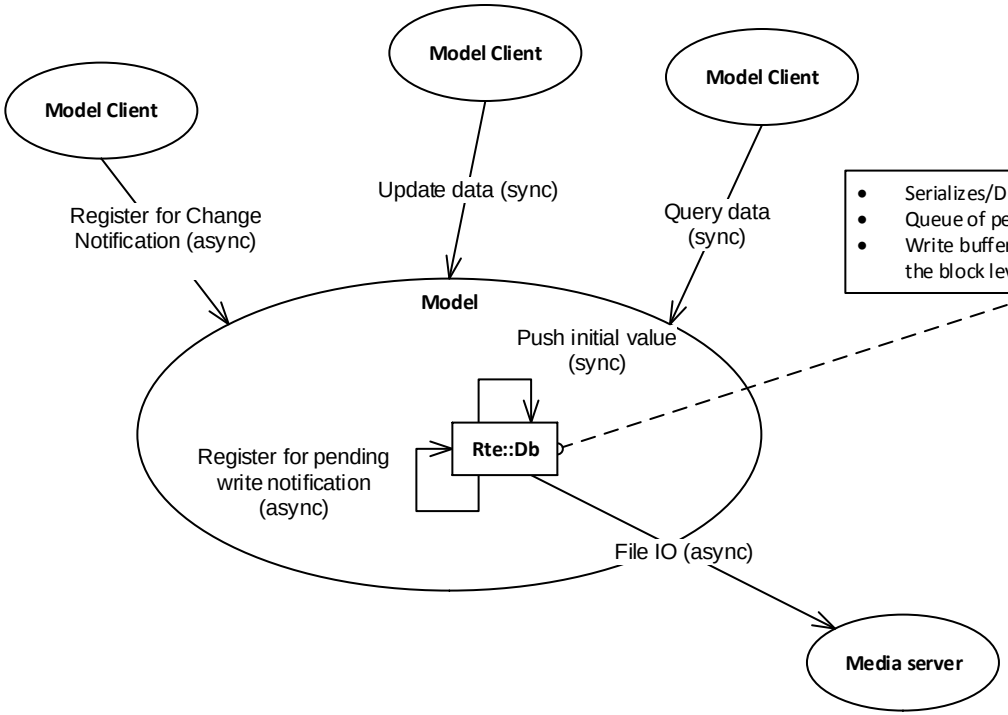
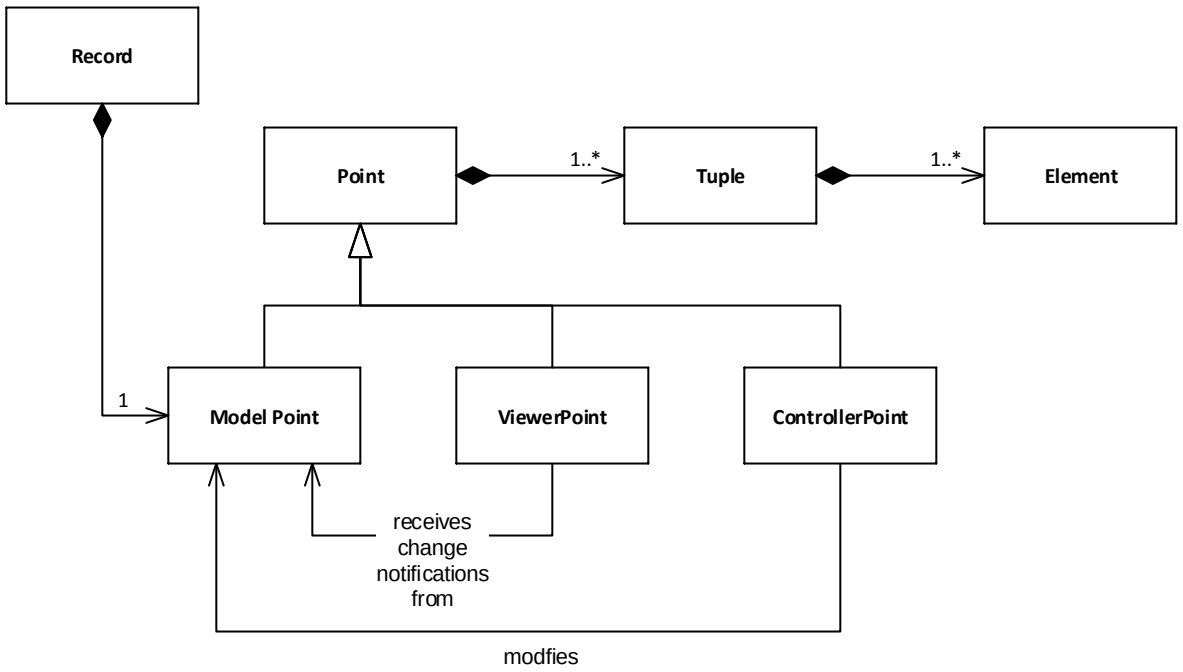
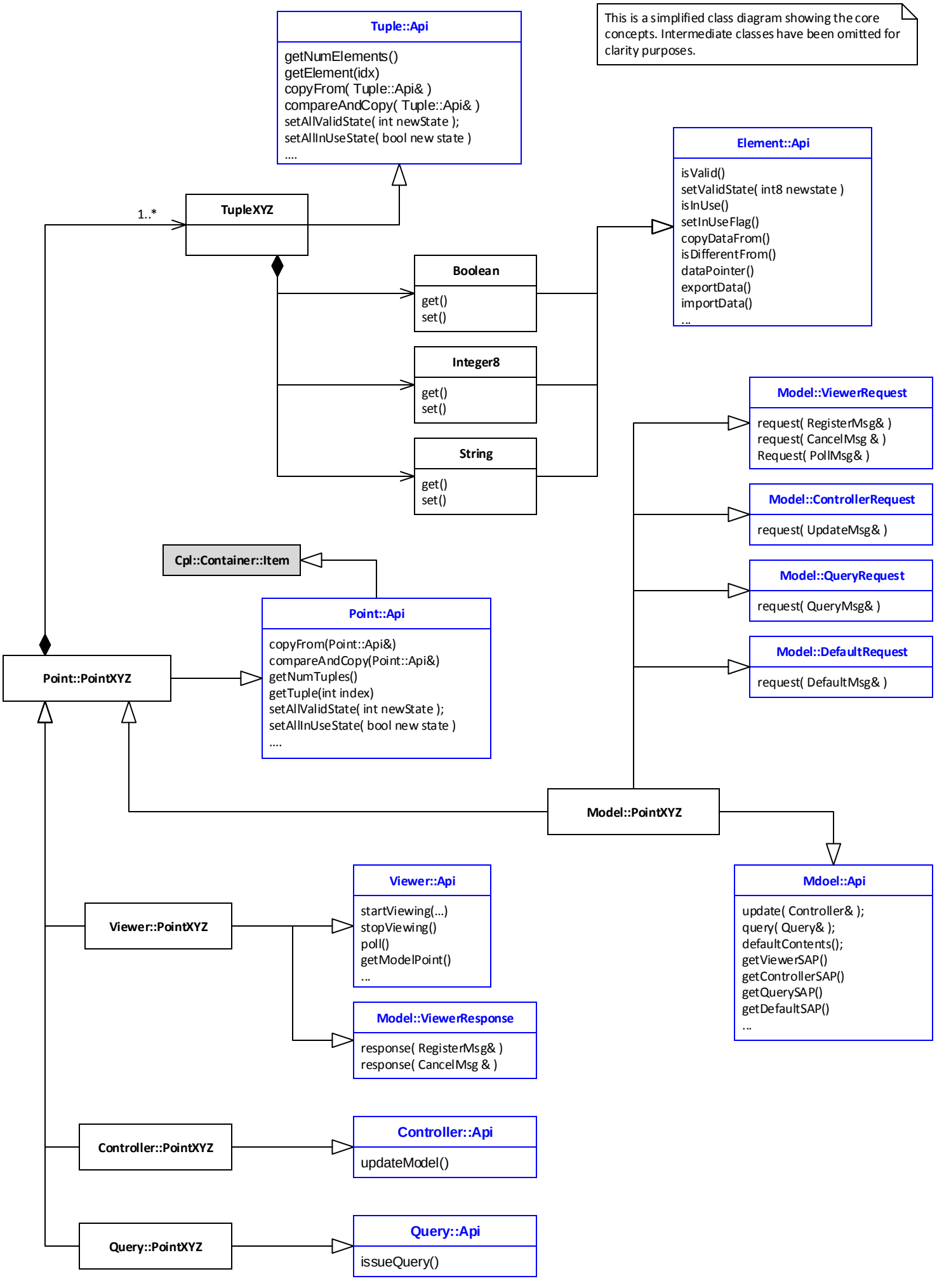
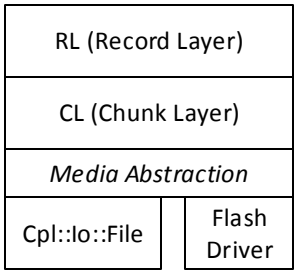


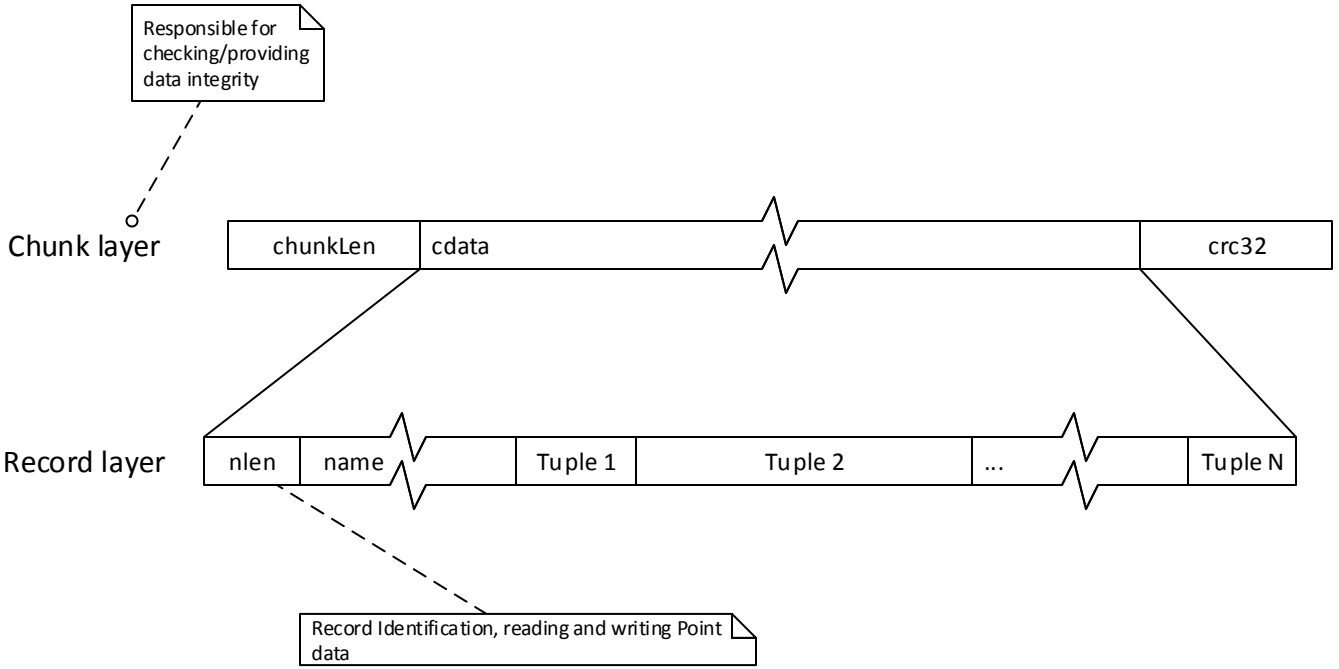
