CERN European Organization for Nuclear Research

Totem Offline Group

Totem Database Service - User Manual

Przemysław Dadel

Meyrin, September 2009 r.

Contents

1 Introduction

This document is intended to provide overview of Totem Database Service. While reading this document it is highly advisable to refer to Doxygen documentation[7] of Totem Database Service, and to get acquainted with documentation for TotODaM[10] Application.

Totem Database Service is a module in CMSSW that is responsible for providing access to databse with non-event data via Event Setup System[3].

This document is intended to be read by people with general understanding of CMS Framework[1][2] with good understanding of Event Setup System[3][4]

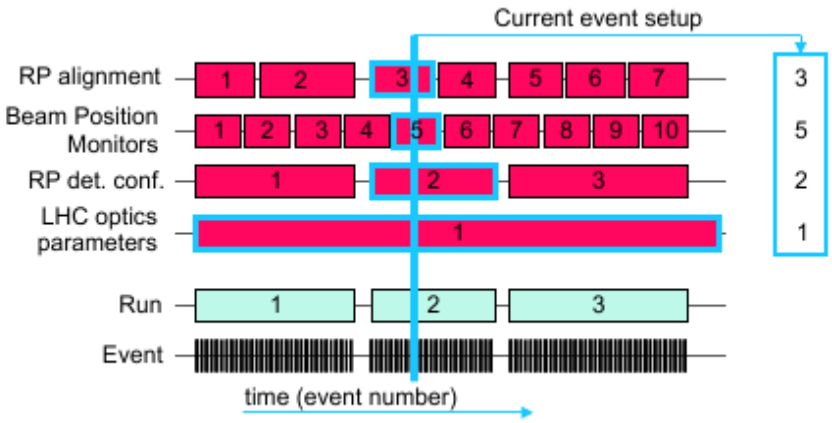


Figure 1: Event Setup System

2 Technology overview

Totem Database Service uses technologies, libraries, modules available in CMS Software. No external libraries are needed.

2.1 Oracle OCCI

Interaction with Oracle database from C++ code is achieved by using Oracle C++ Call Interface[5] (OCCI) - a high-performance and comprehensive API to access the Oracle database.

2.2 Oracle SQL Developer

SQL Developer[11] is convenient tool to manipulate databases. Some of its functionalities are:

- create connection with database
- define database schema using GUI creator for new tables, views.
- display data in tables.
- insert data to tables using GUI.

- run SQL statements, code completion is supported
- export database schema and data in database to files
- replication of entire/part of sch em in one database to another

2.3 Oracle Data Modeler

Oracle Data Modeler[12] is a tool for drawing ERD diagrams(graphical representation of database schema). These diagram can be drown manually or generated from database schema description file generated by SQL Developer

2.4 Problems

boost::posix_time Despite boost posix time[6] library is available in CMS Software libraries folder, I managed link code properly only by adding this flag to BuildFile of module using boost posix time:

```
<flags LDFLAGS="${CMSSW_RELEASE_BASE}/external/slc4_ia32_gcc345/lib/
    libboost_date_time-gcc34-mt-1_38.so" >
```

One should keep in mind of this dependency when porting CMS Framework to newer version

Totem Database Service was developed to work with CMS Software version 3_1_0.

3 Event Setup System

Section gives brief description of how Event Setup System works. More detailed description can be found in CMS Software documentation[3][4]

To be done.

4 Database structure

4.1 Naming conventions

- Structure Element - is a element of totem hardware that can be uniquely identified with a single identifier. e.g. RP Silicon Detector, RP Station, RP Arm. Structure Elements can from a tree-like hierarchy.
- Sensor Part - is the lowest level piece of totem software. Sensor Part is always a part of some Structure Element. Sensor Part is identified by a pair:
    1. Sensor Part Coordinate which is 1,2 or 3 number specifier of Sensor Part position in parent Structure Element.
    2. Type and identifier of parent Structure Element.
- Interval of Validity - describes for which range of event in given run is some data valid.

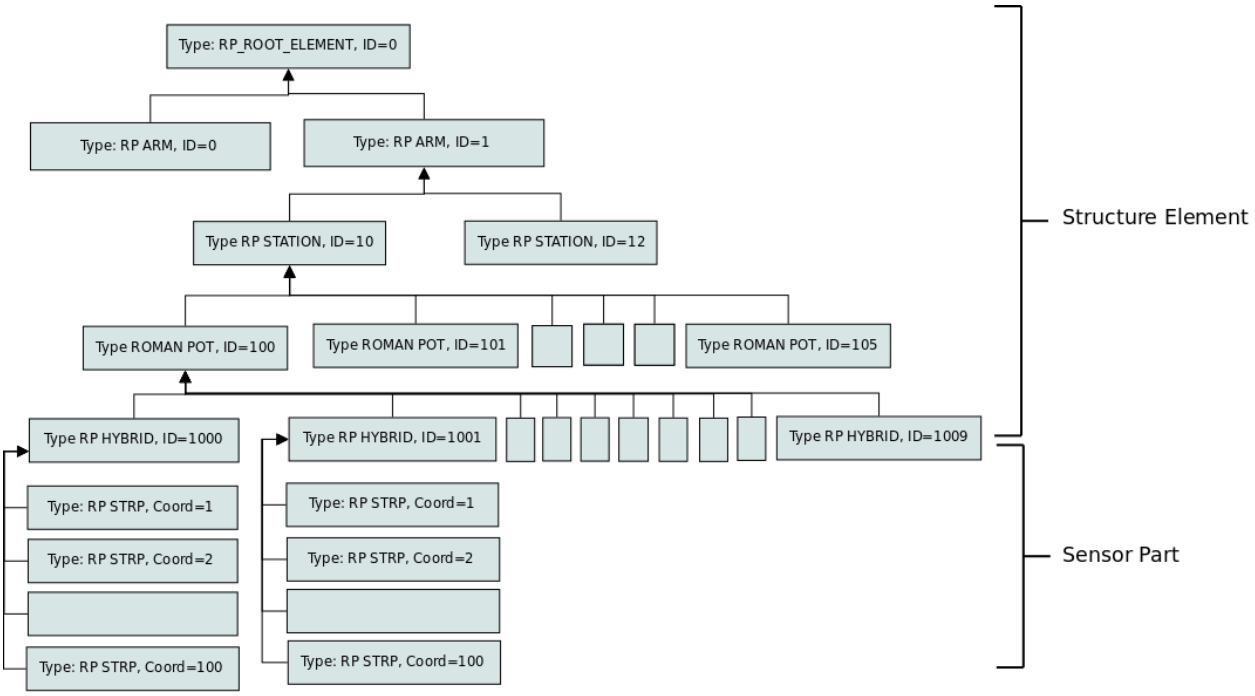


Figure 2: Example Structure Elements ans Sensor Part Hierarchy

4.2 Kinds of data stored in database

Database is intended to store different kind of data:

- **Group Data** - e.g RP Alignment.

