = INTEGER VALUE (REMAINDER1 / 256 ^ 3)
    - REMAINDER2 = REMAINDER1 - BYTE2 * (256 ^ 3)
    - BYTE3 = INTEGER VALUE (REMAINDER2 / 256 ^ 2)
    - REMAINDER3 = REMAINDER2 - BYTE3 * (256 ^ 2)
    - BYTE4 = INTEGER VALUE (REMAINDER3 / 256)
    - BYTE5 = INTEGER VALUE (REMAINDER3 - BYTE4 * 256)

- **BYTE6**
  - 1 (to enable Trigger Out) or (0 to disable Trigger Out)

- **BYTE7 to BYTE63**
  - Can be any value (“don’t care” bytes)

The returned array contains 101 in BYTE0 (the code for “Set Frequency”), BYTE1 to BYTE63 are “don’t care” bytes and could be any value.

Transmit Array

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
<th>Byte 5</th>
<th>Byte 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Code</td>
<td>Freq 0 (MSB)</td>
<td>Freq 1</td>
<td>Freq 2</td>
<td>Freq 3</td>
<td>Freq (LSB)</td>
<td>Trigger Out</td>
</tr>
<tr>
<td>Value</td>
<td>101</td>
<td>Byte</td>
<td>Byte</td>
<td>Byte</td>
<td>Byte</td>
<td>Byte</td>
<td>0 or 1</td>
</tr>
</tbody>
</table>

Returned Array

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Code</td>
</tr>
<tr>
<td>Value</td>
<td>101</td>
</tr>
</tbody>
</table>
Example

To set the generator output to 4100.55MHz and enable the Trigger Out:

4) Frequency = 4,100,550,000 Hz

\[
\text{BYTE1} = \text{INTEGER} \left( \frac{4,100,550,000}{256^4} \right) = \text{INTEGER} (0.9547) = 0
\]

\[
\text{REMAININDER1} = 4,100,550,000 - 0 \times (256^4) = 4,100,550,000
\]

\[
\text{BYTE2} = \text{INTEGER} \left( \frac{4,100,550,000}{256^3} \right) = 244
\]

\[
\text{REMAININDER2} = 4,100,550,000 - 244 \times (256^3) = 6,909,296
\]

\[
\text{BYTE3} = \text{INTEGER} \left( \frac{6,909,296}{256^2} \right) = 105
\]

\[
\text{REMAININDER3} = 6,909,296 - 105 \times (256^2) = 28,016
\]

\[
\text{BYTE4} = \text{INTEGER} \left( \frac{28,016}{256} \right) = 109
\]

\[
\text{BYTE5} = \text{INTEGER} \left( 28,016 - (109 \times 256) \right) = 112
\]

5) Trigger Out

\[
\text{BYTE6} = 1 \text{ to enable the trigger out}
\]

6) The complete transmit array is therefore:

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
<th>Byte 5</th>
<th>Byte 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Code</td>
<td>Freq 0 (MSB)</td>
<td>Freq 1</td>
<td>Freq 2</td>
<td>Freq 3</td>
<td>Freq (LSB)</td>
<td>Trigger Out</td>
</tr>
<tr>
<td>Value</td>
<td>101</td>
<td>0</td>
<td>244</td>
<td>105</td>
<td>109</td>
<td>112</td>
<td>1</td>
</tr>
</tbody>
</table>

See Also

Set Frequency and Power (SSG-6000 only)
Set Frequency (SSG-4000 Series Only)
Set Power
Set RF Power On/Off
Get Generator Output Status
3.2.3 (4) - Set Power

Description

This function sets the RF output power of the signal generator and enables or disables the “trigger out” function. It does not affect the current frequency setting.