Versioning Rte Database:

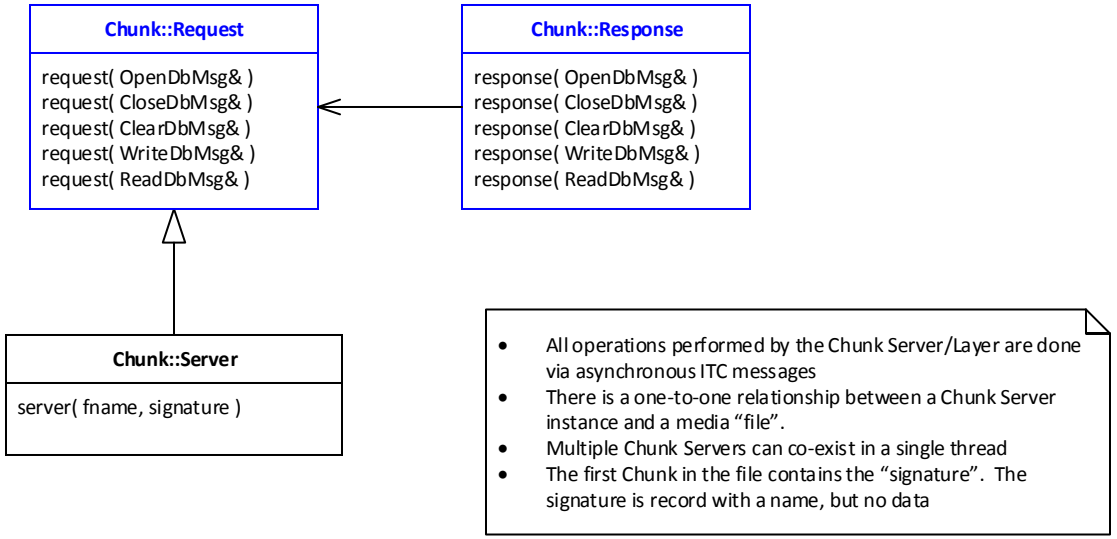
- A change in the major index requires a "external" conversion
- A change in the minor index is done "on the fly"
- The db file's signature name (text string) has the major index encoded in it, i.e. the RTE DB engine will not load a DB of with a non-matching 'major index'
- A change in the minor index can take 3 forms. All these forms are handled automatically by the Record layer.