  This is kind of data for storing numeric data for group of Structure Elements. These data are managed together, they have the same interval of validity, they are stored and retrieved as a whole.

- **General Measurements** - e.g. Air pressure in tunnel.

  This kind of data is designed for storing numeric data that is not associated with any Structure Element. It is ideal for storing global conditions or measurements for detector as a whole. In order to get a General Measurement one has to know the type of that measurement and type of validity interval for that type of measurement.

- **Structure Element Measurement**

  This kind of data is designed for storing numeric data that is associated with a certain Structure Element. It allows to store data that vary among Structure Elements of the same type. One can also query, according to hierarchy of Structure Elements, for measurements of all children Elements of a given parent Structure Element.

  In order to get a single Structure Element Measurement for a Structure Element one has to know the type of that measurement, type of Structure Element, and identifier of that element.

In order to get Structure Element Measurements for all children Elements of a given parent Structure Element one has to specify the type of that measurement, type of children Structure Element, type of parent Structure Element and identifier of parent element.

- **Sensor Part Measurement**

  This kind of data is designed for storing numeric data that is associated with a certain Sensor Part Element. Similarly to Structure Element Measurement, values can be retrieved for a single Sensor Part or ne can query for measurements of all direct children Sensor Parts of a given parent Structure Element.

  In order to get a single Sensor Part Measurement one has to know the type of that measurement, type of parent Structure Element, and identifier of parent element, type of Sensor Part Element, and coordinates of that part.

  In order to get a Sensor Part Measurements for all children Elements of a given **direct** parent Structure Element one has to know the type of that measurement, type of parent Structure Element, and identifier of parent element and type of Sensor Part Element.

4.3 Database schema description

Database schema description can be found in documentation for Totem Offline Database Monitor[10]

5 Using Totem Database Service

This section will explain how to configure connection to database.

5.1 Connecting to Oracle database

5.1.1 Python configuration file

Connection Parameters:

- user login name
- user password
- link to access db
- file with user defined SQL statements

has to be specified in python configuration ParameterSet object and then this object is included in configuration section for modules/plug-ins using the connection to database.

Connection configuration file

```
# -*- coding: utf-8 -*-
import os

#description on connection parameters
database_conf = cms.PSet(
    connection_name = cms.string("OracleConnection"),
```

```
    connection_user = cms.string("TOTEM"),
    connection_password = cms.string("Huber Niewiadowski knows the password to
        that database"),
    connection_link = cms.string("(DESCRIPTION = (ADDRESS = (PROTOCOL=TCP)(
        HOST=oradev10.cern.ch)(PORT=10520))(ENABLE=BROKEN) (CONNECT_DATA= (SID
        =D10) ) )"),
    verbosity = cms.untracked.uint32(1),
    connection_sql_statemnets = cms.string( os.environ['CMSSW_BASE'] + "/src/
        TotemDatabaseService/Sql/sql_statements.sql")
)

#example module that uses connection
process.RPAlignmentCorrectionsESSource = cms.ESSource("
    RPAlignmentCorrectionsESSource",
    database_connection = database_conf,
    verbosity = cms.untracked.uint32(1)
)
```

5.1.2 Establishing connection

The following listing will show how to establish connection from a c++ code.

Establishing connection

```
  try
  {
    boost::shared_ptr<OracleConnection> _connector;
    //pSet is ParameterSet object passed to a constructor of the module
    edm::ParameterSet conf = pSet.getParameterSet ("database_connection");
    //get name of the connection from conf object
    string connection_name = conf.getParameter<string > ("connection_name");
    try
    {
      //try to get the connection from ConnectionPool,
      //if connection is not found, exception is thrown
      _connector = OracleConnectionPool::getConnection (connection_name);
    }
    catch (OracleConnectionPool::ConnectionNotFoundException ex)
    {
      //create new connection
      _connector = boost::shared_ptr<OracleConnection > (new OracleConnection
          (conf));
      //register connection in connection pool
      OracleConnectionPool::registerConnection (_connector, connection_name);
    }
  }
  catch (...)
  {
    throw cms::Exception ("ModuleName") << "Oracle Connection initialization
        failed ";
```

```
}
```

5.2 User Defined SQL Statements

In order to separate SQL statements from c++ code, so that SQL code can be modified independently from c++ code, all SQL statements uses in the code are stored in an external files in form of a list of pair: statement_name SQL_code .

```
STATEMENT_NAME {
   SELECT SOME_COLUMN FROM SOME_TABLE
}
```

These User Defined SQL Statements are available in the code via OracleConnection interface[7].

5.3 Data Source Finders concept

Totem Database Service can provide user with data from different kind of sources like:

- database
- XML file
- etc.

Finder for a certain sort of data is class then implements interface for that sort of data and returns requested data from some source.
For example Finder for RP Strip Status is a class that implement SensorPartStatusFinder and reads a certain strip status from e.g database.

It is easy to write many finders for the same sort of data that read data from different sources and combine them into a single finder according to Composite[9] and Chain of Responsibility[8] object design patters.

For each kind data in database there is defined abstract finder class that describes interface to get this kind of data.

Used finders interfaces are:

- GroupDataFinder

  Returns Group data for requested parameters.

- DefaultGroupDataIDFinder

  Returns a Default GroupData identifier that distinguishes Group Data from other available Group Data in given Validity Interval.

- ValidityIntervalFinder

  Returns a validity of given type that contains given event.

- GeneralMeasurementFinder

  Returns General Measurement for requested parameters.