The transmit array is made up of the following bytes:

- **BYTE0**
  - 102 (code for Set Power)
- **BYTE1**
  - 1 (to set a negative power value) or 0 (to set a positive power value)
- **BYTE2** to **BYTE3**
  - Absolute power in dBm multiplied by 100 (to allow fine resolution)
  - The value is split into MSB (BYTE2) and LSB (BYTE3)
  - BYTE2 = INTEGER VALUE ((ABSOLUTE POWER * 100) / 256)
  - BYTE3 = (ABSOLUTE POWER * 100) - (BYTE2 * 256)
- **BYTE4**
  - 1 (to enable Trigger Out) or 0 (to disable Trigger Out)
- **BYTE5** to **BYTE63**
  - Can be any value (“don’t care” bytes)

The returned array contains 102 in BYTE0 (the code for “Set Frequency and Power”), BYTE1 to BYTE63 are “don’t care” bytes and could be any value.

Transmit Array

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Code</td>
<td>Power</td>
<td>Sign</td>
<td>Power (MSB)</td>
<td>Power (LSB)</td>
</tr>
<tr>
<td>Value</td>
<td>102</td>
<td>0 or 1</td>
<td>Byte</td>
<td>Byte</td>
<td>0 or 1</td>
</tr>
</tbody>
</table>

Returned Array

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Code</td>
</tr>
<tr>
<td>Value</td>
<td>102</td>
</tr>
</tbody>
</table>
Example

To set the generator output power to -5.5dBm and enable the Trigger Out:

1) Power

   BYTE1 = 1 since the power is negative (ignore the sign in the below calculations)

   BYTE2 = INTEGER ((POWER * 100) / 256)
   = INTEGER ((5.5 * 100) / 256)
   = INTEGER (2.15)
   = 2

   BYTE3 = (POWER * 100) - (BYTE2 * 256)
   = (5.5 * 100) - (2 * 256)
   = 38

2) Trigger Out

   BYTE4 = 1 to enable the trigger out

The complete transmit array is therefore:

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Code</td>
<td>Power</td>
<td>Sign</td>
<td>Power</td>
<td>(MSB)</td>
</tr>
<tr>
<td>Value</td>
<td>102</td>
<td>1</td>
<td>2</td>
<td>38</td>
<td>1</td>
</tr>
</tbody>
</table>

See Also

Set Frequency and Power
Set Frequency
Set RF Power On/Off
Get Generator Output Status
3.2.3 (5) - Set RF Power On/Off

Description

This function enables or disables the RF output of the signal generator.

Send code 104 in BYTE0 of the transmit array with BYTE1 as 1 to enable or 0 to disable the RF output. BYTE2 to BYTE63 are “don’t care” bytes and can be any value.

The returned array contains 104 in BYTE0. BYTE1 to BYTE63 are “don’t care” bytes and could be any value.

Transmit Array

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 0</th>
<th>Byte 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Code</td>
<td>Power On/Off</td>
</tr>
<tr>
<td>Value</td>
<td>104</td>
<td>1 or 0</td>
</tr>
</tbody>
</table>

Returned Array

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Code</td>
</tr>
<tr>
<td>Value</td>
<td>104</td>
</tr>
</tbody>
</table>

Example

To enable the RF output, send the following transmit array:

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 0</th>
<th>Byte 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Code</td>
<td>Power On/Off</td>
</tr>
<tr>
<td>Value</td>
<td>104</td>
<td>1</td>
</tr>
</tbody>
</table>

To disable the RF output, send the following transmit array:

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 0</th>
<th>Byte 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Code</td>
<td>Power On/Off</td>
</tr>
<tr>
<td>Value</td>
<td>104</td>
<td>0</td>
</tr>
</tbody>
</table>

See Also

Set Frequency and Power (SSG-4000 Series only)
Set Frequency and Power (SSG-6000 only)
Set Power
Get Generator Output Status (SSG-4000 Series)
Get Generator Output Status (SSG-6000 Only)
3.2.3 (6) - Set Noise/Spur Mode

Description

This function applies to SSG-4000LH and SSG-4000HP only. It sets the generator in either “Low Noise” mode (for best phase noise performance) or “Low Spur” mode (for best spurious performance). The generator defaults to “Low Noise” mode.

Send code 106 in BYTE0 of the transmit array with BYTE1 as 1 to enable “Low Spur” mode or 0 to enable “Low Noise” mode. BYTE2 to BYTE63 are “don’t care” bytes and can be any value.

The returned array contains 106 in BYTE0. BYTE1 to BYTE63 are “don’t care” bytes and could be any value.