  - Entirely new points and/or records being added.
  - APPENDING new data to an existing point.
    - This means: appending a new tuple to a Record, it does NOT mean inserting or changing existing tuple(s) in to an existing point in the Record.
  - Purging and/or deleting obsoleted Records.
- The 'minor index' is essential per Record/point – not a single 'global' minor index per DB concept.

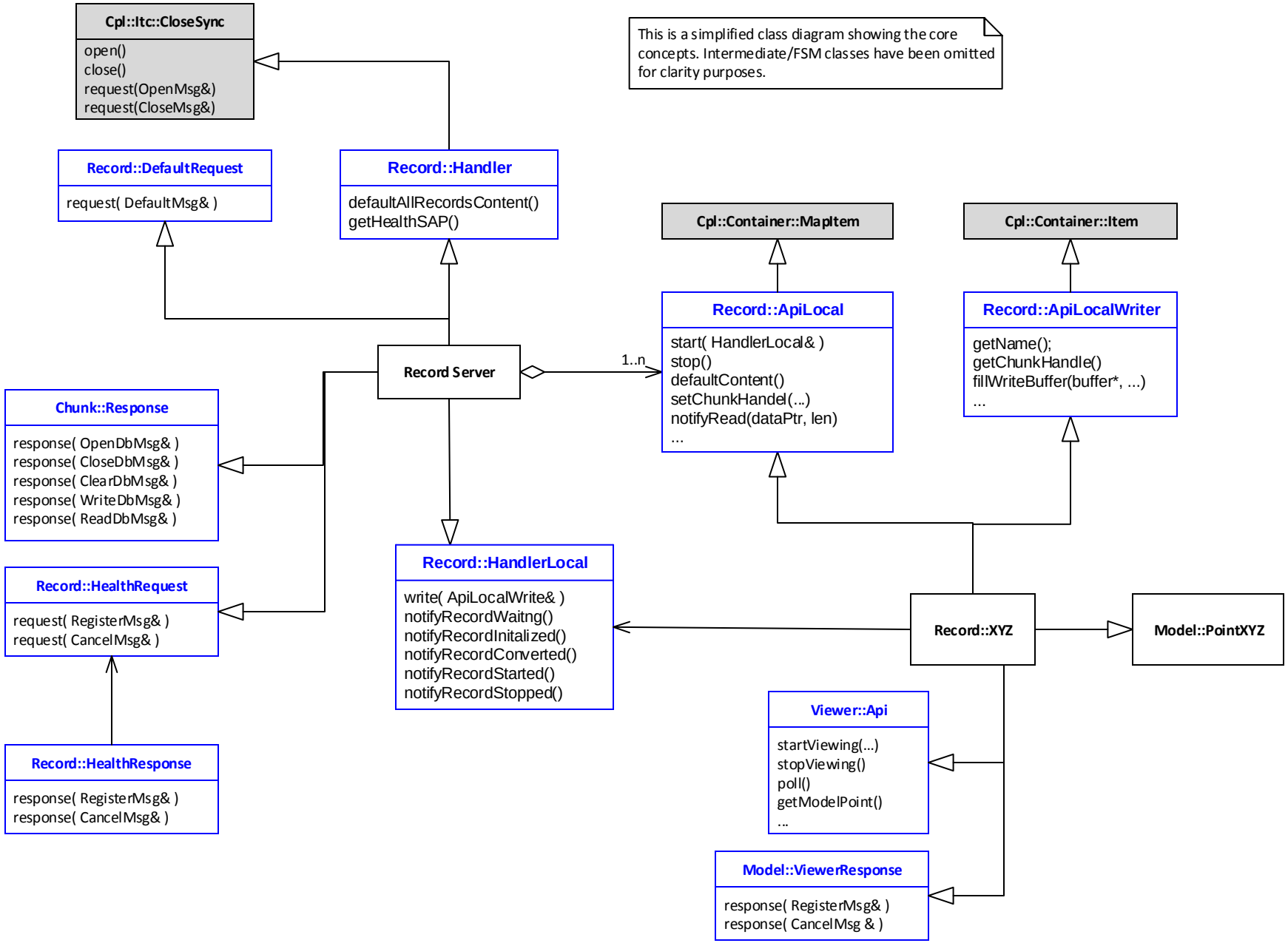



This is a simplified class diagram showing the core concepts. Intermediate classes have been omitted for clarity purposes.