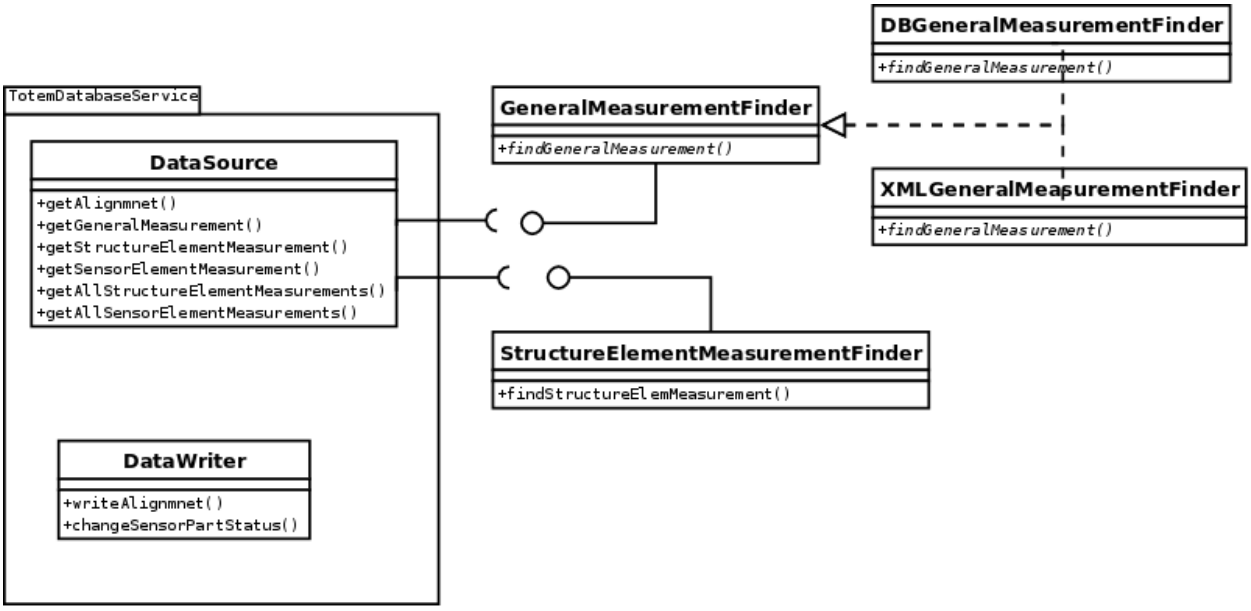


Figure 3: Concept of Data Finders

- StructureElementFinder

  Returns Structure Element Measurement for requested parameters.

- SensorPartMeasurementFinder

  Returns Sensor Part Measurement for requested parameters.

- SensorPartStatusFinder

  Returns Sensor Part Status for requested parameters.

There are implemented finder for each type of Finder that uses database as data source. The class implementing these interfaces is called DatabaseDataSourceFinder

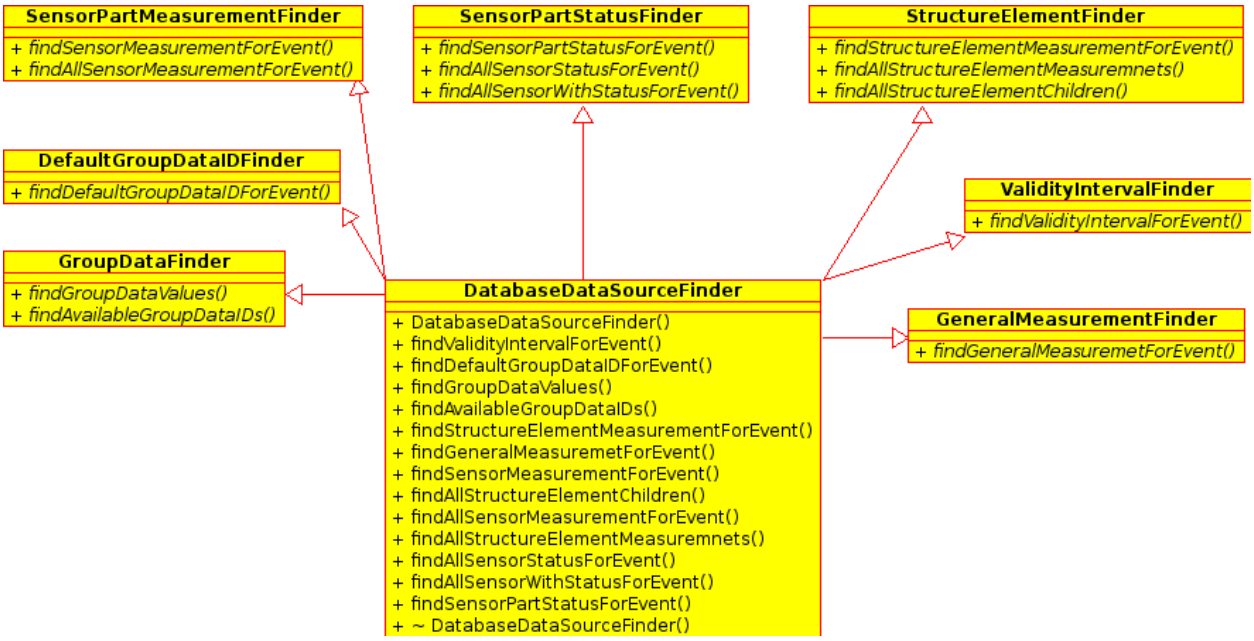
5.4 Developing ESProducer for accessing data

Access to non event data is interfaced in the code through Event Setup System[3] EventSetup object is available in the framework as argument of many methods of CMS Framework modules (e.g EDProducer produce() method EDAnalizer analyze() method), Data stored in EventSetup object is wrapped by EventSetupRecord. For every type of data there has to be defined separate EventSetupRecord. Particular EventSetupRecord is filled with data ESProducer plug-in for that kind of data. The same plug-in also manages Validity Interval of the provided data.

When developer wants to get access to new type of data of certain kind(General Measurement, StructureElementMeasurement etc.) edm::ESProducer[4] for that data must be written.

5.4.1 Group Data(Alignment)

This section will describe how to implement ESProducer for Group Data(Alignment).



The case study is ESProducer called RPAlignmentCorrectionsESSource for Roman Pot Alignment, that is already implemented in module *TotemDatabaseService/RPAlignmentESSource*

Initial conditions.

1. Define classes to hold RP Alignment. They are defined in *Geometry/TotemRPAlignment-DataFormats*

2. Define EventSetupRecord to hold RPAlignmentCorrections.

   Record is called RPAlignmentCorrectionsRecord and is defined in *TotemDatabaseService/-DataRecord*

3. Make sure that types of Validity Interval and Structure Elements for which you get Group Data are defined in database.

   It can be done using TotODaM(Totem Offline Database Monitor)application[10].