Transmit Array

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 0</th>
<th>Byte 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Code</td>
<td>Mode</td>
</tr>
<tr>
<td>Value</td>
<td>106</td>
<td>1 or 0</td>
</tr>
</tbody>
</table>

Returned Array

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Code</td>
</tr>
<tr>
<td>Value</td>
<td>106</td>
</tr>
</tbody>
</table>

Example

To set “Low Spur” mode, send the following transmit array:

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 0</th>
<th>Byte 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Code</td>
<td>Mode</td>
</tr>
<tr>
<td>Value</td>
<td>106</td>
<td>1</td>
</tr>
</tbody>
</table>

To set “Low Noise” mode, send the following transmit array:

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 0</th>
<th>Byte 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Code</td>
<td>Mode</td>
</tr>
<tr>
<td>Value</td>
<td>106</td>
<td>0</td>
</tr>
</tbody>
</table>
3.2.3 (7) - Get Generator Output Status (SSG-4000 Series)

Description

This function returns the current output status of the SSG-4000 series signal generators. The following parameters are checked:

- Generator lock status (locked/unlocked)
- RF output status (on/off)
- Current output frequency
- Current output power
- Current output power relative to user requested level

Send code 105 in BYTE0 of the transmit array. BYTE2 to BYTE63 are “don’t care” bytes and can be any value.

The returned array is made up of the following bytes:

- **BYTE0**
  - 105 (code for Get Generator Status)
- **BYTE1**
  - RF output status (1 if enabled or 0 if disabled)
- **BYTE2**
  - Lock status (1 if frequency is locked or 0 if not locked)
- **BYTE3 to BYTE6**
  - Frequency in Hz, broken up into 4 bytes with MSB in BYTE3 and LSB in BYTE6
  - \( \text{FREQUENCY} = (256 ^ 3) \times \text{BYTE3} + (256 ^ 2) \times \text{BYTE4} + 256 \times \text{BYTE5} + \text{BYTE6} \)
- **BYTE7**
  - Output power sign (1 if power value is negative or 0 if positive)
- **BYTE8 to BYTE9**
  - Power setting in dBm (absolute value), split into MSB (BYTE8) and LSB (BYTE9)
  - Absolute power is calculated as: \( P = (256 \times \text{BYTE8} + \text{BYTE9}) / 100 \)
- **BYTE10**
  - High power request warning (UnLevel High)
  - 1 if the user requested a higher power than the generator can achieve or 0 if the output power is within the correct range
- **BYTE11**
  - Low power request warning (UnLevel Low)
  - 1 if the user requested a lower power than the generator can achieve or 0 if the output power is within the correct range
- **BYTE12 to BYTE63**
  - Could be any value (“don’t care” bytes)

**Transmit Array**

<table>
<thead>
<tr>
<th>Byte</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte 0</td>
<td>Code</td>
<td>105</td>
</tr>
</tbody>
</table>
Returned Array

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
<th>Byte 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Code</td>
<td>RF On/Off</td>
<td>Lock Status</td>
<td>Freq (MSB)</td>
<td>Freq 1</td>
<td>Freq 2</td>
</tr>
<tr>
<td>Value</td>
<td>105</td>
<td>1 or 0</td>
<td>1 or 0</td>
<td>Byte</td>
<td>Byte</td>
<td>Byte</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 6</th>
<th>Byte 7</th>
<th>Byte 8</th>
<th>Byte 9</th>
<th>Byte 10</th>
<th>Byte 11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Freq (LSB)</td>
<td>Power + or -</td>
<td>Power (MSB)</td>
<td>Power (LSB)</td>
<td>UnLevel High</td>
<td>UnLevel Low</td>
</tr>
<tr>
<td>Value</td>
<td>Byte</td>
<td>Byte</td>
<td>Byte</td>
<td>Byte</td>
<td>0 or 1</td>
<td>0 or 1</td>
</tr>
</tbody>
</table>

Example

The following array would be returned if the generator was set with the output enabled at 751.25 MHz, +5.5 dBm:

