CPhyGenCtl RPC

Table of Contents

Contacting **The *Moving Pixel* Company**

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	

Documentation

1 Overview

This document describes details of the communication interface for CPhyGenCtl referred to as RPC, which is a general acronym that stands for “Remote Procedure Call”. CPhyGenCtl support automation and remote control via RPC requests from other applications via a TCP server port.

To support this functionality, a separate .NET DLL is provided, called CphyGenCtlRPC.dll. Any user program that can interface to a .NET DLL can control CPhyGenCtl using RPC calls. This document assumes that the user is conversant with programming in the .NET environment or otherwise knowledgeable about interfacing between his preferred programming environment (e.g. LabView, Matlab, etc.) and .NET.

Alternatively, RPC commands have a script form that can be used in a text script file and executed directly from the CPhyGenCtl GUI using the “RPC Script” command. In this case, no external program is required. Many customers use script files extensively as they are a very powerful and convenient tool to configure and control CPhyGenCtl.

In general, RPC commands can perform almost any task that can be accomplished using the GUI. They can be used to configure all aspects of the CPhyGenCtl environment, including setting options and CPhy timing parameters, adjusting voltages and delays, configuring frequencies and video frame timing, etc. They can be used to build and send MIPI commands, macros, and initiate video mode. They can even be used to build and send custom-defined video frames, allowing implementation of non-standard, user-specific test suites.

2 Configuring CPhyGenCtl as an RPC Server

Before CPhyGenCtl will accept incoming RPC calls from remote applications, it must be enabled to accept them and given a TCP/UDP port number to use on the host computer.[1]

TCP/UDP ports are resources of the computer operating system used for inter-process communication. A port is referenced by its port number, which is a 16-bit integer (1-65525), with certain ports reserved for use by the OS and other "well-known" or registered ports are used by common Windows applications. Generally, these ports have port numbers at the low end of the range and so it is best to select a higher number in the range to avoid conflict.

For more information about TCP ports, please refer to the following link:

http://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers

To enable the CPhyGenCtl to listen on a port as an RPC server, simply select the "Enable RPC" menu option in the Connect menu. If this option is not already checked, a simple dialog will appear requesting the port number to use for incoming RPC requests. To disable CPhyGenCtl as an RPC server, uncheck the "Enable RPC" menu option.

When CPhyGenCtl is enabled as an RPC server, the user can still interact with the GUI normally. In addition, RPC calls from other applications can interact with the application as well, able to perform many of the functions programmatically that the user can perform using the GUI.

Notes:

- Currently there is no access control to prevent multiple applications from connecting and making RPC calls simultaneously to the CPhyGenCtl server even as the user is using the application, which, of course, may cause confusing behavior.
- Where a user initiated action would normally elicit a message box on CPhyGenCtl, usually a warning or a confirmation, the equivalent RPC call will not do so. Instead, the operation proceeds as if the user had responded affirmative to the message.
- RPC calls block until CPhyGenCtl has completed processing them. In some cases, this can take many seconds, especially when building and sending large video command sequences. For client applications that require a responsive interface, RPC calls should be made from a separate thread.

[1] Note that these steps are not required when executing an RPC script file from the PGRemote GUI.

3 CPhyGenCtlRPCClient Library

To facilitate the user in building RPC client programs to control CPhyGenCtl, the CPhyGenCtlRPCClient.dll is provided. This library is written in C# and provides classes to encapsulate error codes, command definitions, command field definitions, and a few simple calls to establish a connection to the CPhyGenCtl server and send commands. These classes are contained in the namespace "CPhyGenCtlRPC" and are described below.

Note: for those interfacing to the CPhyGenCtlRPCClient dll from language other than .NET-aware languages such as C#, managed C++, Visual Basic, etc. a significantly deeper understanding of COM and how to interface to a "managed" .NET dll is required. Example languages that fall into this category are Labview, Matlab, and Python. This may be necessary especially with respect to function signatures, which currently take advantage of C# generic types and may not be easily portable to other languages or environments.

To this last point, an attempt has been made to provide alternate strictly-typed function calls that may be more easily used from non-.NET languages. In addition, a type library (.tlb) is provided along with CPhyGenCtlRPCClient.dll that should help with importing and defining function signatures.

3.1 CPhyGenCtlRPCClient Class Overview

The CPhyGenCtlRPCClient library contains four classes:

- CPhyGenCtlClient – main client instance to use for interfacing with CPhyGenCtl.
- RPCErrs – error code definitions
- RPCCmds – RPC command code definitions
- RPCDefs – RPC command parameter definitions

These classes are further described in the following sections.

3.1.1 CPhyGenCtlClient Class

The CPhyGenCtlClient Class represents the main class of the library, providing an interface to connect to the CPhyGenCtl server, send RPC commands, and obtain results.

The client constructor takes not arguments and is simply created with the call:

```
CPhyGenCtlClient client = new CPhyGenCtlClient();
```

Next, should be a call to connect to a CPhyGenCtl server. The server should be running and enabled for RPC as described in section 2. As an example, the following call connects to a CPhyGenCtl server located on the local host machine and enabled for RPC at port 2799:

```
int rc = client.Connect("", 2799);
```

When finished sending RPC commands, disconnect from the server with

```
int rc = client.Disconnect();
```

3.1.2 RPCErrs Class

The RPCErrs class contains definitions for the possible error codes that can occur in the CPhyGenCtl server (see 5.2). However, note that only a subset of the errors in this class will be returned by an RPC call. No comprehensive documentation of these errors is provided currently – the names are meant to provide a more descriptive indication of the error that occurred. In most cases, the error code returned is also supplemented with an error message that is returned as well by the RPC call.

A few error codes deserve further comment:

FAIL: used as a generic catch-all failure code.

LOCAL_FAILURE: indicates that an error occurred on the client side (before reaching the CPhyGenCtl server). Often this error is associated with an error marshaling unexpected or incorrect command arguments.

CONTROL_IS_DISABLED: returned when setting the underlying value of a control that is disabled in CPhyGenCtl, which indicates the feature is not currently available or supported in the current configuration.

NEED_START_EDIT_COMMAND: indicates a configuration command was sent without a prior START_EDIT_CONFIG command.

3.1.3 RPCCmds Class

The RPCCmds class contains definitions for all the possible RPC commands that can be sent to the CPhyGenCtl server. General categories of RPC commands are:

- **MIPI Command:** commands that send a MIPI command.
- **Command Management**: command definitions management (adding, deleting, sorting, assigning to buttons)
- **PG and Probe Configuration**: commands that set any of the PG Configuration dialog settings (e.g. voltages, delays, etc.), as well the LP frequency, HS Sym Rate, Lane Count, and DT Mode settings.
- **CPhy Timing Configuration**: commands that define CPhy bus timing and protocol symbol sequences.

- **Frame Timing Configuration:** commands that set any of the Timing Configuration dialog settings (e.g. frame component dimensions, video timing and playback options).

- **Menu:** relevant commands from the File, Connect, Standard, and Options menus.

- **Operational:** operational commands to stop and restart the PG.

- **Status:** commands to retrieve status including PG state, DUT response data, and contention state.

- **Support:** miscellaneous support commands to compute CRC values on user data, pause for user acknowledgment, etc.

To send MIPI commands, use the MIPICmd RPC call. All other command categories use signature variants of the RPCCmd and RPCQuery calls. See section 3.1.4 for more details.

3.1.4 RPCDefs Class

The RPCDefs class overview contains constants to be used for RPC command parameters. Predominantly, this class contains all the MIPI command codes that can be used to send MIPI commands. More specifically, this class includes the following types:

- DSI Command Codes
- CSI Command Codes
- DCS Command Codes
- DPhy Command Codes
- Data Transfer Modes
- MIPI Standards
- Menu Options
- Contention Mask Bits
- Trigger commands
- Blanking Modes

See section 5.4 for details.

3.2 Sending a MIPI Command

MIPI command definition constitutes a major command class distinct from all other RPC commands. Because of its numerous arguments, a few special calls in the CPhyGenCtlRPC DLL are dedicated to defining and sending MIPI commands (MIPICmd, ImpairedMIPICmd, and AddMIPICmd). All other commands use generic argument calls (RPCCmd and RPCQuery), having multiple function signatures to support 1, 2, 3, or 4 command arguments.

The MIPICmd RPC call outputs a given command on the CPhy bus (without defining it as a named command) and has the following signature:

```
public int MIPICmd(int cmdCode, int DCSCmdCode, bool BTA, int
DTMode, int VC, int arg1, int arg2, int arg3, string fn, byte[]
data, ref string errMsg, ref string statusMsg);
```

Its parameters are a superset of what the user sees in the CPhyGenCtl GUI, where the cmdCode dictates which parameters are used and which are unused. For example, none of the short packet form DSI commands use the "fn" parameter and only one command (SET_SCROLL_AREA) uses the "arg3" parameter. In general, though not required, unused integer parameters can be set to zero, an unused "fn" parameter can be set to the empty string "", and unused data buffer set to null.

Table 1 describes the parameters of the MIPICmd library call.

Table 1 – MIPICmd Parameters

cmdCode	The RPC command code
DCSCmdCode	The DCS command code (applies only to DCS commands)
BTA	Requests a BTA sequence following the MIPI command
DTMode	Specifies the data-transfer mode for the command
VC	The MIPI virtual channel for the command (0-3)
arg1	cmdCode specific argument
arg2	cmdCode specific argument
arg3	cmdCode specific argument
fn	Specifies the file name, path relative to the CPhyGenCtl application, to use for the command (applies only to commands that use file argument). If the data argument is non-null, this field is ignored.
data	Specifies a data buffer to use for the command instead of a file (applies only to commands that use file argument). If non-null, the fn argument is ignored.
errMsg	If return code is negative, contains the error message string
statusMsg	Returns the last text displayed in the CPhyGenCtl status bar after the command is sent
return code	The return code from the command. A negative return code indicates an error from the RPCErrs class. A zero return code indicates success for most commands. Commands that return a value, e.g. GET_PG_PROBE_TYPE return a non-negative value for their result.

To give some examples, the following command causes CPhyGenCtl to send a Generic Short Write DSI command with 1 parameter (0xaa) as a high-speed packet on a DSI link:

```
rc = client.MIPICmd(RPCDefs.GENERIC_SHORT_WRITE, 0, false,
    RPCDefs.DT_HS, 0, 1, 0xaa, 0, "", null,
    ref errMsg, ref statusMsg);
```

In this case, several of the MIPICmd parameters are unused by CPhyGenCtl, specifically DCSCmdCode, BTA, arg3, fn, and data.

RPC calls mimic the functionality of CPhyGenCtl when used through the GUI. For example, to send a video frame using a BMP file as the source and the current timing parameters of CPhyGenCtl, you could use the following call:

```
rc = client.MIPICmd(RPCDefs.PACKED_PIXEL_STREAM_565,
    0, false, RPCDefs.DT_HS, 0, 0, 0, 0,
    "c:\\pictures\\test12x12.bmp", null,
    ref errMsg, ref statusMsg);
```

If you don't want CPhyGenCtl to parse the file as a BMP image, provide a filename without the ".bmp" extension. The bytes will then be interpreted as raw video data.

