ClassACNet

Object Interface to MOOC/ACNet

Duane C. Voy
Fermi National Accelerator Laboratory
Accelerator Division
Instrumentation Department

Rationale

- C++ interface to MOOC
- Easy to leverage MOOC/ACNet features
- Provide full-featured MOOC support
- Minimize redundant MOOC class code
- Allow user to tailor the level of support
- Off-the-shelf solutions for many cases
- Minimal object oriented mumbo-jumbo so...
- Easy to use with C (using C++ compiler)

ClassACNet

- ClassACNet is a C++ namespace
- Class Portal
  - interface/configure MOOC/ACNet/Triggers
  - supports fundamental MOOC features
  - supports multiple front-end data structures
- Class ObjAccessor
  - interface to front-end data structures
  - supports data and control flow

Abstraction: Device & Object

- Map front-end data to the ACNet model
- Front-end **Device** data are represented by:
  - a **Control Object** (optional)
    - read status (optional)
    - set (& read into bit bucket) control (optional)
  - a **Data Object** (optional)
    - read data (optional)
    - set & read data/parameters (optional)

Abstraction: Device & Object

```
                    ACNet to ClassACNet Property/Method Mapping

|    ACNet     |                          ClassACNet
|<- Property ->|<------------------------- Device ------------------------->|

    basic
    status    -->  reading ----\
                               +---- Control Object ----\
    basic     <--  setting -+--/  (ObjAccessor instance)  \
    control                 |                              \
                  reading   |                               \
                    of    --                                 +--- Device --
                  setting                                   /
                                                           /
    reading   -->  reading ----\                          /
                               +----  Data  Object  ----/
    setting   <--  setting -+--/  (ObjAccessor instance)
                            |
    reading        reading  |
      of      -->    of    --
    setting        setting
```

Class Portal

- A MOOC class
- Each Portal instance provides:
  - support for up to 256 ClassACNet Devices
  - continuous & snapshot plots (15 to 720 Hz)
  - alarms and limits (digital and analog)
  - robust error checking of SSDN, length & offset
  - support for MOOC_WHACK_FORWARD
  - optional double buffer resolves alignment issues
  - all of this support in <u>one line of user code</u>

Class Portal

```
Portal( ePortalStatus * const statusPtr,
        char const * const namePtr,
        unsigned int const portalID,
        unsigned int const maxAlarms,
        bool const fastPlotEnable,
        unsigned int const maxPlots,
        eDiagnosticControl const diagControl );
```

- portalID: `SSDNHX xxxx (0000/0021/0000/0000)`
  - `Portal( &status, “TestPortal”, 0x0021, k64Alarms, k720HzPlot, k32Plots, kDiagOff );`

Class Portal

- Public methods:
  - `ePortalID PortalID( void ) const;`
  - `unsigned int MaxAlarms( void ) const;`
  - `bool FastPlotEnable( void ) const;`
  - `unsigned int MaxPlots( void ) const;`

  - `void DiagnosticControl( eDiagnosticControl const );`
  - `eDiagnosticControl DiagnosticControl( void )const;`
  - `char const *Name( void ) const;`
  - `void PrintVersion( void ) const;`
  - `void Display( bool const showHeader ) const;`

Class ObjAccessor

- ObjAccessors *access* front-end *objects*
- ObjAccessors are automatically attached to the ‘most recently instantiated’ Portal
- Each ObjAccessor instance provides:
  - data read & fast read methods (read accessor),
  - parameter set & read methods (set accessor) or
  - both of the above (read&set accessor).

Class ObjAccessor

- Abstract class - must be subclassed
- `ObjAccessor(ePortalStatus * const statusPtr,`
- `char const * const namePtr,`
- `unsigned int const deviceID,`
- `bool const controlDevice,`
- `unsigned long int const maxReadBuffer,`
- `unsigned long int const maxSetBuffer,`
- `eDiagnosticControl const diagControl);`

- deviceID: `SSDNHX xxxx (0008/0021/0000/0000)`
  - `ObjAccessor( &status, “Test”, 0x0008, kData, sizeof( reading ), sizeof( setting ), kDiagOff );`

Class ObjAccessor

- Public methods:
  - `eDeviceID DeviceID( void ) const;`
  - `void OffsetScale( unsigned long int const );`
  - `unsigned long int OffsetScale( void ) const;`
  - `void DiagnosticControl( eDiagnosticControl const);`
  - `eDiagnosticControl DiagnosticControl( void ) const;`
  - `char const *Name( void ) const;`
  - `void PrintVersion( void ) const;`
  - `void Display( bool const showHeader ) const;`

Class ObjAccessor

- Protected methods:
  - `virtual eDeviceStatus Validate(..);`
  - `virtual eDeviceStatus Set(..);`
  - `virtual eDeviceStatus ReadSetting(..)const;`
  - `virtual eDeviceStatus Read(..) const;`
  - `virtual void FastRead(..) const;`
  - `virtual unsigned long IndexCount(..) const;`
  - `virtual unsigned long SetByteCount(..) const;`
  - `virtual unsigned long ReadByteCount(..) const;`
  - `virtual unsigned long FastReadByteCount(..) const;`
- This is where the ObjAccessor meets the Portal

