```
#include "ScriptRunner/ScriptRunner.hpp"
#include "ScriptRunner/CrateList.hpp"
#include "ScriptRunner/options.hpp"
#include "l3fparser/ParserParameters.hpp"
#ifndef __RUN_CONFIG_MGR__
 #include "run_config_mgr/run_cfg_mgr.h"
#endif
#include "l3run_config/L3RunConfigMgr.h"
#include "l3fchunk/L3Chunk.hpp"
#include "l3fchunk/L3MonChunk.hpp"
#include "ScriptRunner/L3SendLumInfo.hpp"

#include "rcp/RCPManager.hpp"
#include "rcp/RCP.hpp"
#include "edm/Event.hpp"
#include "edm/TKey.hpp"
#include "edm/IDSelector.hpp"
#include "edm/THandle.hpp"
#include "edm/RCPtr.hpp"
#include "d0om/d0_Ref.hpp"
#include "PhysicsVectors/LorentzVector.h"
#ifndef ERROROBJ_H
 #include "ErrorLogger/ErrorObj.h"
#endif
#ifndef ELSEVERITYLEVEL_H
 #include "ErrorLogger/ELseverityLevel.h"
#endif
#ifndef ZMTHROW_H
 #include "Exceptions/ZMthrow.h"
#endif
#ifndef ZMEXERRORLOG_H
 #include "ErrLogEx/ZMexErrorLog.h"
#endif
#ifndef L3ZMXEL_H
 #include "l3utilities/L3ZMxel.hpp"
#endif
#include "l3fregistry/L3TRegistry.hpp"
#include <iostream>
#include <string>
#include <strstream>
// need this include to pass on the event location:
#include "l3ftoolbase/unpackTool.hpp"
#include "l3streaming/L3Streaming.hpp"
#include "ScriptRunner/L3MonData.hpp"
#include "l3fregistry/dynload.hpp"
#include "l3fstatmanager/L3Timer.hpp"

typedef unsigned int UINT;


L3_TOOL_DECL(L3ErrHandle)
L3_TOOL_DECL(L3RunConfigMgr)
L3_TOOL_DECL(L3TCFTTrack)
L3_TOOL_DECL(L3TNullVertex)
  L3_TOOL_DECL(L3TMcVtxFinder)
  L3_TOOL_DECL(L3TBTagIP)
#ifndef NT_MSVCPP
// fixes for t00.92.00
L3_TOOL_DECL(L3TJet)
L3_TOOL_DECL(L3TCalMEt)
L3_TOOL_DECL(L3TPhoton)
L3_TOOL_DECL(L3TauHadronic)
// end t00.92.00 fixes
#endif

#ifdef NT_MSVCPP
#ifndef L3DLL
  L3_TOOL_DECL(L3TSmtUnpack)
  L3_TOOL_DECL(L3TsmtCluster)
```



```
  L3_TOOL_DECL(L3TMuoUnpackTimes)
  L3_TOOL_DECL(L3TMuoUnpack)
  L3_TOOL_DECL(L3TMuoLocal)
  L3_TOOL_DECL(L3TCalCalib)
  L3_TOOL_DECL(L3TMuoHitCoinc)
  L3_TOOL_DECL(L3TGlobalTracker)
  L3_TOOL_DECL(L3TCFTUnpack)
  L3_TOOL_DECL(L3TCalCluster)
  L3_TOOL_DECL(L3TJet)
L3_TOOL_DECL(L3TCalUnp)
L3_TOOL_DECL(L3TSMTTracker)
 // L3_TOOL_DECL(L3TMuoCentralMatch)
#endif
#endif

// These are the local log and an administrator
// While the output streams will be set by the ErrHandleTool,
// ScriptRunner has to set the context (run/event) and the
// node.


