ASTRONOMICAL
RESEARCH
CAMERAS, INC.

V.I.R.U.S. API User Manual

Version 3.0

October 2011

Getting Started

The Virus API is an extension to the standard ARC API C++ libraries.

Development Tools

The API libraries were built using the following development environments.

Linux Applications and API Library: Standard CentOS 5.5 linux distribution using GCC.

Building an Application Using the API

- Include necessary **header** file locations:

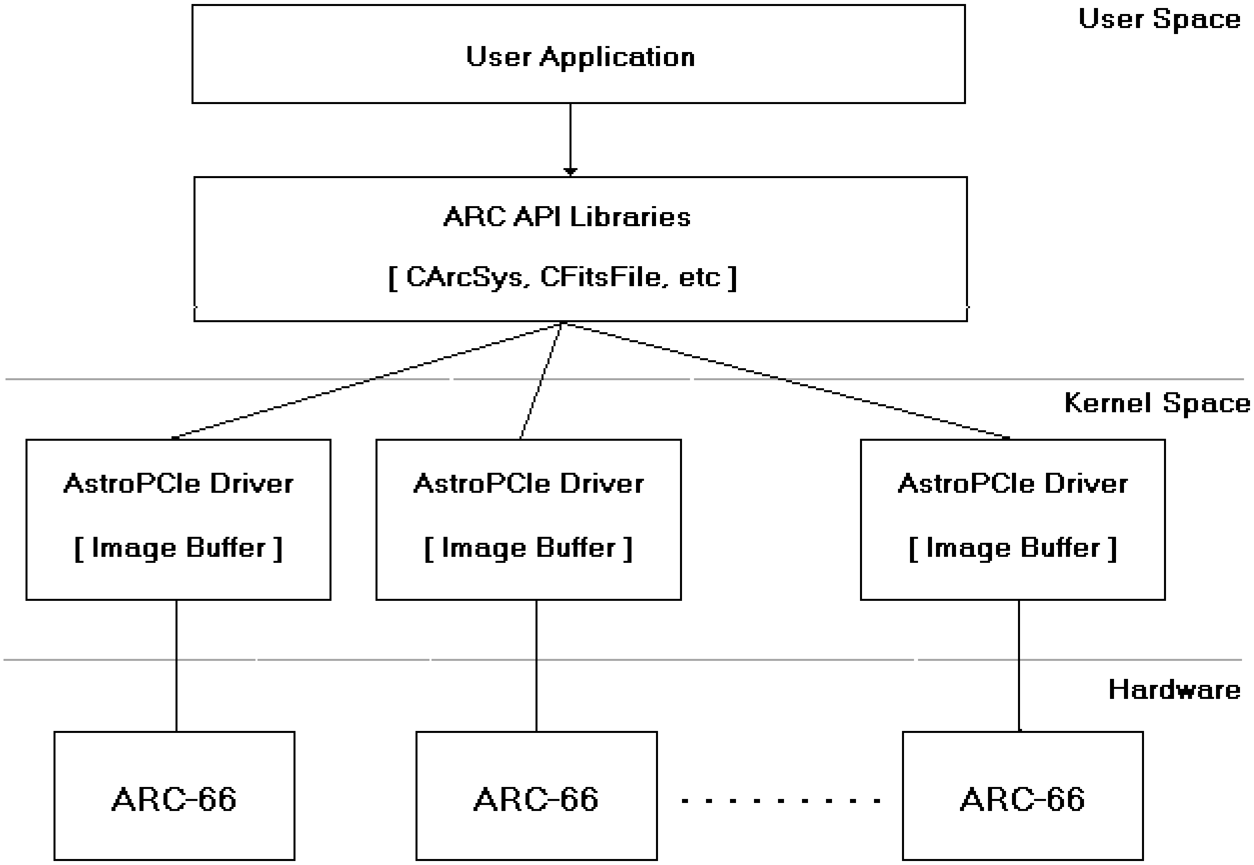
On Linux: /xxx/ARC_API/VIRUS/CArcSys/src
/xxx/ARC_API/3.0/CArcDevice

Where "xxx" is the path to the ARC_API folder. See *Header List* section for a full list of available headers.

- Include necessary **library** locations:

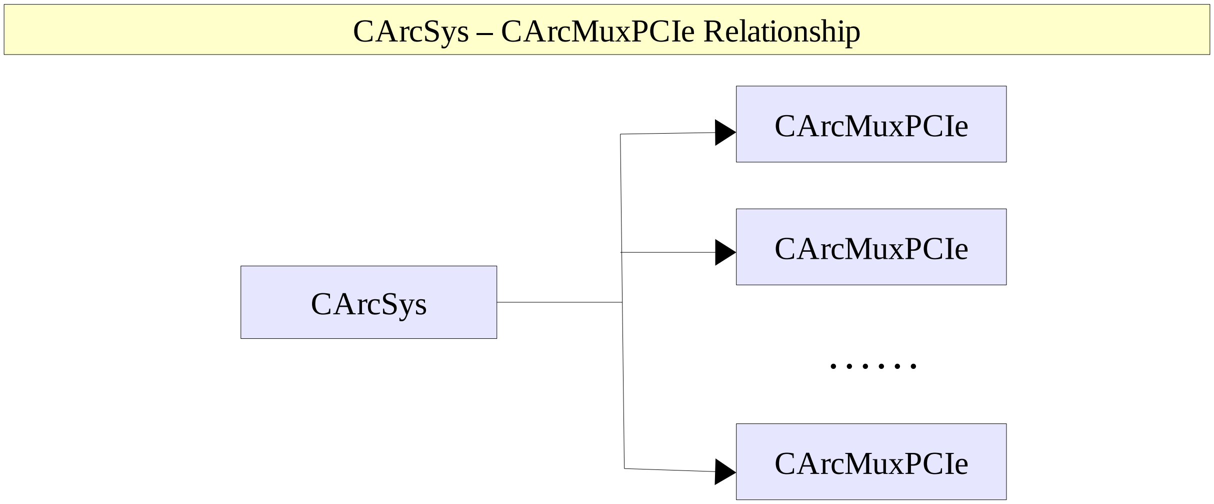
On Linux: /xxx/ARC_API/3.0/Release
/xxx/ARC_API/VIRUS/Release

Where "xxx" is the path to the ARC_API folder.

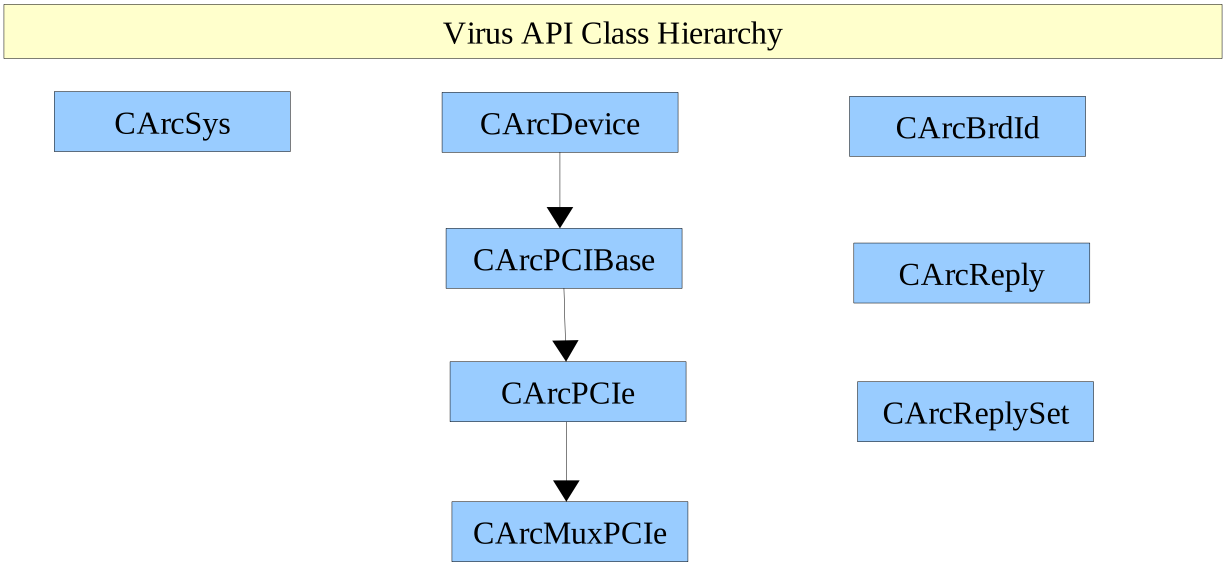


Class Relationships

All device access is accomplished through the implementation of a single class, CArcSys, which manages all PCIe boards, which are implemented in the CArcMuxPCIe class. The basic method call structure is similar to the standard ARC API class interface, except the board id is now a class that can represent a single board, a range of boards, or all boards. Similarly, the reply for any command is now a set of reply values. The CArcSys/CArcMuxPCIe class relationship is shown in the diagram below:



The basic class hierarchy is shown in the diagram below:



CArcBrdId Class

(Include Header File: CArcBrdId.h)

All CArcSys methods that communicate with a controller require this class as a parameter. The CArcBrdId class is used to represent a single or range of controller boards, each of which has its own unique id. The class encapsulates two data values that are the start and end board ids. If the two values are the same, then the class represents a single board, otherwise it represents a range of boards. Board ids are between 8 and 255 inclusive. The class also defines a static member BROADCAST_ALL that represents all boards in the system. Access to the board ids is provided through two methods: *int One()* and *int Two()*. *One()* provides the start board id; *Two()* provides the end board id.

Example: To send test data link ('TDL') to controllers with ids 0x10 to 0x20:

```
CArcSys cArcSys;
CArcReplySet rReplySet;

. . . .

cArcSys.Command( rReplySet,
                 CArcBrdId( 0x10, 0x20 ),
                 TDL,
                 0x112233 );
. . . .
```

Example: To send test data link ('TDL') to ALL controllers:

```
CArcSys cArcSys;
CArcReplySet rReplySet;

. . . .

cArcSys.Command( rReplySet,
                 CArcBrdId::BROADCAST_ALL,
                 TDL,
                 0x112233 );
. . . .
```

Example: To send test data link ('TDL') to only controller number 0x90:

```
CArcSys cArcSys;
CArcReplySet rReplySet;

. . . .

cArcSys.Command( rReplySet,
                 CArcBrdId( 90, 90 ),
                 TDL,
                 0x112233 );
. . . .
```

CArcReply Class

(Include Header File: CArcReply.h)

All CArcSys methods that return a hardware reply (PCIe board or controller) will return a set of CArcReply classes (see CarcReplySet class below). The CArcReply class is data structure used to represent a single hardware reply and is specified by a header and value. The header has the following format: 0xSSDD02, where SS equals source id (2 = broadcast id, 8 to 255 = controller id) and DD equals destination id (should always be 0 = computer id). The reply value is stored as one of the following types: bool, int, void*, or double. The type of the value depends on the command. Most all hardware commands, however, return an integer.

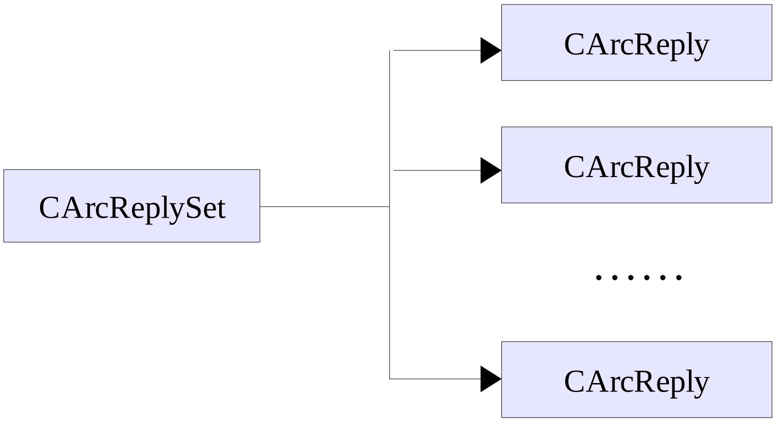
The reply data type can be determined using the xxxType() methods: *bool IntType()*, *bool BoolType()*, *bool PtrType()*, *bool DblType()*. All methods except that of the stored data type will return false. For example, if the reply is an integer, then the IntType() method will return true while all others will return false. Another method, *std::string TypeString()*, will return a string representation of the stored data type; "*int*", "*bool*", "*ptr*", or "*double*".

CArcReplySet Class

(Include Header File: CArcReplySet.h)

All CArcSys methods that return a hardware reply (PCIe board or controller) return a CArcReplySet via a passed-in parameter and/or return value. The CArcReplySet is purely a collection of CArcReply objects and a set of convenience methods for manipulating the collection. See the class definition for method details.

CArcReplySet – CArcReply Relationship



CArcReplySet Example: To send test data link ('TDL') to ALL controllers and check the replies:

```
CArcSys cArcSys;
CArcReplySet rReplySet;

. . . .

cArcSys.Command( rReplySet,
                 CArcBrdId( 0x10, 0x20 ),
                 TDL,
                 0x112233 );

for ( int i=0; i<rReplySet.Length(); i++ )
{
      CArcReply cReply = rReplySet.At( i );

      if ( cReply.Int() != 0x112233 )
      {
            // Handle error
      }
}

. . . .
```

CArcMuxPCIe Class

(Include Header File: CArcMuxPCIe.h)

The CArcMuxPCIe class is an extension of the standard ARC API CArcPCIe class. In general, this class should not be instantiated directly by the user application; instead the CArcSys class should be used, which encapsulates all PCIe boards in the system. The Virus PCIe board is tightly coupled with the MUX board and has a different register set than the standard PCIe board, which can be found in the Arc PCIe documentation. All CArcSys methods call methods in this class.