An alternative available to a program using an RPC call instead of a user using the GUI, is to make use of the "data" parameter of MIPICmd:

```
rc = client.MIPICmd(RPCDefs.PACKED_PIXEL_STREAM_565,
    0, false, RPCDefs.DT_HS, 0, 0, 0, 0,
    "", imageData, ref errMsg, ref statusMsg);
```

In this case, the imageData buffer containing data in the appropriate RGB565 format is used instead of a file. This allows easy programmatic construction of video frames for testing.

Another example causes CPhyGenCtl to build and send a video frame as a sequence of DCS WriteMemoryStart and WriteMemoryContinue commands.

```
rc = client.MIPICmd(RPCDefs.DCS_LONG_WRITE,
    RPCDefs.WRITE_MEMORY_START, false, RPCDefs.DT_HS,
    0, 0, 921600, 0,
    "c:\\pictures\\rgbstripes640x480.bmp",
    null, ref errMsg, ref statusMsg);
```

This call requests that CPhyGenCtl read in the given BMP file (as RGB888 format), break 921,600 bytes of it (640 x 480 x 3) into ~64 KB segments and send successive WriteMemoryStart / WriteMemoryContinue commands in a single PG sequence.

Note the RPC script command equivalent to this call is SEND_MIPI_CMD.

3.3 Sending a MIPI Command With Errors

An extended version of MIPICmd called ImpairedMIPICmd is provided that allows the user to set the ECC and/or CRC fields of the command in order to cause receive errors in the device under test. In addition, this command allows the user to provide a byte offset within the constructed packet and an XOR byte mask to be used to introduce errors in the packet at the given byte offset.

Below is the calling syntax of the ImpairedMIPICmd call:

```
public int ImpairedMIPICmd(int cmdCode, int DCSCmdCode,
         bool BTA, int DTMode, int VC, int arg1, int arg2, int arg3,
         int customPHCRC, int customCRC, int errByteOff,
         int errByteMask, string fn, byte[] data,
         ref string errMsg, ref string statusMsg)
```

The command is similar to the MIPICmd call (see previous section for details), except for the addition of four parameters. These additional parameters are described below:

Table 2 – Additional ImpairedMIPICmd Parameters

customPHCRC	The 16-bit PH CRC value to use for the packet. Set to -1 to use the correct computed PH CRC. Applies to all copies of PHCRC in the packet.
customCRC	The 16-bit CRC value to use for long packets. Set to -1 to use the correct computed CRC.
errByteOff	Specifies the byte offset, starting from the first byte in the packet header, to the byte to XOR with the errByteMask to introduce an error in the packet. Set to -1 to disable this function.
errByteMask	Specifies the XOR error byte mask to use to use at errByteOff in the packet.

The errByteOff, as described above, starts from the beginning of the packet. Byte counts include duplicated PH bytes across all lanes and must take into account byte-pair demultiplexing when lane count is more than 1. For a system with N lanes, the byte offsets of various PH fields for lane L are shown in table 3:

Table 3 – Packet Field Byte Offsets

Pkt Byte Desc	**Byte offset**
DataID, first PH	2*L + 1
Arg0 / WordCnt0, first PH	2*N + 2*L
Arg1 / WordCnt1, first PH	2*N + 2*L + 1
PHCRC0, first PH	4*N + 2*L
PHCRC1, first PH	4*N + 2*L + 1
DataID, second PH	8*N + 2*L + 1
Arg0 / WordCnt0, first PH	10*N + 2*L
Arg1 / WordCnt1, first PH	10*N + 2*L + 1
PHCRC0, first PH	12*N + 2*L
PHCRC1, first PH	12*N + 2*L + 1
First payload byte	14*N

For example, in a 4 lane system, the DataID of the second PH of lane 2 would be byte offset 37 (8*4 + 2*2 + 1). Byte offset 100 would reference payload byte 44 (100 – 14*4).

Note that the customPHCRC and customCRC are set in the packet AFTER the errByteMask is applied. So, for example, if errByteOff references a PHCRC fields and customPHCRC was set to 0145, the PHCRC in the packet would be 0x145. However, if the customECC was set to -1, one of the computed PHCRC bytes for the packet would be XORed with errByteMask to determine the byte value put in the packet header.

Note the RPC script command equivalent to this command is SEND_IMPAIRED_MIPI_CMD.

3.4 Defining a named MIPI Command

The AddMIPICmd RPC call defines a named command in the application and does not send the command on the MIPI bus. AddMIPICmd contains the same arguments as the MIPICmd call with one additional argument – cmdName – that defines the name of the command. The following is the function signature of the AddMIPICmd:

```
public int AddMIPICmd(string cmdName, int cmdCode, int DCSCmdCode,
                      bool BTA, int DTMode, int VC, int arg1,
                      int arg2, int arg3, string fn, byte[] data,
                      ref string errMsg, ref string statusMsg);
```

Once a command has been defined and named, it can be output using the RPCCmd DLL call with the SEND_NAMED_CMD_FROM_NAME command.

3.5 Sending Other RPC Commands

Most RPC commands make the use of generic RPC call RPCCmd, which has three function signatures, accommodating 0, 1, 2, or 3 command arguments respectively:

```
public int RPCCmd(int cmdCode,
                      ref string errMsg, ref string statusMsg)
public int RPCCmd<T1>(int cmdCode, T1 arg1,
                      ref string errMsg, ref string statusMsg)
public int RPCCmd<T1, T2>(int cmdCode, T1 arg1, T2 arg2,
                      ref string errMsg, ref string statusMsg)
public int RPCCmd<T1, T2, T3>(int cmdCode, T1 arg1,
                      T2 arg2, T3 arg3,
                      ref string errMsg, ref string statusMsg)
```

In addition, the RPCQuery calls may be used for calls that return more other than a single integer return code:

```
public int RPCQuery<T1>(int cmdCode, ref T1 respVal,
                      ref string errMsg, ref string statusMsg);
public int RPCQuery<T1, T2>(int cmdCode, T1 arg1, ref T2 respVal,
                      ref string errMsg, ref string statusMsg);
public int RPCQuery<T1, T2, T3>(int cmdCode, T1 arg1,
                      T2 arg2, T3 respVal,
                      ref string errMsg, ref string statusMsg);
```

Which call should be used depends on the RPC command (see RPC command descriptions).

For example, to set the HS symbol rate to 300 Msps, the single-argument signature call is used, e.g.:

```
rc = client.RPCCmd(RPCCmds.SET_HS_SYM_RATE, (float)300e+6,
                   ref errMsg, ref statusMsg);
```

But to set an option, the dual-argument signature call is used, e.g.:

```
rc = client.RPCCmd(RPCCmds.SET_OPTION,
                   RPCDefs.OPT_LOOP_COMMANDS, true,
                   ref errMsg, ref statusMsg);
```

Note that because the RPCCmd and RPCQuery calls have generic arguments, any type may be passed in successfully. However, a side-effect of this is that no implicit type casting is performed (e.g. double-to-float, float-to-int, int-to-byte, etc.). Consequently, because CPhyGenCtl expects command arguments to be a specific type, it will return an ARGTYPE_MISMATCH error if the wrong type parameter is supplied.

3.6 Alternate Command Interface For Non-.NET Languages

Some languages or programming environments cannot support the generic style argument passing (which is a Microsoft C# construct). Accordingly, the CPhyGenCtlRPC DLL supports the following alternate procedure calls:

```
int RPCCmdI(int cmdCode, int arg1,
               ref string errMsg, ref string statusMsg);
int RPCCmdF(int cmdCode, float arg1,
               ref string errMsg, ref string statusMsg);
int RPCCmdS(int cmdCode, string arg1,
               ref string errMsg, ref string statusMsg);
int RPCCmdII(int cmdCode, int arg1, int arg2,
               ref string errMsg, ref string statusMsg);
int RPCCmdIF(int cmdCode, int arg1, float arg2,
               ref string errMsg, ref string statusMsg);
int RPCCmdIS(int cmdCode, int arg1, string arg2,
               ref string errMsg, ref string statusMsg);
int RPCCmdIII(int cmdCode, int arg1, int arg2, int arg3,
               ref string errMsg, ref string statusMsg);
int RPCCmdIIS(int cmdCode, int arg1, int arg2, string arg3,
               ref string errMsg, ref string statusMsg);
int RPCCmdSIS(int cmdCode, string arg1, int arg2, string arg3,
               ref string errMsg, ref string statusMsg);
int RPCCmdIIBA(int cmdCode, int arg1, int arg2, byte[] arg3,
               ref string errMsg, ref string statusMsg);
int RPCCmdIISS(int cmdCode, int arg1, int arg2,
               string arg3, string arg4,
               ref string errMsg, ref string statusMsg);
int RPCCmdSISI(int cmdCode, string arg1, int arg2,
               string arg3, int arg4,
               ref string errMsg, ref string statusMsg);
```

```
int RPCQueryIRetS(int cmdCode, int arg1, ref string respVal,
                  ref string errMsg, ref string statusMsg);
int RPCQuerySRetS(int cmdCode, string arg1, ref string respVal,
                  ref string errMsg, ref string statusMsg);
```

These calls have explicit argument types and can be used instead of their generic equivalents. The naming convention is such that 'F' stands for a floating point argument, 'I' stands for and integer argument, and 'S' stands for a string argument, and 'BA' stands for byte array.

3.7 RPC Notes

The behavior of most commands, their usage, and arguments are self-explanatory, but certain modes and commands some deserve additional comment.

3.7.1 Setting MIPI Standard

Before sending any MIPI commands, make sure to send the SET_MIPI_STANDARD command to select which command set is available (currently CSI or DSI). Otherwise, the CMD_STANDARD_MISMATCH error will be returned for calls not supported by the current standard.

3.7.2 Inserting Commands During Video Playback

Command insertion while video is looping (or a macro having a command insertion point) is achieved in the same way as when using the GUI. Simply set the "Enable Command Insertion" option (SET_OPTION with OPT_ENABLE_CMD_INSERTION argument), start looping video, then send another non-video MIPI command.

The inserted command will be sent during the vertical blanking of one of the looping video frames. Note that if command insertion is not enabled, subsequent commands to a looping video command will stop the PG first. This happens automatically since, as noted earlier, all message box queries from CPhyGenCtl are assumed to be answered affirmatively.

3.7.3 Sending Configuration Commands

Because of how they are implemented, RPC configuration commands must be bracketed with START_EDIT_CONFIG and END_EDIT_CONFIG. These commands behave equivalently to the user clicking on the Inst Cfg… button and the OK button in the dialog respectively. Multiple start/end sequences are allowed and there is no restriction on which subset of configuration commands can be present in each sequence.

