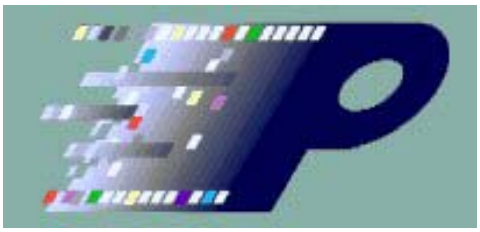

TMPC Pattern Generator Programmatic Interface (PPI) User's Manual

Table of Contents

1 Contacting The Moving Pixel Company

For any questions or comment about PPI or PGApp, you can contact us using one of the methods below:

Phone	+1.503.626.9663	US Pacific Time Zone
Fax	+1.503.626.9653	US Pacific Time Zone
Address	**The *Moving Pixel* Company** 4905 SW Griffith Drive, Suite 106 Beaverton, Oregon 97005 USA	
Email	information@movingpixel.com	
Web site	http://www.movingpixel.com	

2 Introduction

The Moving Pixel Company's Pattern Generator Programmatic Interface (PPI) is based on Microsoft's Component Object Model (COM/DCOM). It gives the TMPC Pattern Generator the ability to be controlled from a separate user program running on any host accessible via a Microsoft network or network running a third party DCOM application.

In PPI, an instance of PGApp (the controlling application for the PG) is called the **server** and the user program is called the **client**. If controlling real hardware, the server always runs and resides on a machine/TLA connected via USB to one or more PG modules. This machine may be the same or different from the host for the client application.

There is a one-to-one relationship between server and client, and each client launches its own instance of PGApp when creating a new COM Application object (defined by the PPI interface). Launched in this way, PGApp always comes up connected to a single offline module (i.e. not connected to a real PG module).

Once created, the client can obtain a COM System object, allowing it to query for PG modules available on the server machine and connect to one in particular. After connection, numerous methods are available to configure and operate the PG. When the client is finished with the PG, it releases its COM objects. This automatically releases the lock on the physical PG module (so others can connect to it) and exits the corresponding instance of PGApp.

This client/server model merely extends the current paradigm that allows for a single local user to run multiple PGApp instances, connecting to and independently controlling multiple PG modules. However, in this case, clients are programmatic users and may possibly be running on a remote computer.

Some general characteristics of the programmatic interface are as follows:

- Client programs may be written in any language or programming environment that supports COM. Common examples are Visual C++ and Visual Basic. PPI has been tested under Windows XP, 2K, and Vista.[1]

- All of the exported server interfaces are *dual* interfaces (they support static and dynamic binding). Depending on the client programming language, it may be only possible (or just easier) to use one interface over the other.

- When a client launches a PGApp server, the main window of the server application will be visible and "Client Connected" will be displayed in the main

[1] Note that at the time of this writing Vista client applications controlling either a local PGApp server or a remote PGApp server running under an OS other than Vista have been successfully tested. Remote clients controlling a PGApp server running under Vista has not yet been made to work, presumably due to one or more of the new security features introduced in Vista. Please contact The Moving Pixel Company for up-to-date information about this issue.

window status bar. Clients can hide the server's main window via PPI. If the window is visible, users can directly interact with the Tektronix Pattern Generator server application.

- PGApp has many instances where the user is notified of something or asked to confirm a particular operation. For example, before loading a system, the user is asked whether the current system should be saved before the load operation. Since it is not possible to ask questions through the programmatic interface, PGApp will always proceed with the original operation as though the question were never asked. In the previous example, the load operation would proceed without saving the current system.

- PPI operates within the main thread of the application.

2.1 Differences between TMPC PPI and legacy Tektronix PPI

While The Moving Pixel Company PPI implementation was derived from the legacy Tektronix PPI implementation, they differ from each other in several ways. This section summarizes the differences between the two.

1. As described already, the TMPC client/server relationship is one-to-one. Each client launches its own instance of PGApp. Tek PPI had a many-to-one client/server model, where only one instance of the PG application ran on the host machine and all clients connected to it.

2. TMPC PPI no longer defines a Module object. Because of the tighter coupling between PGApp and the PG module, the Module object interface has been subsumed into the System object interface.

3. TMPC PG modules are uniquely referenced via serial number whereas Tek PPI generally referred to specific PG modules by TLA slot number.

4. Support for module configuration has been greatly expanded in the TMPC PPI interface. While Tek PPI provided a few core module configuration routines, TMPC PPI provides routines to set up all PGApp module, probe, and signal setup parameters. In addition, the GetRunStatus routine discloses when the PG is waiting on an event and allows the client to advance the PG from the wait state. Similarly, step mode is supported for single-stepping test vectors from a client application.

3 Setting up PPI

3.1 Registering COM interfaces

The first step to getting PPI working is to register the COM interfaces and classes provided by PGApp. Registration needs to be performed on both the client and server machines (if they are different) and is achieved simply by installing and running PGApp on both machines and selecting the System menu option "Register As COM Server".

This menu option enters information about its COM interfaces in the local registry. Even if you do not intend to use PGApp on the client machine you still must install and run it to register its COM interfaces. This information is used by your client application to navigate through COM/DCOM system calls to reach and communicate with a PGApp server.

Note that under Windows Vista, you must run PGApp in administrator mode to successfully register PGApp as a COM server. To do this, right-click the PGApp executable and select "Run As Administrator".

While PGApp can be launched normally for the purposes of COM registration, it can also be launched through the command line. Open a DOS window, change to the application directory (by default c:/program files/TMPC/PGApp) and type the command: **PGApp /RegServer**. Similarly, you may also type **PGApp /UnregServer** to unregister the COM interfaces. (This method will not work under Vista, however, unless you have set the privilege level of the PGApp executable to run the program as an administrator. You can do this by right-clicking on PGApp.exe, select properties, compatibility tab, and check the box "Run this program as an administrator").

3.2 Configuring DCOM

Now that the default COM interfaces have been defined, you can use the Windows utility dcomcnfg.exe to modify the registration and set up security parameters depending on how you plan to use PPI. This needs to be done on both the client and server machines. Note that you will need Administrator privileges on both machines to perform this task.

3.2.1 Launching dcomcnfg

Each Windows operating system has a slightly different method for accessing the COM property page of an application. To bring up the properties for the PGApp, follow one of the procedures outlined below depending on the operating system:

<u>Under Windows Vista/XP:</u>

1. Bring up the Start Menu, select Run, and type dcomcnfg (or bring up DOS window and type dcomcnfg)
2. Double-click on Component Services in right pane
3. Double-click on Computers in right pane
4. Double-click on My Computer in right pane
5. Double-click on DCOM Config folder in right pane

6. Right-click on "TMPC Pattern Generator" icon and select Properties

Under Windows 2K:

1. Bring up the Start Menu, select Run, and type dcomcnfg (or bring up DOS window and type dcomcnfg)
2. Scroll through list, highlight "TMPC Pattern Generator", and click Properties button.

3.2.2 Server Machine Configuration

On the server machine, bring up the property page for the TMPC Pattern Generator (PGApp) and perform the following steps:

1. On the general tab, set the authentication level to None.
2. On the location tab, check "Run application on this computer"
3. On the security tab, under Launch and Activation Permissions click the Customize button and then the Edit… button.
4. Click Add…, type "Everyone", and click OK.
5. Check to allow all four permissions -- Local/Remote Launch and Local/Remote Activation -- and click OK.
6. On the security tab, under Access Permissions click the Customize button and then the Edit… button.
7. Click Add…, type "Everyone", and click OK.
8. Check to allow both permissions -- Local/Remote Access -- and click OK.
9. On the identity tab, click "The interactive user". This means all remote clients connecting to the server will run under the user currently logged in.
10. Click OK to close the properties window, then exit dcomcnfg.
11. Reboot machine.

Note: this configuration turns off authentication and allows any COM client to lauch PGApp on the server under the current user's account. A more restrictive configuration is to set up a user account and password just for running a PGApp server. In this case, you would enter this account information on the identity page instead of using the interactive user. Similarly, this information would need to be entered on the client machine as well instead of the launching user.

3.2.3 Client Machine Configuration

If the client machine is the same as the server machine, no further DCOM configuration is required. If the client machine is different than the server machine, you must perform a similar procedure on it as well. On the client machine, bring up the property page for the TMPC Pattern Generator and perform the following steps (note differences from the server procedure in bold):

1. On the general tab, set the authentication level to None.
2. On the location tab, **check "Run application on the following computer". Type in or browse for the name of the server computer in the space provided**.