4. One probably should fill database with some data so that module can be tested;)

5. Implement ESProducer that fills RPAlignmentCorrectionsRecord with data.

Example ESProducer implementation

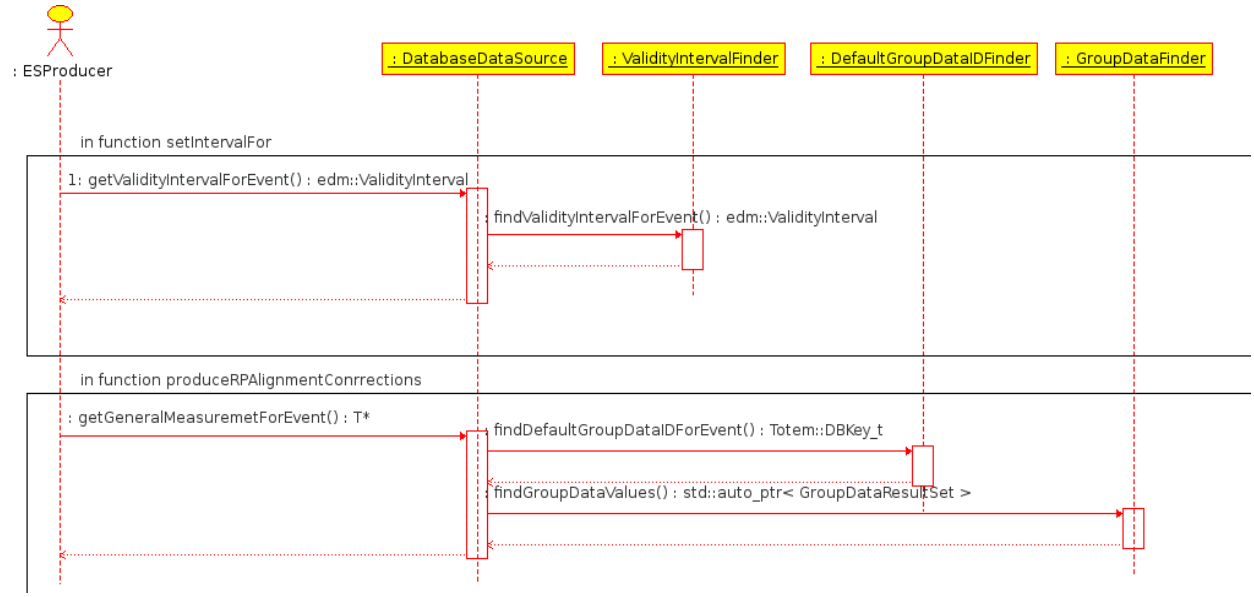


Figure 4: Sequence of calls to get GroupData(Alignment)

Header file for ESProducer

```
#define IOV_TYPE "RP_ALIGNMENT_IOV"
#define ELEMENT_TYPE "RP_HYBRID"

class RPAlignmentCorrectionsESSource: public edm::ESProducer, public edm::
    EventSetupRecordIntervalFinder
{
public:

  RPAlignmentCorrectionsESSource (const edm::ParameterSet &);
  ~RPAlignmentCorrectionsESSource ();

  /**
   * method that will fill object of RPAlignmentCorrectionsRecord class (which
       is derived from EventSetupRecord)
   * with RPAlignmentCorrections
   */
  virtual std::auto_ptr<RPAlignmentCorrections> produceRPAlignmentCorrections
      (const RPAlignmentCorrectionsRecord &);

private:
  unsigned int verbosity;
  boost::shared_ptr<DatabaseDataSource> dataSource;

protected:
  /**
   * method that will set ValidityInterval of RPAlignmentCorrections in
```

```
      RPAlignmentCorrectionsRecord
   */
  virtual void setIntervalFor (const edm::eventsetup::EventSetupRecordKey&,
      const edm::IOVSyncValue&, edm::ValidityInterval&);
};
```

Implementation file for ESProducer

```
RPAlignmentCorrectionsESSource::RPAlignmentCorrectionsESSource (const edm::
    ParameterSet& pSet) :
verbosity (pSet.getUntrackedParameter<unsigned int>("verbosity", 0))
{
  try
  {
    //initializing connection
    boost::shared_ptr<OracleConnection> _connector;
    _connector = ... //initialize  connection

    //creating DataSource object
    dataSource = boost::shared_ptr<DatabaseDataSource > (new
        DatabaseDataSource (_connector));
    boost::shared_ptr<DatabaseDataSourceFinder> finder =
            boost::shared_ptr<DatabaseDataSourceFinder > (new
                DatabaseDataSourceFinder (_connector));

    //registering finders necessary to get GroupData(Alignment)
    dataSource->setGroupDataFinder (finder);
    dataSource->setGroupDataDefaultIDFinder (finder);
    dataSource->setValidityIntervalFinder (finder);
  }
  catch (...)
  {
    throw cms::Exception ("RPAlignmentCorrectionsESSource") << "
        RPAlignmentCorrectionsESSource initialization failed ";
  }

  // set that method produceRPAlignmentCorrections fills
      RPAlignmentCorrectionsRecord with RPAlignmentCorrections
  setWhatProduced (this, &RPAlignmentCorrectionsESSource::
      produceRPAlignmentCorrections);
  // set that this ESProducer produces RPAlignmentCorrectionsRecord
  findingRecord<RPAlignmentCorrectionsRecord > ();
}

void RPAlignmentCorrectionsESSource::setIntervalFor (const edm::eventsetup::
    EventSetupRecordKey& key, const edm::IOVSyncValue& iosv, edm::
    ValidityInterval& oValidity)
{
  //this is needed to prevent occuring problem when last event of
      ValidityInterval is greater than last process event.
```

```
  if (iosv.eventID().event() == IOVSyncValue::endOfTime().eventID().event
      ())
  {
    oValidity = ValidityInterval(iosv, iosv);
  }
  else
  {
    if (key == RPAlignmentCorrectionsRecord::keyForClass())
    {
      //using dataSource to get ValidityInterval of type IOV_TYPE
      ValidityInterval vi = dataSource->getValidityIntervalForEvent(iosv.
          eventID(), IOV_TYPE);
      oValidity = ValidityInterval(iosv, vi.last());
    }
  }
}

std::auto_ptr<RPAlignmentCorrections> RPAlignmentCorrectionsESSource::
    produceRPAlignmentCorrections(const RPAlignmentCorrectionsRecord & record
    )
{
  //extract ValidityInterval of RPAlignmentCorrectionsRecord
  ValidityInterval vi = record.validityInterval();
  //get RPAlignmentCorrections for current ValidityInterval(represented by
      first event of ValidityInterval of IOV_TYPE)
  //for element type = ELEMENT_TYPE
  auto_ptr<RPAlignmentCorrections> data(dataSource->getGroupDataForEvent<
      RPAlignmentCorrections>(vi.first().eventID(), IOV_TYPE, ELEMENT_TYPE
      ));
  return data;
}
//defines RPAlignmentCorrectionsESSource as a plugin of CMS Framework
DEFINE_ANOTHER_FWK_EVENTSETUP_SOURCE(RPAlignmentCorrectionsESSource);
```

5.4.2 General Measurement

This section will describe how to implement ESProducer for General Measurement. The case study is ESProducer called GeneralMeasurementESSource for some Example General Measurement, that is implemented in module *TotemDatabaseService/ExampleESSource*

Initial conditions.

1. Define class to hold Example General Measurement.

   Class called **ExampleGeneralMeasurement** is defined in *TotemDatabaseService/ExampleESSource*

2. Define EventSetupRecord to hold ExampleGeneralMeasurement.

   Record called **ExampleGeneralMeasurementRecord** is defined in *TotemDatabaseService/ExampleESSource*

3. Make sure that types of ValidityInterval and Measurement type are defined in database. It can be done using TotODaM(Totem Offline Database Monitor)application[10].
4. One probably should fill database with some data so that module can be tested;)
5. Implement ESProducer that fills ExampleGeneralMeasurementRecord with data.