CArcSys Class

(Include Header File: CArcSys.h)

Everything should be done through this class (CArcSys), which encapsulates all the PCIe boards into a single system. All device (PCIe) access is done using a set of CArcMuxPCIe objects.

Example: Basic program (setup.cpp) to setup all controllers:

```
#include “CArcSys.h”

using namespace std;
using namespace arc;

int main()
{
      CArcSys cArcSys;

      try
      {
            // Open all PCIe devices
            cArcSys.Open();

            // Setup all controllers
            cArcSys.SetupController( “/home/arc/tim.lod” );
      }
      catch ( exception& e )
      {
            cerr << endl << e.what() << endl;
      }

      // Close all device connections
      cArcSys.Close();

      return 0;
}
```

To Compile:

```
g++ -I/xxx/ARC_API/VIRUS/CArcSys/src -I/xxx/ARC_API/3.0/CArcDevice
    -L/xxx/ARC_API/VIRUS/Release -L/xxx/ARC_API/3.0/Release
    setup.cpp -lCArcDevice -lCArcSys
```

Example: Basic program (expose.cpp) to setup all controllers and take a 0.5 second exposure:

```
#include “CArcSys.h”
#include “CExpIFace.h”
#include “CArcBrdId.h”

using namespace std;
using namespace arc;

class CExposeHandler : public CExpIFace
{
      void ExposeCallback( float fElapsedTime )
      {
            cout << “Elapsed Time: “ << fElapsedTime << endl;
      }

      void ReadCallback( int dPixelCount );
      {
            cout << “Pixel Count: “ << dPixelCount << endl;
      }
};

int main()
{
      CArcSys cArcSys;

      try
      {
            // Open all PCIe devices
            cArcSys.Open();

            // Setup all controllers
            cArcSys.SetupController( “/home/arc/tim.lod” );

            // Fill the buffer with test data
            cArcSys.FillCommonBuffer( int( 0xDEAD ) );

            // Start an exposure using all controllers. NOTE: This method
            // will block, put in a thread to allow other tasks during expose.
            cArcSys.Expose( CArcBrdId::BROADCAST_ALL,
                            0.5f,
                            CArcSys::DEFAULT_ROWS,
                            CArcSys::DEFAULT_COLS,
                            bAbort,
                            new CExposeHandler() );
      }
      catch ( exception& e )
      {
            cerr << endl << e.what() << endl;
      }

      // Close all device connections
      cArcSys.Close();

      return 0;
}
```

To Compile:

```
g++ -I/xxx/ARC_API/VIRUS/CArcSys/src -I/xxx/ARC_API/3.0/CArcDevice
    -L/xxx/ARC_API/VIRUS/Release -L/xxx/ARC_API/3.0/Release
    expose.cpp -lCArcDevice -lCArcSys
```

Header Listing

CArcSys.h

CArcSys class definition. Primary class that should be used for device access. This class encapsulates a set of CarcMuxPCIe classes.

CArcMuxPCIe.h

PCIe class definition. Provides PCIe device access and should not be used directly, the CArcSys class will handle calls to this class.

CArcBrdId.h

Board Id class definition. Used as a parameter for CArcSys methods to specify which controller to command.

CArcReply.h

Command reply class definition. Data structure to store a single controller command reply.

CArcReplySet.h

Command reply set class definition. Data structure to store a set of controller command replies (i.e. a set of CArcReply classes).

CArcSys Methods

This section documents details of the methods available through the CArcSys class (see CArcSys.h). These methods define the standard interface for the PCIe sub-devices. The following is a list of these methods; with details to follow on subsequent pages:

```
void IsOpen( CarcReplySet& rReplySet );

void Open();

void Close();

void Reset();

void GetCommonBufferProperties( CArcReplySet& rReplySet );

void FillCommonBuffer( unsigned short u16Value );

void CommonBufferVA( CArcReplySet& rReplySet );

void CommonBufferPA( CArcReplySet& rReplySet );

void CommonBufferSize( CArcReplySet& rReplySet );

void GetId( CArcReplySet& rReplySet );

void GetStatus( CArcReplySet& rReplySet );

void ClearStatus();

void WriteBar( int dBar, int dAddress, int dValue );

void ReadBar( CArcReplySet& rReplySet, int dBar, int dAddress );

CArcReplySet& Command( CArcReplySet& rReplySet, CArcBrdId cBoardId, int dCommand, int
dArg1, int dArg2, int dArg3, int dArg4 );

void Cmd( CArcBrdId cBoardId, int dCommand, CArcReply dExpectedReply );

void Cmd( CArcBrdId cBoardId, int dCommand, int dArg1, CArcReply dExpectedReply );

void Cmd( CArcBrdId cBoardId, int dCommand, int dArg1, int dArg2, CArcReply
dExpectedReply );

void Cmd( CArcBrdId cBoardId, int dCommand, int dArg1, int dArg2, int dArg3, CArcReply
dExpectedReply );

void Cmd( CArcBrdId cBoardId, int dCommand, int dArg1, int dArg2, int dArg3, int dArg4,
CArcReply dExpectedReply );

void CheckReply( CArcReplySet& rReply, int dReply );

void GetControllerIDs( CArcReplySet& rReplySet );

void ResetController();

void SetupController( const char* pszTimFile, CArcBrdId cBoardId, bool bReset, bool bTdl,
bool bPower, int dRows, int dCols );

void LoadControllerFile( const char* pszFilename, bool bValidate );
void SetImageSize( CArcBrdId cBoardId, int dRows, int dCols );

CArcReplySet& GetImageRows( CArcReplySet& rReplySet, CArcBrdId cBoardId );

CArcReplySet& GetImageCols( CArcReplySet& rReplySet, CArcBrdId cBoardId );

int  GetCCParams();

bool IsCCParamSupported( int dParameter );

void IsBinningSet( CArcReplySet& rReplySet, CArcBrdId cBoardId );

void SetBinning( CArcBrdId cBoardId, int dRows, int dCols, int dRowFactor, int
dColFactor, int* pBinRows, int* pBinCols );

void UnSetBinning( CArcBrdId cBoardId, int dRows, int dCols );

void IsSyntheticImageMode( CArcReplySet& rReplySet, CArcBrdId cBoardId );

void SetSyntheticImageMode( CArcBrdId cBoardId, bool bMode );

void SetOpenShutter( CArcBrdId cBoardId, bool bShouldOpen );

void Expose( CArcBrdId cBoardId, float fExpTime, int dRows, int dCols, const bool&
bAbort, CExpIFace* pExpIFace, bool bOpenShutter );

void EnableTemperatureCtrl( bool bOnOff, int dSide, double gTempture, CArcBrdId
cBoardId );

void GetArrayTemperature( CArcReplySet& rReplySet, int dSide, CArcBrdId cBoardId );

void GetArrayHeaterVoltage( CArcReplySet& rReplySet, int dSide, CArcBrdId cBoardId );
```

CArcSys::IsOpen

Syntax:

`void IsOpen( CArcReplySet& rReplySet );`

Description:

Each reply in the *CArcReplySet* contains true if an application has called *CArcSys::Open* successfully.

Parameters:

rReplySet [OUT]

> A reply set that specifies the open state of each device. Each reply in the set contains the device number and a boolean reply specifying if a device is open or closed.

Throws Exception:

N/A

Return Value	Description
CArcReplySet	Contains true if device is open; false otherwise.

Usage:

```
#include <iostream>
#include “CArcSys.h”

using namespace std;
using namespace arc;

CArcSys *pArcSys = new CArcSys();

CarcReplySet rReplySet;

pArcSys->Open();

if ( pArcSys->IsOpen( rReplySet ) )
{
      try
      {
            // Throws an exception if one or more
            // replies do not match.
            CArcReply cReply = rReplySet.Merge();
      }
      catch ( exception& e )
      {
            cerr << “Not all devices are open! “ << e.what();
      }
}

. . . .
```

CArcSys::Open

Syntax:

`void Open();`

Description:

Opens a connection to all available host interface devices and allocates the image buffer for each device.

Parameters:

N/A

Throws Exception:

N/A

Return Value	Description
N/A	N/A

Usage:

```
#include “CArcSys.h”

using namespace std;
using namespace arc;

CArcSys *pArcSys = new CArcSys();

pArcSys->Open();

. . . .
```

CArcSys::Close

Syntax:

```
void Close();
```

Description:

Closes all host interface devices and frees all image buffers.

Parameters:

N/A

Throws Exception:

N/A

Return Value	Description
N/A	N/A

Usage:

```
#include “CArcSys.h”

using namespace std;
using namespace arc;

CArcSys *pArcSys = new CArcSys();

pArcSys->Open();

. . . .

pArcDev->Close();
```

CArcSys::Reset

Syntax:

```
void Reset();
```

Description:

Resets all host interface devices.

Parameters:

N/A

Throws Exception:

std::runtime_error

Return Value	Description
N/A	N/A

Usage:

```
#include “CArcSys.h”

using namespace std;
using namespace arc;

CArcSys *pArcSys = new CArcSys();

try
{
      pArcSys->Open();

      . . . .

      pArcSys->Reset();

      . . . .
}
catch ( exception& e )
{
      cerr << e.what() << endl;
}
```

CArcSys::GetCommonBufferProperties

Syntax:

```
void GetCommonBufferProperties( CArcReplySet& rReplySet );
```

Description:

Calls the host interface driver to retrieve the common buffer properties: *user virtual address*, *physical address*, and *size* (in bytes) for each device (PCIe board).

Parameters:

rReplySet [OUT]

> A reply set where each reply contains true if the buffer properties of a device was successfully obtained. Each reply contains the device number and the boolean reply.

Throws Exception:

N/A

Return Value	Description
CArcReplySet	Contains true if successfully obtained device buffer properties; false otherwise.

Notes:

The properties are maintained by the CArcSys class and can be retrieved by calling the following methods: *CArcSys::CommonBufferVA*, *CArcSys::CommonBufferPA*, and *CArcSys::CommonBufferSize*.

Usage:

```
CArcSys *pArcSys = new CArcSys();

//
// Open device, etc.
//
. . . .

CarcReplySet rReplySet;

pArcSys->GetCommonBufferProperties( rReplySet );

for ( int i=0; i<rReplySet.Length(); i++ )
{
      CArcReply cReply = rReplySet.At( i );

      if ( cReply.Bool() )
      {
            cout << “Dev “ << cReply.Header() << “ buf virt addr: “
                 << pArcSys->CommonBufferVA() << endl;

            cout << “Dev “ << cReply.Header() << “ buf phys addr: “
                 << pArcSys->CommonBufferPA() << endl;

            cout << “Dev “ << cReply.Header() << “ buf size: “
                 << pArcDev->CommonBufferSize() << endl;
      }
}
```

CArcSys::FillCommonBuffer

Syntax:

`void FillCommonBuffer( unsigned short u16Value = 0 );`

Description:

Fills all device common buffers with the specified 16-bit value.

Parameters:

u16Value

The value to fill the common image buffer with; default = 0

Throws Exception:

std::runtime_error

Return Value	Description
N/A	N/A

Usage:

```
#include “CArcSys.h”

using namespace std;
using namespace arc;

CArcSys *pArcSys = new CArcSys();

//
// Open device, etc
//
. . . .

//
// Fill the buffer with 0xBEEF
//
pArcSys->FillCommonBuffer( 0xBEEF );
```

CArcSys::CommonBufferVA

Syntax:

```
void CommonBufferVA( CArcReplySet& rReplySet );
```

Description:

Returns the common buffer user virtual address for each device (PCIe board).

Parameters:

rReplySet [OUT]

> A reply set where each reply contains the device number and a pointer to the common buffer virtual address for the device.

Throws Exception:

N/A

Return Value	Description
CArcReplySet	Contains a void * pointer for each virtual address

Usage:

```
#include “CArcSys.h”

using namespace std;
using namespace arc;

CArcSys *pArcSys = new CArcSys();

//
// Open device, etc.
//
. . . .

CarcReplySet rReplySet;

//
// Get the virtual address to 16-bit data
//
pArcSys->CommonBufferVA( rReplySet );

for ( int i=0; i<rReplySet.Length(); i++ )
{
      unsigned short* pU16Buf = ( unsigned short * )rReplySet.At( i ).Ptr();

      //
      // Print the first ten values
      //
      for ( int i=0; i<10; i++ )
      {
            cout <<  “Buffer[ “ << i << “ ]: “ << pU16Buf[ i ] << endl;
      }
}
```

CArcSys::CommonBufferPA

Syntax:

```
void CommonBufferPA( CArcReplySet& rReplySet );
```

Description:

Returns the common buffer user physical address for each device (PCIe board).

Parameters:

rReplySet [OUT]

A reply set where each reply contains the device number and a pointer to the common buffer physical address for the device.

Throws Exception:

N/A

Return Value	Description
CArcReplySet	Contains a void * pointer for each physical address

Notes:

The physical address is an invalid address for the user application. It is only available for reference and validation and should never be called upon.