3. On the security tab, under Launch and Activation Permissions click the Customize button and then the Edit… button.
4. Click Add…, type "Everyone", and click OK.
5. Check to allow all four permissions -- Local/Remote Launch and Local/Remote Activation -- and click OK.
6. On the security tab, under Access Permissions click the Customize button and then the Edit… button.
7. Click Add…, type "Everyone", and click OK.
8. Check to allow both permissions -- Local/Remote Access -- and click OK.
9. On the identity tab, click **"The launching user".**
10. Click OK to close the properties window, then exit dcomcnfg.
11. Reboot machine.

4 PPI Client Programming

4.1 PPI Interfaces

The programmatic interface for PGApp consists of two kinds of objects: Application and System.

Application: The Application object is created by the client to initially launch a new instance of a PGApp server and to subsequently obtain a reference to a System object. The Application object exports a single interface called IPGApplication.

System: The System object provides methods for controlling a PG module, including module, probe, and system setup, run configuration and control, and save/load operations. The System object exports a single interface called IPGSystem.

See Chapter 5 for reference information detailing the routines supported by the IPGApplication and IPGSystem interfaces.

Generally, methods are synchronous and wait for the completion of the operation before returning. However, some routines, such as those that deal with module acquisition and module release, return before fully completing. Subsequent PPI calls will return the PGAPP_E_BUSY error if PGApp is not yet ready to accept a new command. The client can use the IsPGBusy call to query if PGApp has completed the previous command before submitting any new PPI requests.

All methods in both PPI interfaces return an HRESULT (or SCODE). Refer to PPIErrors.h for possible error codes. Note that when a method returns an error, output arguments are undefined and should not be used.

4.2 PPI Support Files

Once PGApp has been installed, all PPI examples and support files can be found in the default directory c:/Program Files/TMPC/PPI. These files include:

- PGApp_i.c – defines the PPI COM interface and class IDs
- PGApp_h.h – defines the PPI COM interface routines and type information
- PGApp.tlb – type library for PPI COM interface
- PPIErrors.h – error code defintions
- PPIUsersManualX_XX.doc – this manual
- TestPPI – Visual C++ console client example

Previously, Tektronix provided sample client applications using other languages such as Labview, Matlab, and Visual Basic. While the TMPC version of PPI has not been tested under these languages, it is expected they should work no differently than before. As a convenient reference for those using these languages, these Tektronix client examples are included in the PPI directory under the "LegacyTek" folder. While these examples will

not work "as is" because of extensive interface method changes, they should provide a useful overall framework for writing PPI clients in these languages.

5 PPI Reference

This section is a reference for all the objects and interfaces supported by the TMPC PG Programmatic Interface (PPI). Code examples assume existing pointers to Application and System objects – pApp and pSystem. These pointers would be initialized in VC++ using code as follows:

```
#include <comdef.h>
#include "pgapp_h.h"

HRESULT hr;
IDispatch* pTmp;
IPGApplication* pApp;
IPGSystem* pSystem;

// initialize COM Library
CoInitialize (NULL);

// create Application object
// note: on successful creation, an instance of PGApp will be launched
// on the server machine
hr = CoCreateInstance (CLSID_PGApplication, NULL,
        CLSCTX_LOCAL_SERVER | CLSCTX_REMOTE_SERVER,
        IID_IPGApplication, (void**)&pApp);
assert(SUCCEEDED(hr));

// obtain System object
hr = pApp->GetSystem(&pTmp);
assert(SUCCEEDED(hr));
hr = pTmp->QueryInterface(IID_IPGSystem, (void **)&pSystem);
assert(SUCCEEDED(hr));
pTmp->Release();

// … use objects here

// release objects
// note: after the Application object is released, the instance of PGApp that
// was launched at creation will terminate
pSystem->Release();
pApp->Release();
```

Also, some of the examples below use a routine called WaitForPG that spins waiting for the IsPGBusy routine to return false. The code for it is as follows:

```
HRESULT WaitForPG(IPGSystem* pSystem)
{
        HRESULT hr;
        long Busy;

        for (;;) {
                hr = pSystem->IsPGBusy(&Busy);
                if (!SUCCEEDED(hr) || !Busy) break;
        }

        return(hr);
}
```

5.1 Quick Reference

This section summarizes the objects and methods of PPI. These methods are described in more detail in the Reference section.

Application Object (IPGApplication)

HRESULT GetSystem(ppDispatch)
HRESULT ShowWindow(Show)

System Object (IPGSystem)

Module Discovery and Acquisition

HRESULT GetModuleCount(pModuleCount)
HRESULT GetModuleSerialNumber(ModuleIndex, pSerialNum)
HRESULT AcquireModules(SerialNum0, SerialNum1, SerialNum2, SerialNum3)
HRESULT ReleaseAllModules()
HRESULT GetAcquiredModuleCount(pModuleCount)
HRESULT GetAcquiredModuleProperties(ModulePos, pModuleProperties)
HRESULT GetSWVersion(pVersion)
HRESULT IsPGBusy(pBusy)

Load and Save

HRESULT LoadSystem(SystemPath)
HRESULT SaveSystem(SystemPath)
HRESULT Import(ImportFilePath, BlockNo)
HRESULT Export(ExportFilePath, BlockNo, ExportType)

Operation

HRESULT Run()
HRESULT Stop()

HRESULT Advance()
HRESULT Step()
HRESULT Jump()
HRESULT GetRunStatus(pRunStatus)
HRESULT SetHostRunTrigger()

Module Configuration
HRESULT SetClockMode(ClockMode)
HRESULT SetClockPeriod(Period)
HRESULT SetClockFrequency(Frequency)
HRESULT SetReferenceClockSource(Source)
HRESULT SetInputClockThreshold(Threshold)
HRESULT SetInputClockFilterPeriod(FilterPeriod)
HRESULT SetInputClockInvert(Invert)
HRESULT SetInputClockDelay(Delay)
HRESULT SetDiscontinuousClock(DiscontinuousClock)
HRESULT SetEventThreshold(Threshold)
HRESULT SetEventFilterPeriod(FilterPeriod)
HRESULT SetEventModeForAdvance(EventMode)
HRESULT SetEventModeForJump(EventMode)
HRESULT SetRunMode(RunMode)
HRESULT SetHiZOnStop(HiZOnStop)
HRESULT SetRunTriggerSource(RunTriggerSource)

Probe Configuration
HRESULT GetProbeType(ModulePos, ProbeIndex, ProbeType)
HRESULT SetOutputLevel(ModulePos, ProbeIndex, Level)
HRESULT SetDataDelay(ModulePos, ProbeIndex, ByteIndex, Delay)
HRESULT SetDataDelay180(ModulePos, ProbeIndex, ByteIndex, Delay180)
HRESULT SetStrobeShape(ModulePos, ProbeIndex, StrobeShape)
HRESULT SetVarGroup(ModulePos, ProbeIndex, ChannelIndex, Group)
HRESULT SetVarHighLevel(ModulePos, ProbeIndex, ChannelIndex, Level)
HRESULT SetVarLowLevel(ModulePos, ProbeIndex, ChannelIndex, Level)
HRESULT SetVarSlewRate(ModulePos, ProbeIndex, ChannelIndex, SlewRate)
HRESULT SetVarDifferential(ModulePos, ProbeIndex, ChannelIndex, Differential)
HRESULT SetVarDelay(ModulePos, ProbeIndex, ChannelIndex, Delay)
HRESULT SetVarInhibitThreshold(ModulePos, ProbeIndex, Threshold)
HRESULT SetVarInhibitEnable(ModulePos, ProbeIndex, Enable)

Signals Configuration
HRESULT SetSignalInput(SignalInput)
HRESULT SetSignalOutput(SignalOutput)
HRESULT InvertSignalOutput(Invert)

Group/Channel Configuration
HRESULT GetGroupNames(pGroupNames)

HRESULT GetGroupSize(GroupName, pGroupSize)
HRESULT GetProbeChannelNames(GroupName, pProbeChlNames)

5.2 IPGApplication Methods

5.2.1 IPGApplication::GetSystem

Description:
This method returns the interface pointer for the System object.

IDL Syntax:
HRESULT GetSystem([out,retval] IDispatch** ppDispatch)

Arguments:
ppDispatch – the interface pointer for the System object.

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_FAILED	The operation was unsuccessful.

Example:

```
HRESULT hr;
IDispatch* pTmp;
IPGSystem* pSystem;

// obtain System object
hr = pApp->GetSystem(&pTmp);
assert(SUCCEEDED(hr));
hr = pTmp->QueryInterface(IID_IPGSystem, (void **)&pSystem);
assert(SUCCEEDED(hr));
pTmp->Release();
```

Remarks:
Gets the existing system object if there is one.