For example, the following sequence would be used to set the DTMode, LP and HS frequencies, and lane count:

```
// set configuration parameters
rc = client.RPCCmd(RPCCmds.START_EDIT_CONFIG,
                   ref errMsg, ref statusMsg);
rc = client.RPCCmd(RPCCmds.SET_DT_MODE, RPCDefs.DT_HS,
                   ref errMsg, ref statusMsg);
rc = client.RPCCmd(RPCCmds.SET_LP_FREQ, (float)7.6e+6,
                   ref errMsg, ref statusMsg);
```

```
rc = client.RPCCmd(RPCCmds.SET_HS_SYM_RATE, (float)300e+6,
                   ref errMsg, ref statusMsg);
rc = client.RPCCmd(RPCCmds.SET_LANE_CNT, 4,
                   ref errMsg, ref statusMsg);
rc = client.RPCCmd(RPCCmds.END_EDIT_CONFIG,
                   ref errMsg, ref statusMsg);
```

3.7.4 Defining and Sending Macros

Similarly, macros are defined by bracketing their component commands with START_MACRO and END_MACRO. The START_MACRO call puts CPhyGenCtl in macro-mode and initializes to build a new macro. All MIPI commands that are subsequently sent are added to the current macro (rather than forwarded to the CPhy Generator for playback). Then, when the END_MACRO call is made, macro-mode is exited, and the macro is defined and named according to the string argument provided in the END_MACRO command.

Unlike non-macro commands that can be sent without being named, macros must first be defined as named commands. Once defined, they can then be sent using the SEND_NAMED_CMD_FROM_NAME command.

Only MIPI commands may be components of a macro. Thus, when in macro mode, only commands sent via the following DLL commands are stored in the macro:

- MIPICmd
- ImpairedMIPICmd
- RPCCmd with SEND_NAMED_CMD_FROM_NAME
- RPCCmd with SEND_NAMED_CMD_FROM_INDEX

For example, it is not possible to change the voltage or symbol rate in the middle of a macro. However, as with defining macros using the GUI, LP and HS packet types may be mixed.

Note that, while in the CPhyGenCtl GUI sending packed pixel stream commands result in video mode being invoked (and full frames built as a result), when these commands are sent in a macro, they result in a single packed pixel stream packet being sent. Thus, conceptually, RPC macros can be used to build and send full video frames.

The following sequence defines and plays a simple macro:

```
// begin macro definition
rc = client.CPhyGenCtlCmd(RPCCmds.START_MACRO,
                   ref errMsg, ref statusMsg);

rc = client.MIPICmd(RPCDefs.COLOR_MODE_ON, 0, false,
                   RPCDefs.DT_HS, 0, 0, 0, 0, "", null,
                   ref errMsg, ref statusMsg);
rc = client.MIPICmd(RPCDefs.TURN_ON_PERIPHERAL, 0, false,
                   RPCDefs.DT_HS, 0, 0, 0, 0, "", null,
                   ref errMsg, ref statusMsg);
```

```
rc = client.MIPICmd(RPCDefs.DCS_SHORT_WRITE,
                    RPCDefs.ENTER_NORMAL_MODE, false,
                    RPCDefs.DT_HS, 0, 0, 0, 0, "", null,
                    ref errMsg, ref statusMsg);

// end macro definition and send to PG
rc = client.CPhyGenCtlCmd(RPCCmds.END_MACRO,
                    ref errMsg, ref statusMsg);
```

3.7.5 Using the USER_WAIT Command

Often an RPC script is used for test automation. To facilitate using a single script to perform a sequence of tests, the USER_WAIT command is provided. When this command is sent, a message box is displayed with the supplied string message. For example, the message might read “Click OK to perform test #6”.

While the application is waiting, the user can perform whatever actions are necessary to retrieve the results from the previous test and set hardware up for the subsequent test. When ready, the user can click OK to continue executing script commands or CANCEL to abort script processing.

3.7.6 Using the Custom Commands

Two generic calls are available for the user to define arbitrary commands: the CUSTOM_COMMAND and CUSTOM_LONG_COMMAND calls. These can be used to build arbitrary packets with proper DSI/CSI packet structure. Both have arguments consisting of the DataID to use and a data array containing the payload bytes for the command. For the CUSTOM_COMMAND call, if the payload array is less than or equal to two bytes, a short packet is built with the correct PHCRC. Otherwise, a long packet is built with the correct length field, PHCRC, and checksum given the DataID and payload data. For the CUSTOM_LONG_COMMAND call, a long packet format is always used, even for zero, one and two byte data array lengths.

For example, the following code sends a short packet with DataID = 0xfd:

```
// send custom short packet
byte[] data = new byte[2];
data[0] = 0x1b;
data[1] = 0x2c;
rc = client.MIPICmd(RPCDefs.CUSTOM_COMMAND, 0, false,
                    RPCDefs.DT_HS, 0, 0xfd, 0, 0, "", data,
                    ref errMsg, ref statusMsg);
```

And this code sends a 100-byte long packet with DataID = 0xfd:

```
// send custom long packet
byte[] data = new byte[100];
for (int inx = 0; inx < 100; inx++)
      data[inx++] = (byte)inx;
rc = client.MIPICmd(RPCDefs.CUSTOM_COMMAND, 0, false,
                    RPCDefs.DT_HS, 0, 0xfd, 0, 0, "", data,
                    ref errMsg, ref statusMsg);
```

4 CPhyGenCtlRPC Project

When CPhyGenCtl is installed, a simple Visual Studio 2008 project written in C# is included that demonstrates use of the RPC interface. The project is called CPhyGenCtlRPCTest and is located in the CPhyGenCtl application directory:

c:\Program Files\TMPC\CPhyGenCtl

The Main routine entry point is in the file Program.cs, which starts by connecting to the CPhyGenCtl server which is assumed to be running on the local host and enabled for RPC at port 2799. Once connected, the routine simply progresses through most available RPC commands, segregated according to category (e.g. options, CPhy configuration, instrument configuration, frame timing, named commands, macros, status, video). This code doesn't perform anything particularly useful and is intended to be run in the debugger, stepping or running with breakpoints set at locations of interest.

Note the reference to the CPhyGenCtlRPCClient DLL, which is located in the bin/Debug directory (along with a sample bmp and data file used by the program).

5 RPC Script Files

RPC script files are text files with embedded RPC commands. They are very useful since the RPC script command can be built and saved as a named command and assigned to buttons like any other command. As RPC commands can assign almost all settings in CPhyGenCtl and can send almost all commands to the PG, it allows for powerful reconfiguration and testing capabilities at the push of a single button.

This section describes the syntax used in RPC script files. An easy way to generate proper MIPI_CMD syntax is to use the Script Recording feature of CPhyGenCtl. This is enabled using the "Script->Start Recording…" menu option. Subsequently, any command sent will be recorded in the recorded file. Please refer to the CPhyGenCtl User's Manual for more information.

Note that to use RPC scripts you do not need RPC enabled on CPhyGenCtl and you do not need a programming environment or DLL libraries. All you need is a text editor.

An RPC script file has the following syntax:

- A comment line begins with "//" and is ignored by the parser.
- A blank line that consists solely of spaces or tabs is ignored by the parser.
- A command line begins with the '#' character, followed by a space, followed by an RPC command plus its arguments. Only one RPC command may be defined per line.
- String constants that match any of the definitions in Appendix B or Appendix C may be used instead of numeric values for any command or its arguments.
- String constants are case insensitive (e.g. send_mipi_cmd is equivalent to Send_Mipi_Cmd is equivalent to SEND_MIPI_CMD)
- Numeric values are assumed to be decimal, unless appended with an 'h', in which case they are parsed as hex. So, "10" is interpreted as 10 decimal, "10h" is interpreted as 16, "3f" results in a parsing error, and "3fh" is interpreted as 63.
- File names should have double-quotes, e.g. "c:\rpcScript.txt". A null file should be indicated with a pair of double-quotes, e.g. "".
- or use the string "null" (case-insensitive) to specify no file name.
- Commands that require a data buffer argument, may use the string "null" (case-insensitive) to indicate a null buffer or simply list data values at the end of the command to indicate buffer values. Currently, RPC commands cannot span more than one line in the RPC script file so all data buffer values must be included on the command line.

RPC script files need to adhere to the same rules as RPC programming, and specifically certain command types have some additional requirements. Specifically, configuration commands must be bracketed by START_EDIT_CONFIG and END_EDIT_CONFIG calls, macros must be bracketed by START_MACRO and END_MACRO calls. Please see sections 3.7.3 and section 3.7.4 for more information.

RPC calls have the same arguments and types whether made from a script file or programmatically. Thus, for example, the SEND_MIPI_CMD requires 10 arguments for the MIPICmd call as described in **Table 1**. To send a DSI HS generic short write command with two arguments with values 0x10 and 0x20, you could use the following script line:

SEND_MIPI_CMD GENERIC_SHORT_WRITE 0 0 DT_HS 0 2 10h 20h "" NULL

In this case, you must refer to Appendix C to obtain the command argument definitions for arg1, arg2, and arg3 of the GENERIC_SHORT_WRITE command. Note the string literals "SEND_MIPI_CMD", "GENERIC_SHORT_WRITE", and "DT_HS" are not required and could be replaced by their numeric equivalents if desired.

If you wanted to cause a receive error in your DUT by changing the ECC to 45 (for example) you might use the following:

SEND_IMPAIRED_MIPI_CMD GENERIC_SHORT_WRITE 0 0 DT_HS 0 2 10h 20h 45 -1 -1 0 "" NULL

Other RPC commands have their own number of arguments and argument types depending on the specific command. Thus, for example, you could set the video timing line time with the following command:

SET_TIMING_LINE_TIME 5e-6

Just remember if you are setting any of the configuration commands you must bracket the calls with start and end configuration commands, e.g.

START_EDIT_CONFIG
SET_HS_SYM_RATE 120e+6
SET_LP_FREQ 15e+6
SET_LANE_CNT 4
END_EDIT_CONFIG

Finally, once a script file is defined you can execute it by simply selecting RPC Script from the PktType list-box and type in or browse for the script file name. Then, like any other command, you can click the Send button to execute the script file, or type in a name to save it as a named command and then assign it to a button.

5.1 Example RPC script

A good way to set up and control almost all aspects of CPhyGenCtl is using one or more RPC scripts. In addition to their traditional use in sending a sequence of commands, you can use them to initialize all options, timing configuration, instrument configuration, and CPhy configuration. Define a script file for each unique test configuration you need.

One way to auto-generate a configuration script containing RPC commands representing the current application state using the “Write Current State” menu command.