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
<th>Byte 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Code</td>
<td>RF On/Off</td>
<td>Lock Status</td>
<td>Freq (MSB)</td>
<td>Freq 1</td>
<td>Freq 2</td>
</tr>
<tr>
<td>Value</td>
<td>105</td>
<td>1</td>
<td>1</td>
<td>44</td>
<td>199</td>
<td>42</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 6</th>
<th>Byte 7</th>
<th>Byte 8</th>
<th>Byte 9</th>
<th>Byte 10</th>
<th>Byte 11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Freq (LSB)</td>
<td>Power + or -</td>
<td>Power (MSB)</td>
<td>Power (LSB)</td>
<td>UnLevel High</td>
<td>UnLevel Low</td>
</tr>
<tr>
<td>Value</td>
<td>80</td>
<td>0</td>
<td>2</td>
<td>38</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The returned array is broken down as follows:
- RF output is enabled (BYTE1 = 1)
- Generator is locked (BYTE2 = 1)
- Frequency
  \[ \text{Frequency} = (256^3) \times \text{BYTE3} + (256^2) \times \text{BYTE4} + 256 \times \text{BYTE5} + \text{BYTE6} \]
  \[ = (256^3) \times 44 + (256^2) \times 199 + 256 \times 42 + 80 \]
  \[ = 751,250,000 \text{ Hz} \]
  \[ = 751.25 \text{ MHz} \]
- Absolute power
  \[ = (256 \times \text{BYTE2} + \text{BYTE3}) / 100 \]
  \[ = (256 \times 2 + 38) / 100 \]
  \[ = 5.5 \text{ dBm} \]
- Real power
  \[ = +5.5 \text{ dBm (since BYTE7 = 1)} \]
- Power has settled to user defined level (BYTE10 = 0 and BYTE11 = 0)

See Also

- Set Frequency and Power (SSG-4000 Series only)
- Set Frequency (SSG-4000 Series Only)
- Set Power
- Set RF Power On/Off
3.2.3 (8) - Get Generator Output Status (SSG-6000 Only)

Description

This function returns the current output status of the SSG-6000 signal generator. The following parameters are checked:

- Generator lock status (locked/unlocked)
- RF output status (on/off)
- Current output frequency
- Current output power
- Current output power relative to user requested level

Send code 105 in BYTE0 of the transmit array. BYTE2 to BYTE63 are “don’t care” bytes and can be any value.

The returned array is made up of the following bytes:

- BYTE0 105 (code for Get Generator Status)
- BYTE1 RF output status (1 if enabled or 0 if disabled)
- BYTE2 Lock status (1 if frequency is locked or 0 if not locked)
- BYTE3 to BYTE7 Frequency in Hz, broken up into 4 bytes with MSB in BYTE3 and LSB in BYTE6

\[
FREQUENCY = (256^4) \times \text{BYTE3} + (256^3) \times \text{BYTE4} + (256^2) \times \text{BYTE5} + 256 \times \text{BYTE6} + \text{BYTE7}
\]

- BYTE8 Output power sign (1 if power value is negative or 0 if positive)
- BYTE9 to BYTE10 Power setting in dBm (absolute value), split into MSB (BYTE8) and LSB (BYTE9)

Absolute power is calculated as:

\[
P = (256 \times \text{BYTE9} + \text{BYTE10}) / 100
\]

- BYTE11 High power request warning (UnLevel High)

1 if the user requested a higher power than the generator can achieve or 0 if the output power is within the correct range

- BYTE12 Low power request warning (UnLevel Low)

1 if the user requested a lower power than the generator can achieve or 0 if the output power is within the correct range

- BYTE13 to BYTE63 Could be any value (“don’t care” bytes)

Transmit Array

<table>
<thead>
<tr>
<th>Byte</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Code</td>
<td>105</td>
</tr>
</tbody>
</table>
Returned Array