5.2.2 IPGApplication::ShowWindow

Description:
This method shows/hides the PGApp server's main window.

IDL Syntax:
HRESULT ShowWindow([in] long Show)

Arguments:
Show – set this flag to (0) to hide the server window and (1) to show it.

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_FAILED	The operation was unsuccessful.

Example:

```
// Hide the window
pApp->ShowWindow(0);

// Show the window
pApp->ShowWindow(1);
```

Remarks:
The PGApp server window is shown by default when the Application object is created. If left visible, users may interact with the PGApp server as well as the client.

Note that when the application window is visible, PPI calls are treated as user actions, manipulating windows and updating fields as a user would. Thus, when the application window is visible, PPI calls are significantly slower as window updating occurs. If a lot of configuration via PPI is necessary, the programmer might consider hiding the PGApp server window before configuration and then showing it again when configuration is complete.

5.3 IPGSystem Methods

5.3.1 IPGSystem::AcquireModules

Description:

Given up to 4 serial numbers, this method acquires and merges the PG modules as a single instrument. Once acquired, none of the real modules will be available to other applications until released. The order determines each module's position in the instrument, with the module specified by SerialNum0 designated as the master module.

Using a serial number of 0 requests an offline module to occupy the position. More than one offline module may be specified in the instrument. While offline modules do not output data when the instrument is run, they do have program data associated with them, and allow saving/loading of system files with multiple modules.

If the master module is an offline module, the instrument is considered offline (even if one or more slave modules are real). If the master module is a real module, the instrument is considered real and can be run (even if one or more slave modules are offline).

IDL Syntax:

```
HRESULT AcquireModules([in] long SerialNum0, [in] long SerialNum1,
                       [in] long SerialNum2, [in] long SerialNum3)
```

Arguments:

SerialNum0 – the serial number of the first (master) PG module
SerialNum1 – the serial number of the second (slave) PG module
SerialNum2 – the serial number of the third (slave) PG module
SerialNum3 – the serial number of the fourth (slave) PG module

Use serial number of 0 to designate an offline module.
Use serial number of -1 to designate no module.
Once -1 is used for a module the remaining modules must also be -1

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_E_INVALID_ARG	Duplicate serial numbers > 0
PGAPP_E_FAILED	The operation was unsuccessful. The module may be in use by another user or application.

Example:

```
// build 3-module instrument with PG module with serial number 1005 as master and
two offline slave modules
pSystem->AcquireModuleBySerialNumber(SerialNumber, 0, 0, -1);

// wait for PGApp to complete initialization
WaitForPG(pSystem);
```

Remarks:

When the PGSystem object is created, it is automatically started initially connected to a single offline module. To communicate with and control real hardware, a module or set of modules must first be acquired using this routine. Once this call returns successfully, PGApp may still be busy initializing. To avoid getting the PGAPP_E_BUSY error on the subsequent call, you can use WaitForPG() to ensure PGApp has completed first.

5.3.2 IPGSystem::Advance

Description:
If the PG is waiting on an event, this call triggers the waiting event so the sequence can proceed.

IDL Syntax:
HRESULT Advance()

Arguments:
<none>

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_E_SYSTEM_NOT_RUNNING	The PG is not running

Example:

```
// advance PG module from wait state
pSystem->Advance();
```

Remarks:
This call has no effect if the PG is not waiting on an event.

5.3.3 IPGSystem::Export

Description:
This method exports pattern data for a particular block to an ASCII text file.

IDL Syntax:
HRESULT Export([in] BSTR ExportFilePath,[in] long BlockNo, [in] long ExportType)

Arguments:
ExportFilePath – the file to export data to.
BlockNo – block number of the data to export
ExportType – export file type format.
Options:
PGAPP_EXP_TLA_DATA_EXCHANGE

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_E_SYSTEM_RUNNING	PG is running.
PGAPP_E_UNKNOWN_EXPORT_TYPE	Invalid file type format.
PGAPP_E_INVALID_BLOCK_NUMBER	Invalid block number.
PGAPP_E_INVALID_EXPORT_FILE	Error creating/opening file.
PGAPP_E_FAILED	The operation was unsuccessful

Example:

```
// create file name string
BSTR exportFileName = SysAllocString(L"c:/my documents/BlockSave.tpg");

// export block 1 to file in TLA Data Exchange Format
pSystem->Export(exportFileName, 1, PGAPP_EXP_TLA_DATA_EXCHANGE);

// free file name string
SysFreeString(exportFileName);
```

Remarks:
None

5.3.4 IPGSystem::GetAcquiredModuleCount

Description:
Returns the number of acquired modules.

IDL Syntax:
HRESULT GetAcquiredModuleCount([out, retval] long* pAcquiredModuleCount)

Arguments:
pAcquiredModuleCount – returned module count

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.

Example:

```
long ModuleCount;

// obtain acquired module count
pSystem->GetModuleCount(&ModuleCount);
```

Remarks:
This routine is intended for future use. Currently, always returns (1).

5.3.5 IPGSystem::GetAcquiredModuleProperties

Description:
Given an acquired module index, this method acquires a PG module's properties.

IDL Syntax:
HRESULT GetAcquiredModuleProperties([in] long ModulePos, [out, retval] BSTR* pModuleProperties)

Arguments:
ModulePos – the acquired module position, i.e. an ordinal in the range of 0 to the value returned by GetAcquiredModuleCount - 1. Currently, this must be (0) since only one module may be acquired at a time.
pModuleProperties – returned string describing acquired module properties.

Formatting of the module properties string is as follows:
"<manufacturer>,<model>,<firmware version>,<serial number>,
<max frequency>,<memory depth>". i.e.
"The Moving Pixel Company,TMPCPG3A,1.00,1005,300 MHz,33554432"

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_E_INVALID_ARG	Invalid module index
PGAPP_E_FAILED	The operation was unsuccessful.

Example:

```
HRESULT hr;
char msg[255];
wchar_t* szOut;

// obtain acquired module properties
hr = pSystem->GetAcquiredModuleProperties(0, &szOut);
if (SUCCEEDED(hr)) {
        wcstombs(msg, szOut, 255);
        printf("GetAcquiredModuleProperties returned %s\n", msg);
        ::SysFreeString(szOut);
}
```

Remarks:
Client is responsible for deallocating the returned properties string when finished.

5.3.6 IPGSystem::GetGroupNames

Description:
Retrieves a list of group names from the PG.

IDL Syntax:
HRESULT GetGroupNames([out, retval] VARIANT* pGroupNames)

Arguments:
pGroupNames – returned list of PG group names.

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_E_FAILED	The operation was unsuccessful. The module may be in use by another user or application.

Example:

```
HRESULT hr;
SAFEARRAY FAR* pNameArray = NULL;
SAFEARRAYBOUND bound;
BSTR HUGEP *pbstr = NULL;
VARIANT names;

VariantInit( &names );
hr = pSystem->GetGroupNames( &names);
if (SUCCEEDED(hr)) {
        pNameArray = V_ARRAY( &names );
        SafeArrayAccessData( pNameArray, (void HUGEP* FAR*)&pbstr );

        bound = pNameArray->rgsabound[0];
        for ( i = 0; i < bound.cElements; i++ ) {
                _bstr_t groupName( pbstr[ bound.lLbound + i ], TRUE );
                Printf("Group name %d is %s\n", I, groupName);
        }

        SafeArrayUnaccessData( pNameArray );
}
```

Remarks:
None

5.3.7 IPGSystem::GetGroupSize

Description:
Gets the size in bits of a given group.

IDL Syntax:
HRESULT GetGroupSize([in] BSTR GroupName, [out, retval] long* pGroupSize)

Arguments:
GroupName – name of group to get size of
pGroupSize – returned group size in bits

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_E_FAILED	The operation was unsuccessful. The module may be in use by another user or application.

Example:

```
long GroupSize;

// obtain the group size for the group "UserGrp1"
BSTR GroupName = SysAllocString(L"UserGrp1");
pSystem->GetGroupSize(GroupName, &GroupSize);
SysFreeString(GroupName);
```

Remarks:
None

5.3.8 IPGSystem::GetModuleCount

Description:
Gets the number of modules present on the server machine.

IDL Syntax:
HRESULT GetModuleCount([out, retval] long* pModuleCount)

Arguments:
pModuleCount – returned number of modules present

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.