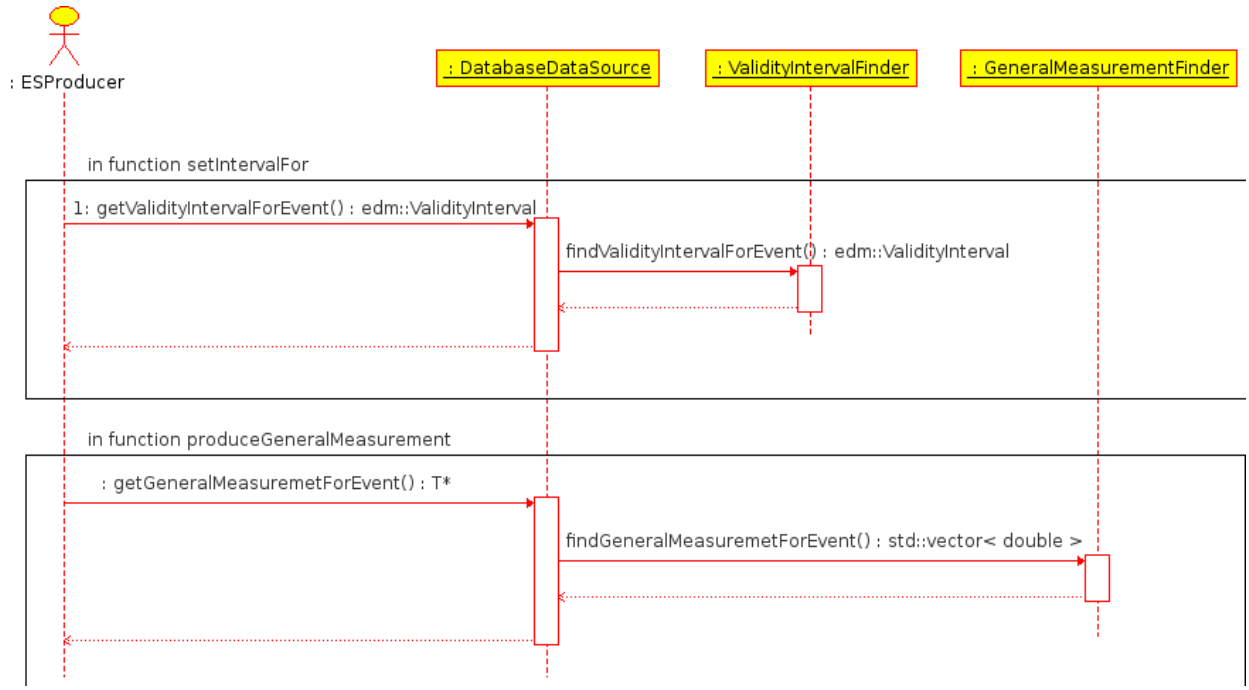


Figure 5: Sequence of method calls to retrieve General Measurement

Header file for ESProducer

```
class GeneralMeasurementESSource : public edm::ESProducer, public edm::
    EventSetupRecordIntervalFinder
{
public:

  GeneralMeasurementESSource (const edm::ParameterSet &);
  ~GeneralMeasurementESSource ();

  virtual std::auto_ptr<ExampleGeneralMeasurement>
      produceExampleGeneralMeasurement (const ExampleGeneralMeasurementRecord
      &);

private:
  unsigned int verbosity;
  boost::shared_ptr<DatabaseDataSource> dataSource;
```

```
protected:
  virtual void setIntervalFor (const edm::eventsetup::EventSetupRecordKey&,
      const edm::IOVSyncValue&, edm::ValidityInterval&);
};
```

Implementation of ESProducer

```
GeneralMeasurementESSource::GeneralMeasurementESSource (const edm::
    ParameterSet& pSet) :
verbosity (pSet.getUntrackedParameter<unsigned int>("verbosity", 0))
{
  try
  {
    //initializing connection
    boost::shared_ptr<OracleConnection> _connector;
    _connector = ... //initialize  connection

    dataSource = boost::shared_ptr<DatabaseDataSource > (new
        DatabaseDataSource (_connector));
    boost::shared_ptr<DatabaseDataSourceFinder> finder =
            boost::shared_ptr<DatabaseDataSourceFinder > (new
                DatabaseDataSourceFinder (_connector));

    dataSource->setGeneraltMeasurementFinder (finder);
    dataSource->setValidityIntervalFinder (finder);
  }
  catch (...)
  {
    throw cms::Exception ("GeneralMeasurementESSource") << "
        GeneralMeasurementESSource initialization failed ";
  }

  setWhatProduced (this, &GeneralMeasurementESSource::
      produceExampleGeneralMeasurement);
  findingRecord<ExampleGeneralMeasurementRecord > ();
}

//
   ____________________________________________________________________________


void GeneralMeasurementESSource::setIntervalFor (const edm::eventsetup::
    EventSetupRecordKey& key, const edm::IOVSyncValue& iosv, edm::
    ValidityInterval& oValidity)
{
  if (iosv.eventID ().event () == IOVSyncValue::endOfTime ().eventID ().event
      ())
  {
    oValidity = ValidityInterval (iosv, iosv);
  }
  else
```

```
  {
    if (key == ExampleGeneralMeasurementRecord::keyForClass ())
    {
      ValidityInterval vi = dataSource->getValidityIntervalForEvent (iosv.
          eventID (), "LHC_TEMP_IOV");
      oValidity = ValidityInterval (iosv, vi.last ());
    }
  }
}

std::auto_ptr<ExampleGeneralMeasurement> GeneralMeasurementESSource::
    produceExampleGeneralMeasurement (const ExampleGeneralMeasurementRecord &
    record)
{
  edm::EventID event = record.validityInterval ().first ().eventID ();
  return std::auto_ptr<ExampleGeneralMeasurement > (dataSource->
      getGeneralMeasuremetForEvent<ExampleGeneralMeasurement > (event, "
      ExampleMeasuremntType", "ExampleGeneralMeasurement_IOV"));
}

DEFINE_ANOTHER_FWK_EVENTSETUP_SOURCE (GeneralMeasurementESSource);
```

5.4.3 Available Group Data(Alignment)

This section will describe how to implement ESProducer for that will produce object that will allow to list all available Group Data(Alignments) for given Validity Interval.