Usage:

```
#include “CArcSys.h”

using namespace std;
using namespace arc;

CArcSys *pArcSys = new CArcSys();

CArcReplySet rReplySet;

//
// Open device, etc.
//
. . . .

//
// Get the physical address of the common buffer
//
pArcSys->CommonBufferPA( rReplySet );

for ( int i=0; i<rReplySet.Length(); i++ )
{
      int physAddr = rReplySet.At( i ).Int();

      cout << “Buffer PA: 0x“ << hex << “ << physAddr << dec << endl;
}
```

CArcSys::CommonBufferSize

Syntax:

`void CommonBufferSize( CArcReplySet& rReplySet );`

Description:

Returns the common buffer size (in bytes) for each device (PCIe board).

Parameters:

rReplySet [OUT]

> A reply set where each reply contains the device number and an integer that represents the common buffer size (in bytes) for the device.

Throws Exception:

N/A

Return Value	Description
CArcReplySet	Contains the size, as an int, for each device common buffer

Notes:

The size (in bytes) of each allocated common image buffer.

Usage:

```
#include “CArcSys.h”

using namespace std;
using namespace arc;

CArcSys *pArcSys = new CArcSys();

CArcReplySet rReplySet;

//
// Open device, etc.
//
. . . .

//
// Get the size ( in bytes ) of the common buffer
//
pArcSys->CommonBufferSize( rReplySet );

for ( int i=0; i<rReplySet.Length(); i++ )
{
      cout << “Dev “ << rReplySet.At( i ).Header()
           << “ Buffer Size: “ << “ << rReplySet.At( i ).Int()
           << endl;
}
```

CArcSys::GetId

Syntax:

```
void GetId( CArcReplySet& rReplySet );
```

Description:

Returns the hardware device ID for each PCIe board.

Parameters:

rReplySet [OUT]

> A reply set where each reply contains the device number and an integer ID for the device.

Throws Exception:

std::runtime_error

Return Value	Description
CArcReplySet	Contains the device ID for each device. Must match PCIe::ID.

Notes:

The *CArcPCIe* class contains a static constant (CarcPCIe::ID) against which the return value can be compared.

Usage:

```
#include “CarcSys.h”
#include “CarcPCIe.h”

using namespace std;
using namespace arc;

CArcSys *pArcSys = new CArcSys();

CarcReplySet rReplySet;

. . . .

pArcSys->GetId( rReplySet );

for ( int i=0; i<rReplySet.Length(); i++ )
{
      if ( rReplySet.At( i ).Int() == CArcPCIe::ID )
      {
            cout << “Found PCIe board!” << endl;
      }
}
```

CArcSys::GetStatus

Syntax:

`void GetStatus( CArcReplySet& rReplySet );`

Description:

Returns the hardware device status for each PCIe board.

Parameters:

rReplySet [OUT]

A reply set where each reply contains the device number and an integer that represents the status for the device.

Throws Exception:

std::runtime_error

Return Value	Description
CArcReplySet	The hardware device status for each device (PCIe board).

Notes:

See the PCIe status register documentation for bit definition details.

Usage:

```
#include “CArcSys.h”

using namespace std;
using namespace arc;

CArcSys *pArcSys = new CArcSys();

CarcReplySet rReplySet;

. . . .

pArcSys->GetStatus( rReplySet );

for ( int i=0; i<rReplySet.Length(); i++ )
{
      cout << “Dev ” << i << “ Status: 0x” << hex
           << rReplySet.At( i ).Int() << endl;
}
```

CArcSys::ClearStatus

Syntax:

`void ClearStatus();`

Description:

Clears the device status for each PCIe board.

Parameters:

N/A

Throws Exception:

std::runtime_error

Return Value	Description
N/A	N/A

Notes:

Not generally useful in user applications.

Usage:

```
#include “CArcSys.h”

using namespace std;
using namespace arc;

CArcSys *pArcSys = new CArcSys();

. . . .

pArcSys->ClearStatus();
```

CArcSys::WriteBar

Syntax:

```
void WriteBar( int dBar, int dOffset, int dValue );
```

Description:

Write a value to a base address register (BAR) on all connected PCIe boards.

Parameters:

dBar

The base address register number.

dOffset

The offset within the base address register.

dValue

The value to write.

Throws Exception:

std::runtime_error

Return Value	Description
N/A	N/A

Notes:

In general, this method should never be called by a user application. See *CArcMuxPCIe.h* for definitions.

The dBar parameter can be one of the values:

`CArcPCIe::LCL_CFG_BAR`	(Local Configuration Registers)
`CArcPCIe::DEV_REG_BAR`	(FPGA (Device) Registers)

The dOffset parameter can be one of the values:

`CArcMuxPCIe::REG_MUX_ON_OFF`	(Enable/Disable MUX Register)
`CArcMuxPCIe::REG_FRAME_CNTR_RESET`	(Reset Frame Counter Register)
`CArcMuxPCIe::REG_SET_PIXEL_COUNT`	(Set Total Pixel Count Register)
`CArcMuxPCIe::REG_INIT_IMAGE_ADDR`	(Initialize Image Address Register)
`CArcMuxPCIe::REG_IMAGE_MODE_CTRL`	(Image/Command Mode Register)
`CArcMuxPCIe::REG_REPLY_HEADER_0`	(Reply Header 0 Register)
`CArcMuxPCIe::REG_REPLY_0`	(Reply 0 Register)
`CArcMuxPCIe::REG_REPLY_HEADER_1`	(Reply Header 1 Register)
`CArcMuxPCIe::REG_REPLY_1`	(Reply 1 Register)
`CArcMuxPCIe::REG_REPLY_HEADER_2`	(Reply Header 2 Register)
`CArcMuxPCIe::REG_REPLY_2`	(Reply 2 Register)
`CArcMuxPCIe::REG_REPLY_HEADER_3`	(Reply Header 3 Register)
`CArcMuxPCIe::REG_REPLY_3`	(Reply 3 Register)
`CArcMuxPCIe::REG_REPLY_HEADER_4`	(Reply Header 4 Register)
`CArcMuxPCIe::REG_REPLY_4`	(Reply 4 Register)
`CArcMuxPCIe::REG_REPLY_HEADER_5`	(Reply Header 5 Register)
`CArcMuxPCIe::REG_REPLY_5`	(Reply 5 Register)
`CArcMuxPCIe::REG_REPLY_HEADER_6`	(Reply Header 6 Register)

CArcMuxPCIe::REG_REPLY_6	(Reply 6 Register)
CArcMuxPCIe::REG_REPLY_HEADER_7	(Reply Header 7 Register)
CArcMuxPCIe::REG_REPLY_7	(Reply 7 Register)
CArcPCIe::REG_CMD_HEADER	(Command Header Register)
CArcPCIe::REG_CMD_COMMAND	(Command Register)
CArcPCIe::REG_CMD_ARG0	(Command Argument #1 Register)
CArcPCIe::REG_CMD_ARG1	(Command Argument #2 Register)
CArcPCIe::REG_CMD_ARG2	(Command Argument #3 Register)
CArcPCIe::REG_CMD_ARG3	(Command Argument #4 Register)
CArcPCIe::REG_CMD_ARG4	(Command Argument #5 Register)
CArcPCIe::REG_INIT_IMG_ADDR	(Image Buffer Physical Address Register)
CArcPCIe::REG_STATUS	(Status Register)
CArcPCIe::REG_PIXEL_COUNT	(Image Pixel Count Register)
CArcPCIe::REG_FRAME_COUNT	(Continuous Readout Frame Count Register)
CArcPCIe::REG_ID_LO	(Device ID LSW Register)
CArcPCIe::REG_ID_HI	(Device ID MSW Register)
CArcPCIe::REG_CTLR_SPECIAL_CMD	(Controller Special Command Register)

Usage:

```
CArcSys* pArcSys = new CArcSys();

. . . .

//
//  Send a TDL command to the controller
// +---------------------------------------------------+

//
// Send the command header
//
pArcSys->WriteBar( DEV_REG_BAR, REG_CMD_HEADER, 0x203 );

//
// Send the command
//
pArcSys->WriteBar( DEV_REG_BAR, REG_CMD_COMMAND, TDL );

//
// Send the argument
//
pArcSys->WriteBar( DEV_REG_BAR, REG_CMD_ARG0, 0x112233 );

. . . .

//
//  Instead, this should be done as follows:
// +---------------------------------------------------+
pArcSys->Command( rReplySet, CArcBrdId::BROADCAST_ALL, TIM_ID, TDL, 0x112233 );

. . . .
```

CArcSys::ReadBar

Syntax:

```
void ReadBar( CarcReplySet& rReplySet, int dBar, int dOffset );
```

Description:

Read a value from a PCIe base address register (BAR).

Parameters:

rReplySet [OUT]

A reply set where each reply contains the returned value.

dBar

The base address register number.

dOffset

The offset within the base address register.

Throws Exception:

std::runtime_error

Return Value	Description
CArcReplySet	Set of values read from the specified base address register (dBar + dOffset)

Notes:

In general, this method should never be called by a user application.

The dBar parameter can be one of the values:

CArcPCIe::LCL_CFG_BAR	(Local Configuration Registers)
CArcPCIe::DEV_REG_BAR	(FPGA (Device) Registers)

The dOffset parameter can be on the values:

CArcMuxPCIe::REG_MUX_ON_OFF	(Enable/Disable MUX Register)
CArcMuxPCIe::REG_FRAME_CNTR_RESET	(Reset Frame Counter Register)
CArcMuxPCIe::REG_SET_PIXEL_COUNT	(Set Total Pixel Count Register)
CArcMuxPCIe::REG_INIT_IMAGE_ADDR	(Initialize Image Address Register)
CArcMuxPCIe::REG_IMAGE_MODE_CTRL	(Image/Command Mode Register)
CArcMuxPCIe::REG_REPLY_HEADER_0	(Reply Header 0 Register)
CArcMuxPCIe::REG_REPLY_0	(Reply 0 Register)
CArcMuxPCIe::REG_REPLY_HEADER_1	(Reply Header 1 Register)
CArcMuxPCIe::REG_REPLY_1	(Reply 1 Register)
CArcMuxPCIe::REG_REPLY_HEADER_2	(Reply Header 2 Register)
CArcMuxPCIe::REG_REPLY_2	(Reply 2 Register)
CArcMuxPCIe::REG_REPLY_HEADER_3	(Reply Header 3 Register)
CArcMuxPCIe::REG_REPLY_3	(Reply 3 Register)
CArcMuxPCIe::REG_REPLY_HEADER_4	(Reply Header 4 Register)
CArcMuxPCIe::REG_REPLY_4	(Reply 4 Register)
CArcMuxPCIe::REG_REPLY_HEADER_5	(Reply Header 5 Register)
CArcMuxPCIe::REG_REPLY_5	(Reply 5 Register)

CArcMuxPCIe::REG_REPLY_HEADER_6	(Reply Header 6 Register)
CArcMuxPCIe::REG_REPLY_6	(Reply 6 Register)
CArcMuxPCIe::REG_REPLY_HEADER_7	(Reply Header 7 Register)
CArcMuxPCIe::REG_REPLY_7	(Reply 7 Register)

CArcPCIe::REG_CMD_HEADER	(Command Header Register)
CArcPCIe::REG_CMD_COMMAND	(Command Register)
CArcPCIe::REG_CMD_ARG0	(Command Argument #1 Register)
CArcPCIe::REG_CMD_ARG1	(Command Argument #2 Register)
CArcPCIe::REG_CMD_ARG2	(Command Argument #3 Register)
CArcPCIe::REG_CMD_ARG3	(Command Argument #4 Register)
CArcPCIe::REG_CMD_ARG4	(Command Argument #5 Register)
CArcPCIe::REG_INIT_IMG_ADDR	(Image Buffer Physical Address Register)
CArcPCIe::REG_STATUS	(Status Register)
CArcPCIe::REG_PIXEL_COUNT	(Image Pixel Count Register)
CArcPCIe::REG_FRAME_COUNT	(Continuous Readout Frame Count Register)
CArcPCIe::REG_ID_LO	(Device ID LSW Register)
CArcPCIe::REG_ID_HI	(Device ID MSW Register)
CArcPCIe::REG_CTLR_SPECIAL_CMD	(Controller Special Command Register)

Usage:

```
#include “CArcSys.h”
#include “CArcPCIe.h”

using namespace std;
using namespace arc;

CArcSys *pArcSys = new CArcSys();

. . . .

CArcReplySet rLoIdSet, rHiIdSet;

// Read all of the PCIe board IDs
//
pArcSys->ReadBar( rHiIdSet, CarcPCIe::DEV_REG_BAR, CarcPCIe::REG_ID_HI );
pArcSys->ReadBar( rLoIdSet, CarcPCIe::DEV_REG_BAR, CarcPCIe::REG_ID_LO );

if ( rHiIdSet.Length() == rLoIdSet.Length() )
{
      for ( int i=0; i<rHiIdSet.Length(); i++ )
      {
            cout << “Dev: “ << i << “ Id Hi: “ << rHiIdSet.At( i ).Int()
                 << “ Id Lo: “ << rLoIdSet.At( i ).Int() << endl;
      }
}

. . . .
```

CArcSys::Command

Syntax:

```
CArcReplySet& Command( CArcReplySet& rReplySet, CArcBrdId cBoardId, int dCommand, int
dArg1, int dArg2, int dArg3, int dArg4 );
```

Description:

Sends an ASCII command to the specified board(s).