Example:

```
long ModuleCount;

// get the number of modules present
pSystem->GetModuleCount(&ModuleCount);
```

Remarks:
None.

5.3.9 IPGSystem::GetModuleSerialNumber

Description:
Gets the serial number of a module given its module index. i.e. an ordinal in the range 0 to GetModuleCount().

IDL Syntax:
HRESULT GetModuleSerialNumber([in] long ModuleIndex, [out, retval] long* pSerialNumber)

Arguments:
ModuleIndex – the module index, i.e. an ordinal in the range of 0 to the value returned by GetModuleCount - 1.
pSerialNumber – returned serial number

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_INVALID_ARG	Module index is out of range

Example:

```
long SerialNumber;

// get the serial number of the first module present on the server system
pSystem->GetModuleSerialNumber(0, &SerialNumber);
```

Remarks:
Note: the serial number returned for offline modules is 0.

5.3.10 IPGSystem::GetProbeChannelNames

Description:
Gets the size in bits of a given group.

IDL Syntax:
HRESULT GetProbeChannelNames([in] BSTR GroupName, [out, retval] BSTR* pProbeChlNames)

Arguments:
GroupName – name of group to get probe channels from
pProbeChlNames – returned string representing probe channels

Channel name string format is identical to the entry in the channel setup page of the Setup Window in PGApp. For example, the string returned from a default system in the example below would be: "A1(7-0),A0(7-0)". Please see the PGApp User Manual for more description.

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
char msg[255];
wchar_t* szOut = NULL ;
HRESULT hr;
BSTR GroupName = SysAllocString(L"UserGrp1");

hr = pSystem->GetProbeChannelNames( GroupName, &szOut );
if (SUCCEEDED(hr)) {
        wcstombs(msg, szOut, 255);
        printf("UserGrp1 channel names: %s\n", msg);
        SysFreeString( szOut );
}

SysFreeString(GroupName);
```

Remarks:
None

5.3.11 IPGSystem::GetProbeType

Description:
Gets the probe type for the given acquired module position and probe index.

IDL Syntax:
HRESULT GetProbeType([in] long ModulePos, [in] long ProbeIndex, [out, retval] long* pProbeType)

Arguments:
ModulePos – acquired module position, i.e. an ordinal in the range of 0 to the value returned by GetAcquiredModuleCount - 1. Currently, this must be (0) since only one module may be acquired at a time.
ProbeIndex – probe index, use 0=A, 1=B, 2=C, 3=D
pProbeType – returned probe type, one of the PGAPPProbeType enumeration:

PGAPP_NONE = 0,
PGAPP_P370LV = 5,
PGAPP_LVDS = 6,
PGAPP_P375 = 9,
PGAPP_P370 = 11,
PGAPP_P370LV2 = 12

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
long ProbeType;

// get probe type for acquired module 0, probe index 2
pSystem->GetProbeType(0, 2, &ProbeType);
```

Remarks:
None

5.3.12 IPGSystem::GetRunStatus

Description:
Gets the probe type for the given acquired module position and probe index.

IDL Syntax:
HRESULT GetRunStatus([out, retval] long* pRunStatus)

Arguments:
pRunStatus – returned status of the PG, one of the RunStatus enumeration

```
PGAPP_IDLE      = 0,    // PG is not running
PGAPP_WAITING   = 2,    // PG is waiting on an event
PGAPP_RUNNING   = 3,    // PG is running
PGAPP_ARMED     = 4     // PG is waiting for a run trigger
```

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
long RunStatus;

// get current PG run status
pSystem->GetRunStatus(&RunStatus);
```

Remarks:
None

5.3.13 IPGSystem::GetSWVersion

Description:
Gets the PGAPP software version.

IDL Syntax:
HRESULT GetSWVersion([out, retval] BSTR* pVersion)

Arguments:
pVersion – returned string containing the software version of the form "X.X.XXX", for example "2.0.007".

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
char msg;
wchar_t* szOut = NULL;
HRESULT hr;

hr = pSystem->GetSWVersion( &szOut );
if (SUCCEEDED(hr)) {
        wcstombs(msg, szOut, 255);
        printf("GetSWVersion returned the software version: %s\n", msg);
        SysFreeString( szOut );
} else {
        printf("GetSWVersion returned error 0x%x\n", hr);
}
```

Remarks:
User is responsible for deallocating returned string.

5.3.14 IPGSystem::Import

Description:
This method imports pattern data from an ASCII text file into a given block.

IDL Syntax:
HRESULT Import([in] BSTR ImportFilePath, [in] long BlockNo)

Arguments:
ImportFilePath – file to be imported
BlockNo – block number to import into

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_E_SYSTEM_RUNNING	PG is running.
PGAPP_E_UNKNOWN_IMPORT_TYPE	Invalid file type format.
PGAPP_E_INVALID_BLOCK_NUMBER	Invalid block number.
PGAPP_E_INVALID_IMPORT_FILE	Error opening file.
PGAPP_E_FAILED	The operation was unsuccessful

Example:

```
BSTR ImportFileName = SysAllocString(L"c:/my documents/block.tpg");

// import data from block.tpg into block 2
pSystem->Import(ImportFileName, 2);

SysFreeString(ImportFileName);
```

Remarks:
The file type is automatically detected. Import file types supported are:
Tek TLA Data Exchange Format
Synapticad Spreadsheet Format
Agilent HPD Format

5.3.15 IPGSystem::InvertSignalOutput

Description:
Inverts the signal output if SignalOut is currently set to ExtTrigOut.

IDL Syntax:
HRESULT InvertSignalOutput([in] long Invert)

Arguments:
Invert – set to (0) to not invert, (1) to invert.

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
// set SignalOut to ExtTrigOut and invert
pSystem->SetSignalOutput(PGAPP_EXT_TRIG_OUT);
pSystem->InvertSignalOutput(1);
```

Remarks:
Only the ExtTrigOut signal can be inverted. If ExtTrigOut is not the current SignalOut, this call has no effect.

5.3.16 IPGSystem::IsPGBusy

Description:
Indicates whether the PG has finished initialization after acquiring or releasing a module.

IDL Syntax:
HRESULT IsPGBusy([out, retval] long* pBusy)

Arguments:
pBusy – returned flag indicating busy status: (1) for busy, (0) for not busy.

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.

Example:

```
HRESULT hr;
long Busy;

// wait for PG to complete initialization after module acquistion/release
for (;;) {
        hr = pSystem->IsPGBusy(&Busy);
        if (SUCCEEDED(hr) || !Busy) break;
        Sleep(10);
}
```

Remarks:
Use after the following calls:
AcquireModuleBySerialNumber(),
AcquireModuleByIndex(),
ReleaseAllModules()

5.3.17 IPGSystem::Jump

Description:
This call forces the next branch test to succeed and thus take the branch.

IDL Syntax:
HRESULT Jump()

Arguments:
<none>

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_E_SYSTEM_NOT_RUNNING	The PG is not running

Example:

```
//  force next branch test to succeed and the branch to be taken
pSystem->Jump();
```

Remarks:
This call is only really useful when the PG is in an infinite loop waiting on a branch event to break out of the loop.

5.3.18 IPGSystem::LoadSystem

Description:
Loads the specified system file.

IDL Syntax:
HRESULT LoadSystem([in] BSTR SystemPath)

Arguments:
SystemPath – full or relative path of system file to load.

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_E_SYSTEM_RUNNING	PG is running
PGAPP_E_LOAD_INVALID_FILE	Error opening file
PGAPP_E_LOAD_ERROR	Error loading information from file
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
BSTR loadSystemName = SysAllocString(L"c:/my documents/EventTest.tpg");

// load system file into PG
hr = pSystem->LoadSystem(loadSystemName);

SysFreeString(loadSystemName);
```

Remarks:
All file paths without machine qualifiers refer to drives mapped on the server computer.

All current PG data and settings will be lost.

Probe information will be automatically updated if probe configuration in system file does not match hardware. This has the side effect of resetting the ouput levels of probes that do not match to their minimum levels.

5.3.19 IPGSystem::ReleaseAllModules

Description:
Releases any acquired modules.