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
<th>Byte 5</th>
<th>Byte 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Code</td>
<td>RF On/Off</td>
<td>Lock Status</td>
<td>Freq (MSB)</td>
<td>Freq 1</td>
<td>Freq 2</td>
<td>Freq 3</td>
</tr>
<tr>
<td>Value</td>
<td>105</td>
<td>1 or 0</td>
<td>1 or 0</td>
<td>Byte</td>
<td>Byte</td>
<td>Byte</td>
<td>Byte</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 7</th>
<th>Byte 8</th>
<th>Byte 9</th>
<th>Byte 10</th>
<th>Byte 11</th>
<th>Byte 12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Freq (LSB)</td>
<td>Power + or -</td>
<td>Power (MSB)</td>
<td>Power (LSB)</td>
<td>UnLevel High</td>
<td>UnLevel Low</td>
</tr>
<tr>
<td>Value</td>
<td>Byte</td>
<td>0 or 1</td>
<td>Byte</td>
<td>0 or 1</td>
<td>0 or 1</td>
<td></td>
</tr>
</tbody>
</table>

Example

The following array would be returned if the generator was set with the output enabled at 4980.50MHz, +5.5dBm):

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
<th>Byte 5</th>
<th>Byte 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Code</td>
<td>RF On/Off</td>
<td>Lock Status</td>
<td>Freq (MSB)</td>
<td>Freq 1</td>
<td>Freq 2</td>
<td>Freq 3</td>
</tr>
<tr>
<td>Value</td>
<td>105</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>40</td>
<td>220</td>
<td>102</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 7</th>
<th>Byte 8</th>
<th>Byte 9</th>
<th>Byte 10</th>
<th>Byte 11</th>
<th>Byte 12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Freq (LSB)</td>
<td>Power + or -</td>
<td>Power (MSB)</td>
<td>Power (LSB)</td>
<td>UnLevel High</td>
<td>UnLevel Low</td>
</tr>
<tr>
<td>Value</td>
<td>32</td>
<td>0</td>
<td>2</td>
<td>38</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The returned array is broken down as follows:
- RF output is enabled (BYTE1 = 1)
- Generator is locked (BYTE2 = 1)
- Frequency
  \[
  \text{Frequency} = (256 \times 4) \times \text{BYTE3} + (256 \times 3) \times \text{BYTE4} + (256 \times 2) \times \text{BYTE5} + 256 \times \text{BYTE6} + \text{BYTE7} \\
  = (256 \times 4) \times 1 + (256 \times 3) \times 40 + (256 \times 2) \times 220 + 256 + 102 + 32 \\
  = 751,250,000 \text{ Hz} \\
  = 751.25 \text{ MHz}
  \]
- Absolute power
  \[
  \text{Absolute power} = (256 \times \text{BYTE2} + \text{BYTE3}) / 100 \\
  = (256 \times 2 + 38) / 100 \\
  = 5.5\text{dBm}
  \]
- Real power
  \[
  \text{Real power} = +5.5\text{dBm} \text{ (since BYTE7 = 1)}
  \]
- Power has settled to user defined level (BYTE10 = 0 and BYTE11 = 0)

See Also

Set Frequency and Power (SSG-6000 only)
Set Frequency (SSG-6000 Only)
Set Power
Set RF Power On/Off
3.2.3 (9) - Get Generator Minimum Frequency

Description

This function reports the signal generator minimum frequency specification in Hz.

Send code 42 in BYTE0 of the transmit array. BYTE2 to BYTE63 are “don’t care” bytes and can be any value.

The returned array is made up of the following bytes:

- BYTE0
  - 42 (code for Get Generator Minimum Frequency)
- BYTE1 to BYTE4
  - Frequency in Hz, broken up into 4 bytes with MSB in BYTE3 and LSB in BYTE6
  - FREQUENCY = (256 ^ 3) * BYTE1 + (256 ^ 2) * BYTE2 + 256 * BYTE3 + BYTE4
- BYTE5 to BYTE63
  - Could be any value (“don’t care” bytes)

Transmit Array

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 0</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>42</td>
<td>Code</td>
<td></td>
</tr>
</tbody>
</table>

Returned Array

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>42</td>
<td>Byte</td>
<td>Byte</td>
<td>Byte</td>
<td>Byte</td>
</tr>
<tr>
<td>Description</td>
<td>Code</td>
<td>Freq (MSB)</td>
<td>Freq 1</td>
<td>Freq 2</td>
<td>Freq (LSB)</td>
</tr>
<tr>
<td>Value</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