Parameters:

rReplySet [OUT]

A reply set where each reply contains the reply header and an integer that represents the command reply.

cBoardId

A CArcBrdId class instance representing the ID of the board or range of boards to send the command to.

dCommand

A valid ASCII controller command. See *ArcDefs.h* for command and reply definitions.

dArg1 – dArg4

Arguments for the command; default = -1

Throws Exception:

std::runtime_error

Return Value	Description
CArcReplySet	The command reply set

Usage:

```
#include “CArcSys.h”

using namespace std;
using namespace arc;

CArcSys *pArcSys = new CArcSys();

. . . .

CArcReplySet rReplySet;

//
// Broadcast Test Data Link ( ‘TDL’ )
//
pArcSys->CheckReply(
      pArcSys->Command( rReplySet,
                        CArcBrdId::BROADCAST_ALL,
                        TDL,
                        0x112233 ),
      0x112233 );

. . . .
```

CArcSys::Cmd

Syntax:

```
void Cmd( CArcBrdId cBoardId, int dCommand, CArcReply dExpectedReply );

void Cmd( CArcBrdId cBoardId, int dCommand, int dArg1, CArcReply dExpectedReply );

void Cmd( CArcBrdId cBoardId, int dCommand, int dArg1, int dArg2, CArcReply
dExpectedReply );

void Cmd( CArcBrdId cBoardId, int dCommand, int dArg1, int dArg2, int dArg3, CArcReply
dExpectedReply );

void Cmd( CArcBrdId cBoardId, int dCommand, int dArg1, int dArg2, int dArg3, int dArg4,
CArcReply dExpectedReply );
```

Description:

Sends an ASCII command to the specified board(s) and compares the reply set against the expected value. An exception is thrown if one or more replies in the set fails to match the expected reply value.

Parameters:

cBoardId

A CArcBrdId class instance representing the id (8 – 255) of the board or range of boards to send the command.

dCommand

A valid ASCII controller command. See *ArcDefs.h* for command and reply definitions.

dArg1 – dArg4

Arguments for the command; default = -1

rReplySet

The expected reply that will be compared against each board reply; default = *CArcReply::DON_REPLY*

Throws Exception:

std::runtime_error

Return Value	Description
N/A	N/A

Usage:

```
#include “CArcSys.h”

using namespace std;
using namespace arc;

CArcSys *pArcSys = new CarcSys();

. . . .

//
// Broadcast Test Data Link ( ‘TDL’ )
//
pArcSys->Cmd( CArcBrdId::BROADCAST_ALL, TDL, 0x112233, CArcReply( 0, 0x112233 ) );

. . . .
```

CArcSys::CheckReply

Syntax:

```
void CheckReply( CArcReplySet& rReplySet, int dReply );
```

Description:

Checks each reply in the set against the integer reply value. An exception is thrown if any of the reply values in the set do not match.

Parameters:

rReplySet

A reply set whose values will be checked against the dReply parameter.

dReply

A reply value used to check all the values in the rReplySet parameter; default = '*DON' (0x444F4E)*

Throws Exception:

std::runtime_error

Return Value	Description
N/A	N/A

Usage:

```
#include “CArcSys.h”

using namespace std;
using namespace arc;

CArcSys *pArcSys = new CArcSys();

. . . .

CArcReplySet rReplySet;

//
// Broadcast Test Data Link ( ‘TDL’ )
//
pArcSys->CheckReply(
      pArcSys->Command( rReplySet,
                        CArcBrdId::BROADCAST_ALL,
                        TDL,
                        0x112233 ),
      0x112233 );

. . . .
```

CArcSys::GetControllerIDs

Syntax:

```
void GetControllerIDs( CArcReplySet& rReplySet );
```

Description:

Returns the hardware IDs from all available timing boards.

Parameters:

rReplySet [OUT]

A reply set which will contain all the controller ids.

Throws Exception:

std::runtime_error

Return Value	Description
CArcReplySet	The controller IDs

Notes:

This method reads the FPGA encoded board id and is the same for all ARC-12 boards. This DOES NOT read the one-wire individual id for a board; that id can be obtained by using the 'RID' ascii command. This method is not useful for the Virus system and should not be used.

Usage:

CArcSys::ResetController

Syntax:

```
void ResetController();
```

Description:

Resets all connected controllers.

Parameters:

N/A

Throws Exception:

std::runtime_error

Return Value	Description
N/A	N/A

Notes:

Usage:

```
#include “CArcSys.h”

using namespace std;
using namespace arc;

CArcSys *pArcSys = new CarcSys();

pArcSys->Open();

pArcSys->ResetController();

. . . .
```

CArcSys::SetupController

Syntax:

```
void SetupController( const char *pszTimFile, CArcBrdId cBoardId, bool bReset, bool bTdl,
bool bPower, int dRows, int dCols );
```

Description:

Convenience function to initialize a camera controller.

Parameters:

pszTimFile

DSP timing board file (.lod)

cBoardId

A CArcBrdId class instance representing the id (8 – 255) of the board or range of boards to setup. Default = CArcBrdId::BROADCAST_ALL

bReset

True to reset the controller. Typically be set to true. Default = true

bTdl

True to test the data link between the host computer and the host device (PCI, PCIe), and the host device and the camera controller. Typically set to true. Default = true

bPower

True to power-on the camera controller. Typically set to true. Default = true

dRows

Image row dimension (in pixels). Default = DEFAULT_ROWS

dCols

Image column dimension (in pixels). Default = DEFAULT_COLS

Throws Exception:

std::runtime_error

Return Value	Description
N/A	N/A

Notes:

This method must be called before any exposures or commands other than test data link (‘TDL’) can occur.

Usage:

```
#include <iostream>
#include “CArcSys.h”

using namespace std;
using namespace arc;

int main()
{
      CArcSys *pArcSys = NULL;

      try
      {
            pArcSys = new CArcSys();

            pArcSys->Open();

            pArcDev->SetupController( “/home/arc/tim.lod” );

            . . . .
      }
      catch ( exception& e )
      {
            cerr << endl << e.what() << endl;
      }

      pArcDev->Close();

      return 0;
}
```

CArcSys::LoadControllerFile

Syntax:

```
void LoadControllerFile( const char* pszFilename, bool bValidate );
```

Description:

Loads a DSP timing or utility file onto all connected controllers.

Parameters:

pszFilename

> The DSP timing or utility file to load onto the controller. Typically tim.lod (timing board) or util.lod (utility board).

bValidate

> True to verify that each data word is written successfully. Default = true

Throws Exception:

std::runtime_error

Return Value	Description
N/A	N/A

Notes:

Calling this method will effectively wipe out any existing controller settings. This method is called from within the SetupController() method and generally doesn't need to be called directly.

Usage:

```
#include <iostream>
#include “CArcSys.h”
#include “CarcReply.h”

using namespace std;
using namespace arc;

CArcSys *pArcSys = new CArcSys();

. . . .

// The following is essentially what SetupController() does:
pArcSys->ResetController();

for ( int i=0; i<123; i++ )
{
      pArcSys->Cmd( CArcBrdId::BROADCAST_ALL, TIM_ID, TDL,
                    0x123456, CArcReply( 0, 0x123456 ) );
}

pArcSys->LoadControllerFile( “tim.lod” );
pArcSys->Cmd( CArcBrdId::BROADCAST_ALL, TIM_ID, PON );
pArcSys->SetImageSize();

. . . .
```

CArcSys::SetImageSize

Syntax:

```
void SetImageSize( CArcBrdId cBoardId, int dRows, int dCols );
```

Description:

Set the image dimensions on all connected camera controllers.

Parameters:

cBoardId

A CArcBrdId class instance representing the id (8 – 255) of the board or range of boards to apply the new image size. Default = CarcBrdId::BROADCAST_ALL

dRows

The row image dimension (in pixels). Default = DEFAULT_ROWS

dCols

The column image dimension (in pixels). Default = DEFAULT_COLS

Throws Exception:

std::runtime_error

Return Value	Description
N/A	N/A

Notes:

This method is called from within the SetupController() method and generally doesn't need to be called directly.

Usage:

```
#include <iostream>
#include “CArcSys.h”

using namespace std;
using namespace arc;

CArcSys *pArcSys = new CArcSys();

. . . .

pArcSys->SetImageSize();

. . . .
```

CArcSys::GetImageRows

Syntax:

`CArcReplySet& GetImageRows( CArcReplySet& rReplySet, CArcBrdId cArcBrdId );`

Description:

Get the image row dimension from the camera controller.

Parameters:

rReplySet [OUT]

The reply set that will contain the controller row dimensions.

cBoardId

A CArcBrdId class instance representing the id (8 – 255) of the board or range of boards to send the command. Default = CArcBrdId::BROADCAST_ALL

Throws Exception:

std::runtime_error

Return Value	Description
CArcReplySet	The set of image row dimensions (in pixels) for each connected controller

Notes:

N/A

Usage:

```
#include <iostream>
#include “CArcSys.h”
#include “CarcReplySet.h”

using namespace std;
using namespace arc;

CArcSys *pArcSys = new CArcSys();

. . . .

CArcReplySet rRowSet = pArcSys->GetImageRows( rRowSet );
CArcReplySet rColSet = pArcSys->GetImageCols( rColSet );

if ( rRowSet.Length() == rColSet.Length() )
{
      for ( int i=0; i<rRowSet.Length(); i++ )
      {
            cout << “Current Image Size: “ << rRowSet.At( i ).Int()
                 << “x” << rColSet.At( i ).Int() << endl;
      }
}

. . . .
```

CArcSys::GetImageCols

Syntax:

```
CArcReplySet& GetImageCols( CArcReplySet& rReplySet, CArcBrdId cArcBrdId );
```

Description:

Get the image column dimension from the camera controller.

Parameters:

rReplySet [OUT]

The reply set that will contain the controller column dimensions.

cBoardId

A CArcBrdId class instance representing the id (8 – 255) of the board or range of boards to send the command. Default = CArcBrdId::BROADCAST_ALL

Throws Exception:

std::runtime_error

Return Value	Description
CArcReplySet	The set of image column dimensions (in pixels) for each connected controller

Notes:

N/A

Usage:

```
#include <iostream>
#include “CArcSys.h”
#include “CarcReplySet.h”

using namespace std;
using namespace arc;

CArcSys *pArcSys = new CArcSys();

. . . .

CArcReplySet rRowSet = pArcSys->GetImageRows( rRowSet );
CArcReplySet rColSet = pArcSys->GetImageCols( rColSet );

if ( rRowSet.Length() == rColSet.Length() )
{
      for ( int i=0; i<rRowSet.Length(); i++ )
      {
            cout << “Current Image Size: “ << rRowSet.At( i ).Int()
                 << “x” << rColSet.At( i ).Int() << endl;
      }
}

. . . .
```

CArcSys::GetCCParams

Syntax:

```
int GetCCParams();
```

Description:

Get the controller configuration parameter value from the first available camera controller.

Parameters:

N/A

Throws Exception:

std::runtime_error

Return Value	Description
int	The current controller configuration parameter value. Should be non-zero.

Notes:

The controller configuration parameter value bits specify the DSP firmware capabilities. The capabilities include binning, sub-array, temperature readout, shutter existence, which ARC boards are in the system, etc. The current bit definitions can be found in *ArcDefs.h*.

Call method *CArcSys::IsCCParamSupported(int)* to determine if individual capabilities are available.

Virus Only: The value returned should be the same for all controllers, so only the first non-zero value is returned. A zero return value means there is a problem with the loaded DSP code.

Usage:

```
#include “CArcSys.h”
#include “ArcDefs.h”

CArcSys *pArcSys = new CArcSys();

. . . .

int dCCParam = pArcSys->GetCCParam();

if ( pArcSys->IsCCParamSupported( SHUTTER_CC ) )
{
      cout << “Shutter support!” << endl;
}
else if ( pArcSys->IsCCParamSupported( SPLIT_SERIAL ) )
{
      cout << “Serial readout supported!” << endl;
}
else if ( pArcSys->IsCCParamSupported( BINNING ) )
{
      cout << “Binning supported!” << endl;
}
else if ( pArcSys->IsCCParamSupported( SUBARRAY ) )
{
      cout << “Sub-Array supported!” << endl;
}
. . . .
```

CArcSys::IsCCParamSupported

Syntax:

```
bool IsCCParamSupported( int dParameter );
```

Description:

Determines if the specified controller configuration parameter is available on the camera controller.

Parameters:

dParameter

The controller configuration parameter to check. A list of parameters can be found in *ArcDefs.h*.

Throws Exception:

N/A

Return Value	Description
true	The specified parameter is supported
false	The specified parameter is NOT supported

Notes:

N/A

Usage:

```
#include “CArcSys.h”
#include “ArcDefs.h”

using namespace std;
using namespace arc;

CArcSys *pArcSys = new CArcSys();

. . . .

int dCCParam = pArcSys->GetCCParam();

if ( pArcSys->IsCCParamSupported( SHUTTER_CC ) )
{
      cout << “Shutter support!” << endl;
}
else if ( pArcSys->IsCCParamSupported( SPLIT_SERIAL ) )
{
      cout << “Serial readout supported!” << endl;
}
else if ( pArcSys->IsCCParamSupported( BINNING ) )
{
      cout << “Binning supported!” << endl;
}
else if ( pArcSys->IsCCParamSupported( SUBARRAY ) )
{
      cout << “Sub-Array supported!” << endl;
}
      . . . .
```

CArcSys::IsBinningSet

Syntax:

```
void IsBinningSet( CArcReplySet& rReplySet, CArcBrdId cArcBrdId );
```

Description:

Determines if the camera controller identified by the board id parameter is currently set for binning.