IDL Syntax:
HRESULT ReleaseAllModules()

Arguments:
None

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
// release all acquired modules
pSystem->ReleaseAllModules( );
```

Remarks:
After this call, all acquired modules will be available for acquisition and the PGApp server will be reset to the default instrument with a single offline module.

5.3.20 IPGSystem::Run

Description:
Runs/Arms the PG depending on the run trigger source.

IDL Syntax:
HRESULT Run( )

Arguments:
None

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_E_SYSTEM_RUNNING	The PG is already running.
PGAPP_E_PROBES_CANT_SUPPORT FREQ	The current clock frequency is greater than the maximum supported by the current probes.
PGAPP_E_FAILED	The operation was unsuccessful.

Example:

```
// run PG
pSystem->Run();
```

Remarks:
If the run trigger source is set to None (see SetRunTriggerSource), this command runs the PG. If the run trigger source is set to a signal other than None, the PG enters Arm mode, where the PG is fully prepared to run with outputs enabled and the first sequence vector driving at the outputs. Only the output clock is not enabled. The output clock is enabled when the trigger source or HostRunTrigger asserts.

A PG in Offline mode compiles when receiving the Run command, but does not actually run.

5.3.21 IPGSystem::SaveSystem

Description:
Saves the PG system to a file.

IDL Syntax:
HRESULT SaveSystem([in] BSTR SystemPath, [in] BSTR UserComment, [in] long SaveData)

Arguments:
SystemPath – full or relative filename path to save to
UserComment – arbitrary user comment to save in file
SaveData – flag to save program data (1) or not save program data (0) in file

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_E_SYSTEM_RUNNING	The PG is running.
PGAPP_E_SAVE_ERROR	An error occurred during the save operation.
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
BSTR saveSystemName = SysAllocString(L"c:/my documents/EventTestSave.tpg");
BSTR comment = SysAllocString(L"Sample Comment");

// save PG configuration and data to system file
pSystem->SaveSystem(saveSystemName, comment, 1);

// free strings
SysFreeString(saveSystemName);
SysFreeString(comment);
```

Remarks:
All file paths without machine qualifiers refer to drives mapped on the server computer.

5.3.22 IPGSystem::SetClockFrequency

Description:
Sets the clock frequency.

IDL Syntax:
HRESULT SetClockFrequency([in] double Frequency)

Arguments:
Frequency – clock frequency setting in Hz. The valid frequency range for the PG3A is 100 to 300E+6.

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_E_SYSTEM_RUNNING	The PG is running
PGAPP_INVALID_ARG	Invalid frequency.
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
// set clock frequency to 10 MHz
pSystem->SetClockFrequency(10E+6);
```

Remarks:
Note that the clock and all data delays will be clipped to a new maximum of 17.25 ns if the clock frequency is changed from less than 29 MHz to greater than 29 MHz.

This setting is only allowed to be changed when the PG is idle.

5.3.23 IPGSystem::SetClockMode

Description:
Sets the clocking mode.

IDL Syntax:
HRESULT SetClockMode([in] long ClockMode)

Arguments:
ClockMode – clocking mode setting, one of the following:
PGAPP_INTERNAL_CLOCKING = 0
PGAPP_EXTERNAL_CLOCKING = 1

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_E_SYSTEM_RUNNING	The PG is running
PGAPP_INVALID_ARG	Invalid clock mode.
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
// set PG in internal clocking mode
pSystem->SetClockMode(PGAPP_INTERNAL_CLOCKING);
```

Remarks:
This setting is only allowed to be changed when the PG is idle.

5.3.24 IPGSystem::SetClockPeriod

Description:
Sets the clock period.

IDL Syntax:
HRESULT SetClockPeriod([in] double Period)

Arguments:
Period – clock period setting in seconds. The valid period range for the PG3A module is 1E-2 to 3.3333E-9.

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_E_SYSTEM_RUNNING	The PG is running
PGAPP_INVALID_ARG	Invalid period
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
// set clock period to 100 ns
pSystem->SetClockPeriod(100E-9);
```

Remarks:
Note that the clock and all data delays will be clipped to a new maximum of 17.25 ns if the clock period is changed from greater than 34.48 ns (29 MHz) to less than 34.48 ns.

This setting is only allowed to be changed when the PG is idle.

5.3.25 IPGSystem::SetDataDelay

Description:
Sets the delay for the data byte lane specified by the given module, probe, and byte-lane indexes.

IDL Syntax:
HRESULT SetDataDelay([in] long ModulePos, [in] long ProbeIndex, [in] long ByteIndex, [in] double Delay)

Arguments:
ModulePos – acquired module position, i.e. an ordinal in the range of 0 to the value returned by GetAcquiredModuleCount - 1. Currently, this must be (0) since only one module may be acquired at a time.
ProbeIndex – probe index, use 0 = A, 1 = B, 2 = C, 3 = D
ByteIndex – byte-lane within probe, use 0 = LSB, 1 = MSB
Delay – data delay in seconds. The valid range for Delay depends on clock frequency:

Clock Frequency	**Delay Range**
>= 29 MHz	0 to 17.25 ns
< 29 MHz	0 to 500 ns

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_E_INVALID_ARG	An invalid argument was provided
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
// set the data delay for module 0, probe 2, MS byte lane to 5 ns
pSystem->SetDataDelay(0, 2, 1, 5.0E-9);
```

Remarks:
This delay is in addition to the data delay setting from SetDataDelay180.

Note that this setting will be clipped to a maximum of 17.25 ns if the clock frequency is changed from less than 29 MHz to greater than 29 MHz.

This setting is allowed to be changed while the PG is running. However, the user should be aware that adjusting the data delay when the PG is running can have undesirable side

effects. Internally, data delays are implemented by delaying the clock associated with the data. Because changes to the data delay are asynchronous to the associated data clock, it is possible to generate a clock glitch. If a glitch occurs, subsequent vectors output by the PG cannot be guaranteed to be correct (until the sequence is stopped and restarted). Thus, the real-time adjustment of data delay is currently intended to be used for calibration purposes only (e.g. to align edge positions relative to reference signals when using a looping test sequence).

We are working to improve this situation. Please call us to discuss your critical application needs for glitch-less, run-time, data delay adjustment.

5.3.26 IPGSystem::SetDataDelay180

Description:
Sets the data delay 180 parameter which, when set, inverts the clock used for the data byte lane specified by the given module, probe, and byte-lane indexes.

IDL Syntax:
HRESULT SetDataDelay180([in] long ModulePos, [in] long ProbeIndex, [in] long ByteIndex, [in] long Delay180)

Arguments:
ModulePos – acquired module position, i.e. an ordinal in the range of 0 to the value returned by GetAcquiredModuleCount - 1. Currently, this must be (0) since only one module may be acquired at a time.
ProbeIndex – probe index, use 0 = A, 1 = B, 2 = C, 3 = D
ByteIndex – byte-lane within probe, use 0 = LSB, 1 = MSB
Delay180 – flag to clear (0) or set (1) the delay.

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_E_INVALID_ARG	An invalid argument was provided
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
// set the data delay 180 for module 0, probe 2, MS byte lane
// if using a 50% duty cycle 40 MHz clock, this will add a 12.5 ns delay to the data byte
pSystem->SetDataDelay(0, 2, 1, 1);
```

Remarks:
For a continuous, 50% duty-cycle clock, this has the effect of delaying the data byte by half the clock period.

This delay is in addition to the data delay setting from SetDataDelay180.

This setting is allowed to be changed while the PG is running.

5.3.27 IPGSystem::SetDiscontinuousClock

Description:
Sets the discontinuous clock flag.

IDL Syntax:
HRESULT SetDiscontinuousClock([in] long DiscontinuousClock)

Arguments:
DiscontinuousClock – flag to clear (0) or set (1) the discontinuous clock setting.

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_E_INVALID_ARG	The DiscontinuousClock value was invalid
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
// clear the discontinuous clock setting
pSystem->SetDiscontinuousClock(0);
```

Remarks:
Set this flag when in external clocking mode and the input clock is discontinuous. This disables frequency checking and related warnings. Notably, it also prevents the output clocks from automatically being inverted on output (this provides for an optimal setup/hold clocking window for the sytem under test). Please see PGApp User Manual for more discussion.

This setting is allowed to be changed while the PG is running.

5.3.28 IPGSystem::SetEventFilterPeriod

Description:
Sets the event filter period.