ELadministrator * ScriptRunner::l3erradmin = ELadministrator::instance();

ErrorLog ScriptRunner::_ScriptRunnerLog;

ScriptRunner::ScriptRunner() : is_init(false),
                               _port_nb(52283),
                               _addr_name("d0ol03.fnal.gov"),
                               _online(true) {}

ScriptRunner::ScriptRunner(bool onl,int port, std::string addr) : is_init(false)
 {
     _port_nb = port;
     _addr_name =addr;
     _online = onl;
}


bool ScriptRunner::d_constructor() {

     is_init = true;
     clock_t begin ;
     _timer = new L3Timer;
     evtcnt = 0;

     // l3erradmin->setContextSupplier(_context);
     l3erradmin->setProcess("Node xx");

     // Set up ErrorLogging
    _ScriptRunnerLog.setPackage("ScriptRunner");
     L3ZMxel::setLogger( ZMexErrorLog(_ScriptRunnerLog) );

     // instantiate the External Iterator:
     It = new ExternIterator();
     //this must be just after construct ExternIterator
     It->Set_Online(_online);

     _lumInfo = new char[maxbuffersize];
     lum_write = new MonWriter(_lumInfo);
     sendLumInfo = L3SendLumInfo::Instance();
     sendLumInfo->startSender(_port_nb, _addr_name);

     _timeElaps.resize(MAXTIME+1);

     //allow for different versions of filter and tool definitions.
     edm::RCPManager *rcpman = edm::RCPManager::instance();
     edm::RCP rcpFile = rcpman->extract("ScriptRunner", "ScriptRunner_process");

     l3string toollist = rcpFile.getString("ToolList");
```

```
     l3string filterlist = rcpFile.getString("FilterList");
     ParserParameters parparm;
     parparm.setToollist(toollist);
     parparm.setFilterlist(filterlist);

     // Begin framework time :
     fwkTime = clock();
     L3Streaming::instance()->clearStreamIDMap();
     return true;
}

bool ScriptRunner::d_destructor() {

      _ScriptRunnerLog (ELinfo, "Begin ScriptRunner destruction ")
             << endmsg;

       _timeElaps.clear();
       delete _timer;
       It->Destroy();
       delete It;
       delete [] _lumInfo;
       delete lum_write;
       return true;
}


bool ScriptRunner::d_processEvent(edm::Event& event ) {

       // check if SR was initialized
       if( ! is_init ) {
          // Set up ErrorLogging
          _ScriptRunnerLog.setPackage("ScriptRunner");
          L3ZMxel::setLogger( ZMexErrorLog(_ScriptRunnerLog) );

          ErrorObj maperror( ELsevere, "Fatal - SR not initialized! ");
                   maperror << endmsg;
            //  throw it!!
           //ZMthrow( L3ZMxel (maperror));
           return false;
      }

     //check for a excution tree:
     if( ! It->isThere() ) {
            ErrorObj maperror( ELerror, "No trigger_list on duty -> event may be out of order ");
            maperror <<"discarded" << endmsg;
            //  throw it!!
            //  ZMthrow( L3ZMxel (maperror));
            return false;
     }

     // moved the Tools&Filters reset to the begining of the event processing
     // Obs - other hooks (methods) that call Tools&Filters, other than
     //       d_process_event MUST reset them. M.Souza 10/24/2001
     It->ResetTools();
     It->ResetFilters();

      // this is only local counter
     evtcnt++;

     //beginning l3filter timing for this event
     clock_t start= clock()*1000;
     _timer->start();

     // Tells the unpacker the event location
     unpackTool::setEvent(&event);

     // Here we implement a try-catch exception handler - we may want to get rid
 of it after, because
     // of time consuming overhead
```



```
          // moved to include getL1L2Info to catch exceptions from l3base (JMH)
     try {

      // get L1/L2 info
      It->getL1L2Info(event);

 #ifdef L3FDEBUG
      std::cout // << instanceName()
          << "\n//********************************* Event # "
          << evtcnt << std::endl;
 #endif

       // External (Event) Traverse Loop
       for( It->Begin() ; It->isFinish() ; It->operator ++() ) {

#ifdef L3FDEBUG
           std::cout << "\n ---------------------------" << std::endl;
           It->_Current()->_GetPt()->hello();