Parameters:

rReplySet [OUT]

The reply set containing bool values that indicate if binning is set.

cBoardId

A CArcBrdId class instance representing the id (8 – 255) of the board or range of boards. Default = CArcBrdId::BROADCAST_ALL

Throws Exception:

std::runtime_error

Return Value	Description
CArcReplySet	Each reply in the set will be true if binning is set; false otherwise

Notes:

N/A

Usage:

```
      #include “CArcSys.h”
      #include “CArcReplySet.h”

      using namespace arc;

      CArcSys *pArcSys = new CArcSys();

      CArcReplySet rReplySet;

      . . . .

      //  Set binning mode to 2x2
      // +--------------------------------------------+
      pArcSys->SetBinning( CarcBrdId::BROADCAST_ALL, DEFAULT_ROWS, DEFAULT_COLS, 2, 2 );

      pArcSys->IsBinningSet( rReplySet );

      cout << “ Binning is ”
           << ( rReplySet.Merge().Bool() == true ? “SET” : “UNSET” )
           << endl;

      . . . .

      //  Un-Set binning mode
      // +--------------------------------------------+
      pArcSys->UnSetBinning( CarcBrdId::BROADCAST_ALL, DEFAULT_ROWS, DEFAULT_COLS );
```

CArcSys::SetBinning

Syntax:

```
void SetBinning( CArcBrdId cBoardId, int dRows, int dCols, int dRowFactor, int
dColFactor, int* pBinRows, int* pBinCols );
```

Description:

Sets the camera controller to binning mode.

Parameters:

cBoardId

A CArcBrdId class instance representing the id (8 – 255) of the board or range of boards.

dRows

The number of rows in the un-binned image.

dCols

The number of columns in the un-binned image.

dRowFactor

The row binning factor.

dColFactor

The column binning factor.

dBinRows

Optional pointer to return the binned image row size to the caller. Default = NULL

dBinCols

Optional pointer to return the binned image column size to the caller. Default = NULL

Throws Exception:

std::runtime_error

Return Value	Description
N/A	N/A

Notes:

Binning is used to combine pixels together on the chip and results in a smaller image. The number of pixels that are combined is determined by the row and column parameters, which do not need to match. A binning factor of 1 means no binning occurs along that image axis.

Usage:

```
#include <iostream>
#include “CArcSys.h”
#include “CArcReplySet.h”

using namespace std;
using namespace arc;

. . . .

CArcSys *pArcSys = NULL;

try
{
      pArcSys = new CArcSys();

      CArcReplySet rReplySet;

      . . . .

      //  Set binning mode to 2x2
      // +-----------------------------------------------+
      pArcSys->SetBinning( CArcBrdId::BROADCAST_ALL,
                           DEFAULT_ROWS,
                           DEFAULT_COLS,
                           2,
                           2 );

      pArcSys->IsBinningSet( rReplySet );

      cout << “ Binning is ”
           << ( rReplySet.Merge().Bool() == true ? “SET” : “UNSET” )
           << endl;

      . . . .

      //  Un-Set binning mode
      // +-----------------------------------------------+
      pArcSys->UnSetBinning( CArcBrdId::BROADCAST_ALL,
                             DEFAULT_ROWS,
                             DEFAULT_COLS );
}
catch ( exception& e )
{
      cerr << e.what() << endl;
}
```

CArcSys::UnSetBinning

Syntax:

```
void UnSetBinning( CArcBrdId cBoardId, int dRows, int dCols );
```

Description:

Sets the camera controller from binning mode back to normal image readout.

Parameters:

cBoardId

A CArcBrdId class instance representing the id (8 – 255) of the board or range of boards.

dRows

The number of rows in the un-binned image.

dCols

The number of columns in the un-binned image.

Throws Exception:

std::runtime_error

Return Value	Description
N/A	N/A

Notes:

N/A

Usage:

```
CArcSys *pArcSys = new CArcSys();

. . . .


//  Set binning mode to 2x2
// +-------------------------------------------+
pArcSys->SetBinning( CArcBrdId::BROADCAST_ALL,
                     DEFAULT_ROWS,
                     DEFAULT_COLS,
                     2,
                     2 );

. . . .


//  Un-Set binning mode
// +-------------------------------------------+
pArcSys->UnSetBinning( CArcBrdId::BROADCAST_ALL,
                       DEFAULT_ROWS,
                       DEFAULT_COLS );

. . . .
```

CArcSys::IsSyntheticImageMode

Syntax:

```
void IsSyntheticImageMode( CArcReplySet& rReplySet, CArcBrdId cBoardId );
```

Description:

Determines if the camera controller is currently set for synthetic image mode.

Parameters:

rReplySet [OUT]

> The reply set containing bool values that indicate if synthetic image mode is set.

cBoardId

> A CArcBrdId class instance representing the id (8 – 255) of the board or range of boards. Default = CArcBrdId::BROADCAST_ALL

Throws Exception:

std::runtime_error

Return Value	Description
CArcReplySet	Each reply in the set will be true if synthetic mode is set; false otherwise

Notes:

See *CArcSys::SetSyntheticImageMode()* notes for more details.

Usage:

```
#include <iostream>
#include “CArcSys.h”
#include “CArcReplySet.h”

using namespace std;
using namespace arc;

. . . .

CArcSys *pArcSys = new CArcSys();

. . . .

CArcReplySet rReplySet;

pArcSys->IsSyntheticImageMode( rReplySet );

cout << “Synthetic image mode is ”
     << ( rReplySet.Merge().Bool() ? “true” : “false” )
     << endl;

. . . .
```

CArcSys::SetSyntheticImageMode

Syntax:

```
void SetSyntheticImageMode( CArcBrdId cBoardId, bool bMode );
```

Description:

Sets the camera controller into synthetic image mode.

Parameters:

cBoardId

A CArcBrdId class instance representing the id (8 – 255) of the board or range of boards.

bMode

True to turn synthetic image mode on; false to turn off.

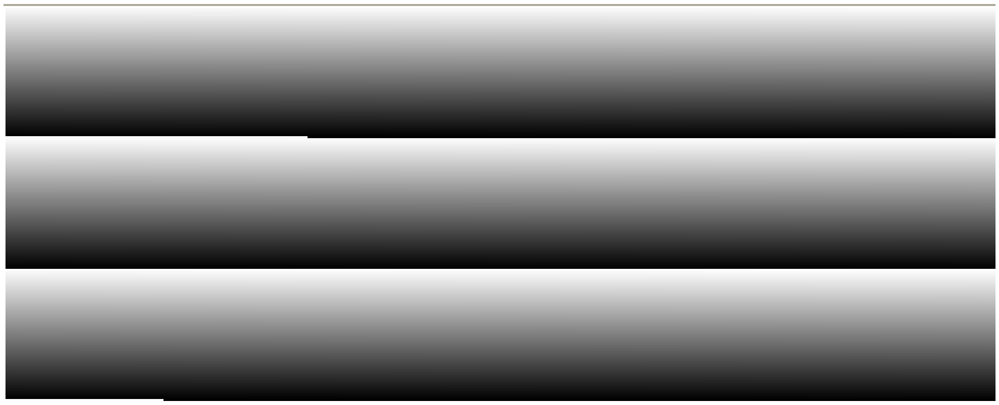
Throws Exception:

std::runtime_error

Return Value	Description
N/A	N/A

Notes:

Synthetic image mode causes the controller DSP to bypass the A/D converters and generate an artificial image pattern. The image data will have the following pattern: 0, 1, 2, 3… 65535, 0, 1, 2, 3… 65535, 0, 1, 2, 3… 65535 … See the figure below for an example of the pattern. The number and size of the pattern depends on the image dimensions.



Usage:

```
#include “CArcSys.h”

CArcSys *pArcSys = new CArcSys();

. . . .

pArcSys->SetSyntheticImageMode( CArcBrdId::BROADCAST_ALL, true );


. . . .
```

CArcSys::SetOpenShutter

Syntax:

```
void SetOpenShutter( CArcBrdId cBoardId, bool bMode );
```

Description:

Determines whether or not to open the shutter during an exposure.

Parameters:

cBoardId

A CArcBrdId class instance representing the id (8 – 255) of the board or range of boards.

bMode

True to open the shutter during exposure; false to keep it closed.

Throws Exception:

std::runtime_error

Return Value	Description
N/A	N/A

Notes:

N/A

Usage:

```
#include “CarcSys.h”

. . . .

CArcSys *pArcSys = new CArcSys();

. . . .

pArcSys->SetOpenShutter( CArcBrdId::BROADCAST_ALL, true );

. . . .
```

CArcSys::Expose

Syntax:

```
void Expose( float fExpTime, int dRows, int dCols, const bool& bAbort, CExpIFace* pExpIFace, int* pTotalPixelCount );

void Expose( CArcBrdId cBoardId, float fExpTime, int dRows, int dCols, const bool& bAbort, CExpIFace* pExpIFace, int* pTotalPixelCount, bool bOpenShutter );
```

Description:

Starts an image exposure. The first method is a convenience method that broadcasts expose to all controllers.

Parameters:

cBoardId

A CArcBrdId class instance representing the id (8 – 255) of the board or range of boards.

fExpTime

The exposure time (in seconds).

dRows

The number of rows in the image.

dCols

The number of columns in the image.

bAbort

External reference to allow the user to abort the method. Default = false

pExpIFace

A *CExpIFace* pointer that can be used to provide elapsed time and pixel count information. Default = NULL

pTotalPixelCount [OUT]

An integer that will be filled by the method with the total pixel count. Default = NULL

bOpenShutter

Set to true to open the shutter during an exposure. Default = true

Throws Exception:

std::runtime_error

Return Value	Description
N/A	N/A

Notes:

See the *CExpIFace.h* header file for details.

This is a convenience method that handles both the exposure and readout of an image. The elapsed exposure time and pixel count callback methods of the *CExpIFace* parameter (provided it's not NULL) will be used to provide feedback to the user application. The user application may extend the *CExpIFace* class or implement a separate extension class to handle the callback methods.

Note that this method will block until the exposure is complete. To allow the application to continue to process user input the Expose method should be run from within a thread.

Usage:

```
#include “CExpIFace.h”
#include “CArcSys.h”

class CMyExpIFace : public CExpIFace
{
      void ExposeCallback( float fElapsedTime )
      {
            cout << “Elapsed Time: “ << fElapsedTime << endl;
      }

      void ReadCallback( int dPixelCount )
      {
            cout << “Pixel Count: “ << dPixelCount << endl;
      }
};

CArcSys *pArcSys = new CArcSys();
CMyExpIFace cMyExpIFace;

. . . .

pArcSys->Expose( 0.5f, DEFAULT_ROWS, DEFAULT_COLS, false, &cMyExpIFace );

. . . .
```

In the above example, the expose and read callbacks will be called from the *Expose()* method during exposure and readout respectively. The *CExpIFace* and *CArcPCIe* classes can be combined into a single class as follows:

```
#include “CExpIFace.h”
#include “CArcSys.h”

class CMySys : public CExpIFace, public CArcSys
{
      void ExposeCallback( float fElapsedTime )
      {
            cout << “Elapsed Time: “ << fElapsedTime << endl;
      }

      void ReadCallback( int dPixelCount )
      {
            cout << “Pixel Count: “ << dPixelCount << endl;
      }

      void Expose( CArcBrdId cBoardId, float fExpTime, const bool& bAbort = false,
                   bool bOpenShutter = true )
      {
            CArcSys::Expose( cBoardId, fExpTime, DEFAULT_ROWS,
                             DEFAULT_COLS, bAbort, this );
      }

      . . . .
};

CMySys cMySys;

. . . .

cMySys.Expose( CArcBrdId::BROADCAST_ALL, 0.5f );
```

CArcSys::EnableTemperatureCtrl

Syntax:

```
void EnableTemperatureCtrl( bool bOnOff, int dSide, double gTemp, CArcBrdId cBoardId );
```

Description:

Enables/Disables array temperature control to the specified value.

Parameters:

bOnOff

> 'true' to control the temperature to the specified value, 'false' to disable control.

dSide

> Either LEFT_SIDE_ID or RIGHT_SIDE_ID. Defined in *CArcMuxPCIe.h*.

gTemp

> The temperature value to control the array too. Default = 0.0 (Celcius)

cBoardId

> A CArcBrdId class instance representing the id (8 – 255) of the board or range of boards. Default = CArcBrdId::BROADCAST_ALL

Throws Exception:

std::runtime_error

Return Value	Description
N/A	N/A

Notes:

N/A

Usage:

```
#include “CArcSys.h”
#include “CArcMuxPCIe.h”

using namespace arc;

. . . .

CArcSys* pArcSys = new CArcSys();

. . . .

// The following enables the left side to control the
// temperature to a value of -190 celcius on all boards.
pArcSys->EnableTemperatureCtrl( true, LEFT_SIDE, -190 );

. . . .
```

CArcSys::GetArrayTemperature

Syntax:

```
void GetArrayTemperature( CArcReplySet& rReplySet, int dSide, CArcBrdId cBoardId );
```

Description:

Returns the array temperature (in Celcius).