IDL Syntax:
HRESULT SetEventFilterPeriod([in] long FilterPeriod)

Arguments:
FilterPeriod – filter period setting; use the PGAPPEventFilterPeriod enumeration:
PGAPP_EVENT_FILTER_0_NS = 0,
PGAPP_EVENT_FILTER_25_NS = 1,
PGAPP_EVENT_FILTER_50_NS = 2

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_E_INVALID_ARG	The FilterPeriod argument was invalid
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
// set the event filter period to 25 ns
pSystem->SetEventFilterPeriod(PGAPP_EVENT_FILTER_25_NS);
```

Remarks:
This setting is allowed to be changed while the PG is running.

5.3.29 IPGSystem::SetEventModeForAdvance

Description:
Determines how WaitFor events are processed, either edge or level triggered.

IDL Syntax:
HRESULT SetEventModeForAdvance([in] long EventMode)

Arguments:
EventMode – the event mode setting; use the PGAPPEventMode enumeration:
PGAPP_EDGE = 0,
PGAPP_LEVEL = 1

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_E_SYSTEM_RUNNING	The PG is running
PGAPP_E_INVALID_ARG	The EventMode argument was invalid
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
// set the event mode for advance to LEVEL
pSystem->SetEventModeForAdvance(PGAPP_LEVEL);
```

Remarks:
This setting is only allowed to be changed when the PG is idle.

5.3.30 IPGSystem::SetEventModeForJump

Description:
Determines how events are processed for branches, either edge or level triggered.

IDL Syntax:
HRESULT SetEventModeForJump([in] long EventMode)

Arguments:
EventMode – the event mode setting; use the PGAPPEventMode enumeration:
PGAPP_EDGE = 0,
PGAPP_LEVEL = 1

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_E_SYSTEM_RUNNING	The PG is running
PGAPP_E_INVALID_ARG	The EventMode argument was invalid
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
// set the event mode for branches to LEVEL
pSystem->SetEventModeForJump(PGAPP_LEVEL);
```

Remarks:
This setting is only allowed to be changed when the PG is idle.

5.3.31 IPGSystem::SetEventThreshold

Description:
Sets the threshold for input events.

IDL Syntax:
HRESULT SetEventThreshold([in] double Threshold)

Arguments:
Threshold – input event threshold in volts. The legal range for Threshold is -5.0V to 5.0V.

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_E_INVALID_ARG	The Threshold argument was invalid
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
// set the event input threshold to 1.5 volts
pSystem->SetEventThreshold(1.5);
```

Remarks:
This setting is allowed to be changed while the PG is running.

5.3.32 IPGSystem::SetHiZOnStop

Description:
Sets the HiZOnStop flag.

IDL Syntax:
HRESULT SetHiZOnStop([in] long HiZOnStop)

Arguments:
HiZOnStop – flag to clear (0) or set (1) the HiZOnStop setting.

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_E_SYSTEM_RUNNING	The PG is running
PGAPP_E_INVALID_ARG	The HiZOnStop argument was invalid
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
// set HiZOnStop
pSystem->SetHiZOnStop(1);
```

Remarks:
Puts output data and strobe signals into high-impedance state and sets flag to return signals to high-impedance state when Run completes.

This setting is only allowed to be changed when the PG is idle.

5.3.33 IPGSystem::SetHostRunTrigger

Description:
Sets the HostRunTrigger flag, causing the PG to run if armed.

IDL Syntax:
HRESULT SetHostRunTrigger()

Arguments:
none

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
// set host run trigger flag
pSystem->SetHostRunTrigger();
```

Remarks:
This routine has no effect if the PG is not in arm mode.

The PG enters arm mode when the run trigger source is set to a signal other than None and then run. At the time the PG is run, the run trigger flag is cleared and the PG waits for the run trigger to occur. The HostRunTrigger is effectively a master trigger, forcing a trigger regardless of run trigger source.

5.3.34 IPGSystem::SetInputClockDelay

Description:
Sets the external input clock delay.

IDL Syntax:
HRESULT SetInputClockDelay([in] double Delay)

Arguments:
Delay – input clock delay in seconds. The valid range for Delay depends on clock frequency:

Clock Frequency	**Delay Range**
>= 29 MHz	0 to 17.25 ns
< 29 MHz	0 to 500 ns

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_E_INVALID_ARG	The Delay argument was invalid
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
// set the input clock delay to 5 ns
pSystem->SetInputClockDelay(5.0E-9);
```

Remarks:
This setting will delay all output clocks relative to the input clock but will have no effect on output clock/data/strobe timing relationships.

Note that this setting will be clipped to a maximum of 17.25 ns if the clock frequency is changed from less than 29 MHz to greater than 29 MHz.

This setting is allowed to be changed while the PG is running. However, the user should be aware that adjusting the input clock delay when the PG is running can have undesirable side effects. Because changes to the clock delay are asynchronous to the clock, it is possible to generate a glitch. If a glitch occurs, subsequent vectors output by the PG cannot be guaranteed to be correct (until the sequence is stopped and restarted). Thus, the real-time adjustment of clock delay is currently intended to be used for

calibration purposes only (e.g. to align edge positions relative to reference signals when using a looping test sequence).

We are working to improve this situation. Please call us to discuss your critical application needs for glitch-less, run-time, clock delay adjustment.

5.3.35 IPGSystem::SetInputClockFilterPeriod

Description:
Sets the external input clock filter period to filter glitches of width less than the filter period from the input clock.

IDL Syntax:
HRESULT SetInputClockFilterPeriod([in] long FilterPeriod)

Arguments:
FilterPeriod – input clock filter period; use enumeration PPAPPClockFilterPeriod:
PGAPP_CLOCK_FILTER_0_NS = 0,
PGAPP_CLOCK_FILTER_1_NS = 1,
PGAPP_CLOCK_FILTER_4_NS = 2,
PGAPP_CLOCK_FILTER_10_NS = 3

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_E_INVALID_ARG	The FilterPeriod argument was invalid
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
// set clock filter period to 4 ns
pSystem->SetInputClockFilterPeriod(PGAPP_CLOCK_FILTER_4_NS);
```

Remarks:
This setting is allowed to be changed while the PG is running.

5.3.36 IPGSystem::SetInputClockInvert

Description:
Sets/clears the external input clock inversion flag

IDL Syntax:
HRESULT SetInputClockInvert([in] long Invert)

Arguments:
Invert -- flag to clear (0) or set (1) input clock inversion.

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_E_INVALID_ARG	The Invert argument was invalid
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
// invert the input clock
pSystem->SetInputClockInvert(1);
```

Remarks:
This setting will effectively delay all output clocks relative to the input clock by 180 degrees but will have no effect on output clock/data/strobe timing relationships.

This setting is allowed to be changed while the PG is running.

5.3.37 IPGSystem::SetInputClockThreshold

Description:
Sets the threshold for the external input clock.

IDL Syntax:
HRESULT SetInputClockThreshold([in] double Threshold)

Arguments:
Threshold – the external input clock threshold in volts. Valid range is from -2V to 2.5V.

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_E_INVALID_ARG	The Threshold argument was invalid
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
// set the input clock threshold to 1.5V
pSystem->SetInputClockThreshold(1.5);
```

Remarks:
This setting is allowed to be changed while the PG is running.

5.3.38 IPGSystem::SetOutputLevel

Description:
Sets the output level of the given probe.

IDL Syntax:
HRESULT SetOutputLevel([in] long ModulePos, [in] long ProbeIndex, [in] double Level)

Arguments:
ModulePos – acquired module position, i.e. an ordinal in the range of 0 to the value returned by GetAcquiredModuleCount - 1. Currently, this must be (0) since only one module may be acquired at a time.
ProbeIndex – probe index, use 0 = A, 1 = B, 2 = C, 3 = D
Level – output level in volts. The valid range for output level depends on probe type as follows:

Probe Type	Min (V)	Min Spec (V)	Max (V)
P370	0.30	4.50	5.50
P370LV	0.50	1.65	3.60
P370LV2	0.54	0.80	2.50
LVDS	N/A	N/A	N/A
P375	N/A	N/A	N/A

Level must fall within Min/Max range or SetOutputLevel will return an error. If it is less than Min Spec but greater than or equal to Min it is allowed, but falls outside of probe's guaranteed operating range.

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_E_INVALID_ARG	An argument was invalid.
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
// set the output level of probe 2 to 3.3 V
pSystem->SetOutputLevel(0, 2, 3.3);
```

Remarks:
This setting is allowed to be changed while the PG is running.

5.3.39 IPGSystem::SetReferenceClockSource

Description:
Sets the reference clock source for internal clocking.

IDL Syntax:
HRESULT SetReferenceClockSource([in] long Source)

Arguments:
Source – the reference clock source; use PGAPPRefClkSource enumeration
PGAPP_INTERNAL_REFCLK = 0,
PGAPP_EXTERNAL_REFCLK = 1,
PGAPP_TLA_BACKPLANE_REFCLK = 2,
PGAPP_TEKLINK_REFCLK = 3

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_E_SYSTEM_RUNNING	The PG is running
PGAPP_INVALID_ARG	Invalid Source argument
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
// set the reference clock source to internal
pSystem->SetReferenceClockSource(PGAPP_INTERNAL_REFCLK);
```

Remarks:
A PGAPP_INVALID_ARG error is returned if the TLA backplane reference clock is set as the run trigger source and the TLA backplane is not detected (i.e. the PG module is running in a stand-alone cabinet).