Class ObjAccessor

- Three device types:
  - array (type 0)
    - 1..n elements of identical size
  - channel (type 1)
    - 256 channels of possibly different sizes
  - length offset (type 2)
    - length in bytes, offset in offsetScale * bytes
- deviceType: `SSDNHX xxxx (0008/0021/0000/0100)`
- deviceChan: `SSDNHX xxxx (0008/0021/0000/0123)`

Off-the-shelf ObjAccessors

- In situ - access front-end data in memory
- Object/Array/Matrix - allocated front-end memory objects
- Constructor - reading via object construction
- Function - calls your read and set functions
- Method - calls your class' accessor/mutator
- String - 2D array of char properly swapped
- File Directory - list of *.typ files in directory

In situ ObjAccessor

- Access front-end data in memory
- Scalar & array versions:
  - read&set (with and without set bounds)
  - read
  - set (with and without set bounds)
  - write only (with and without set bounds)
- Data flow only - best for accessing RAM based data, or hardware registers

Object ObjAccessor

- Instantiate & access a front-end object
- Scalar and 1 & 2-D array versions:
  - read&set (with and without set bounds)
  - read
  - set (with and without set bounds)
- Indexing operators provided
- Data flow only - best for data-only objects
- Set objects with polling scheme?

Constructor ObjAccessor

- Calls your object's constructor to build reading reply
- One to three constructor parameters
- Scalar & array versions
- Data and control (via constructor call) flow
- Your custom processing

Function & Method ObjAccessors

- Calls your function or accessor/mutator
- Scalar & array versions:
  - read & set (with and without set bounds)
  - read
  - set (with and without set bounds)
- Data and control flow
- Your custom processing
- <u>You must know ObjAccessor status codes</u>

String & File Directory ObjAccessors

- String Array ObjAccessor
  - 2D array of char properly swapped
  - read data flow only
- File Directory ObjAccessor
  - listing of all *.typ files in /path
  - read data flow only

Locking ObjAccessor

- You inherit this class which provides
  - simple read locking protocol
    - lock others out until you are done reading data
  - simple set-read locking protocol
    - set setting to lock others out until you are done reading the result
- Data and control flow

Example - dabbel

```
!
ADD R:TDTEST ( "Damper Test Device", RRXDMP,         , 07FFFFFE )
!
 SSDNHX SETTNG (0008/0021/0000/0123)
!               |||| |||| |||| ||||
!               |||| |||| |||| ||-->>>>chan (8 bits) :== channel
!               |||| |||| |||| -->>>>typ (8 bits) :== deviceType
!               |||| |||| ---->>>>misc (16 bits)
!               |||| ---->>>>oid (14 bits: 8192 max) :== portalID
!               ||-->>>>misc2 (8 bits: 256 max) :== deviceID
!               -->>>>misc2 (8 bits) - reserved must be zero
!
```

Example - foo.h

```
#include "classacnet.h"    // object interface to accelerator control network
class Foo {
  public:
    typedef enum {         // legal setting range
      kFooMin = -128, kFooNominal = 0, kFooMax = 127
    } tFooType;
    typedef enum {         // Foo Status
      kNoModuleFound = -129, kERROR = -1, kOK = 0
    } eFooStatus;

    // constructor
    Foo( eFooStatus * const statusPtr, tFooType const initialValue );

    // accessors
    int Read( tFooType *destPtr, unsigned long int const misc ) const {
      *destPtr = static_cast<tFooType>( _aToD ); return( ClassACNet::kDeviceOk );
    };
    int Get( tFooType *destPtr, unsigned long int const misc ) const {
      *destPtr = static_cast<tFooType>( _dToA ); return( ClassACNet::kDeviceOk );
    };
    // mutator
    int Set( tFooType const &value, unsigned long int const misc ) {
      _dToA = static_cast<short int>( value ); return( ClassACNet::kDeviceOk );
    };

    inline void *operator new( size_t size, void *p ) {
      return( p );
    }
  private:
    short int  _aToD, _dToA;
};
```

Example - foo.cpp

```
#include "classacnetinsitu.h"           // ClassAcnet stuff
using namespace ClassACNet;

Foo::Foo( eFooStatus * const statusPtr, tFooType const initialValue ) {

  // probe for control registers
  short int  garbage;     // scratch for probing hardware
  if ( OK != vxMemProbe( reinterpret_cast<char *>( this ), VX_READ, 2, reinterpret_cast<char *>( &garbage ) ) ) {
    *statusPtr = kNoModuleFound;
    return;
  }
  _dToA = initialValue;

  // add this to ACNet
  ePortalStatus           status;
  (void) new Portal( &status, "Foo", 0x0023, k3Alarms, k720HzPlot, k3Plots, kDiagOff );

  // read A/D
  (void) new ReadInSitu<short int>
    ( &status, "M:ATOD", 0x0000, kData, kUnaligned, _aToD, kDiagOff );

  // set with bounds & read setting of D/A
  (void) new BndsSetInSitu<short int>
    ( &status, "M:DTOA",  0x0001, kData, kUnaligned, _dToA, kFooMin, kFooMax, kDiagOff );

  // read A/D and set with bounds & read setting of D/A
  (void) new ReadBndsSetInSitu<short int,short int>
    ( &status, "M:BOTH",  0x0002, kData, kUnaligned, _aToD, _dToA, kFooMin, kFooMax, kDiagOff );

  *statusPtr = kOK;
  return;
}
```