The case study is ESProducer called RPAlignmentCorrectionsESSource(in TotemDatabaseService/RPAlignmentESSource) for Roman Pot Alignment. ESProducer produces RPAlignmentDBReader(in TotemDatabaseService/DBUserInterface) object that has method that allows to list available Group Data for given Validity Interval. Record for that class is called RPAlignmentDBReaderRecord(in TotemDatabaseService/DataRecord)

Initial conditions for element,iov types definitions are the same as for ESProducer for GroupData

Header file for ESProducer

```
class RPAlignmentCorrectionsESSource : public edm::ESProducer, public edm::
    EventSetupRecordIntervalFinder
{
public:

  RPAlignmentCorrectionsESSource (const edm::ParameterSet &);
  ~RPAlignmentCorrectionsESSource ();

  virtual std::auto_ptr<RPAlignmentDBReader> produceDBReader (const
      RPAlignmentDBReaderRecord &);

private:
  unsigned int verbosity;
  boost::shared_ptr<DatabaseDataSource> dataSource;
```

```
protected:
  virtual void setIntervalFor (const edm::eventsetup::EventSetupRecordKey&,
     const edm::IOVSyncValue&, edm::ValidityInterval&);
};
```

Implementation of ESProducer

```
RPAlignmentCorrectionsESSource::RPAlignmentCorrectionsESSource (const edm::
    ParameterSet& pSet) :
verbosity (pSet.getUntrackedParameter<unsigned int>("verbosity", 0))
{
  try
  {
    boost::shared_ptr<OracleConnection> _connector;

    dataSource = boost::shared_ptr<DatabaseDataSource > (new
        DatabaseDataSource (_connector));
    boost::shared_ptr<DatabaseDataSourceFinder> finder =
            boost::shared_ptr<DatabaseDataSourceFinder > (new
                DatabaseDataSourceFinder (_connector));

    dataSource->setGroupDataFinder (finder);
    dataSource->setGroupDataDefaultIDFinder (finder);
    dataSource->setValidityIntervalFinder (finder);
  }
  catch (...)
  {
    throw cms::Exception ("RPAlignmentCorrectionsESSource") << "
        RPAlignmentCorrectionsESSource initialization failed ";
  }

  setWhatProduced (this, &RPAlignmentCorrectionsESSource::produceDBReader);
  findingRecord<RPAlignmentDBReaderRecord > ();
}

void RPAlignmentCorrectionsESSource::setIntervalFor (const edm::eventsetup::
    EventSetupRecordKey& key, const edm::IOVSyncValue& iosv, edm::
    ValidityInterval& oValidity)
{
   /**
    * the ValidityInterval is set ot infinity because the returned
    * object does not store data but rather then provides some service to read
        data.
    */
   oValidity = ValidityInterval (IOVSyncValue::beginOfTime (), IOVSyncValue::
       endOfTime ());
}

std::auto_ptr<RPAlignmentDBReader> RPAlignmentCorrectionsESSource::
    produceDBReader (const RPAlignmentDBReaderRecord &)
{
```

```
  return std::auto_ptr<RPAlignmentDBReader > (new RPAlignmentDBReader (
      dataSource));
}
```

5.4.4 Structure Element Measurement

This section will describe how to implement ESProducer for Structure Element Measurement. The case study is ESProducer called StructureMeasurementESSource for example data type RPStation12ExampleMeasurement that represents some Example Measurement for RP Hybrids(Silicon Detectors) in RP Stations 12.

StructureMeasurementESSource implemented in module *TotemDatabaseService/ExampleESSource*

Initial conditions.

1. Define class to hold Structure Element Measurement.

   Class called **RPStation12ExampleMeasurement** is defined in *TotemDatabaseService/ExampleESSource*

2. Define EventSetupRecord to hold RPStation12ExampleMeasurement.

   Record called **ExampleStructureMeasurementRecord** is defined in *TotemDatabaseService/ExampleESSource*

3. Make sure that Validity Interval type, Measurement type, and necessary Structure Elements are defined in database.

   It can be done using TotODaM(Totem Offline Database Monitor)application[10].

4. One probably should fill database with some data so that module can be tested;)

5. Implement ESProducer that fills ExampleStructureMeasurementRecord with data.

Header file for ESProducer

```
class StructureMeasurementESSource : public edm::ESProducer, public edm::
    EventSetupRecordIntervalFinder
{
public:

  StructureMeasurementESSource (const edm::ParameterSet &);
  ~StructureMeasurementESSource ();

  virtual std::auto_ptr<RPStation12ExampleMeasurement> produceRPStat12ExMeas (
      const ExampleStructureMeasurementRecord&);

private:
  unsigned int verbosity;
  boost::shared_ptr<DatabaseDataSource> dataSource;
```

```
  typedef std::auto_ptr<RPStation12ExampleMeasurement>
      RPStation12ExampleMeasurementPtr;
  RPStation12ExampleMeasurementPtr _temp_rp12_measurement;

protected:
  virtual void setIntervalFor (const edm::eventsetup::EventSetupRecordKey&,
      const edm::IOVSyncValue&, edm::ValidityInterval&);
};
```

Implementation of ESProducer

```
StructureMeasurementESSource::StructureMeasurementESSource (const edm::
    ParameterSet& pSet) :
verbosity (pSet.getUntrackedParameter<unsigned int>("verbosity", 0))
{
  try
  {
    //initializing connection
    boost::shared_ptr<OracleConnection> _connector;
    _connector = ... //initialize  connection

    //creating DataSource object
    dataSource = boost::shared_ptr<DatabaseDataSource > (new
        DatabaseDataSource (_connector));
    boost::shared_ptr<DatabaseDataSourceFinder> finder =
            boost::shared_ptr<DatabaseDataSourceFinder > (new
                DatabaseDataSourceFinder (_connector));

    //set Structure Element Finder
    dataSource->setStructureElementFinder (finder);
  }
  catch (...)
  {
    throw cms::Exception ("StructureMeasurementESSource") << "
        StructureMeasurementESSource initialization failed ";
  }

  // set that method produceRPStat12ExMeas fills
      ExampleStructureMeasurementRecord with RPStation12ExampleMeasurement
  setWhatProduced (this, &StructureMeasurementESSource::produceRPStat12ExMeas)
      ;
  // set that this ESProducer produces ExampleStructureMeasurementRecord
  findingRecord<ExampleStructureMeasurementRecord > ();
}

void StructureMeasurementESSource::setIntervalFor (const edm::eventsetup::
    EventSetupRecordKey& key, const edm::IOVSyncValue& iosv, edm::
    ValidityInterval& oValidity)
{
  if (iosv.eventID ().event () == IOVSyncValue::endOfTime ().eventID ().event
      ())
```

```
  {
    oValidity = ValidityInterval (iosv, iosv);
  }
  else
  {
    if (key == ExampleStructureMeasurementRecord::keyForClass ())
    {
      const std::string measurement_type = "RP_SOME_DATA";
      const std::string parent_element_type = "RP_STATION";
      const std::string parent_element_id = "12";
      const std::string element_type = "RP_SILICON_DETECTOR";

      std::pair< RPStation12ExampleMeasurement*, edm::ValidityInterval> p =
              dataSource->getAllStructureElementMeasurements<
                  RPStation12ExampleMeasurement > (iosv.eventID (),
                  measurement_type, parent_element_type, element_type,
                  parent_element_id);

      oValidity = p.second;
      //RPStation12ExampleMeasurement is stored to be returned in
          produceRPStat12ExMeas method
      _temp_rp12_measurement = RPStation12ExampleMeasurementPtr (p.first);
    }
  }
}
//returns a RPStation12ExampleMeasurement retrieved from database together
    with ValidityInterval for that data.
std::auto_ptr<RPStation12ExampleMeasurement> StructureMeasurementESSource::
    produceRPStat12ExMeas (const ExampleStructureMeasurementRecord & record)
{
  if (_temp_rp12_measurement.get () != NULL)
    return _temp_rp12_measurement;
  else throw cms::Exception ("StructureMeasurementESSource") << "No valid
      data for record ExampleStructureMeasurementRecord";
}

DEFINE_ANOTHER_FWK_EVENTSETUP_SOURCE (StructureMeasurementESSource);
```

5.4.5 Sensor Part Measurement

This section will describe how to implement ESProducer for Sensor Part Measurement . The case study is ESProducer called SensorPartMeasurementESSource for example data type RP1207StripThreshold that represents Threshold for RP Strips of Roman Pot 1207.