1. Select Script->”Write Current State”
2. Browse for the script file name to use

This creates a script like the following:

```
// Script file created 3/24/2016 14:32:13.5328688

// CPhyGenCtl version: 2.0.0.0
// Connected to CPhyGen instrument serial # 1502
// Instrument version is: 1.5
# SET_MIPI_STANDARD STD_DSI

// INSTRUMENT CONFIGURATION
# START_EDIT_CONFIG
# SET_DT_MODE DT_HS
# SET_LP_FREQ 10.00E+6
# SET_HS_SYM_RATE 100.00E+6
# SET_LANE_CNT 1
# SET_ENABLE_ALL_HS_VOLT_COMMON 1
# SET_HS_LOW_VOLT 0 0.00
# SET_HS_HIGH_VOLT 0 0.40
# SET_LP_LOW_VOLT 0.00
# SET_LP_HIGH_VOLT 1.20
# SET_LANE_DELAY 0 0E-12
# SET_LP_LOW_CONT_THRESH 0.45
# SET_LP_HIGH_CONT_THRESH 0.55
# SET_BTA_WAIT_TIME 20.00E-6
# SET_TRIG_PULSE_WIDTH 1E-05
# SET_LANE_MAP 0000h
# END_EDIT_CONFIG

// TIMING CONFIGURATION
// HTOTAL 296
// VTOTAL 85
# SET_TIMING_HSYNC 96
# SET_TIMING_HACTIVE 50
# SET_TIMING_VSYNC 2
# SET_TIMING_VACTIVE 40
# SET_TIMING_LINE_TIME 1.2432E-6
# SET_TIMING_PIX_CLK 238.0952E+6
# SET_TIMING_FRAME_RATE 9463.2448
# SET_TIMING_HS_SYM_RATE 2500000000.0000
# SET_TIMING_ENABLE_CONTINUOUS_MODE 1
# SET_TIMING_HBPORCH 100
# SET_TIMING_HFPORCH 50
# SET_TIMING_VBPORCH 33
# SET_TIMING_VFPORCH 10
# SET_TIMING_ENABLE_DSI_PULSE_MODE 1
# SET_TIMING_ENABLE_DSI_BURST_MODE 0
# SET_TIMING_TOP_FIELD_FIRST 0
# SET_TIMING_HSYNC_BLANKING_MODE LP11_BLANK_MODE
```

```
# SET_TIMING_HBPORCH_BLANKING_MODE LP11_BLANK_MODE
# SET_TIMING_HFPORCH_BLANKING_MODE HS_BLANK_MODE
# SET_TIMING_VERTICAL_BLANKING_MODE LP11_BLANK_MODE
# SET_3D_PARAMETER PARAM_3D_ENABLE 0
# SET_ADVANCE_FRAME_ON_EXTERNAL_EVENT 0
# SET_TIMING_MASTER MASTER_HS_BIT_RATE

// OPTION CONFIGURATION
# SET_OPTION OPT_LOOP_COMMANDS 1
# SET_OPTION OPT_ENABLE_HOLD_LAST_SYMBOL_TEST_MODE 0
# SET_OPTION OPT_WAIT_ON_EXT_EVENT_TO_START 0
# SET_OPTION OPT_DISABLE_EVENT_TIMEOUT 0
# SET_OPTION OPT_ENABLE_CMD_INSERTION 0
# SET_OPTION OPT_DISCARD_DURING_INSERTION 0
# SET_OPTION OPT_SEND_SINGLE_PKT_PER_HS_BURST 0
# SET_OPTION OPT_ALLOW_IMAGE_RESCALING 1

// DSI-only options
# SET_OPTION OPT_ENABLE_EOT_PKTS 0
# SET_OPTION OPT_ENABLE_DSI_SCRAMBLING 0
# SET_OPTION OPT_SEND_PPS_WITH_COMPRESSED_VIDEO 1

// WRITE_MEMORY CONFIGURATION
# SET_OPTION OPT_ENABLE_WM_PARTITIONING 1
# SET_OPTION OPT_WM_QUANTIZE_TO_LINE_LENGTH_MULTIPLE 1
# SET_WM_PARTITION_LENGTH 1000
# SET_WM_PARTITION_INTERVAL 0.00E-6
# SET_WM_IMAGE_DECODE_FORMAT FMT_RGB888_24

// DSC CONFIGURATION
# SET_DSC_SLICE_WIDTH 0
# SET_DSC_SLICE_HEIGHT 8
# SET_DSC_USE_BLOCK_PREDICTION 1
# SET_DSC_ONE_CHUNK_PER_PACKET 0
# SET_DSC_CONFIG_FILENAME rc_8bpc_8bpp.cfg

// CPHY CONFIGURATION
# SET_CPHY_ALL_LANES_COMMON 1
# SET_CPHY_LANE_DEFAULT 0 0
# SET_CPHY_SYMBOL_SEQUENCE 0 CPHY_SEQ_START_PREAMBLE 3 3 3 3 3 3 3
# SET_CPHY_SYMBOL_SEQUENCE 0 CPHY_SEQ_USER_PREAMBLE NULL
# SET_CPHY_SYMBOL_SEQUENCE 0 CPHY_SEQ_END_PREAMBLE 3 3 3 3 3 3 3
# SET_CPHY_SYMBOL_SEQUENCE 0 CPHY_SEQ_POSTAMBLE 4 4 4 4 4 4 4
# SET_CPHY_SYMBOL_SEQUENCE 0 CPHY_SEQ_SYNC 3 4 4 4 4 4 3
# SET_CPHY_SYMBOL_SEQUENCE 0 CPHY_SEQ_SYNC1 3 4 4 4 4 4 3
# SET_CPHY_SYMBOL_SEQUENCE 0 CPHY_SEQ_SYNC2 3 4 4 4 4 4 3
# SET_CPHY_SYMBOL_SEQUENCE 0 CPHY_SEQ_SYNC3 3 4 4 4 4 4 3
# SET_CPHY_LANE_DEFAULT 1 0
# SET_CPHY_SYMBOL_SEQUENCE 1 CPHY_SEQ_START_PREAMBLE 3 3 3 3 3 3 3
# SET_CPHY_SYMBOL_SEQUENCE 1 CPHY_SEQ_USER_PREAMBLE NULL
# SET_CPHY_SYMBOL_SEQUENCE 1 CPHY_SEQ_END_PREAMBLE 3 3 3 3 3 3 3
# SET_CPHY_SYMBOL_SEQUENCE 1 CPHY_SEQ_POSTAMBLE 4 4 4 4 4 4 4
# SET_CPHY_SYMBOL_SEQUENCE 1 CPHY_SEQ_SYNC 3 4 4 4 4 4 3
# SET_CPHY_SYMBOL_SEQUENCE 1 CPHY_SEQ_SYNC1 3 4 4 4 4 4 3
# SET_CPHY_SYMBOL_SEQUENCE 1 CPHY_SEQ_SYNC2 3 4 4 4 4 4 3
# SET_CPHY_SYMBOL_SEQUENCE 1 CPHY_SEQ_SYNC3 3 4 4 4 4 4 3
```

```
# SET_CPHY_LANE_DEFAULT 2 0
# SET_CPHY_SYMBOL_SEQUENCE 2 CPHY_SEQ_START_PREAMBLE 3 3 3 3 3 3 3
# SET_CPHY_SYMBOL_SEQUENCE 2 CPHY_SEQ_USER_PREAMBLE NULL
# SET_CPHY_SYMBOL_SEQUENCE 2 CPHY_SEQ_END_PREAMBLE 3 3 3 3 3 3 3
# SET_CPHY_SYMBOL_SEQUENCE 2 CPHY_SEQ_POSTAMBLE 4 4 4 4 4 4 4
# SET_CPHY_SYMBOL_SEQUENCE 2 CPHY_SEQ_SYNC 3 4 4 4 4 4 3
# SET_CPHY_SYMBOL_SEQUENCE 2 CPHY_SEQ_SYNC1 3 4 4 4 4 4 3
# SET_CPHY_SYMBOL_SEQUENCE 2 CPHY_SEQ_SYNC2 3 4 4 4 4 4 3
# SET_CPHY_SYMBOL_SEQUENCE 2 CPHY_SEQ_SYNC3 3 4 4 4 4 4 3
# SET_CPHY_LANE_DEFAULT 3 0
# SET_CPHY_SYMBOL_SEQUENCE 3 CPHY_SEQ_START_PREAMBLE 3 3 3 3 3 3 3
# SET_CPHY_SYMBOL_SEQUENCE 3 CPHY_SEQ_USER_PREAMBLE NULL
# SET_CPHY_SYMBOL_SEQUENCE 3 CPHY_SEQ_END_PREAMBLE 3 3 3 3 3 3 3
# SET_CPHY_SYMBOL_SEQUENCE 3 CPHY_SEQ_POSTAMBLE 4 4 4 4 4 4 4
# SET_CPHY_SYMBOL_SEQUENCE 3 CPHY_SEQ_SYNC 3 4 4 4 4 4 3
# SET_CPHY_SYMBOL_SEQUENCE 3 CPHY_SEQ_SYNC1 3 4 4 4 4 4 3
# SET_CPHY_SYMBOL_SEQUENCE 3 CPHY_SEQ_SYNC2 3 4 4 4 4 4 3
# SET_CPHY_SYMBOL_SEQUENCE 3 CPHY_SEQ_SYNC3 3 4 4 4 4 4 3
# SET_CPHY_PARAMETER CPHY_PARAM_HS_PREPARE 50 0
# SET_CPHY_PARAMETER CPHY_PARAM_HS_EXIT 120 0
# SET_CPHY_PARAMETER CPHY_PARAM_TA_GO 0 4
# SET_CPHY_PARAMETER CPHY_PARAM_TA_GET 0 5
# SET_CPHY_PARAMETER CPHY_PARAM_WAKEUP 1000000 0
# SET_TGR_PRE_LENGTH 63
# SET_TGR_POST_LENGTH 31
# SET_ENABLE_LANE_DELAYS 0
```