The following array would be returned for SSG-4000HP:

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Code</td>
<td>Freq (MSB)</td>
<td>Freq 1</td>
<td>Freq 2</td>
<td>Freq (LSB)</td>
</tr>
<tr>
<td>Value</td>
<td>42</td>
<td>14</td>
<td>230</td>
<td>178</td>
<td>128</td>
</tr>
</tbody>
</table>

The minimum frequency spec can be calculated as follows:

\[
\text{Min Frequency} = (256^3) \times \text{BYTE}_1 + (256^2) \times \text{BYTE}_2 + 256 \times \text{BYTE}_3 + \text{BYTE}_4 \\
= (256^3) \times 14 + (256^2) \times 230 + 256 \times 178 + 128 \\
= 250,000,000 \text{ Hz} \\
= 250 \text{ MHz}
\]

See Also

- Get Generator Maximum Frequency (SSG-4000 Series Only)
- Get Generator Maximum Frequency (SSG-6000 Only)
- Get Generator Step Size
- Get Generator Minimum Power
- Get Generator Maximum Power
3.2.3 (10) - Get Generator Maximum Frequency (SSG-4000 Series Only)

Description

This function reports the maximum frequency specification in Hz for the SSG-4000 series signal generators.

Send code 43 in BYTE0 of the transmit array. BYTE2 to BYTE63 are “don’t care” bytes and can be any value.

The returned array is made up of the following bytes:
- BYTE0
  - 43 (code for Get Generator Maximum Frequency)
- BYTE1 to BYTE4
  - FREQUENCY = (256 ^ 3) * BYTE1 + (256 ^ 2) * BYTE2 + 256 * BYTE3 + BYTE4
- BYTE5 to BYTE63
  - Could be any value (“don’t care” bytes)

Transmit Array

<table>
<thead>
<tr>
<th>Byte</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Code</td>
<td>43</td>
</tr>
</tbody>
</table>

Returned Array

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Code</td>
<td>Freq (MSB)</td>
<td>Freq 1</td>
<td>Freq 2</td>
<td>Freq (LSB)</td>
</tr>
<tr>
<td>Value</td>
<td>43</td>
<td>Byte</td>
<td>Byte</td>
<td>Byte</td>
<td>Byte</td>
</tr>
</tbody>
</table>
Example

The following array would be returned for SSG-4000HP:

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Code</td>
<td>Freq (MSB)</td>
<td>Freq 1</td>
<td>Freq 2</td>
<td>Freq (LSB)</td>
</tr>
<tr>
<td>Value</td>
<td>43</td>
<td>238</td>
<td>107</td>
<td>40</td>
<td>0</td>
</tr>
</tbody>
</table>

The maximum frequency spec can be calculated as follows:

\[
\text{Max Frequency} = (256 ^ 3) \times \text{BYTE1} + (256 ^ 2) \times \text{BYTE2} + 256 \times \text{BYTE3} + \text{BYTE4}
\]

\[
= (256 ^ 3) \times 238 + (256 ^ 2) \times 107 + 256 \times 40 + 0
\]

\[
= 4,000,000,000 \text{ Hz}
\]

\[
= 4,000 \text{ MHz}
\]

See Also

- Get Generator Maximum Frequency (SSG-6000 Only)
- Get Generator Minimum Frequency
- Get Generator Step Size
- Get Generator Minimum Power
- Get Generator Maximum Power
3.2.3 (11) - Get Generator Maximum Frequency (SSG-6000 Only)

Description

This function reports the maximum frequency specification in Hz for the SSG-6000 signal generators.

Send code 43 in BYTE0 of the transmit array. BYTE2 to BYTE63 are “don’t care” bytes and can be any value.