SensorPartMeasurementESSource implemented in module *TotemDatabaseService/ExampleESSource*

Initial conditions.

1. Define class to hold Strip Threshold.

   Class called **RP1207StripThreshold** is defined in *TotemDatabaseService/ExampleESSource*

2. Define EventSetupRecord to hold RP1207StripThreshold.

   Record called **ExampleSensorMeasurementRecord** is defined in *TotemDatabaseService/ExampleESSource*

3. Make sure that Validity Interval type, Measurement type, and necessary Structure Elements and Sensor Parts are defined in database.

   It can be done using TotODaM(Totem Offline Database Monitor)application[10].

4. One probably should fill database with some data so that module can be tested;)

5. Implement ESProducer that fills ExampleStructureMeasurementRecord with data.

Implementation ESProducer is implemented in *TotemDatabaseService/ExampleESSource*

5.4.6 Sensor Part Status

ESProducer for Sensor Part Status is almost in all ways identical with ESProducer for Sensor Part Measurement. The difference is that you do not specify Measurement type and the returned value for some SensorPart is not a vector of doubles but a string describing Sensor Part Status.

Additional Initial condition is that status types has to be defined in database.

Example files implementing ESProducer for Sensor Status are in *TotemDatabaseService/ExampleESSource* Example ESProducer produces status of Strips in RP Hybrid(Silicon Detector) 1207 These files are:

- ExampleRP1207StripsStatus - class for storing status for Sensor Part Elements.
- ExampleSensorStatusRecord - record for holding ExampleRP1207StripsStatus
- SensorStatusESSource - ESProducer

5.5 Developing ESProducer for writing data

5.5.1 Group Data

This section will describe how to implement ESProducer for that will produce object that will allow to insert Group Data(Alignments) to database.

The case study is ESProducer called RPAlignmentCorrectionsESSource(in TotemDatabaseService/RPAlignmentESSource) for Roman Pot Alignment. ESProducer produces RPAlignmentDBWriter(in TotemDatabaseService/DBUserInterface) object that allows to insert data to database. Record for that class is called RPAlignmentDBWriter(in TotemDatabaseService/DataRecord)

Initial conditions for element,iov types definitions are the same as for ESProducer for GroupData

Header file for ESProducer

```
class RPAlignmentCorrectionsESSource : public edm::ESProducer, public edm::
    EventSetupRecordIntervalFinder
{
public:
```

```
  RPAlignmentCorrectionsESSource (const edm::ParameterSet &);
  ~RPAlignmentCorrectionsESSource ();

  virtual std::auto_ptr<RPAlignmentDBWriter> produceDBWriter (const
      RPAlignmentDBWriterRecord &);

private:
  unsigned int verbosity;
  boost::shared_ptr<DatabaseDataWriter> dataWriter;

protected:
  virtual void setIntervalFor (const edm::eventsetup::EventSetupRecordKey&,
      const edm::IOVSyncValue&, edm::ValidityInterval&);
};
```

Implementation of ESProducer

```
RPAlignmentCorrectionsESSource::RPAlignmentCorrectionsESSource (const edm::
    ParameterSet& pSet) :
verbosity (pSet.getUntrackedParameter<unsigned int>("verbosity", 0))
{
  try
  {
    boost::shared_ptr<OracleConnection> _connector;
    dataWriter = boost::shared_ptr<DatabaseDataWriter > (new
        DatabaseDataWriter (_connector));
  }
  catch (...)
  {
    throw cms::Exception ("RPAlignmentCorrectionsESSource") << "
        RPAlignmentCorrectionsESSource initialization failed ";
  }

  setWhatProduced (this, &RPAlignmentCorrectionsESSource::produceDBWriter);
  findingRecord<RPAlignmentDBWriterRecord > ();
}

void RPAlignmentCorrectionsESSource::setIntervalFor (const edm::eventsetup::
    EventSetupRecordKey& key, const edm::IOVSyncValue& iosv, edm::
    ValidityInterval& oValidity)
{
   /**
    * the ValidityInterval is set ot infinity because the returned
    * object does not store data but rather then provides some service to
        write data.
    */
   oValidity = ValidityInterval (IOVSyncValue::beginOfTime (), IOVSyncValue::
       endOfTime ());
}
```

```
std::auto_ptr<RPAlignmentDBWriter> RPAlignmentCorrectionsESSource::
    produceDBWriter (const RPAlignmentDBWriterRecord &)
{
  return std::auto_ptr<RPAlignmentDBWriter > (new RPAlignmentDBWriter (
      dataWriter));
}
```

5.5.2 Sensor Part Status

Similar to RPAlignmentDBWriter there can be implemented SenorPartStatusWriter that will forward changeStatus method calls to DatabaseDataWriter method *changeSensorPartStatusForIOV*

5.6 Reading data from EventSetup

This section describes how to extract data from EventSetupRecord. This example will show Extracting RPAlignmentCorrections from RPAlignmentCorrectionsRecord.

Extracting data from EventSetup

```
//EventSetup es object is provided by CMS Framework
ESHandle<RPAlignmentCorrections> handle
es.get<RPAlignmentCorrectionsRecord>().get(handle);
handle-> //ESHandle<RPAlignmentCorrections> work like pointer to
    RPAlignmentCorrections
```

5.7 Listing available Group Data(Alignments)

List available Alignments

```
ESHandle<RPAlignmentDBReader > reader;
es.get<RPAlignmentDBReaderRecord > ().get (reader);
RPAlignmentDBReader const& read = *(reader.product ());
map<DBKey_t, shared_ptr<RPAlignmentCorrections> > avail =
  reader->getAvailableGroupData<RPAlignmentCorrections > (vi, "
      RP_ALIGNMENT_IOV", "RP_HYBRID");
```

5.8 Inserting Group Data to Database

Inserting Alignments to Database

```
ESHandle<RPAlignmentDBWriter> handle
es.get<RPAlignmentDBWriterRecord>().get(handle);
GroupDataWriter writer = handle->createGroupDataWriter("RP_HYBRID");
writer.setDescription("Test_alignment")

//set Alignment for a RP HYBRID 1207
```