5.2 RPCErrs Class Reference

```
namespace CPhyGenCtlRPC
{
    public class RPCErrs
    {
        public const int FAIL = -1;
        public const int UNKNOWN_CMD = -3;
        public const int NOT_LICENSED = -6;
        public const int ARGTYPE_MISMATCH = -7;

        public const int USB_OPEN_FAILED = -100;
        public const int DEVICE_NOT_PRESENT = -101;
        public const int USB_TIMEOUT = -102;
        public const int MAX_LEN_EXCEEDED = -103;
        public const int CANT_OPEN_FILE = -104;
        public const int FILE_DOESNT_EXIST = -105;
        public const int FILE_TOO_SHORT = -106;
        public const int IO_ERROR = -107;
        public const int INVALID_PARAM = -108;
        public const int BAD_RBF_FILE_LEN = -109;
        public const int BAD_MAGIC_NUMBER = -110;
        public const int DDRBW_FIFO_OVERFLOW = -111;
        public const int DDRBW_DID_NOT_COMPLETE = -112;
        public const int VERIFY_FAILED = -113;
        public const int ABORT = -114;
        public const int TIMEOUT = -115;
```

```
public const int CANT_ACQUIRE_SEMA = -116;
public const int NO_PROGSTATE_DEFINED = -117;
public const int BAD_SERIAL_NUMBER = -118;
public const int BAD_FIRMWARE_FILE_LEN = -119;
public const int BAD_SECURITY_FILE = -120;
public const int INVALID_FIRMWARE_FILE = -121;
public const int FLASH_VERIFY_ERROR = -122;
public const int CAPT_STATE_MISMATCH = -123;
public const int NO_CONNECTION = -124;
public const int REMOTE_PG_CALL_FAILED = -125;
public const int BAD_FRAME_PARAMETERS = -126;
public const int BAD_LINE_TIME = -127;
public const int INVALID_SYNC_LINE_CNT = -128;
public const int INVALID_ACTIVE_PIX_CNT = -129;
public const int INVALID_HEIGHT_OR_WIDTH = -130;
public const int CANT_CONFIGURE_P375 = -131;
public const int PG_IS_RUNNING = -132;
public const int INSUFFICIENT_PG_MEMORY = -133;
public const int THREAD_IS_BUSY = -134;
public const int INVALID_TIMING_PARAM = -136;
public const int INVALID_HSYNC_PARAM = -137;
public const int INVALID_HFRONTPORCH_PARAM = -138;
public const int INVALID_HBACKPORCH_PARAM = -139;
public const int AGGREGATE_HS_PKT_LANE_MISMATCH = -140;
public const int NO_BLANKING_ADJ = -141;
public const int HSA_TOO_SHORT = -142;
public const int HBP_TOO_SHORT = -143;
public const int HFP_TOO_SHORT = -144;
public const int MAX_PAYLOAD_EXCEEDED = -145;
public const int TEST_DID_NOT_COMPLETE = -146;
public const int SESSION_EXPIRED = -148;
public const int INVALID_VBACKPORCH_PARAM = -152;
public const int BAD_HS_MODE = -153;
public const int BAD_LP_REPL = -154;
public const int SEG_UNALIGNED = -155;
public const int INSERTION_CODE_NOT_FOUND = -161;
public const int INSERTION_CMD_TOO_LONG = -162;
public const int BTA_TIMEOUT = -163;
public const int CANT_LOCK_HS_PLL = -166;
public const int INVALID_HS_BITRATE = -169;
public const int CANT_RESTART = -170;
public const int PARSE_ERR = -172;
public const int UNKNOWN_MIPI_CMD = -173;
public const int NEED_START_EDIT_CMD = -174;
public const int CONTROL_IS_DISABLED = -175;
public const int UNEXPECTED_ARG_TYPE = -176;
public const int PAYLOAD_TOO_SHORT = -177;
public const int CMD_STANDARD_MISMATCH = -178;
public const int UNKNOWN_DCS_CMD = -180;
public const int CANT_PARTITION_FILE_CMD = -181;
public const int LP_DELAY_TOO_LONG = -182;
public const int INSERTION_BLOCK_TOO_LONG = -183;
public const int TOO_FEW_TOKENS = -191;
public const int VALUE_OUT_OF_RANGE = -192;
public const int OBJECT_NOT_FOUND = -193;
public const int USER_CANCELED = -194;
public const int BLANKING_SEGMENT_TOO_SHORT = -199;
```

```
        public const int BLANKING_SEGMENT_TOO_LONG = -205;
        public const int INVALID_VACTIVE_PARAM = -206;
        public const int INVALID_VFRONTPORCH_PARAM = -207;
        public const int UNEXPECTED_PG_COUNT = -208;

        public const int PORT_IS_UNAVAILABLE = -300;
        public const int CANT_PARSE_VALUE = -305;
        public const int NOT_BINARY_GEN_FILE = -310;

        public const int SILENT_ERROR = -999;

        public const int LOCAL_FAILURE = -5000;
    }
}
```