PGAPP_TEKLINK_REFCLK is not currently supported.

This setting is only allowed to be changed when the PG is idle.

5.3.40 IPGSystem::SetRunMode

Description:
Sets the PG run mode.

IDL Syntax:
HRESULT SetRunMode([in] long RunMode)

Arguments:
RunMode – run mode setting; use from PGAPPRunMode enumeration:
PGAPP_CONTINUOUS = 0,
PGAPP_STEP = 1

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_E_SYSTEM_RUNNING	The PG is running
PGAPP_INVALID_ARG	Invalid RunMode argument
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
// set the run mode to continuous
pSystem->SetRunMode(PGAPP_CONTINUOUS);
```

Remarks:
Set the Run mode to Step to allow single-vector stepping using the Step() command when the PG is running. Set to Continuous mode for normal operation.

This setting is only allowed to be changed when the PG is idle.

5.3.41 IPGSystem::SetRunTriggerSource

Description:
Sets the run trigger source for the PG.

IDL Syntax:
HRESULT SetRunTriggerSource([in] long RunTriggerSource)

Arguments:
RunTriggerSource – signal to use for the run trigger; use the PGAPPSignalType enumeration:

PGAPP_SIGNAL_NONE	= 0,
PGAPP_TLA_BACKPLANE_SIGNAL1	= 1,
PGAPP_TLA_BACKPLANE_SIGNAL2	= 2,
PGAPP_TLA_BACKPLANE_SIGNAL3	= 3,
PGAPP_TLA_BACKPLANE_SIGNAL4	= 4,
PGAPP_EXT_TRIG_IN	= 6,
PGAPP_HOST_TRIGGER	= 8

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_INVALID_ARG	Invalid RunTriggerSource argument
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
// set the run trigger to ExtTrigIn
pSystem->SetRunTriggerSource(PGAPP_EXT_TRIG_IN)
```

Remarks:
A PGAPP_INVALID_ARG error is returned if a TLA backplane signal is set as the run trigger source and the TLA backplane is not detected (i.e. the PG module is running in a stand-alone cabinet).

The PG enters arm mode when the run trigger source is set to a signal other than None and then run. At the time the PG is run, the run trigger flag is cleared and the PG waits for the run trigger to occur. Regardless of run trigger source, SetHostRunTrigger can always be used to trigger the PG.

5.3.42 IPGSystem::SetSignalInput

Description:
Selects the signal input to be used as part of the event definition equations.

IDL Syntax:
HRESULT SetSignalInput([in] long InputSource)

Arguments:
InputSource –SignalIn input source; use the PGAPPSignalType enumeration:

PGAPP_SIGNAL_NONE	= 0,
PGAPP_TLA_BACKPLANE_SIGNAL1	= 1,
PGAPP_TLA_BACKPLANE_SIGNAL2	= 2,
PGAPP_TLA_BACKPLANE_SIGNAL3	= 3,
PGAPP_TLA_BACKPLANE_SIGNAL4	= 4,
PGAPP_EXT_TRIG_IN	= 6

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_E_SYSTEM_RUNNING	The PG is running
PGAPP_INVALID_ARG	Invalid InputSource argument
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
// set the input source to ExtTrigIn
pSystem->SetSignalInput(PGAPP_EXT_TRIG_IN);
```

Remarks:
The signal selected here is used as the 9^{th} bit in the event definition equations.

This setting is only allowed to be changed when the PG is idle.

A PGAPP_INVALID_ARG error is returned if a TLA backplane signal is set as the signal input and the TLA backplane is not detected (i.e. the PG module is running in a stand-alone cabinet).

5.3.43 IPGSystem::SetSignalOutput

Description:
Selects the destination for the SignalOut signal defined in the sequence definition.

IDL Syntax:
HRESULT SetSignalOutput([in] long OutputDest)

Arguments:
OutputDest – SignalOut output destination; use the PGAPPSignalType enumeration:

PGAPP_SIGNAL_NONE	= 0,
PGAPP_TLA_BACKPLANE_SIGNAL1	= 1,
PGAPP_TLA_BACKPLANE_SIGNAL2	= 2,
PGAPP_TLA_BACKPLANE_SIGNAL3	= 3,
PGAPP_TLA_BACKPLANE_SIGNAL4	= 4,
PGAPP_EXT_TRIG_OUT	= 7

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_E_SYSTEM_RUNNING	The PG is running
PGAPP_INVALID_ARG	Invalid OutputDest argument
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
// set the output destination for SignalOut to ExtTrigOut
pSystem->SetSignalOutput(PGAPP_EXT_TRIG_OUT);
```

Remarks:
This setting is only allowed to be changed when the PG is idle.

A PGAPP_INVALID_ARG error is returned if a TLA backplane signal is set as the signal output and the TLA backplane is not detected (i.e. the PG module is running in a stand-alone cabinet).

5.3.44 IPGSystem::SetStrobeShape

Description:
Selects the output strobe shape for the given probe.

IDL Syntax:
HRESULT SetStrobeShape([in] long ModulePos, [in] long ProbeIndex, [in] long StrobeShape)

Arguments:
ModulePos – acquired module position, i.e. an ordinal in the range of 0 to the value returned by GetAcquiredModuleCount - 1. Currently, this must be (0) since only one module may be acquired at a time.
ProbeIndex – probe index, use 0 = A, 1 = B, 2 = C, 3 = D
StrobeShape – the selected stobe shape; use the PGAPPStrobeShape enumeration:

PGAPP_FULL	= 0,
PGAPP_HALF_CLK_POS	= 1,
PGAPP_HALF_CLK_NEG	= 2,
PGAPP_INV_FULL	= 4,
PGAPP_INV_HALF_CLK_POS	= 5,
PGAPP_INV_HALF_CLK_NEG	= 6

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_INVALID_ARG	Invalid argument
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
// set the strobe shape for probe 1 to be inverted, half-clock, during the negative half of
// the clock cycle
pSystem->SetStrobeShape(0, 1, PGAPP_INV_HALF_CLK_NEG);
```

Remarks:
See the PGApp user manual for a detailed description of strobes their shapes.

This setting is allowed to be changed while the PG is running.

5.3.45 IPGSystem::SetVarDelay

Description:
Sets the output delay for a variable probe channel.

IDL Syntax:
HRESULT SetVarDelay([in] long ModulePos, [in] long ProbeIndex, [in] long ChannelIndex, [in] double Delay)

Arguments:
ModulePos – acquired module position, i.e. an ordinal in the range of 0 to the value returned by GetAcquiredModuleCount - 1. Currently, this must be (0) since only one module may be acquired at a time.
ProbeIndex – probe index, use 0 = A, 1 = B, 2 = C, 3 = D
ChannelIndex – channel index, use the PGAPPVarChannel enumeration:
PGAPP_CHANNEL_A0 = 0,
…
PGAPP_CHANNEL_A15 = 15,
PGAPP_CHANNEL_STROBE = 16,
PGAPP_CHANNEL_CLK = 18
Delay – output delay in ps. For the P375, the valid range is 0 to 2400 ps.

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_INVALID_ARG	Invalid argument
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
// set delay for channel 10 of variable probe 2 to 1 ns
pSystem->SetVarDelay(0, 2, PGAPP_CHANNEL_A10, 1000.0);
```

Remarks:
The channel delay is in addition to the DataDelay setting for the probe.

This setting is allowed to be changed while the PG is running.

5.3.46 IPGSystem::SetVarDifferential

Description:
Sets the differential mode for a variable probe channel.

IDL Syntax:
HRESULT SetVarDifferential([in] long ModulePos, [in] long ProbeIndex, [in] long ChannelIndex, [in] long Differential)

Arguments:
ModulePos – acquired module position, i.e. an ordinal in the range of 0 to the value returned by GetAcquiredModuleCount - 1. Currently, this must be (0) since only one module may be acquired at a time.
ProbeIndex – probe index, use 0 = A, 1 = B, 2 = C, 3 = D
ChannelIndex – channel index, use the PGAPPVarChannel enumeration (event channels only):
PGAPP_CHANNEL_A0 = 0,
PGAPP_CHANNEL_A2 = 2,
…
PGAPP_CHANNEL_A14 = 14,
PGAPP_CHANNEL_STROBE = 16,
PGAPP_CHANNEL_CLK = 18
Differential – mode flag. Set to (0) for single-ended, (1) for differential operation.