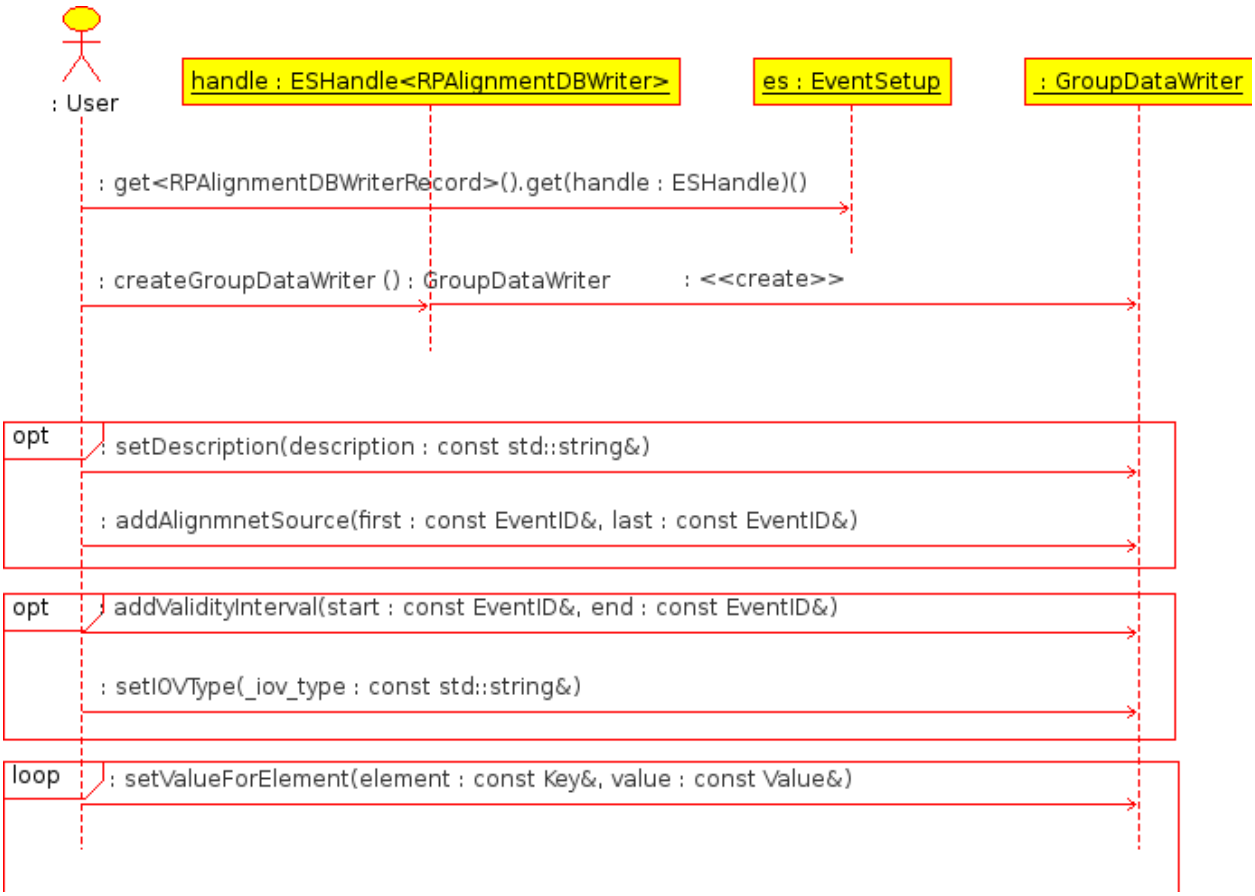


Figure 6: Sequence diagram for writing alignment

```
writer.setValueForElement<std::string, RPAlignmentCorrection> ("1207",
    RPAlignmentCorrection(1,2,13));
//set Alignment for a RP HYBRID 1208
writer.setValueForElement<std::string, RPAlignmentCorrection> ("1208",
    RPAlignmentCorrection(2,3,13));
writer.setValueForElement ...

//set on basis of which events was Alignment computed,
writer.addAlignmnetSource (EventID (1, 0), EventID (1, 200));
writer.addAlignmnetSource (EventID (2, 0), EventID (2, 100));

//optional, set type of interval associated with given Alignment
writer.setIOVType ("RP_SILICON_DETECTOR_ALIGNMENT_IOV");
//optional, associated Alignment with ValidityInterval
writer.addValidityInterval (EventID (3, 0), EventID (3, 1000));
//writes data to database
writer.commit ();
```

5.9 DatabaseDataSource Interface

DatabaseDataSource Interface is described in Doxygen for TotemDatabaseService

6 Code Structure

In TotemDatabaseService directory there are modules of TotemDatabaseService
These modules are:

1. Connection

   Module contains classes that manages connection with Oracle database

2. DatabaseCommon

   Module contains classes that provide common functionalities for DataSource and DataWriter modules.

3. DataRecord

   Here are stored EventSetupRecords for produced data.

4. DataSource

   Module contains classes that are responsible for getting data from some data source, mostly from database. Here are definitions of Data Finders and their implementations for reading data from database.

5. DataTypes

   Module contains DataTypes used/defined by TotemDatabaseService for use for other modules.

6. DataWriter

   Module contains classes that allows you to write some kinds of data to database. These kinds are now GroupData(Alignment) and Sensor Part Status.

7. DBUserInterface

   Module contains classes that provided by Event Setup System to give interface to specified methods of DataSource or DataWriter.

8. Doc

   Doc directory contains this user manual and doxygen documentation for the code.

9. ExampleESSource

   Module contains example ESProducer, EventSetupRecords, and Classes to store various sort of data like General Measurement, Sensor Part Status etc.

10. Examples

    Module contains classes example EDProducer that shows how to read and write data with objects provided by Event Setup System

11. RPAlignmentESSource

12. Sql

    Here are files with User Defined Sql Statements

13. Tools

    Module contains classes that facilitate configuring and work of other modules, like loading and parsing User Defined SQL Statements from file.

7 References

References

[1] https://twiki.cern.ch/twiki/bin/view/CMS/WorkBook

[2] https://twiki.cern.ch/twiki/bin/view/CMS/SWGuideFrameWork

[3] https://twiki.cern.ch/twiki/bin/view/CMS/WorkBookMoreOnCMSSWFramework#EventSetupLink

[4] https://twiki.cern.ch/twiki/bin/view/CMS/SWGuideFrameWork#EventSetup_and_Services

[5] http://www.oracle.com/technology/tech/oci/occi/index.html

[6] http://www.boost.org/doc/libs/1_40_0/doc/html/date_time/posix_time.html

[7] Doxygen generated refman(html and pdf) can be found in TotemDatabaseService/Doc

[8] http://sourcemaking.com/design_patterns/chain_of_responsibility

[9] http://sourcemaking.com/design_patterns/composite

[10] Totem Offline Database Monitor by Maciej Zalelwski. https://twiki.cern.ch/twiki/bin/view/TOTEM/TotemDatabase

[11] http://www.oracle.com/technology/products/database/sql_developer/index.html

[12] http://www.oracle.com/technology/products/database/datamodeler/index.html