Parameters:

rReplySet [OUT]

The set of returned temperature values (doubles).

dSide

Either LEFT_SIDE_ID or RIGHT_SIDE_ID. Defined in *CArcMuxPCIe.h*.

cBoardId

A CArcBrdId class instance representing the id (8 – 255) of the board or range of boards. Default = CArcBrdId::BROADCAST_ALL

Throws Exception:

std::runtime_error

Return Value	Description
CArcReplySet	The set of array temperatures (in Celcius)

Notes:

N/A

Usage:

```
#include “CArcSys.h”
#include “CArcMuxPCIe.h”

using namespace arc;

. . . .

CArcSys* pArcSys = new CArcSys();

. . . .

pArcSys->GetArrayTemperature( rReplySet, LEFT_SIDE );

for ( int i=0; i<rReplySet.Length(); i++ )
{
      cout << “Ctlr Id: “ << rReplySet.At( i ).SrcID() << “ temp( C ): “
           << rReplySet.At( i ).Double() << endl;
}

. . . .
```

CArcSys::GetArrayHeaterVoltage

Syntax:

```
void GetArrayHeaterVoltage( CArcReplySet& rReplySet, int dSide, CArcBrdId cBoardId );
```

Description:

Returns the array heater voltage.

Parameters:

rReplySet [OUT]

The set of returned temperature values (doubles).

dSide

Either LEFT_SIDE_ID or RIGHT_SIDE_ID. Defined in *CArcMuxPCIe.h*.

cBoardId

A CArcBrdId class instance representing the id (8 – 255) of the board or range of boards. Default = CArcBrdId::BROADCAST_ALL

Throws Exception:

std::runtime_error

Return Value	Description
CArcReplySet	The set of heater voltages

Notes:

Usage:

```
#include “CArcSys.h”
#include “CArcMuxPCIe.h”

using namespace arc;

. . . .

CArcSys* pArcSys = new CArcSys();

. . . .

pArcSys->GetArrayHeaterVoltage( rReplySet, LEFT_SIDE );

for ( int i=0; i<rReplySet.Length(); i++ )
{
      cout << “Ctlr Id: “ << rReplySet.At( i ).SrcID() << “ temp( C ): “
           << rReplySet.At( i ).Double() << endl;
}

      . . . .
```

CExpIFace Interface

This section documents details of the methods available through the *CExpIFace* class (see CExpIFace.h). This class is an abstract interface that provides exposure callbacks for user applications. The user may extend this class and pass it into the *CArcSys::Expose()* method for elapsed time and pixel count information.

The following is a list of these methods; with details to follow on subsequent pages:

```
void ExposeCallback( float fElapsedTime );

void ReadCallback( int dPixelCount );
```

CExpIFace::ExposeCallback

Syntax:

```
void ExposeCallback( float fElapsedTime );
```

Description:

Called from the *CArcDevice::Expose()* method to supply the application with elapsed time info.

Parameters:

fElapsedTime

The current elapsed time.

Throws Exception:

std::runtime_error

Return Value	Description
N/A	N/A

Notes:

This class must be sub-classed by the user application. The sub-class can then be passed into the *CArcDevice::Expose()* method. This is the only way to get elapsed exposure time info from the *CArcDevice::Expose()* method.

Usage:

```
class CExpInfo : public CExpIFace
{
      void ExposeCallback( float fElapsedTime )
      {
            cout << “Elapsed Time: “ << fElapsedTime << endl;
      }

      void ReadCallback( int dPixelCount )
      {
            cout << “Pixel Count: “ << dPixelCount << endl;
      }

} cExpInfo;

. . . .

CArcSys* pArcSys = new CArcSys();

. . . .

pArcSys->Expose( 0.5f, dRows, dCols, false, &cExpInfo );

. . . .
```

The *CExpIFace* and *CArcSys* classes can be simultaneously sub-classed.

For example, to sub-class *CArcSys*

```
class CMySys : public CExpIFace, public CArcSys
{
      void ExposeCallback( float fElapsedTime )
      {
            cout << “Elapsed Time: “ << fElapsedTime << endl;
      }

      void ReadCallback( int dPixelCount )
      {
            cout << “Pixel Count: “ << dPixelCount << endl;
      }

};

. . . .

CMySys* pArcSys = new CMySys();

. . . .

pArcSys->Expose( 0.5f, dRows, dCols, false, pArcSys );

. . . .
```

CExpIFace::ReadCallback

Syntax:

```
void ReadCallback( int dPixelCount );
```

Description:

Called from the *CArcDevice::Expose()* method to supply the application with pixel count info.

Parameters:

dPixelCount

The current pixel count.

Throws Exception:

std::runtime_error

Return Value	Description
N/A	N/A

Notes:

This class must be sub-classed by the user application. The sub-class can then be passed into the *CArcDevice::Expose()* method. This is the only way to get pixel count info from the *CArcDevice::Expose()* method.

Usage:

```
class CExpInfo : public CExpIFace
{
      void ExposeCallback( float fElapsedTime )
      {
            cout << “Elapsed Time: “ << fElapsedTime << endl;
      }

      void ReadCallback( int dPixelCount )
      {
            cout << “Pixel Count: “ << dPixelCount << endl;
      }

} cExpInfo;

. . . .

CArcSys* pArcSys = new CArcSys();

. . . .

pArcSys->Expose( 0.5f, dRows, dCols, false, &cExpInfo );

. . . .
```

The *CExpIFace* and *CArcSys* classes can be simultaneously sub-classed.

For example, to sub-class *CArcSys*:

```
class CMySys : public CExpIFace, public CArcSys
{
      void ExposeCallback( float fElapsedTime )
      {
            cout << “Elapsed Time: “ << fElapsedTime << endl;
      }

      void ReadCallback( int dPixelCount )
      {
            cout << “Pixel Count: “ << dPixelCount << endl;
      }

};

. . . .

CMySys* pArcSys = new CMySys();

. . . .

pArcSys->Expose( 0.5f, dRows, dCols, false, pArcSys );

. . . .
```

CArcTools Methods

This section documents details of the methods available through the *CArcTools* class (see CArcTools.h). This provides utility functions used by the library and user applications.

Note that all methods are class methods, that is, all methods are static.

The following is a list of these methods; with details to follow on subsequent pages:

```
static std::string  ReplyToString( int dReply );

static std::string  CmdToString( int dCmd );

static std::string  CmdToString( int dReply, int dBoardId, int dCmd, int dArg1,
                                 int dArg2, int dArg3, int dArg4, int dSysErr );

static int StringToCmd( std::string sCmd );

static std::string  FormatString( const char *szFmt, ... );

static const std::string StringToUpper( std::string sStr );

static std::string  GetSystemMessage( int dCode );

static std::string  ConvertIntToString( int dNumber );

static std::string  ConvertWideToAnsi( wchar_t wcharString[] );

static std::string  ConvertWideToAnsi( const std::wstring& wsString );

static std::wstring ConvertAnsiToWide( const char *szString );

static long StringToHex( std::string sStr );

static char StringToChar( std::string sStr );

static void ThrowException( std::string sClassName, std::string sMethodName,
                            std::string sMsg );

static void ThrowException( std::string sClassName, std::string sMethodName,
                            const char *szFmt, ... );
```

CArcTools::ReplyToString

Syntax:

```
std::string ReplyToString( int dReply );
```

Description:

Returns the std::string representation of the specified command reply.

Parameters:

dReply

The command reply to convert to a std::string.

Throws Exception:

std::runtime_error

Return Value	Description
std::string	A text version of the reply parameter

Notes:

The hexadecimal value of the reply is returned as a character string if the reply is not a standard value.

Example: dReply = 0x455252 -> returns “ERR”

Example: dReply = 0x112233 -> returns “0x112233”

Usage:

```
CArcDevice* pArcDev = new CArcPCIe();

. . . .

int dReply = pArcDev->Command( TIM_ID, WRM, ( X_MEM | 0x3 ) );

//
// Outputs “WRM reply: DON” on success
//
cout << “WRM reply: “ << CArcTools::ReplyToString( dReply );

. . . .

dReply = pArcDev->Command( TIM_ID, TDL, 0x123456 );

//
// Outputs “TDL reply: 0x123456” on success
//
cout << “TDL reply: “ << CArcTools::ReplyToString( dReply );

. . . .
```

CArcTools::CmdToString

Syntax:

```
std::string CmdToString( int dCmd );
```

Description:

Returns the std::string representation of the specified command.

Parameters:

dCmd

The command to convert to a std::string.

Throws Exception:

N/A

Return Value	Description
std::string	A text version of the command parameter

Notes:

The hexadecimal value of the command is returned as a character string if the command is not a three letter ASCII command.

Example: dCmd = 0x54444C -> returns “TDL”

Example: dCmd = 0x112233 -> returns “0x112233”

Usage:

```
CArcDevice* pArcDev = new CArcPCIe();

. . . .

cout << “Sending “ << CArcTools::CmdToString( WRM ) << endl;
int dReply = pArcDev->Command( TIM_ID, WRM, ( X_MEM | 0x3 ) );

. . . .

cout << “Sending “ << CArcTools::CmdToString( TDL ) << endl;
dReply = pArcDev->Command( TIM_ID, TDL, 0x123456 );

. . . .
```

CArcTools::CmdToString

Syntax:

```
std::string CmdToString( int dReply, int dBoardId, int dCmd, int dArg1,
                         int dArg2, int dArg3, int dArg4, int dSysErr );
```

Description:

Method used to bundle command values into a string.

Parameters:

dReply

The command reply.

dBoardId

The command board ID.

dCmd

The command.

dArg1

The command argument #1.

dArg2

The command argument #2.

dArg3

The command argument #3.

dArg4

The command argument #4.

dSysErr

The system error code if the command failed.

Throws Exception:

N/A

Return Value	Description
std::string	A text version of the command

Notes:

The command is returned as a character string of the following form:

```
[ CmdHeader   Cmd  Arg1  Arg2  Arg3  Arg4 ] -> Reply \n System message
```

Usage:

```
CArcTools::CmdToString( 0x112233, TIM_ID, TDL, 0x112233 );
```

Produces the following output:

```
[ 0x203 TDL 0x112233 ] -> 0x112233
```

```
CArcTools::CmdToString( ERR, TIM_ID, TDL, 0x112233 );
```

Produces the following output:

```
[ 0x203 TDL 0x112233 ] -> ERR
```

CArcTools::StringToCmd

Syntax:

```
int StringToCmd( string sCmd );
```

Description:

Method to convert an ASCII command string, such as 'TDL' to the equivalent integer value.

Parameters:

sCmd

The command string to convert.

Throws Exception:

std::runtime_error

Return Value	Description
int	The integer version of the command string parameter

Notes:

Throws std::runtime_error if ASCII command parameter is not three characters in length.

Example: sCmd = “TDL” -> returns 0x54444C

Example: sCmd = “112233” -> returns 0x112233

Usage:

```
string sCmd;

cout << “Enter a command: “;
cin  >> sCmd;
cout << endl;

int dCmd = CArcTools::StringToCmd( sCmd );

cout << “The user entered the command: 0x”
     << hex << dCmd << dec << endl;
```

CArcTools::FormatString

Syntax:

```
std::string FormatString( const char *pszFmt, ... );
```

Description:

Method to format a std::string using C printf-style formatting.

Parameters:

pszFmt

A printf-style format string, followed by variables.

Throws Exception:

N/A

Return Value	Description
std::string	The formatted string

Notes:

Acceptable format parameters:

Format Specifier	Description
%d	integer
%f	double
%s	char * string
%e	system message
%x or %X	lower or upper case hexadecimal integer

Usage:

```
cout << “Two plus Two equals: ”
     << CArcTools::FormatString( “%d”, ( 2 + 2 ) )
     << endl;
```

CArcTools::StringToUpper

Syntax:

```
const std::string StringToUpper( std::string sStr );
```

Description:

Function to transform a string into all uppercase letters.

Parameters:

sStr

The string to convert.

Throws Exception:

N/A

Return Value	Description
const std::string	The converted string

Notes:

Usage:

```
cout << “The string \”lowercase\” as uppercase: ”
     << CArcTools::StringToUpper( “lowercase” )
     << endl;
```

CArcTools::GetSystemMessage

Syntax:

```
std::string GetSystemMessage( int dCode );
```

Description:

Used to get a formatted message string from the specified system error code.

Parameters:

dCode

A system error code.

Throws Exception:

N/A

Return Value	Description
std::string	The system error code string

Notes:

Usage:

CArcTools::ConvertWideToAnsi

Syntax:

```
std::string ConvertWideToAnsi( wchar_t wcharString[] );
```

Description:

Converts the specified wide char string (unicode) to an ANSI std::string.

Parameters:

wcharString

Wide character string to be converted to std::string.

Throws Exception:

N/A

Return Value	Description
std::string	The converted string

Notes:

Usage:

CArcTools::ConvertWideToAnsi

Syntax:

```
std::string ConvertWideToAnsi( const std::wstring& wsString );
```

Description:

Converts the specified wide string (unicode) to an ansi std::string.