The returned array is made up of the following bytes:
- BYTE0
  - 43 (code for Get Generator Maximum Frequency)
- BYTE1 to BYTE5
  - FREQUENCY = (256 ^ 4) * BYTE1 + (256 ^ 3) * BYTE2 + (256 ^ 2) * BYTE3 + 256 * BYTE4 + BYTE5
- BYTE6 to BYTE63
  - Could be any value (“don’t care” bytes)

Transmit Array

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Code</td>
</tr>
<tr>
<td>Value</td>
<td>43</td>
</tr>
</tbody>
</table>

Returned Array

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
<th>Byte 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Code</td>
<td>Freq (MSB)</td>
<td>Freq 1</td>
<td>Freq 2</td>
<td>Freq 3</td>
<td>Freq (LSB)</td>
</tr>
<tr>
<td>Value</td>
<td>43</td>
<td>Byte</td>
<td>Byte</td>
<td>Byte</td>
<td>Byte</td>
<td>Byte</td>
</tr>
</tbody>
</table>
Example

The following array would be returned for SSG-6000:

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
<th>Byte 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Code</td>
<td>Freq (MSB)</td>
<td>Freq 1</td>
<td>Freq 2</td>
<td>Freq 3</td>
<td>Freq (LSB)</td>
</tr>
<tr>
<td>43</td>
<td>1</td>
<td>101</td>
<td>160</td>
<td>188</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

The maximum frequency spec can be calculated as follows:

Max Frequency = (256 ^ 4) * BYTE1 + (256 ^ 3) * BYTE2 + (256 ^ 2) * BYTE3 + 256 * BYTE4 + BYTE5
= (256 ^ 4) * 1 + (256 ^ 3) * 101 + (256 ^ 2) * 160 + 256 * 188 + 0
= 6,000,000,000 Hz
= 6,000 MHz

See Also

Get Generator Maximum Frequency (SSG-4000 Series Only)
Get Generator Minimum Frequency
Get Generator Step Size
Get Generator Minimum Power
Get Generator Maximum Power
3.2.3 (12) - Get Generator Step Size

Description

This function reports the signal generator’s step size in Hz.

Send code 44 in BYTE0 of the transmit array. BYTE2 to BYTE63 are “don’t care” bytes and can be any value.

The returned array is made up of the following bytes:

- BYTE0
  - 44 (code for Get Generator Step Size)
- BYTE1 to BYTE4
  - Frequency in Hz, broken up into 4 bytes with MSB in BYTE3 and LSB in BYTE6
  - FREQUENCY = (256 ^ 3) * BYTE1 + (256 ^ 2) * BYTE2 + 256 * BYTE3 + BYTE4
- BYTE5 to BYTE63
  - Could be any value (“don’t care” bytes)

Transmit Array

<table>
<thead>
<tr>
<th>Byte</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Code</td>
<td>44</td>
</tr>
</tbody>
</table>

Returned Array

<table>
<thead>
<tr>
<th>Byte</th>
<th>Description</th>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Code</td>
<td>Freq (MSB)</td>
<td>Freq 1</td>
<td>Freq 2</td>
<td>Freq (LSB)</td>
<td></td>
</tr>
</tbody>
</table>
Example

The following array would be returned for SSG-4000HP:

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Code</td>
<td>Freq (MSB)</td>
<td>Freq 1</td>
<td>Freq 2</td>
<td>Freq (LSB)</td>
</tr>
<tr>
<td>Value</td>
<td>43</td>
<td>0</td>
<td>0</td>
<td>19</td>
<td>136</td>
</tr>
</tbody>
</table>

The step size frequency spec can be calculated as follows:

\[
\text{Step Size} = (256^3) \times \text{BYTE1} + (256^2) \times \text{BYTE2} + 256 \times \text{BYTE3} + \text{BYTE4}
\]
\[
= (256^3) \times 0 + (256^2) \times 0 + 256 \times 19 + 136
\]
\[
= 5,000 \text{ Hz}
\]
\[
= 5 \text{ KHz}
\]

See Also

- Get Generator Minimum Frequency
- Get Generator Maximum Frequency (SSG-4000 Series Only)
- Get Generator Maximum Frequency (SSG-6000 Only)
- Get Generator Minimum Power
- Get Generator Maximum Power
3.2.3 (13) - Get Generator Minimum Power

Description

This function reports the minimum output power in dBm that the generator is capable of providing.

Send code 45 in BYTE0 of the transmit array. BYTE2 to BYTE63 are “don’t care” bytes and can be any value.