Tuple::Api
getNumElements()
getElement(idx)
copyFrom(Tuple::Api&)
compareAndCopy(Tuple::Api&)
setAllValidState(int newState);
setAllInUseState(bool new state)
....
Element::Api
isValid()
setValidState(int8 newstate)
isInUse()
setInUseFlag()
copyDataFrom()
isDifferentFrom()
dataPointer()
exportData()
importData()
...
1..*
TupleXYZ
Boolean
get()
set()
Integer8
get()
set()
String
get()
set()
Model::ViewerRequest
request(RegisterMsg&)
request(CancelMsg &)
Request(PollMsg&)
Model::ControllerRequest
request(UpdateMsg&)
Model::QueryRequest
request(QueryMsg&)
Model::DefaultRequest
request(DefaultMsg&)
Cpl::Container::Item
Point::Api
copyFrom(Point::Api&)
compareAndCopy(Point::Api&)
getNumTuples()
getTuple(int index)
setAllValidState(int newState);
setAllInUseState(bool new state)
....
Point::PointXYZ
Model::PointXYZ
Mdoel::Api
update(Controller&);
query(Query&);
defaultContents();
getViewerSAP()
getControllerSAP()
getQuerySAP()
getDefaultSAP()
...
Viewer::Api
startViewing(...)
stopViewing()
poll()
getModelPoint()
...
Viewer::PointXYZ
Model::ViewerResponse
response(RegisterMsg&)
response(CancelMsg &)
Controller::PointXYZ
Controller::Api
updateModel()
Query::PointXYZ
Query::Api
issueQuery()


RL (Record Layer)
CL (Chunk Layer)
Media Abstraction
Cpl::Io::File
Flash Driver


Responsible for checking/providing data integrity
Chunk layer
chunkLen
cdata
crc32
Record layer
nlen
name
Tuple 1
Tuple 2
...
Tuple N
Record Identification, reading and writing Point data


Chunk::Request
request(OpenDbMsg&)
request(CloseDbMsg&)
request(ClearDbMsg&)
request(WriteDbMsg&)
request(ReadDbMsg&)
Chunk::Response
response(OpenDbMsg&)
response(CloseDbMsg&)
response(ClearDbMsg&)
response(WriteDbMsg&)
response(ReadDbMsg&)
Chunk::Server
server(fname, signature)
All operations performed by the Chunk Server/Layer are done via asynchronous ITC messages
There is a one-to-one relationship between a Chunk Server instance and a media “file”.
Multiple Chunk Servers can co-exist in a single thread
The first Chunk in the file contains the “signature”. The signature is record with a name, but no data


Cpl::Itc::CloseSync
open()
close()
request(OpenMsg&)
request(CloseMsg&)
This is a simplified class diagram showing the core concepts. Intermediate/FSM classes have been omitted for clarity purposes.
Record::DefaultRequest
request(DefaultMsg&)
Record::Handler
defaultAllRecordsContent()
getHealthSAP()
Cpl::Container::MapItem
Cpl::Container::Item
Record::ApiLocal
start(HandlerLocal&)
stop()
defaultContent()
setChunkHandel(...)
notifyRead(dataPtr, len)
...
Record::ApiLocalWriter
getName();
getChunkHandle()
fillWriteBuffer(buffer*, ...)
...
Record Server
1..n
Chunk::Response
response(OpenDbMsg&)
response(CloseDbMsg&)
response(ClearDbMsg&)
response(WriteDbMsg&)
response(ReadDbMsg&)
Record::HandlerLocal
write(ApiLocalWrite&)
notifyRecordWaitng()
notifyRecordInitalized()
notifyRecordConverted()
notifyRecordStarted()
notifyRecordStopped()
Record::HealthRequest
request(RegisterMsg&)
request(CancelMsg&)
Record::XYZ
Model::PointXYZ
Viewer::Api
startViewing(...)
stopViewing()
poll()
getModelPoint()
...
Record::HealthResponse
response(RegisterMsg&)
response(CancelMsg&)
Model::ViewerResponse
response(RegisterMsg&)
response(CancelMsg &)