5.3 RPCCmds Class Reference

```
public class RPCCmds
{
    ////////////////////////////////////////////////
    // command management
    ////////////////////////////////////////////////

    public const int SEND_MIPI_CMD = 1;

    // Create a named command in the application
    // (Will delete an existing command by the same name if it exists)
    // (If sent while defining macro, only names the component
    // command being added to macro)
    // Uses same arguments as SEND_MIPI_CMD with one additional
    // argument (first argument): command name (string)
    public const int ADD_MIPI_CMD = 0x2;

    public const int COMPUTE_CRC = 3;

    public const int SEND_IMPAIRED_MIPI_CMD = 4;

    // arg1: Cmd name (string)
    // sends the (already defined) named command given its Cmd name
    public const int SEND_NAMED_CMD_FROM_NAME = 5;

    // arg1: Cmd index (integer)
    // sends the (already defined) named command given its Cmd index
    public const int SEND_NAMED_CMD_FROM_INDEX = 6;

    // Deletes a named command in the application
    // arg1: command name (string)
    public const int DELETE_NAMED_CMD = 7;

    // Deletes all named commands in the application
    // no arguments
    public const int DELETE_ALL_NAMED_CMDS = 8;

    // arg1 = 0 (disable), 1 (enable)
    public const int SET_SORT_CMDS = 9;
```

```
    // no arguments, returns number of named Cmds defined (integer)
    public const int GET_NAMED_CMD_COUNT = 10;

    // arg1: Cmd index (integer)
    // returns the Cmd name associated with the Cmd index (string)
    public const int GET_NAMED_CMD_NAME = 11;

    // arg1 = command name (string)
    public const int SELECT_NAMED_CMD = 12;

    ////////////////////////////////////////////////
    // instrument configuration commands
    ////////////////////////////////////////////////

    // arg1 = mode (DT_HS, DT_LP, DT_DEFAULT)
    public const int SET_DT_MODE = 0x101;

    // arg1 = LP_FREQ (float, Hz, 0.2 - 30 MHz)
    public const int SET_LP_FREQ = 0x102;

    // arg1 = HS_SYM_RATE (float, sym/sec, 23.44 to 2600 Msps)
    public const int SET_HS_SYM_RATE = 0x103;

    // arg1 = lane count (int, 1-4)
    public const int SET_LANE_CNT = 0x104;

    // arg1 = 0 (disable), 1 (enable)
    public const int SET_FORCE_TEST_PATTERN = 0x105;
    public const int SET_ENABLE_FORCE_TEST_PATTERN = 0x105; // depr

    // arg1 = 0 (disable), 1 (enable)
    public const int SET_ALL_HS_VOLT_COMMON = 0x106;
    public const int SET_ENABLE_ALL_HS_VOLT_COMMON = 0x106; // depr

    // arg1 = channel (int, 0-3)
    // arg2 = voltage (float, volts, -0.6 to 1.2 V)
    public const int SET_HS_LOW_VOLT = 0x107;
    public const int SET_HS_HIGH_VOLT = 0x108;

    // arg1 = voltage (float, volts, -0.28 to 1.8 V)
    public const int SET_LP_LOW_VOLT = 0x109;
    public const int SET_LP_HIGH_VOLT = 0x10a;

    // arg1 = lane (int, 0-3)
    // arg2 = delay (float, sec, 0-15 ns)
    // (lane delays must be enabled in CPhy timing)
    // (lane delay will be quantized based on HSFreq)
    public const int SET_LANE_DELAY = 0x10b;

    // arg1 = voltage (float, volts, 0.0 to 1.0 V)
    public const int SET_LP_LOW_CONT_THRESH = 0x10d;
    public const int SET_LP_HIGH_CONT_THRESH = 0x10e;

    // arg1 = time (float, sec, 0.1 to 10000 us)
    public const int SET_BTA_WAIT_TIME = 0x10f;

    // arg1 = time (float, sec, 0.1 us to 80000 us)
```

```
public const int SET_TRIG_PULSE_WIDTH = 0x110;

// arg1 = lane map
// laneMap[3..0] source data lane to use for phy lane 0
// laneMap[7..4] source data lane to use for phy lane 1
// laneMap[11..8] source data lane to use for phy lane 2
// laneMap[15..12] source data lane to use for phy lane 3
// Example, unity mapping is: 3210h
public const int SET_LANE_MAP = 0x111;

// Asserts TrigOut on instrument
//
// NOTE: this is for immediate probe trigger control;
// (use ASSERT_TRIG command to set TrigOut from MIPI stream)
// arg1 = trigger command: 0 (low), 1 (high),
//                         2 (toggle), 3 (pulse)
// Use SET_TRIG_PULSE_WIDTH to set the pulse width
public const int FORCE_TRIG = 0x112;

// no arguments
// bracket any instrument configuration calls with
// START_EDIT_CONFIG/END_EDIT_CONFIG
public const int START_EDIT_CONFIG = 0x160;

// no arguments
// initializes instrument with new configuration settings
public const int END_EDIT_CONFIG = 0x161;

////////////////////////////////////////////////
// timing configuration commands
////////////////////////////////////////////////

// arg1 = pixel count (int)
public const int SET_TIMING_HSYNC = 0x181;
public const int SET_TIMING_HBPORCH = 0x182;
public const int SET_TIMING_HFPORCH = 0x183;
public const int SET_TIMING_HACTIVE = 0x184;

// arg1 = line count (int)
public const int SET_TIMING_VSYNC = 0x185;
public const int SET_TIMING_VBPORCH = 0x186;
public const int SET_TIMING_VFPORCH = 0x187;
public const int SET_TIMING_VACTIVE = 0x188;

// arg1 = line time (float, sec)
public const int SET_TIMING_LINE_TIME = 0x189;

// arg1 = pixel clock frequency (float, Hz)
public const int SET_TIMING_PIX_CLK = 0x18a;

// arg1 = frame rate (float, Hz)
public const int SET_TIMING_FRAME_RATE = 0x18b;

// arg1 = 0 (disable), 1 (enable)
public const int SET_TIMING_ENABLE_DSI_PULSE_MODE = 0x18c;
public const int SET_TIMING_ENABLE_DSI_BURST_MODE = 0x18d;
public const int SET_TIMING_ENABLE_CSI_LSLE_MODE = 0x18e;
```

```
public const int SET_TIMING_ENABLE_CSI_FRAME_NUMBERING = 0x18f;
public const int SET_TIMING_ENABLE_CSI_LINE_NUMBERING = 0x190;

// arg1 = blanking mode
//   (AUTO_BLANK_MODE, HS_BLANK_MODE, LP11_BLANK_MODE)
public const int SET_TIMING_HSYNC_BLANKING_MODE = 0x191;
public const int SET_TIMING_HBPORCH_BLANKING_MODE = 0x192;
public const int SET_TIMING_HFPORCH_BLANKING_MODE = 0x193;
public const int SET_TIMING_VERTICAL_BLANKING_MODE = 0x194;

// arg1 = 0 (disable), 1 (enable)
public const int SET_TIMING_TOP_FIELD_FIRST = 0x195;

// arg1 = timing configuration name (string)
public const int SELECT_TIMING_CONFIG = 0x196;
public const int SET_TIMING_CONFIG = 0x196; // depr

// arg1 = timing configuration name (string)
public const int SAVE_TIMING_CONFIG = 0x197;

// arg1 = timing configuration name (string)
public const int DELETE_TIMING_CONFIG = 0x198;

// no arguments
public const int DELETE_ALL_TIMING_CONFIGS = 0x199;

// arg1 = 0 (disable), 1 (enable)
public const int SET_SORT_TIMING_CONFIGS = 0x19a;

// arg1 = HS Sym Rate (float, syms / sec)
public const int SET_TIMING_HS_SYM_RATE = 0x19b;

// arg1 = 3D parameter number (int, PARAM_3D_x defines)
// arg2 = 3D parameter value (int)
public const int SET_3D_PARAMETER = 0x19d;

// arg1 = timing master (int, MASTER_x defines)
public const int SET_TIMING_MASTER = 0x19e;

// arg1 = 0 (disable), 1 (enable)
public const int SET_ADVANCE_FRAME_ON_EXTERNAL_EVENT = 0x19f;

// arg1 = DSC configuration name (string)
public const int SELECT_DSC_CONFIG = 0x1a0;

// arg1 = DSC configuration name (string)
public const int SAVE_DSC_CONFIG = 0x1a1;

// arg1 = DSC configuration name (string)
public const int DELETE_DSC_CONFIG = 0x1a2;

// no arguments
public const int DELETE_ALL_DSC_CONFIGS = 0x1a3;

// arg1 = 0 (disable), 1 (enable)
public const int SET_SORT_DSC_CONFIGS = 0x1a4;
```

```
    // arg1 = field value (int)
    public const int SET_DSC_SLICE_WIDTH = 0x1a5;
    public const int SET_DSC_SLICE_HEIGHT = 0x1a6;
    public const int SET_DSC_ENABLE_422 = 0x1a7;
    public const int SET_DSC_USE_YUV_INPUT = 0x1a8;
    public const int SET_DSC_USE_BLOCK_PREDICTION = 0x1a9;
    public const int SET_DSC_ONE_CHUNK_PER_PACKET = 0x1aa;

    // arg1 = configuration filename (string)
    public const int SET_DSC_CONFIG_FILENAME = 0x1ab;

    // arg1 = source filename (string)
    // arg2 = DSC filename (string)
    public const int DSC_COMPRESS_FILE = 0x1ac;

    // arg1 = DSC filename (string)
    // arg2 = source filename (string)
    public const int DSC_UNCOMPRESS_FILE = 0x1ad;

    ////////////////////////////////////////////////
    // CPhy Timing configuration commands
    ////////////////////////////////////////////////

    // arg1 = lane index (int, 0-3)
    // arg2 = sequence index (CPHY_SEQ_XX defines)
    // arg3 = symbol seqeunce (byte array)
    public const int SET_CPHY_SYMBOL_SEQUENCE = 0x200;

    // arg1 = parameter (CPHY_PARAM_* definition)
    // arg2 = nsValue (int)
    // arg3 = TLPXValue (int)
    public const int SET_CPHY_PARAMETER = 0x201;

    // arg2 = 0 (disable), 1 (enable)
    public const int SET_CPHY_ALL_LANES_COMMON = 0x202;

    // arg1 = lane index (int, 0-3)
    // arg2 = 0 (disable), 1 (enable)
    public const int SET_CPHY_LANE_DEFAULT = 0x203;

    // arg1 = TGR Pre Length (int, 0-127)
    public const int SET_TGR_PRE_LENGTH = 0x204;

    // arg1 = TGR Post Length (int, 0-127)
    public const int SET_TGR_POST_LENGTH = 0x205;

    // arg1 = 0 (disable), 1 (enable)
    public const int SET_ENABLE_LANE_DELAYS = 0x206;

    // arg1 = timing configuration name (string)
    public const int SELECT_CPHY_CONFIG = 0x207;
    public const int SET_CPHY_CONFIG = 0x207; // depr

    // arg1 = timing configuration name (string)
    public const int SAVE_CPHY_CONFIG = 0x208;

    // arg1 = timing configuration name (string)
```

```
public const int DELETE_CPHY_CONFIG = 0x209;

// no arguments
public const int DELETE_ALL_CPHY_CONFIGS = 0x20a;

// arg1 = 0 (disable), 1 (enable)
public const int SET_SORT_CPHY_CONFIGS = 0x20b;

////////////////////////////////////////////////////
// macro defintion
////////////////////////////////////////////////////

// no arguments; puts application in macro mode
// any (non-video) Cmds sent will append to current macro
public const int START_MACRO = 0x280;

// Create a named macro in the application
// (Deletes an existing macro by the same name if it exists)
// arg1: macro name (string)
public const int END_MACRO = 0x281;

// Measures the current macro (must still be in macro_mode).
// Macro is subsequently discarded
public const int MEASURE_MACRO = 0x282;

// arg1: zero-based component command index (int)
// arg2: component command name (string)
public const int RENAME_MACRO_COMPONENT_COMMAND = 0x283;

// Ends and sends the current macro (must still be in macro_mode).
// Macro is discarded after sending.
public const int SEND_MACRO = 0x284;

////////////////////////////////////////////////////
// status commands
////////////////////////////////////////////////////

// no arguments; returns 0 (not connected), 1 (connected)
public const int IS_INST_CONNECTED = 0x300;

// no arguments; returns 0 (not running), 1 (running)
public const int IS_PROGRAM_RUNNING = 0x301;

// no arguments; returns contention mask
// mask consists of six bits (CTN_XX bits)
public const int GET_LP_CONTENTION = 0x302;

// no arguments
public const int RESET_LP_CONTENTION = 0x303;

// no arguments
public const int GET_DUT_RESPONSE = 0x304;

// arg1 = fileName: name of file to save DUT response (as text)
// arg2 = appendFlag (integer): set to non-zero to append to file
// arg3 = commentString: comment to save with response
//        set to "" for no comment
```

```
public const int SAVE_DUT_RESPONSE = 0x305;

// no arguments, returns inputState[1..0] (integer)
public const int GET_INST_GP_INPUT_STATE = 0x306;

// arg1: fileName: name of file to save PG event state as int
// arg2: appendFlag (integer): set to non-zero to append to file
// arg3: commentString: comment to save with response
//       set to "" for no comment
public const int SAVE_INST_GP_INPUT_STATE = 0x307;

////////////////////////////////////////////////
// frame buffer commands
////////////////////////////////////////////////

// arg1: buffer number, (integer, 0-3)
// arg2: dataType specifying video format (int)
// arg3: file name (string)
// Loads the frame from the given file, converts to the given
// video format, and stores into the given frame buffer number
// (use DEALLOC_FRAME to deallocate when done)
public const int LOAD_FRAME = 0x380;

// arg1: buffer number, (integer, 0-3)
// Deallocates the frame stored in the given frame buffer
public const int DEALLOC_FRAME = 0x381;

////////////////////////////////////////////////
// WriteMemory configuration commands
////////////////////////////////////////////////

// arg1 = maximum number of bytes per DCS WriteMemory partition
public const int SET_WM_PARTITION_LENGTH = 0x3a0;

// arg1 = LP11 time between DCS WriteMemory partitions (float, sec)
public const int SET_WM_PARTITION_INTERVAL = 0x3a1;

// arg1 = DCS WriteMemory image decode format (FMT_xxx defines)
public const int SET_WM_IMAGE_DECODE_FORMAT = 0x3a2;
public const int SET_WM_BMP_DECODE_FORMAT = 0x3a2; // depr

// NO LONGER SUPPORTED
// arg1 = line length in pixels used to decode binary files
// SET_WM_BINARY_FILE_LINE_LENGTH = 0x3a3

////////////////////////////////////////////////
// RPC commands
////////////////////////////////////////////////

// arg1: Cmd name (string)
// returns the RPC string for the given Cmd name (string)
public const int GET_RPC_CMD_STRING_FROM_NAME = 0x400;

// arg1: Cmd index (integer)
// returns the RPC string for the given Cmd index (string)
public const int GET_RPC_CMD_STRING_FROM_INDEX = 0x401;
```

```
    // arg1: RPC command string (string)
    // executes the given RPC command string
    // the string may have multiple RPC commands separated by
    // <CR><LF> or <CR> (all rules of RPC script files can be followed)
    public const int EXECUTE_RPC_CMD_STRING = 0x402;

    ////////////////////////////////////////////////
    // options
    ////////////////////////////////////////////////

    // arg1 = option (OPT_XXX defines)
    // arg2 = 0 (disable), 1 (enable)
    public const int SET_OPTION = 0x480;

    // arg1 = standard (STD_XXX defines)
    public const int SET_MIPI_STANDARD = 0x481;

    // Sets the Bayer encode type for CSI RAW images
    // (when Bayer is enabled)
    // arg1: string representing bayer format
    // ("GRBG", "RGGB", "BGGR", "GBRG")
    public const int SET_BAYER_ENCODE_TYPE = 0x482;

    ////////////////////////////////////////////////
    // other commands
    ////////////////////////////////////////////////

    // arg1 = hostname (string)
    public const int INST_CONNECT = 0x500;

    // no arguments
    public const int INST_DISCONNECT = 0x501;


    // arg1 = user message string enclosed in double-quotes
    // displays message box to user with OK button (no timeout)
    public const int USER_WAIT = 0x502;

    // no arguments
    public const int STOP_PG = 0x503;
    public const int RESTART_PG = 0x504;

    // Loads button/command config from the a cfg file.
    // arg1: configuration file name (string)
    public const int LOAD_CONFIG = 0x505;

    // Saves the current button/command config to a cfg file
    // arg1: configuration file name (string)
    public const int SAVE_CONFIG = 0x506;

    // Assigns command to a button in the application
    // (Will replace an existing button assign, use "" to unassign)
    // (use button page = -1 for all pages, e.g. for group unassign)
    // (use button number = -1 for all buttons on page)
    // arg1: button page (0-3, int)
    // arg2: button number in page (0-29, int)
    // arg3: command name (string)
```

```
    // arg4: tooltip (string)
    public const int ASSIGN_BUTTON = 0x507;

    // Writes current state as a RPC commands to script file
    // (overwrites existing file)
    // arg1: script file name (string)
    public const int WRITE_CURRENT_STATE = 0x508;
}
```