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_INVALID_ARG	Invalid argument
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
// set channel 10 to of variable probe 2 to be differential
// (uses channel 11 as negative differential signal component)
pSystem->SetVarDelay(0, 2, PGAPP_CHANNEL_A10, 1);
```

Remarks:
Only even channels may be set to differential mode. If set to differential, the channel data for the subsumed odd channel is not output.

This setting is allowed to be changed while the PG is running.

5.3.47 IPGSystem::SetVarGroup

Description:
Sets the group number for a variable probe channel. Any parameter setting to one variable channel is automatically propagated to other channels in the same group.

IDL Syntax:
HRESULT SetVarGroup([in] long ModulePos, [in] long ProbeIndex, [in] long ChannelIndex, [in] long Group)

Arguments:
ModulePos – acquired module position, i.e. an ordinal in the range of 0 to the value returned by GetAcquiredModuleCount - 1. Currently, this must be (0) since only one module may be acquired at a time.
ProbeIndex – probe index, use 0 = A, 1 = B, 2 = C, 3 = D
ChannelIndex – channel index, use the PGAPPVarChannel enumeration:
PGAPP_CHANNEL_A0 = 0,
…
PGAPP_CHANNEL_A15 = 15,
PGAPP_CHANNEL_STROBE = 16,
PGAPP_CHANNEL_CLK = 18
Group – group number. Use 0-3 for group index or -1 for no group.

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_INVALID_ARG	Invalid argument
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
// set channels 0-7 of variable probe 2 to group 0, then set their delays to 1 ns
for (i = 0; i < 8; i++) {
        pSystem->SetVarDelay(0, 2, i, 0);
}
pSystem->SetVarDelay(0, 2, 0, 1000.0);
```

Remarks:
This setting is allowed to be changed while the PG is running.

5.3.48 IPGSystem::SetVarHighLevel

Description:
Sets the high-level voltage for a variable probe channel.

IDL Syntax:
HRESULT SetVarHighLevel([in] long ModulePos, [in] long ProbeIndex, [in] long ChannelIndex, [in] double Level)

Arguments:
ModulePos – acquired module position, i.e. an ordinal in the range of 0 to the value returned by GetAcquiredModuleCount - 1. Currently, this must be (0) since only one module may be acquired at a time.
ProbeIndex – probe index, use 0 = A, 1 = B, 2 = C, 3 = D
ChannelIndex – channel index, use the PGAPPVarChannel enumeration:
PGAPP_CHANNEL_A0 = 0,
…
PGAPP_CHANNEL_A15 = 15,
PGAPP_CHANNEL_STROBE = 16,
PGAPP_CHANNEL_CLK = 18
Level – high-level output voltage. For the P375, the valid range is -2.2 to 6.5 V.

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_INVALID_ARG	Invalid argument
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
// set high-level voltage for channel 10 of variable probe 2 to 5.0 V
pSystem->SetVarHighLevel(0, 2, PGAPP_CHANNEL_A10, 5.0);
```

Remarks:
This setting is allowed to be changed while the PG is running.

5.3.49 IPGSystem::SetVarInhibitEnable

Description:
Enables/disables the inhibit inputs of a variable probe.

IDL Syntax:
HRESULT SetVarInhibitEnable([in] long ModulePos, [in] long ProbeIndex, [in] long Enable)

Arguments:
ModulePos – acquired module position, i.e. an ordinal in the range of 0 to the value returned by GetAcquiredModuleCount - 1. Currently, this must be (0) since only one module may be acquired at a time.
ProbeIndex – probe index, use 0 = A, 1 = B, 2 = C, 3 = D
Enable – flag to enable (1) or disable (0) the inhibit inputs

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_INVALID_ARG	Invalid argument
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
// enable inhibit inputs for variable probe 2
pSystem->SetVarInhibitEnable(0, 2, 1);
```

Remarks:
This setting is allowed to be changed while the PG is running.

5.3.50 IPGSystem::SetVarInhibitThreshold

Description:
Sets the inhibit threshold for a variable probe.

IDL Syntax:
HRESULT SetVarInhibitThreshold([in] long ModulePos, [in] long ProbeIndex, [in] double Threshold)

Arguments:
ModulePos – acquired module position, i.e. an ordinal in the range of 0 to the value returned by GetAcquiredModuleCount - 1. Currently, this must be (0) since only one module may be acquired at a time.
ProbeIndex – probe index, use 0 = A, 1 = B, 2 = C, 3 = D
Threshold – threshold voltage in volts. For the P375, the valid range is -2.5 to 5 V.

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_INVALID_ARG	Invalid argument
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
// set the inhibit threshold for variable probe 2 to 3.3V
pSystem->SetVarInhibitThreshold(0, 2, 3.3);
```

Remarks:
This setting is allowed to be changed while the PG is running.

5.3.51 IPGSystem::SetVarLowLevel

Description:
Sets the low-level voltage for a variable probe channel.

IDL Syntax:
HRESULT SetVarLowLevel([in] long ModulePos, [in] long ProbeIndex, [in] long ChannelIndex, [in] double Level)

Arguments:
ModulePos – acquired module position, i.e. an ordinal in the range of 0 to the value returned by GetAcquiredModuleCount - 1. Currently, this must be (0) since only one module may be acquired at a time.
ProbeIndex – probe index, use 0 = A, 1 = B, 2 = C, 3 = D
ChannelIndex – channel index, use the PGAPPVarChannel enumeration:
PGAPP_CHANNEL_A0 = 0,
…
PGAPP_CHANNEL_A15 = 15,
PGAPP_CHANNEL_STROBE = 16,
PGAPP_CHANNEL_CLK = 18
Level – high-level output voltage. For the P375, the valid range is -2.2 to 6.5 V.

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_INVALID_ARG	Invalid argument
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
// set low-level voltage for channel 10 of variable probe 2 to -1.0 V
pSystem->SetVarLowLevel(0, 2, PGAPP_CHANNEL_A10, -1.0);
```

Remarks:
This setting is allowed to be changed while the PG is running.

5.3.52 IPGSystem::SetVarSlewRate

Description:
Sets the slew rate of a variable probe channel.

IDL Syntax:
HRESULT SetSlewRate([in] long ModulePos, [in] long ProbeIndex, [in] long ChannelIndex, [in] long SlewRate)

Arguments:
ModulePos – acquired module position, i.e. an ordinal in the range of 0 to the value returned by GetAcquiredModuleCount - 1. Currently, this must be (0) since only one module may be acquired at a time.
ProbeIndex – probe index, use 0 = A, 1 = B, 2 = C, 3 = D
ChannelIndex – channel index, use the PGAPPVarChannel enumeration:
PGAPP_CHANNEL_A0 = 0,
…
PGAPP_CHANNEL_A15 = 15,
PGAPP_CHANNEL_STROBE = 16,
PGAPP_CHANNEL_CLK = 18
SlewRate – flag to select normal (0) or slow (1) slew rate. For the P375, the normal slew rate is approximately 3500 V/us and slow slew rate is approximately 1750 V/us.

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_INVALID_ARG	Invalid argument
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
// set  slew rate for channel 10 of variable probe 2 to normal
pSystem->SetVarSlewRate(0, 2, PGAPP_CHANNEL_A10, 0);
```

Remarks:
This setting is allowed to be changed while the PG is running.

5.3.53 IPGSystem::Step

Description:
Force the output clock to transition for one cycle.

IDL Syntax:
HRESULT Step()

Arguments:
None

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_E_SYSTEM_NOT_RUNNING	The PG is not running
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
// step the output clock for 100 clocks
// assumes PG is running and in Step mode
for (i = 0; i < 100; i++) {
        pSystem->Step();
}
```

Remarks:
Assumes PG is running and in Step mode. (or PGAPP_E_FAILED will be returned)

5.3.54 IPGSystem::Stop

Description:
Stops the PG.

IDL Syntax:
HRESULT Stop()

Arguments:
None

HRESULT Return Codes:

Return Code	Description
S_OK	The operation succeeded.
PGAPP_E_BUSY	PGApp is busy initializing from previous call.
PGAPP_E_SYSTEM_NOT_RUNNING	The PG is not running
PGAPP_E_FAILED	The operation was unsuccessful. .

Example:

```
// stop the PG
pSystem->Stop();
```

Remarks:
None