Pds Xtc

Package Xtc

The Xtc package defines several classes.

Class XtcFileIterator

An xtc file is a collection of datagrams (Dgram). This is an iterator to loop through all datagrams in a file.

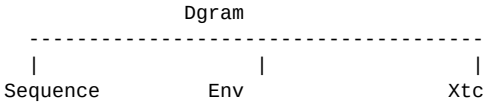
Public Member Functions:

```
// Constructor and destructor
XtcFileIterator(int fd, size_t maxDgramSize);
~XtcFileIterator();

Dgram* next();
```

Class Dgram

Datagram - structure of a piece of information within the xtc file.
A datagram contains: Sequence, Env and Xtc.



Public Attributes:

```
  PDS_DGRAM_STRUCT // #define PDS_DGRAM_STRUCT Sequence seq; Env env; Xtc xtc
```

Class Sequence

Defined in `pdsdata/xtc/Sequence.hh`.

Public Types:

```
  enum Type    {Event = 0, Occurrence = 1, Marker = 2};
  enum         {NumberOfTypes = 3};
```

Public Member Functions:

```
// Constructors
Sequence() {}
Sequence(const Sequence&);
Sequence(const ClockTime& clock, const TimeStamp& stamp);
Sequence(Type, TransitionId::Value, const ClockTime&, const TimeStamp&);

Type type() const;
TransitionId::Value  service() const;
bool isExtended() const;
bool isEvent() const;
const ClockTime& clock() const {return _clock;}
const TimeStamp& stamp() const {return _stamp;}

Sequence& operator=(const Sequence&);
```

Class Env

Defined in `pdsdata/xtc/Env.hh`

Public Member Functions

```
// Constructors
Env() {}
Env(const Env& in) : _env(in._env) {}
Env(uint32_t env);

uint32_t value() const;

const Env& operator=(const Env& that);
```

Class Xtc

This class defines a container for data stored in an xtc file. An event is a collection of such containers. The size and location of the unspecified data contained are defined by the "sizeofPayload" and "payload" functions. To iterate over a set of such containers see "XtcIterator.hh".

Public Member Functions:
Constructors:

```
//Constructor used to create an empty, unlabeled
Xtc() : damage(0), extent(0) {};

// Copy constructor, will copy everything, but NOT the payload.
Xtc(const Xtc& xtc) :
  damage(xtc.damage), src(xtc.src), contains(xtc.contains), extent(sizeof(Xtc)) {}

//
Xtc(const TypeId& type) :
  damage(0), contains(type), extent(sizeof(Xtc)) {}

Xtc(const TypeId& type, const Src& _src) :
  damage(0), src(_src), contains(type), extent(sizeof(Xtc)) {}

Xtc(const TypeId& _tag, const Src& _src, unsigned _damage) :
  damage(_damage), src(_src), contains(_tag), extent(sizeof(Xtc)) {}

Xtc(const TypeId& _tag, const Src& _src, const Damage& _damage) :
  damage(_damage), src(_src), contains(_tag), extent(sizeof(Xtc)) {}
```

Overloaded allocation operators to allow the insertion of a dummy "Xtc" within a datagram.

```
void* operator new(size_t size, char* p)     { return (void*)p; }
void* operator new(size_t size, Xtc* p)      { return p->alloc(size); }
```

```
// Return a pointer (of unspecified type) to the payload associated with this class
char*        payload()       const { return (char*)(this+1); }

// Return the size (in bytes) of the payload associated with the class.
int          sizeofPayload() const { return extent - sizeof(Xtc); }

// Return the Xtc which is immediately following an object of this class.
Xtc*       next()               { return (Xtc*)((char*)this+extent); }
const Xtc* next()         const { return (const Xtc*)((char*)this+extent); }

void*      alloc(uint32_t size)  { void* buffer = next(); extent += size; return buffer; }
```

Public Attributes:

```
Damage   damage;
Src      src;
TypeId   contains;
uint32_t extent;
```

Class XtcIterator

This class allows iteration over a collection of "Xtc" objects. An "event" generated from DataFlow consists of data described by a collection of "Xtc" objects. Therefore, this class is instrumental in the navigation of an event's data. The set of "Xtc"s is determined by passing (typically to the constructor) a root "Xtc" which describes the collection of "Xtc"s to process. This root, for example is provided by an event's datagram.

As this is an Abstract-Base-Class, it is expected that an application will subclass from this class, providing an implementation of the "process" method. This method will be called back for each "Xtc" in the collection. Note that the "Xtc" to process is passed as an argument. If the "process" method wishes to abort the iteration a zero (0) value is returned. The iteration is initiated by calling the "iterate" member function.

Public Member Functions:
Constructors and destructor:

```
XtcIterator(Xtc* root);
XtcIterator() {}
virtual ~XtcIterator() {}
```

The first constructor takes an argument the "Xtc" which defines the collection to iterate over.

```
virtual int process(Xtc* xtc) = 0;
```

This function is pure virtual and must be implemented by all derived classes.

```
void      iterate();
```

This function commences iteration over the collection specified by the constructor.

```
void      iterate(Xtc*);
```

Iterate over the collection specifed as an argument to the function. For each "Xtc" found call back the "process" function. If the "process" function returns zero (0) the iteration is aborted and control is returned to the caller. Otherwise, control is returned when all elements of the collection have been scanned.

```
const Xtc* root()  const;
```

This function returns the collection specified by the constructor.