Parameters:

wsString

Wide string to be converted to std::string.

Throws Exception:

N/A

Return Value	Description
std::string	The converted string

Notes:

Usage:

CArcTools::ConvertAnsiToWide

Syntax:

```
std::wstring ConvertAnsiToWide( const char *szString );
```

Description:

Converts the specified ANSI char string to a unicode std::wstring.

Parameters:

szString

ANSI C character string to be converted to std::wstring.

Throws Exception:

N/A

Return Value	Description
std::wstring	The converted wide string

Notes:

Usage:

CArcTools::ConvertIntToString

Syntax:

```
std::string ConvertIntToString( int dNumber );
```

Description:

Converts the specified integer value to a std::string.

Parameters:

dNumber

The integer to convert to std::string.

Throws Exception:

N/A

Return Value	Description
std::string	The converted string

Notes:

This is a convenience method.

Usage:

```
cout << “The number 10 as a string: “
     << CArcTools::ConvertIntToString( 10 )
     << endl;
```

CArcTools::StringToHex

Syntax:

```
long StringToHex( std::string sStr );
```

Description:

Converts the specified std::string to a long integer value.

Parameters:

sStr

The std::string to convert.

Throws Exception:

N/A

Return Value	Description
long	The converted long value

Notes:

This is a convenience method.

Usage:

```
      cout << “The string \“10\” as a hex value: “
           << CArcTools::StringToHex( “10” )
           << endl;
```

CArcTools::StringToChar

Syntax:

```
char StringToChar( std::string sStr );
```

Description:

Converts the specified std::string, which represents a single character, to a C char value.

Parameters:

sStr

The std::string to convert to a char.

Throws Exception:

N/A

Return Value	Description
char	The converted character

Notes:

This is a convenience method.

Usage:

```
char c = CArcTools::StringToHex( “P” );

cout << “c = “ << c << endl;
```

CArcTools::ThrowException

Syntax:

```
void ThrowException( string sClassName, string sMethodName, string sMsg );
```

Description:

Throws a std::runtime_error based on the supplied cfitsion status value.

Parameters:

sClassName

Name of the class where the exception occurred.

sMethodName

Name of the method where the exception occurred.

sMsg

The exception message.

Throws Exception:

std::runtime_error

Return Value	Description
N/A	N/A

Notes:

Throws a std::runtime_error exception with the message formatted as follows:

(ClassName::ClassMethod()): Message

If the sClassName parameter is empty, then the string “?Class?” will be used. Similarly, if the sMethodName parameter is empty, then the string “?Method?” will be used.

For Example: “ (CArcDevice::Command()): Incorrect reply: 0x112233”

For Example: “ (?Class?::?Method?()): Some message goes here”

Usage:

```
CArcDevice* pArcDev = new CArcPCIe();

. . . .

int dReply = pArcDev->Command( TIM_ID, TDL, 0x112233 );

if ( dReply != 0x112233 )
{
     CArcTools::ThrowException( “CArcDevice”, “Command”, “TDL Failed!” );
}

. . . .
```

CArcTools::ThrowException

Syntax:

```
void ThrowException( string sClassName, string sMethodName, const char *pszFmt, ... );
```

Description:

Method uses printf-style formatting that is then used to throw a std::runtime_error exception.

Parameters:

sClassName

Name of the class where the exception occurred.

sMethodName

Name of the method where the exception occurred.

pszFmt

A C printf-style format string, followed by variables.

Throws Exception:

std::runtime_error

Return Value	Description
N/A	N/A

Notes:

Throws a std::runtime_error exception with the message formatted as follows:

(ClassName::ClassMethod()): Message

If the sClassName parameter is empty, then the string “?Class?” will be used. Similarly, if the sMethodName parameter is empty, then the string “?Method?” will be used.

Acceptable format parameters:

Format Specifier	Description
%d	integer
%f	double
%s	char * string
%e	system message
%x or %X	lower or upper case hexadecimal integer

For Example: “ (CArcDevice::Command()): Incorrect reply: 0x112233”

For Example: “ (?Class?::?Method?()): Some message goes here”

Usage:

```
CArcDevice* pArcDev = new CArcPCIe();

. . . .

int dReply = pArcDev->Command( TIM_ID, TDL, 0x112233 );

if ( dReply != 0x112233 )
{
      CArcTools::ThrowException( “CArcDevice”,
                                 “Command”,
                                 “TDL failed, reply: 0x%X”,
                                  dReply );
}

. . . .
```

CArcTools CTokenizer Class

This section documents details of the methods available through the *CArcTools::CTokenizer* class (see CArcTools.h). This class provides a string tokenizer that uses string streams instead of the older C *strtok()* function.

The following is a list of these methods; with details to follow on subsequent pages:

```
CArcTools::CTokenizer();

void Victim( std::string str );

std::string Next();

bool IsEmpty();
```

CArcTools::CTokenizer

Syntax:

```
CArcTools::CTokenizer();
```

Description:

Default class constructor. Seperates a string into individual tokens deliminated by spaces.

Parameters:

N/A

Throws Exception:

N/A

Return Value	Description
N/A	N/A

Notes:

Class to separate a string deliminated spaces.

Usage:

```
CArcTools::CTokenizer tokenizer = new CArcTools::CTokenizer();

tokenizer.Victim( “This is a message!” );

while ( !tokenizer.IsEmpty() )
{
     cout << “Token: “ << tokenizer.Next() << endl;
}

. . . .
```

<u>Results in the following output</u>:

```
Token: This
Token: is
Token: a
Token: message!
```

CArcTools::CTokenizer::Victim

Syntax:

```
void Victim( std::string str );
```

Description:

Method used to break a string into individual tokens.

Parameters:

str

The string to parse.

Throws Exception:

std::runtime_error

Return Value	Description
N/A	N/A

Notes:

Seperates the specified string deliminated by spaces.

Usage:

```
CArcTools::CTokenizer tokenizer = new CArcTools::CTokenizer();

tokenizer.Victim( “This is a message!” );

while ( !tokenizer.IsEmpty() )
{
      cout << “Token: “ << tokenizer.Next() << endl;
}

. . . .
```

<u>Results in the following output</u>:

```
Token: This
Token: is
Token: a
Token: message!
```

CArcTools::CTokenizer::Next

Syntax:

```
std::string Next();
```

Description:

Method used to return the next token.

Parameters:

N/A

Throws Exception:

std::runtime_error

Return Value	Description
std::string	The next token from the string

Notes:

Seperates the specified string deliminated by spaces.

Usage:

```
CArcTools::CTokenizer tokenizer = new CArcTools::CTokenizer();

tokenizer.Victim( “This is a message!” );

while ( !tokenizer.IsEmpty() )
{
     cout << “Token: “ << tokenizer.Next() << endl;
}

. . . .
```

Results in the following output:

```
Token: This
Token: is
Token: a
Token: message!
```

CArcTools::CTokenizer::IsEmpty

Syntax:

```
bool IsEmpty();
```

Description:

Method used to determine if there are anymore tokens available.

Parameters:

N/A

Throws Exception:

std::runtime_error

Return Value	Description
true	More tokens are available
false	No more tokens are available

Notes:

Usage:

```
CArcTools::CTokenizer tokenizer = new CArcTools::CTokenizer();

tokenizer.Victim( “This is a message!” );

while ( !tokenizer.IsEmpty() )
{
     cout << “Token: “ << tokenizer.Next() << endl;
}

. . . .
```

Results in the following output:

```
Token: This
Token: is
Token: a
Token: message!
```

General Command and Controller Constants and Macros (ArcDefs.h)

This section documents details of the command and controller constants and macros as defined in *ArcDefs.h*.

PCI_ID

Type:

`Integer`

Description:

PCI(e) board id. Defined as 1.

TIM_ID

Type:

`Integer`

Description:

Timing board id. Defined as 2.

UTIL_ID

Type:

`Integer`

Description:

Utility board id. Defined as 3.

SMALLCAM_DLOAD_ID

Type:

`Integer`

Description:

SmallCam DSP download id. Defined as 3.

`X_MEM`

Type:

`Integer`

Description:

DSP X memory space. Used as part of the address parameter for read (‘RDM’) and write (‘WRM’) memory commands.

`Y_MEM`

Type:

`Integer`

Description:

DSP Y memory space. Used as part of the address parameter for read (‘RDM’) and write (‘WRM’) memory commands.

`P_MEM`

Type:

`Integer`

Description:

DSP program memory space. Used as part of the address parameter for read (‘RDM’) and write (‘WRM’) memory commands.

`R_MEM`

Type:

`Integer`

Description:

DSP ROM. Used as part of the address parameter for read (‘RDM’) and write (‘WRM’) memory commands.

DON

Type:

`Integer`

Description:

Success reply. Most device/controller commands return DON on success. See the command description document for details on commands and replies.

Defined as 0x444F4E

ERR

Type:

`Integer`

Description:

Error reply. See the command description document for details on commands and replies.

Defined as 0x455252

SYR

Type:

`Integer`

Description:

System reset reply. This reply means a system reset occurred. The *CArcDevice::ResetController()* method return SYR on success. See the command description document for details on commands and replies.

Defined as 0x535952

RST

Type:

`Integer`

Description:

Reset reply. See the command description document for details on commands and replies.

Defined as 0x525354

HERR

Type:

`Integer`

Description:

Header error reply. This reply means the command header is improperly formatted. See the command description document for details on commands and replies.

Defined as 0x48455252

TOUT

Type:

`Integer`

Description:

Timeout reply. This reply means the device or controller did not respond with a reply within a reasonable amount of time. See the command description document for details on commands and replies.

Defined as 0x544F5554

ROUT

Type:

`Integer`

Description:

Readout reply. This reply means the controller is currently reading an image. See the command description document for details on commands and replies.

Defined as 0x524F5554

`IS_ARC12`

Type:

`Macro`

Syntax:

`IS_ARC12( int id );`

Parameter:

Integer ID as returned from *CArcDevice::GetControllerId()*.

Description:

Macro that returns true if the ID parameter represents the SmallCam (ARC-12) controller. Returns false otherwise.