5.4 RPCDefs Class Reference

```
public class RPCDefs
{
    // DT modes
    public const int DT_DEFAULT = 0;
    public const int DT_LP = 1;
    public const int DT_HS = 2;

    // MIPI standards
    public const int STD_DSI = 0;
    public const int STD_CSI = 1;

    // SET_TIMING_MASTER arg1 values
    public const int MASTER_LINE_TIME = 0;
    public const int MASTER_PIX_CLK = 1;
    public const int MASTER_FRAME_RATE = 2;
    public const int MASTER_HS_BIT_RATE = 3;

    // SET_3D_PARAMETER arg1 values
    public const int PARAM_3D_ENABLE = 0;
    public const int PARAM_3D_STREAM_MODE = 1;
    public const int PARAM_3D_DISPLAY_ORIENTATION = 2;
    public const int PARAM_3D_LR_ORDER = 3;
    public const int PARAM_3D_FORMAT = 4;
    public const int PARAM_3D_VSYNC = 5;
    public const int PARAM_3D_INTER_FRAME_VACT = 6;
    public const int PARAM_3D_COUNT = 7;

    // 3D_STREAM_MODE and 3D_DISPLAY_ORIENTATION
    public const int PORTRAIT_MODE = 0;
    public const int LANDSCAPE_MODE = 1;

    // 3D_LR_ORDER
    public const int LEFT_FIRST = 0;
    public const int RIGHT_FIRST = 1;

    // 3D_FORMAT
    public const int FORMAT_3D_LINE = 0;
    public const int FORMAT_3D_FRAME = 1;
    public const int FORAMT_3D_PIXEL = 2;

    // options
    public const int OPT_LOOP_COMMANDS = 1;
    public const int OPT_WAIT_ON_EXT_EVENT_TO_START = 2;
    public const int OPT_DISABLE_EVENT_TIMEOUT = 3;
    public const int OPT_DISABLE_CMD_TIMEOUT = 3; // deprecated
```

```
public const int OPT_ENABLE_CMD_INSERTION = 4;
public const int OPT_DISCARD_DURING_INSERTION = 5;
public const int OPT_ENABLE_EOT_PKTS = 6;
public const int OPT_ENCODE_RAW_FORMAT_AS_BAYER = 8;
public const int OPT_ALWAYS_USE_PREDICTOR1_FOR_COMPRESSION = 9;
public const int OPT_ENABLE_WM_PARTITIONING = 10;
public const int OPT_ENABLE_VIDEO_MODE_IN_MACROS = 11;
public const int OPT_SEND_SINGLE_PKT_PER_HS_BURST = 12;
public const int OPT_ALLOW_IMAGE_RESCALING = 13;
public const int OPT_WM_QUANTIZE_TO_LINE_LENGTH_MULTIPLE = 14;
public const int OPT_ENABLE_HOLD_LAST_SYMBOL_TEST_MODE = 15;
public const int OPT_SEND_PPS_WITH_COMPRESSED_VIDEO = 16;
public const int OPT_ENABLE_DSI_SCRAMBLING = 17;

// DCS Write Memory BMP decode formats
public const int FMT_RGB565_16 = 1;
public const int FMT_RGB666_18 = 2;
public const int FMT_RGB666_24 = 3;
public const int FMT_RGB888_24 = 4;
public const int FMT_YCBCR_420_12 = 11;
public const int FMT_YCBCR_422_16 = 13;
public const int FMT_RGB101010_30 = 23;
public const int FMT_RGB121212_36 = 24;
public const int FMT_YCBCR_422_LOOSE_20 = 25;
public const int FMT_YCBCR_422_24 = 26;
public const int FMT_DSC_COMPRESSED = 33;

// contention mask bits
public const int CTN_TX0A = (1 << 0);
public const int CTN_TX0B = (1 << 1);
public const int CTN_TX0C = (1 << 2);
public const int CTN_TX1A = (1 << 3);
public const int CTN_TX1B = (1 << 4);
public const int CTN_TX1C = (1 << 5);

// MIPI standard cmd masks
public const int DSI_CMD_MASK = 0x400;
public const int CSI_CMD_MASK = 0x500;

// DPhy parameters
public const int CPHY_PARAM_HS_PREPARE = 0;
public const int CPHY_PARAM_HS_EXIT = 1;
public const int CPHY_PARAM_TA_GO = 2;
public const int CPHY_PARAM_TA_GET = 3;
public const int CPHY_PARAM_WAKEUP = 4;

// CPhy sequence parameters
public const int CPHY_SEQ_START_PREAMBLE = 1;
public const int CPHY_SEQ_USER_PREAMBLE = 2;
public const int CPHY_SEQ_END_PREAMBLE = 3;
public const int CPHY_SEQ_POSTAMBLE = 4;
public const int CPHY_SEQ_SYNC = 5;
public const int CPHY_SEQ_SYNC2 = 6;
public const int CPHY_SEQ_SYNC1 = 7;
public const int CPHY_SEQ_SYNC3 = 8;

// Blanking modes
```

```
public const int AUTO_BLANK_MODE = 0;
public const int LP11_BLANK_MODE = 1;
public const int HS_BLANK_MODE = 2;

// trigger commands for ASSERT_PROBE_TRIG and FORCE_PROBE_TRIG
public const int TRIG_CMD_LOW = 1;
public const int TRIG_CMD_HIGH = 2;
public const int TRIG_CMD_TOGGLE = 3;
public const int TRIG_CMD_PULSE = 4;


////////////////////////////////////////////////
// DSI commands
//
// Note: the command values defined here are applicable when
// using these command codes in RPC calls via the
// CPhyGenCtlRPCClient DLL.  All DSI commands will have 0x400
// and all CSI commands will have 0x500 added to them
// (to make them unique across standards).
////////////////////////////////////////////////

// 0 argument commands
public const int VSYNC_START = 0x401;
public const int VSYNC_END = 0x411;
public const int HSYNC_START = 0x421;
public const int HSYNC_END = 0x431;
public const int EOT_PKT = 0x408;
public const int COLOR_MODE_OFF = 0x402;
public const int COLOR_MODE_ON = 0x412;
public const int SHUT_DOWN_PERIPHERAL = 0x422;
public const int TURN_ON_PERIPHERAL = 0x432;
public const int DCS_READ = 0x406;

// arg1 = ParamCnt (0, 1, 2)
// arg2 = Params[15:0]
public const int GENERIC_SHORT_WRITE = 0x403;

// arg1 = ParamCnt (0, 1, 2)
// arg2 = Params[15:0]
public const int GENERIC_READ = 0x404;

// arg1 = ParamCnt (0, 1)
// arg2 = Params[15:0]
// (see DCS short write commands)
public const int DCS_SHORT_WRITE = 0x405;

// arg1 = MaxReturnPktSize[15:0]
public const int SET_MAX_RETURN_PKT_SIZE = 0x437;

// arg1 = Mode[15:0]
public const int SCRAMBLING_MODE = 0x427;

// arg1 = WriteLen
public const int DSI_NULL_PKT = 0x409;

// arg1 = WriteLen
public const int DSI_BLANKING_PKT = 0x419;
```

```
// option1: non-zero length payload array
// option2: filename
public const int GENERIC_LONG_WRITE = 0x429;

// (see DCS long write commands)
public const int DCS_LONG_WRITE = 0x439;

// arg1 = FrameCnt
// arg2 = Interlaced (0 == not interlaced,
//        1 == interlaced; only applies to YCBCR formats)
// option1: non-zero length payload array
// (length = FrameCnt * HTotal * HVTotal * bytesPerPix)
// option2: filename
public const int PACKED_PIXEL_STREAM_565 = 0x40e;
public const int PACKED_PIXEL_STREAM_666 = 0x41e;
public const int LOOSE_PIXEL_STREAM_666 = 0x42e;
public const int PACKED_PIXEL_STREAM_888 = 0x43e;
public const int LOOSE_PIXEL_STREAM_20_YCBCR_422 = 0x40c;
public const int PACKED_PIXEL_STREAM_24_YCBCR_422 = 0x41c;
public const int PACKED_PIXEL_STREAM_16_YCBCR_422 = 0x42c;
public const int PACKED_PIXEL_STREAM_101010 = 0x40d;
public const int PACKED_PIXEL_STREAM_121212 = 0x41d;
public const int PACKED_PIXEL_STREAM_12_YCBCR_420 = 0x43d;
public const int COMPRESSED_PIXEL_STREAM = 0x40b;

// arg1 = Data 0
// arg2 = Data 1
public const int COMPRESSION_MODE = 0x407;

// option1: non-zero length payload array
// option2: non-empty filename
public const int PICTURE_PARAMETER_SET = 0x40a;

// No arguments
public const int EXECUTE_QUEUE = 0x416;

////////////////////////////////////////////////
// CSI commands
//
// Note: the command values defined here are applicable when
// using these command codes in RPC calls via the
// CPhyGenCtlRPCClient DLL.  All DSI commands will have 0x400
// and all CSI commands will have 0x500 added to them
// (to make them unique across standards).
////////////////////////////////////////////////

// arg1 = FrameNum[15:0]
public const int FRAME_START = 0x500;
public const int FRAME_END = 0x501;

// arg1 = LineNum[15:0]
public const int LINE_START = 0x502;
public const int LINE_END = 0x503;

// arg1 = Value[15:0]
public const int GENERIC_SHORT_PKT1 = 0x508;
```

```
public const int GENERIC_SHORT_PKT2 = 0x509;
public const int GENERIC_SHORT_PKT3 = 0x50a;
public const int GENERIC_SHORT_PKT4 = 0x50b;
public const int GENERIC_SHORT_PKT5 = 0x50c;
public const int GENERIC_SHORT_PKT6 = 0x50d;
public const int GENERIC_SHORT_PKT7 = 0x50e;
public const int GENERIC_SHORT_PKT8 = 0x50f;

// arg1 = WriteLen
public const int CSI_NULL_PKT = 0x510;
public const int CSI_BLANKING_PKT = 0x511;

// option1: non-zero length payload array
// option2: filename
public const int LONG_PKT = 0x512;

// arg1 = FrameCnt
// option1: non-zero length payload array
// (length = FrameCnt * HTotal * HVTotal * bytesPerPix)
// option2: filename
//
// A special filename of "USERFRAME<n>" can be used to reference
// internal frame buffers loaded with the LOAD_FRAME command, where
// <n> is 0 to 3.  When "USERFRAME<n>" is used as the filename,
// arg1 = byte offset in frame buffer
// arg2 = byte length of packet
public const int PIXEL_STREAM_YUV420L_8 = 0x51A;
public const int PIXEL_STREAM_YUV422_8 = 0x51E;
public const int PIXEL_STREAM_YUV422_10 = 0x51F;
public const int PIXEL_STREAM_RGB444 = 0x520;
public const int PIXEL_STREAM_RGB555 = 0x521;
public const int PIXEL_STREAM_RGB565 = 0x522;
public const int PIXEL_STREAM_RGB666 = 0x523;
public const int PIXEL_STREAM_RGB888 = 0x524;
public const int PIXEL_STREAM_RAW6 = 0x528;
public const int PIXEL_STREAM_RAW7 = 0x529;
public const int PIXEL_STREAM_RAW8 = 0x52A;
public const int PIXEL_STREAM_RAW10 = 0x52B;
public const int PIXEL_STREAM_RAW12 = 0x52C;
public const int PIXEL_STREAM_RAW14 = 0x52D;

// option1: non-zero length payload array
// option2: filename
public const int USER_8BIT_TYPE1 = 0x530;
public const int USER_8BIT_TYPE2 = 0x531;
public const int USER_8BIT_TYPE3 = 0x532;
public const int USER_8BIT_TYPE4 = 0x533;
public const int USER_8BIT_TYPE5 = 0x534;
public const int USER_8BIT_TYPE6 = 0x535;