Class Level

Defined in `pdsdata/xtc/Level.hh`

Public Types:

```
enum Type{Control, Source, Segment, Event, Recorder, Observer, Reporter,
          NumberOfLevels};
```

Static Public Member Functions

```
static const char* name(Type type);
```

Class TransitionId

Public Types:

```
enum Value {
  Unknown, Reset,
  Map, Unmap,
  Configure, Unconfigure,
  BeginRun, EndRun,
  BeginCalibCycle, EndCalibCycle,
  Enable, Disable,
  L1Accept,
  NumberOf };
```

Static Public Member Functions:

```
static const char* name(TransitionId::Value id);
```

Class Src

This is a base class for any source of information in the xtc file (e.g. Detector, Beamline, Process). Defined in `pdsdata/xtc/Src.hh`

Public Member Functions:

```
// Constructors
Src();
Src(Level::Type level);

uint32_t log()   const;    // logical identifier
uint32_t phy()   const;    // physical identifier

Level::Type level() const; // level

bool operator==(const Src& s) const;
```

Protected Attributes

```
uint32_t _log; // logical  identifier
uint32_t _phy; // physical identifier
```

Class BldInfo

Beamline data Information. This is a derived class that inherits from Src.
Defined in `pdsdata/xtc/BldInfo.hh`

Public Types:

```
enum Type { EBeam = 0, PhaseCavity = 1, FEEGasDetEnergy = 2, NumberOf = 3 };
```

Public Member Functions

- Constructors and Destructor

```
BldInfo() {}
BldInfo(uint32_t processId, Type type);
```

- Accessors

```
uint32_t processId() const;
Type     type()  const;
```

Static Public Member Functions

```
static const char* name(const BldInfo&);
```

Class ProcInfo

Process information class. This is a derived class that inherits from Src.
Holds process info for all Levels except Source.

Public Member Functions:

```
// Constructor
ProcInfo(Level::Type level,     //
         uint32_t processId,    //
         uint32_t ipAddr);      //

uint32_t processId() const;
uint32_t ipAddr()    const;
void     ipAddr(int);
```

Class DetInfo

Detector information. Inherits from Src. Defined in `pdsdata/xtc/DetInfo.hh` and implemented in `pdsdata/xtc/src/DetInfo.cc`.

Public Types:

```
enum Detector {
  NoDetector    = 0,
  AmoIms        = 1,  // AMO Ion Momentum Spectrometer
  AmoGasdet     = 2,  // AMO Gas Detector (in FrontEnd Enclosure)
  AmoETof       = 3,  // AMO Electron Time-of-flight
  AmoITof       = 4,  // AMO Ion Time-of-flight
  AmoMbes       = 5,  // AMO Magnetic bottle electron spectrometer
  AmoVmi        = 6,  // AMO Velocity map imaging
  AmoBps        = 7,  // AMO Beam position screen
  Camp          = 8,  // CFel-ASG-Multi-Purpose EndStation
  EpicsArch     = 9,  //
  BldEb         = 10,
  SxrBeamline   = 11,
  SxrEndstation = 12,
  XppSb1Ipm     = 13,
  XppSb1Pim     = 14,
  XppMonPim     = 15,
  XppSb2Ipm     = 16,
  XppSb3Ipm     = 17,
  XppSb3Pim     = 18,
  XppSb4Pim     = 19,
  XppGon        = 20,
  XppLas        = 21,
  XppEndstation = 22,
  AmoEndstation = 23,
  CxiEndstation = 24,
  XcsEndstation = 25,
  MecEndstation = 26,
  NumDetector   = 27
};
enum Device {
  NoDevice  = 0,
  Evr       = 1,
  Acqiris   = 2,
  Opal1000  = 3,
  TM6740    = 4,
  pnCCD     = 5,
  Princeton = 6,
  Fccd      = 7,
  Ipimb     = 8,
  Encoder   = 9,
  Cspad     = 10,
  NumDevice = 11
};
```

Public Member Functions

```
// Constructors
DetInfo() {}
DetInfo(uint32_t   processId,   //
        Detector   det,         //
        uint32_t   detId,       //
        Device     dev,         //
        uint32_t   devId);      //

bool operator==(const DetInfo &) const;

uint32_t processId() const;
Detector detector() const;
Device device() const;
uint32_t detId() const;
uint32_t devId() const;
```

Static Public Member Functions

```
static const char *name(Detector);
static const char *name(Device);
static const char *name(const DetInfo &);
```

Class ClockTime

Defined in `pdsdata/xtc/ClockTime.hh`.