Simple Example

This section demonstrates a simple use of the ARC API VIRUS libraries.

```
The following compiles the program, where /xxx is the location of the ARC_API folder:

g++ -I/xxx/ARC_API/2.0/CFitsFile -I/xxx/ARC_API/3.0/CArcDevice
-I/xxx/ARC_API/2.0/cfitsio/linux -I/xxx/ARC_API/VIRUS/CArcSys/src
-L/xxx/ARC_API/VIRUS/CArcSys/Release -L/xxx/ARC_API/3.0/Release/x64
-L/xxx/ARC_API/2.0/Release/x64 CArcSysTest.cpp -lCArcSys -lCArcDevice -lcfitsio
-lCFitsFile
```

```
//============================================================================
// Name        : CArcSysTest.cpp
// Author      : Scott Streit
// Version     : 1.0
// Copyright   : ARC, Inc.
// Description : Test application for the VIRUS CArcSys class
//============================================================================

#include <iostream>
#include <sstream>
#include <iomanip>
#include <memory>
#include <valarray>
#include "CArcDevice.h"
#include "CArcPCIe.h"
#include "CArcTools.h"
#include "CArcReply.h"
#include "CArcBrdId.h"
#include "CArcMuxPCIe.h"
#include "CArcSys.h"
#include "CFitsFile.h"
#include "ArcDefs.h"

using namespace std;
using namespace arc;


#define DO_EXP( c ) c = 't'; while ( c != 'y' && c != 'n' ) { cout << "Do expose [y/n] ...
"; cin >> c; } cout << endl

#define PRINT_REPLY( text, rReplySet ) \
                        cout << endl;  \
                        for ( int i=0; i<rReplySet.Length(); i++ ) \
                        { \
                              cout << text << " -> 0x" << hex \
                                   << rReplySet.At( i ).Header() \
                                   << " 0x" << hex << rReplySet.At( i ).Int() \
                                   << dec << endl; \
                        } \
                        cout << endl


#define PRINT_DEC_REPLY( text, rReplySet ) \
                        cout << endl; \
                        for ( int i=0; i<rReplySet.Length(); i++ ) \
                        { \
                              cout << text << " -> 0x" << hex \
                                   << rReplySet.At( i ).Header() \
                                   << dec << " " << rReplySet.At( i ).Int() << endl; \
                        } \
                        cout << endl

#define PRINT_DBL_REPLY( text, rReplySet ) \
                        cout << endl; \
                        for ( int i=0; i<rReplySet.Length(); i++ ) \
                        { \
                              cout << text << " -> 0x" << hex
                                   << rReplySet.At( i ).Header() \
                                   << dec << " " << rReplySet.At( i ).Dbl() << endl; \
                        } \
                        cout << endl

#define PRINT_BOOL_REPLY( text, rReplySet ) \
                        cout << endl; \
                        for ( int i=0; i<rReplySet.Length(); i++ ) \
                        { \
                              cout << text << " -> 0x" << hex \
                                   << rReplySet.At( i ).Header() << " " \
                                   << ( rReplySet.At( i ).Bool() ? "true" : "false" ) \
                                     << dec << endl; \
                        } \
                        cout << endl


#define TIM_FILE        "/home/arc/test.lod"

void PrintReply( CArcReply* pReplySet );
void PrintCCParams( CArcSys* pCArcSys );
void WriteFits( CArcSys* pCArcSys, int dRows, int dCols );
std::string SetDots( const char *cStr );



class CExpCallBk : public CExpIFace
{
      void ExposeCallback( float fElapsedTime )
      {
            cout << "Elapsed Time: " << fElapsedTime << endl;
      }

      void ReadCallback( int dPixelCount )
      {
            cout << "\rPixel Count: " << setfill( ' ' )
                   << setw( 12 ) << right << dPixelCount
                   << flush;
      }
};





 //Main program
```

```
int main( int argc, char** argv )
{
      auto_ptr<CArcSys> pCArcSys;
      CArcReplySet cReplySet;
      CExpCallBk cExpCallBk;

      cout << endl;

      try
      {
            pCArcSys.reset( new CArcSys() );

            //+--------------------------------------------------------+
            //|  OPEN DEVICES                                          |
            //+--------------------------------------------------------+
            cout << "Opening devices .... ";
            pCArcSys.get()->Open();
            cout << "done!" << endl;

            pCArcSys.get()->IsOpen( cReplySet );
            PRINT_BOOL_REPLY( "IsOpen", cReplySet );

            // +--------------------------------------------------------+
            // |  GET COMMON BUFFER PROPERTIES                          |
            // +--------------------------------------------------------+
            cout << "Get common buffer properties .... ";
            pCArcSys.get()->GetCommonBufferProperties( cReplySet );
            cout << "done!" << endl;

            PRINT_BOOL_REPLY( "Common Buffer Properties Exist", cReplySet );

            pCArcSys.get()->CommonBufferVA( cReplySet );
            PRINT_REPLY( "CommonBufferVA", cReplySet );

            pCArcSys.get()->CommonBufferPA( cReplySet );
            PRINT_REPLY( "CommonBufferPA", cReplySet );

            pCArcSys.get()->CommonBufferSize( cReplySet );
            PRINT_DEC_REPLY( "CommonBufferSize", cReplySet );

            if ( cReplySet.Merge().Int() < CArcSys::DEFAULT_BUFFER_SIZE )
            {
                  pCArcSys.get()->Close();

                  ostringstream oss;
                  oss << "Image buffer not large enough! Size: "
                        << cReplySet.Merge().Int() << " Expected: "
                        << CArcSys::DEFAULT_BUFFER_SIZE
                        << ends;

                  throw runtime_error( oss.str().c_str() );
            }

            pCArcSys.get()->GetId( cReplySet );
            PRINT_REPLY( "GetId", cReplySet );

            pCArcSys.get()->GetStatus( cReplySet );
            PRINT_REPLY( "GetStatus", cReplySet );

            pCArcSys.get()->ClearStatus();
            pCArcSys.get()->GetStatus( cReplySet );
            PRINT_REPLY( "GetStatus", cReplySet );

            // +---------------------------------------------------------+
            // |  FILL COMMON BUFFER                                     |
            // +---------------------------------------------------------+
            cout << "Fill common buffer .... ";
            pCArcSys.get()->FillCommonBuffer( 0xDEAD );
            cout << "done!" << endl;

            WriteFits( pCArcSys.get(),
                           CArcSys::DEFAULT_ROWS,
                           CArcSys::DEFAULT_COLS );

            // +---------------------------------------------------------+
            // |  RESET DEVICES                                          |
            // +---------------------------------------------------------+
            cout << "Reset device .... ";
            pCArcSys.get()->Reset();
            cout << "done!" << endl;

            // +---------------------------------------------------------+
            // |  SETUP CONTROLLER                                       |
            // +---------------------------------------------------------+
            cout << "Setup controller .... ";
            pCArcSys.get()->SetupController( TIM_FILE );
            cout << "done!" << endl;

            // +---------------------------------------------------------+
            // |  PRINT CCPARAMS                                         |
            // +---------------------------------------------------------+
            PrintCCParams( pCArcSys.get() );

            // +---------------------------------------------------------+
            // |  READ IMAGE SIZE                                        |
            // +---------------------------------------------------------+
            cout << "Image rows .... ";
            int dRows = pCArcSys.get()->GetImageRows( cReplySet ).Merge().Int();
            cout << dRows << endl;

            cout << "Image cols .... ";
            int dCols = pCArcSys.get()->GetImageCols( cReplySet ).Merge().Int();
            cout << dCols << endl;

            // +---------------------------------------------------------+
            // |  PRINT VIRUS ID                                         |
            // +---------------------------------------------------------+
            pCArcSys.get()->PrintIDList();

            // +---------------------------------------------------------+
            // |  CONTROLLER ID                                          |
            // +---------------------------------------------------------+
            pCArcSys.get()->GetControllerIDs( cReplySet );

            PRINT_REPLY( "Controller ID", cReplySet );




            // +---------------------------------------------------------+
            // |  RDM                                                    |
            // +---------------------------------------------------------+
            cReplySet = pCArcSys.get()->Command( cReplySet,
                                                 CArcBrdId::BROADCAST_ALL,
                                                 RDM,
                                                 ( X_MEM | 0 ) );

            PRINT_REPLY( "RDM X:0", cReplySet );

            // +---------------------------------------------------------+
            // |  TEST DATA LINK                                         |
            // +---------------------------------------------------------+
            int dData = 0x112200;

            cout << "Test Data Link .... ";
            for ( int t=0; t<1911; t++ )
            {
                  dData += t;

                  pCArcSys.get()->Cmd( CArcBrdId::BROADCAST_ALL,
                                                 TDL,
                                                 dData,
                                                 CArcReply( 0, dData ) );
            }
            cout << "done!" << endl;

            cout << endl;

            // +---------------------------------------------------------+
            // |  SYNTHETIC IMAGE MODE                                   |
            // +---------------------------------------------------------+
            cout << "Setting synthetic image mode .... ";
            pCArcSys.get()->SetSyntheticImageMode( CArcBrdId::BROADCAST_ALL, true );
            cout << "done!" << endl;

            pCArcSys.get()->IsSyntheticImageMode( cReplySet );

            PRINT_BOOL_REPLY( "Is Synthetic Mode", cReplySet );

            // +---------------------------------------------------------+
            // |  EXPOSE                                                 |
            // +---------------------------------------------------------+
            char c; DO_EXP( c );

            if ( c == 'y' )
            {
                  cout << "Exposing .... ";
                  pCArcSys.get()->Expose( CArcBrdId::BROADCAST_ALL,
                                                      0.5,
                                                      dRows,
                                                      dCols,
                                                      false,
                                                      &cExpCallBk );
                  cout << " .... done!" << endl;

                  WriteFits( pCArcSys.get(), dRows, dCols );
            }



            // +---------------------------------------------------------+
            // |  BINNING                                                  |
            // +-----------------------------------------------------------+
            pCArcSys.get()->IsBinningSet( cReplySet );

            PRINT_BOOL_REPLY( "Is binning set", cReplySet );

            int dBinRows = 0, dBinCols = 0;

            cout << "Setting binning .... ";
            pCArcSys.get()->SetBinning( CArcBrdId::BROADCAST_ALL, dRows, dCols,
                                        2, 2, &dBinRows, &dBinCols );
            cout << "done! Binned Rows: " << dBinRows << " Binned Cols: "
                 << dBinCols << endl;

            pCArcSys.get()->IsBinningSet( cReplySet );

            PRINT_BOOL_REPLY( "Is binning set", cReplySet );

            cout << "UnSet binning .... ";
            pCArcSys.get()->UnSetBinning( CArcBrdId::BROADCAST_ALL, dRows, dCols );
            cout << "done! Rows: "
                 << pCArcSys.get()->GetImageRows( cReplySet ).Merge().Int()
                 << " Cols: " << pCArcSys.get()->GetImageCols( cReplySet ).Merge().Int()
                 << endl;

            pCArcSys.get()->IsBinningSet( cReplySet );

            PRINT_BOOL_REPLY( "Is binning set", cReplySet );

            // +-----------------------------------------------------------+
            // |  ENABLE/DISABLE ARRAY TEMPERATURE CONTROL                 |
            // +-----------------------------------------------------------+
            cout << "Enable LEFT array temperature .... ";
            pCArcSys.get()->EnableTemperatureCtrl( true, LEFT_SIDE_ID, -167.5 );
            cout << "done!" << endl;

            cout << "Disable RIGHT array temperature .... ";
            pCArcSys.get()->EnableTemperatureCtrl( false, RIGHT_SIDE_ID );
            cout << "done!" << endl;

            cout << endl;

            // +-----------------------------------------------------------+
            // |  READ TEMPERATURE                                         |
            // +-----------------------------------------------------------+
            cout << "Reading LEFT array temperature .... ";
            pCArcSys.get()->GetArrayTemperature( cReplySet, LEFT_SIDE_ID );
            cout << "done!" << endl;

            PRINT_DBL_REPLY( "Array Temperature LEFT", cReplySet );

            cout << "Reading RIGHT array temperature .... ";
            pCArcSys.get()->GetArrayTemperature( cReplySet, RIGHT_SIDE_ID );
            cout << "done!" << endl;

            PRINT_DBL_REPLY( "Array Temperature RIGHT", cReplySet );




            // +-----------------------------------------------------------+
            // |  READ HEATER VOLTAGE                                    |
            // +-------------------------------------------------------+
            cout << "Reading LEFT heater voltage .... ";
            pCArcSys.get()->GetArrayHeaterVoltage( cReplySet, LEFT_SIDE_ID );
            cout << "done!" << endl;

            PRINT_DBL_REPLY( "Array Heater Voltage LEFT", cReplySet );

            cout << "Reading RIGHT heater voltage .... ";
            pCArcSys.get()->GetArrayHeaterVoltage( cReplySet, RIGHT_SIDE_ID );
            cout << "done!" << endl;

            PRINT_DBL_REPLY( "Array Heater Voltage RIGHT", cReplySet );
      }
      catch ( exception& e )
      {
            cout << endl << e.what() << endl;
      }

      cout << endl;

      return 0;
}

void PrintCCParams( CArcSys* pCArcSys )
{
            cout << endl << "CCParam value .... 0x"
                 << hex << pCArcSys->GetCCParams()
                 << dec << endl;

            cout << left << setfill( '-' ) << setw( 35 ) << "" << endl;

            cout << left << setfill( '.' ) << setw( 30 ) << "BINNING"
                 << ( pCArcSys->IsCCParamSupported( BINNING ) ? "true" : "false" )
                 << endl;

            cout << left << setfill( '.' ) << setw( 30 ) << "SHUTTER_CC"
                 << ( pCArcSys->IsCCParamSupported( SHUTTER_CC ) ? "true" : "false" )
                 << endl;

            cout << left << setfill( '.' ) << setw( 30 ) << "TEMP_SIDIODE"
                 << ( pCArcSys->IsCCParamSupported( TEMP_SIDIODE ) ? "true" : "false" )
                 << endl;

            cout << left << setfill( '.' ) << setw( 30 ) << "SPLIT_SERIAL & PARALLEL"
                 << ( pCArcSys->IsCCParamSupported( ALL_READOUTS ) ? "true" : "false" )
                 << endl;

            cout << left << setfill( '.' ) << setw( 30 ) << "SUBARRAY"
                 << ( pCArcSys->IsCCParamSupported( SUBARRAY ) ? "true" : "false" )
                 << endl;

            cout << left << setfill( '.' ) << setw( 30 ) << "MPP_CC"
                 << ( pCArcSys->IsCCParamSupported( MPP_CC ) ? "true" : "false" )
                 << endl;

            cout << left << setfill( '.' ) << setw( 30 ) << "FO_2X_TRANSMITR"
                 << ( pCArcSys->IsCCParamSupported( FO_2X_TRANSMITR ) ? "true" : "false" )
                 << endl;

            cout << left << setfill( '.' ) << setw( 30 ) << "CONT_RD"
```

```
                 << ( pCArcSys->IsCCParamSupported( CONT_RD ) ? "true" : "false" )
                 << endl;

            cout << left << setfill( '-' ) << setw( 35 ) << "" << endl;

            cout << endl;
}

void PrintReply( CArcReplySet* pReplySet )
{
      if ( pReplySet != NULL )
      {
            cout << "Reply Length -> " << pReplySet->Length() << endl;

            for ( int i=0; i<pReplySet->Length(); i++ )
            {
                  cout << "REPLY[ " << i << " ] -> 0x" << hex
                         << pReplySet->At( i ).Int() << dec << endl;
            }
      }
}

void WriteFits( CArcSys* pCArcSys, int dRows, int dCols )
{
      CArcReplySet cReplySet;

      cout << "Writing FITS .... ";
      pCArcSys->CommonBufferVA( cReplySet );

      CFitsFile* pFits = new CFitsFile( "VirusImage.fit",
                                                          dRows,
                                                          dCols,
                                                          CFitsFile::BPP16,
                                                          true );

      for ( int i=0; i<cReplySet.Length(); i++ )
      {
            pFits->Write3D( cReplySet.At( i ).Ptr() );
      }

      delete pFits;
      cout << "done!" << endl;
}

// +--------------------------------------------------------------------------------
// |  SetDots
// +--------------------------------------------------------------------------------
// |  This function just prints dots (...) for the output.
// +--------------------------------------------------------------------------------
std::string SetDots( const char *cStr )
{
      std::string sStr( cStr );

      for ( int i=sStr.length(); i<40; i++ )
            sStr.append( "." );

      return sStr;
}
```