public const int USER_8BIT_TYPE7 = 0x536;
public const int USER_8BIT_TYPE8 = 0x537;

// arg1 = FrameCnt
// arg2 = User DataType
// option1: non-zero length payload array
// option2: filename
public const int RAW_COMPRESSED_10_8_10 = 0x5f4;
```

```
public const int RAW_COMPRESSED_10_7_10 = 0x5f5;
public const int RAW_COMPRESSED_10_6_10 = 0x5f6;
public const int RAW_COMPRESSED_12_8_12 = 0x5f7;
public const int RAW_COMPRESSED_12_7_12 = 0x5f8;
public const int RAW_COMPRESSED_12_6_12 = 0x5f9;

////////////////////////////////////////////////
// DPHY and generic commands
////////////////////////////////////////////////

// no arguments
public const int SEND_BTA = 0x102;
public const int RESET_TRIGGER = 0x104;
public const int TRIGGER1 = 0x105;
public const int TRIGGER2 = 0x106;
public const int TRIGGER3 = 0x107;

// arg1 = CmdByte[7:0]
public const int ESCAPE = 0x108;

// arg1 = delay in ns
// arg2 = LPValue[15:0] (4 lanes, 4-bits used for LP State)
public const int LP_DELAY = 0x109;

// no arguments
public const int ULPS = 0x10a;

// arg1 = seed
public const int PRBS9_SEQ = 0x10b;
public const int PRBS11_SEQ = 0x10c;
public const int PRBS18_SEQ = 0x10d;

public const int WAIT_FOR_BTA = 0x10e;

// waits for input event
// arg1 = LPValue[15:0] (4 lanes, 4-bits used for LP State)
public const int WAIT_EXT_EVENT = 0x110;

// Defines a command insertion point when used in looping macro
// arg1 = delay in ns
// arg2 = LPValue[15:0] (4 lanes, 4-bits used for LP State)
// arg3 = discard flag
//        (0 == suspend program while inserting;
//         1 == discard input while inserting)
public const int CMD_INSERTION_POINT = 0x112;

// Asserts TrigOut on the instrument
// NOTE: this is for probe trigger control during the command
// (use FORCE_TRIG command to affect TrigOut immediately)
// arg1 = delay in ns
// arg2 = LPValue[15:0] (4 lanes, 4-bits used for LP State)
// arg3 = trigger command: 0 (low voltage), 1 (high voltage),
//                         2 (toggle voltage), 3 (pulse voltage)
// Note: Toggle command starts from current voltage
// Note: Pulse command starts from current voltage
// Use SET_TRIG_DURATION command to set the pulse width
public const int ASSERT_TRIG = 0x113;
```

```
// Asserts LP state onto the bus until the specified position
// in the current macro.  The delay parameter is interpreted
// relative to the "zero position" as indicated by the last call
// (in the macro) of MARK_ZERO_POS.  If there is no last
// call to MARK_ZERO_POS, the start of the macro is used.
//
// arg1 = delay in ns
// arg2 = LPValue[15:0] (4 lanes, 4-bits used for LP State)
public const int LP_DELAY_TO_POS = 0x114;

// no arguments
// Records the current position when compiling a macro to
// use as a time reference.  (Subsequent calls to
// LP_DELAY_TO_POS use this "zero position" as the reference
// to allow exact timing for constructing video lines).
public const int MARK_ZERO_POS = 0x115;

// arg1 = LoopCnt (int)
// Params[] contained in non-zero length payload array
public const int TGR_DATA_SEQ = 0x119;
public const int TGR_SYM_SEQ = 0x11a;
public const int TGR_STATE_SEQ = 0x11b;

// Same as LP_DELAY with UI units
// arg1 = duration (UI, int)
// arg2 = LPValue[9:0] (4 lanes, 2-bits used per LP State)
public const int LP_DELAY_UI = 0x11e;

// Same as LP_DELAY_TO_POS with UI units
// arg1 = duration (UI, int)
// arg2 = LPValue[9:0] (4 lanes, 2-bits used per LP State)
public const int LP_DELAY_TO_POS_UI = 0x11f;

// arg1 = loop iterations
public const int LOOP_START = 0x120;

// no arguments
public const int LOOP_END = 0x121;

// arg1 = ParamCnt (1, 2)
// arg2 = Parameter byte 1
// arg3 = Parameter byte 2 (if necessary)
public const int GENERIC_SHORT_READ_RESPONSE = 0x1f5;

// option1: non-zero length payload array
// option2: filename
public const int GENERIC_LONG_READ_RESPONSE = 0x1f6;

// arg1 = ParamCnt (1, 2)
// arg2 = Parameter byte 1
// arg3 = Parameter byte 2 (if necessary)
public const int DCS_SHORT_READ_RESPONSE = 0x1f7;

// option1: non-zero length payload array
// option2: filename
public const int DCS_LONG_READ_RESPONSE = 0x1f8;
```

```
        // arg1 = ErrorBits[15:0]
        public const int ACK_AND_ERROR_REPORT = 0x1f9;

        // arg1 = DataID
        // data block with command data
        public const int CUSTOM_LONG_COMMAND = 0x1fa;

        // option1: filename
        public const int FILE_COMMAND = 0x1fb;

        // arg1 = DataID
        // data block with command data
        public const int CUSTOM_COMMAND = 0x1fc;

        // arg1: filename
        public const int RPC_SCRIPT = 0x1fd;

        ////////////////////////////////////////////
        // DCS_SHORT_WRITE cmd types
        ////////////////////////////////////////////

        // 0 argument commands
        public const int ENTER_IDLE_MODE = 0x39;
        public const int ENTER_INVERT_MODE = 0x21;
        public const int ENTER_NORMAL_MODE = 0x13;
        public const int ENTER_PARTIAL_MODE = 0x12;
        public const int ENTER_SLEEP_MODE = 0x10;
        public const int EXIT_IDLE_MODE = 0x38;
        public const int EXIT_INVERT_MODE = 0x20;
        public const int EXIT_SLEEP_MODE = 0x11;
        public const int NOP = 0;
        public const int SET_DISPLAY_OFF = 0x28;
        public const int SET_DISPLAY_ON = 0x29;
        public const int SET_TEAR_OFF = 0x34;
        public const int SOFT_RESET = 0x01;

        // arg1 = Mode[7:0]
        public const int SET_ADDRESS_MODE = 0x36;

        // arg1 = 1, arg2 = Mode[7:0]
        public const int SET_3D_CONTROL = 0x3d;

        // arg1 = GammaCurve[7:0]
        public const int SET_GAMMA_CURVE = 0x26;

        // arg1 = Format[7:0]
        public const int SET_PIXEL_FORMAT = 0x3a;

        // arg1 = M[0]
        public const int SET_TEAR_ON = 0x35;

        // arg1 = Timing[7:0]
        public const int SET_VSYNC_TIMING = 0x40;

        // arg1 = Timing[7:0]
        public const int SET_VSYNC_TIMING = 0x40;
```

```
        // arg1 = CMB[15:0]
        public const int SET_CABC_MIN_BRIGHTNESS = 0x5e;

        // arg1 = DBV[15:0]
        public const int SET_DISPLAY_BRIGHTNESS = 0x51;

        // arg1 = Ctl[7:0]
        public const int WRITE_CONTROL_DISPLAY = 0x53;

        // arg1 = PS[7:0]
        public const int WRITE_POWER_SAVE = 0x55;

        ////////////////////////////////////////////
        // DCS_READ cmd types
        ////////////////////////////////////////////

        public const int GET_ADDRESS_MODE = 0x0b;
        public const int GET_3D_CONTROL = 0x3f;
        public const int GET_COMPRESSION_MODE = 0x03;
        public const int GET_BLUE_CHANNEL = 0x08;
        public const int GET_DIAGNOSTIC_RESULT = 0x0f;
        public const int GET_DISPLAY_MODE = 0x0d;
        public const int GET_GREEN_CHANNEL = 0x07;
        public const int GET_PIXEL_FORMAT = 0x0c;
        public const int GET_POWER_MODE = 0x0a;
        public const int GET_RED_CHANNEL = 0x06;
        public const int GET_SCANLINE = 0x45;
        public const int GET_SIGNAL_MODE = 0x0e;
        public const int READ_DDB_CONTINUE = 0xa8;
        public const int READ_DDB_START = 0xa1;
        public const int READ_MEMORY_CONTINUE = 0x3e;
        public const int READ_MEMORY_START = 0x2e;
        public const int READ_PPS_CONTINUE = 0xa9;
        public const int READ_PPS_START = 0xa2;
        public const int GET_CABC_MIN_BRIGHTNESS = 0x5f;
        public const int GET_CONTROL_DISPLAY = 0x54;
        public const int GET_DISPLAY_BRIGHTNESS = 0x52;
        public const int GET_ERROR_COUNT_ON_DSI = 0x05;
        public const int GET_IMAGE_CHECKSUM_CT = 0x15;
        public const int GET_IMAGE_CHECKSUM_RGB = 0x14;
        public const int GET_POWER_SAVE = 0x56;

        ////////////////////////////////////////////
        // DCS_LONG_WRITE cmd types
        ////////////////////////////////////////////

        // arg1 = VertScrollPtr[15:0]
        public const int SET_SCROLL_START = 0x37;

        // arg1 = Line[15:0]
        public const int SET_TEAR_SCANLINE = 0x44;

        // arg1 = StartColumn[15:0]
        // arg2 = EndColumn[15:0]
        public const int SET_COLUMN_ADDRESS = 0x2a;
```

```
    // arg1 = StartPage[15:0]
    // arg2 = EndPage[15:0]
    public const int SET_PAGE_ADDRESS = 0x40;

    // arg1 = StartColumn[15:0]
    // arg2 = EndColumn[15:0]
    public const int SET_PARTIAL_COLUMNS = 0x31;

    // arg1 = StartRow[15:0]
    // arg2 = EndRow[15:0]
    public const int SET_PARTIAL_ROWS = 0x30;

    // arg1 = TopFixedArea[15:0];
    // arg2 = VertScrollArea[15:0];
    // arg3 = BotFixedArea[15:0];
    public const int SET_SCROLL_AREA = 0x33;

    // option1: non-zero length payload array
    // option2: filename
    public const int WRITE_LUT = 0x2d;

    // arg1 = FileOffset
    // arg2 = WriteLen
    // option1: non-zero length payload array
    //          (FileOffset and WriteLen are ignored)
    // option2: filename
    public const int WRITE_MEMORY_CONTINUE = 0x3c;

    // arg1 = FileOffset
    // arg2 = WriteLen
    // option1: non-zero length payload array
    //          (FileOffset and WriteLen are ignored)
    // option2: filename
    public const int WRITE_MEMORY_START = 0x2c;
}
```