Example - bar.cpp

```
#include "foo.h"            // class Foo stuff
#include "classacnet.h"     // object interface to accelerator control network

using namespace ClassACNet;

void * kFooBase = reinterpret_cast<void *>( 0x08000000 );

void Bar( void ) {

  Foo::eFooStatus           status;
  class Foo                 *fooPtr = new ( kFooBase ) Foo( &status, Foo::kFooNominal );

  ePortalStatus    stat;
  (void) new Portal( &stat, "Foo", 0x0020, k3Alarms, k720HzPlot, k3Plots, kDiagOff );

  // read A/D
  (void) new ReadMethod<class Foo, Foo::tFooType>
    ( &stat, "M:ATOD", 0x0000, kData, kUnaligned,
      fooPtr, &Foo::Read, kDiagOff );

  // set with bounds & read setting of D/A
  (void) new BndsSetMethod<class Foo, Foo::tFooType>
    ( &stat, "M:DTOA", 0x0001, kData, kUnaligned,
      fooPtr, &Foo::Get, &Foo::Set, Foo::kFooMin, Foo::kFooMax, kDiagOff );

  // read A/D and set with bounds & read setting of D/A
  (void) new ReadBndsSetMethodr<class Foo , Foo::tFooType , Foo::tFooType >
    ( &stat, "M:BOTH", 0x0002, kData, kUnaligned,
      fooPtr, &Foo::Read, &Foo::Get, &Foo::Set, Foo::kFooMin, Foo::kFooMax, kDiagOff );

  return;
}
```

ObjAccessor Model

- Source code is classacnetmodel.cpp in the classacnetexample CVS repository (make a copy)
- Fully functional example for all methods
- Edit to grow your own custom ObjAccessor
- Data and control flow
- Your custom processing
- <u>You must know how ObjAccessors work</u>

ClassACNet - classacnetremote.h

- Class ACNetState

```
//
// Template remote ACNet state send support
//
// Template parameters:
//  int di - Device Index of states device
//
    template<int di>
    class ACNetState { … };


Example:
    class ACNetState<195228>  stateDevice;

    stateDevice.Announce( 5 );

    int value = stateDevice.Value();
```

ClassACNet - classacnetremote.h

- Class ACNetRead

```
//
// Template remote ACNet reading support
//
// Template parameters:
//  Tdata - type of data being read
//  int trunknode - trunk and node of remote system (e.g. 0x09cd)
//  int oid - oid of remote device (2nd word of ssdn)
//  int omsp - omsp of remote device (3rd and 4th words of ssdn)
//  int misc2 - first word of SSDN (often ignored)
//  int di - Device Index of remote device
//  int pi - Property Index of remote device (PI_READNG or PI_BASTAT)
//
    template<class Tdata, int trunknode, int oid, int omsp, int misc2, int di, int pi = PI_READNG>
    class ACNetRead { … };


Example:
    class ACNetRead<float, 0x0a22, 0x0018, 0x00000000, 0x0001, 146554, PI_READNG> testReading;

    status = testReading.Read();
    printf( "Z:MTSTST_READ status = 0x%08x, data.status = 0x%08x, data.value = %f\n",
            status, testReading.Status(), testReading.Value() );
```

ClassACNet - classacnetremote.h

Class ACNetSet

```
//
// Template remote ACNet setting support
//
// Template parameters:
//  Tdata - type of data being read
//  int trunknode - trunk and node of remote system (e.g. 0x09cd)
//  int oid - oid of remote device (2nd word of ssdn)
//  int omsp - omsp of remote device (3rd and 4th words of ssdn)
//  int misc2 - first word of SSDN (often ignored)
//  int di - Device Index of remote device
//  int pi - Property Index of remote device (PI_SETTNG or PI_BCNTRL)
//  int forward -- 0 or 1, indicating whether remote FE should forward setting to database
//
    template<class Tdata, int trunknode, int oid, int omsp, int misc2, int di, int pi=PI_SETTNG, int forward=1>
    class ACNetSet { … };


Example:
    class ACNetSet<float, 0x0a22, 0x0018, 0x00000000, 0x0001, 146554, PI_SETTNG, FORWARD> testSetting;

    status = testSetting.Set( 456.0 );
    printf( "Z:MTSTST_SET Set() status = 0x%08x, data.status = 0x%08x, data.value = %f\n",
            status, testSetting.Status(), testSetting.Value() );
    status = testSetting.ReadSetting();
    printf( "Z:MTSTST_SET ReadSetting() status = 0x%08x, data.status = 0x%08x, data.value = %f\n",
            status, testSetting.Status(), testSetting.Value() );
```