The returned array is made up of the following bytes:
- BYTE0
  - 45 (code for Get Generator Minimum Power)
- BYTE1
  - Output power sign (1 if power value is negative or 0 if positive)
- BYTE2 to BYTE3
  - Absolute power in dBm, split into MSB (BYTE2) and LSB (BYTE3)
  - Absolute power is calculated as: \( P = \frac{256 \times \text{BYTE2} + \text{BYTE3}}{100} \)
- BYTE4 to BYTE63
  - Could be any value (“don’t care” bytes)

Transmit Array

<table>
<thead>
<tr>
<th>Byte</th>
<th>Description</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td></td>
<td>45</td>
</tr>
</tbody>
</table>

Returned Array

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Code</td>
<td>Power + or -</td>
<td>Power (MSB)</td>
<td>Power (LSB)</td>
</tr>
<tr>
<td>Value</td>
<td>45</td>
<td>0 or 1</td>
<td>Byte</td>
<td>Byte</td>
</tr>
</tbody>
</table>
Example

The following array would be returned for SSG-4000HP:

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td>Power</td>
<td>Power</td>
<td>Power</td>
<td></td>
</tr>
<tr>
<td>+ or -</td>
<td>(MSB)</td>
<td>(LSB)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>45</td>
<td>1</td>
<td>19</td>
<td>136</td>
</tr>
</tbody>
</table>

The minimum power spec can be calculated as follows:

Absolute Power = (256 * BYTE2 + BYTE3) / 100
                = (256 * 19 + 136) / 100
                = 50 dBm

Real power    = -50 dBm (since BYTE1 = 1)

See Also

Get Generator Minimum Frequency
Get Generator Maximum Frequency (SSG-4000 Series Only)
Get Generator Maximum Frequency (SSG-6000 Only)
Get Generator Step Size
Get Generator Maximum Power
3.2.3 (14) - Get Generator Maximum Power

Description

This function reports the maximum output power in dBm that the generator is capable of providing.

Send code 46 in BYTE0 of the transmit array. BYTE2 to BYTE63 are “don’t care” bytes and can be any value.

The returned array is made up of the following bytes:

- BYTE0
  - 46 (code for Get Generator Maximum Power)
- BYTE1
  - Output power sign (1 if power value is negative or 0 if positive)
- BYTE2 to BYTE3
  - Absolute power in dBm, split into MSB (BYTE2) and LSB (BYTE3)
  - Absolute power is calculated as:
    \[ P = \frac{256 \times \text{BYTE2} + \text{BYTE3}}{100} \]
- BYTE4 to BYTE63
  - Could be any value (“don’t care” bytes)

Transmit Array

<table>
<thead>
<tr>
<th>Byte</th>
<th>Code</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Code</td>
<td>46</td>
</tr>
</tbody>
</table>

Returned Array

<table>
<thead>
<tr>
<th>Byte</th>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Code</td>
<td>Power + or -</td>
<td>Power (MSB)</td>
<td>Power (LSB)</td>
</tr>
<tr>
<td>Value</td>
<td>46</td>
<td>0 or 1</td>
<td>Byte</td>
<td>Byte</td>
</tr>
</tbody>
</table>
Example

The following array would be returned for SSG-4000HP:

<table>
<thead>
<tr>
<th>Byte Description</th>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Code</td>
<td>Power + or -</td>
<td>Power (MSB)</td>
<td>Power (LSB)</td>
</tr>
<tr>
<td>Value</td>
<td>46</td>
<td>0</td>
<td>7</td>
<td>208</td>
</tr>
</tbody>
</table>

The maximum power spec can be calculated as follows:

Absolute Power = \((256 \times \text{BYTE2} + \text{BYTE3}) / 100\)

= \((256 \times 7 + 208) / 100\)

= 20 dBm

Max Power = +20 dBm (since \(\text{BYTE1} = 1\))

See Also

- Get Generator Minimum Frequency
- Get Generator Maximum Frequency (SSG-4000 Series Only)
- Get Generator Maximum Frequency (SSG-6000 Only)
- Get Generator Step Size
- Get Generator Minimum Power