Public Member Functions:

- Constructors (ClockTime has no destructor)

```
ClockTime();
ClockTime(const ClockTime& t);
ClockTime(unsigned sec, unsigned nsec);
```

- Accessors:

```
unsigned seconds    () const {return _high;}
unsigned nanoseconds() const {return _low;}
```

- Operators

```
ClockTime& operator=(const ClockTime&);
bool operator> (const ClockTime&) const;
bool operator==(const ClockTime&) const;
```

Class Damage

Defined in `pdsdata/xtc/Damage.hh`, inline implementation.

Public Types:

```
enum Value {
  DroppedContribution    = 1,
  OutOfOrder             = 12,
  OutOfSynch             = 13,
  UserDefined            = 14,
  IncompleteContribution = 15,
  ContainsIncomplete     = 16
};
// reserve the top byte to augment user defined errors
enum {NotUserBitsMask=0x00FFFFFF, UserBitsShift = 24};
```

- *DroppedContribution* - means that event is "split" and contains only partial data from incomplete set of detectors. There may be another (or even few) datagram with the same timestamp containing remaining data from the same event.
- *OutOfOrder* - In this case the event may contain data from one or more detector which actually belong to a different event.
- *OutOfSynch*
- *UserDefined* - reserved for user-defined conditions
- *IncompleteContribution* - means that data in that particular container is badly damaged, are incomplete, and cannot be used
- *ContainsIncomplete* - means that it contains one or more containers at deeper XTC levels which have IncompleteContribution damage, events with this damage may be partially useful

```
Bit   Mask   Name
-----------------------------------
 1  0x00002  DroppedContribution
12  0x01000  OutOfOrder
13  0x02000  OutOfSynch
14  0x04000  UserDefined
15  0x08000  IncompleteContribution
16  0x10000  ContainsIncomplete

Bit = n means 1<<n, i.e. 1 shifted left n bits:

                    Decimal     Binary
int32_t value =     1           00000000000000000000000000000001
                                |       |       |       |
DroppedContribution             00000000000000000000000000000010
OutOfOrder                      00000000000000000001000000000000
OutOfSynch                      00000000000000000010000000000000
UserDefined                     00000000000000000100000000000000
IncompleteContribution          00000000000000001000000000000000
ContainsIncomplete              00000000000000010000000000000000
```

Public Member Functions:

```
    // Constructor
    Damage(uint32_t v) : _damage(v) {}

    uint32_t  value() const             { return _damage; }
    void     increase(Damage::Value v)  { _damage |= ((1<<v) & NotUserBitsMask); }
    void     increase(uint32_t v)       { _damage |= v & NotUserBitsMask; }
    uint32_t bits() const               { return _damage & NotUserBitsMask;}
    uint32_t userBits() const           { return _damage >> UserBitsShift; }
    void     userBits(uint32_t v) {
      _damage &= NotUserBitsMask;
      _damage |= (v << UserBitsShift);
    }
```

Class TimeStamp

Public Types:

```
    enum {NumFiducialBits = 17};
    enum {MaxFiducials = (1<<17)-32};
    enum {ErrFiducial = (1<<17)-1};
```

Public Member Functions:

```
TimeStamp();
TimeStamp(const TimeStamp&);
TimeStamp(const TimeStamp&, unsigned control);
TimeStamp(unsigned ticks, unsigned fiducials, unsigned vector, unsigned control=0);

unsigned ticks    () const;
unsigned fiducials() const;
unsigned control  () const;
unsigned vector   () const;

TimeStamp& operator= (const TimeStamp&);
bool       operator==(const TimeStamp&) const;
bool       operator>=(const TimeStamp&) const;
bool       operator<=(const TimeStamp&) const;
bool       operator< (const TimeStamp&) const;
bool       operator> (const TimeStamp&) const;
```

Class TypeId

Public Types:

```
enum Type {
  Any,
  Id_Xtc,          // generic hierarchical container
  Id_Frame,        // raw image
  Id_AcqWaveform,
  Id_AcqConfig,
  Id_TwoDGaussian, // 2-D Gaussian + covariances
  Id_Opal1kConfig,
  Id_FrameFexConfig,
  Id_EvrConfig,
  Id_TM6740Config,
  Id_ControlConfig,
  Id_pnCCDframe,
  Id_pnCCDconfig,
  Id_Epics,        // Epics Data Type
  Id_FEEGasDetEnergy,
  Id_EBeam,
  Id_PhaseCavity,
  Id_PrincetonFrame,
  Id_PrincetonConfig,
  Id_EvrData,
  Id_FrameFccdConfig,
  Id_FccdConfig,
  Id_IpimbData,
  Id_IpimbConfig,
  Id_EncoderData,
  Id_EncoderConfig,
  Id_EvrIOConfig,
  Id_PrincetonInfo,
  Id_CspadElement,
  Id_CspadConfig,
  Id_IpmFexConfig,  // LUSI Diagnostics
  Id_IpmFex,
  Id_DiodeFexConfig,
  Id_DiodeFex,
  Id_PimImageConfig,
  NumberOf};
```

Public Member Functions:

```
    TypeId() {}
    TypeId(const TypeId& v);
    TypeId(Type type, uint32_t version);

    Type     id()      const;
    uint32_t version() const;
    uint32_t value()   const;

    static const char* name(Type type);
```