#endif
           // update itc-l2 bits
           It->ITCL2Bits();

          // Internal traversing
          if( ! It->Traverse() )
             return false;  //error in traversing, discarding this event


       } // End External traversing

     }// End try

     catch( L3ZMxel &Z) {

         if( Z.errorObject().xid().severity > ELerror ) {
              // rethrow the exception. This will be caught by the Supervisor
             // ZMthrow( Z ) ;
             return false;
         } else {
           _ScriptRunnerLog(ELerror, "ScriptRunner caught an exception < 'severe'")
                << " event discarded" << endmsg;
           // kill timers
           //l3StatManager::Instance()->kill_clocks();
           _timer->stop();
           //reset
           It->ResetTools();
           It->ResetFilters();
           It->ResetMySelf();
           return false;
         }
     }

     catch(...) {
        _ScriptRunnerLog(ELerror, "ScriptRunner caught an unknown exception for this event")
           << " event discarded" << endmsg;
       // kill timers
       //l3StatManager::Instance()->kill_clocks();
       _timer->stop();
       //reset
       It->ResetTools();
       It->ResetFilters();
       It->ResetMySelf();
       return false;
     }

     bool _passedStatus = false;
     if( It->Passed() ) {

       _passedStatus = true;
       It->L3ChunkGen( &event);
```

```
        It->genEventHeader(event);
        if(It->Debug_On())
            It->L3DebugChunkGen( &event);
      }

      _timer->stop();
      clock_t finish = clock()*1000;
      //this will fill histograms with bin size of 10ms.
      int _telaps =  _timer->clkTime()*100;
      if ( _telaps >= 20 )
          _timeElaps[21]++;
      else if( _telaps <= 0 )
          _timeElaps[0]++;
      else
          _timeElaps[_telaps+1]++;

    It->upTime(_telaps);


      // Send Lum Data, if new Luminosity index
      if( It->getLumData() ) {
            L3MonChunk lumchnk;
            lumchnk.insertTimeData(_timeElaps);
            It->getLumData(lumchnk);

            //here we add Client(s) Information for Luminosity studies
            std::map<int, ClientDef *>::const_iterator m_iter;

            ClientData _cData;
            L3MonChunk::L3ClientData _clData;

            for( m_iter = _currentSt.begin();
                     m_iter!= _currentSt.end(); m_iter++) {
                 int runNb = m_iter->second->_runnb;
                 _cData._l2map= m_iter->second->_runL2bits;
                 _cData._l2l1map = m_iter->second->_l21mapbit;
                 _cData._l3map = m_iter->second->_l3map;
                 _clData[runNb]=&_cData;
            }
            lumchnk.insertClientData(_clData );

            //have new luminosity index
            lum_write->reset();
            lum_write->write(lumchnk,"this is a test");
            size_t size = lum_write->size();
            sendLumInfo->SendIt(It->getMonId(), It->GetLumIndex(), _lumInfo, size
);

    }

    //reset
    It->ResetMySelf();

#ifdef L3FDEBUG
  std::cout << "->>>>>>>>>>>>>>>>>>>>TIME:" << _telaps << std::endl;
#endif

      if( _passedStatus ) {
#ifdef L3FDEBUG
            std::cout << "PASSED" << std::endl;
#endif
            return true;
      } else{
#ifdef L3FDEBUG
           std::cout << "FAILED" << std::endl;
#endif
            return false;
          }
}
```



```
//-----------------------------------------
bool ScriptRunner::d_jobSummary() {

       _ScriptRunnerLog( ELinfo,
                    "\n\n     ********************\n      Job Summary section")
                    << "\n********************\n\n"
                   << endmsg;

       It->Summary();

      _ScriptRunnerLog( ELinfo,
                    "\n\n     *****************\n      End Job Summary ")
                    << "\n*****************\n\n\n"
                   << endmsg;

       _ScriptRunnerLog( ELinfo, "Total Filtering time(secs) ")
                << (double)(It->ElapsedTime())/CLOCKS_PER_SEC << endmsg;

       clock_t finish = clock();
       _ScriptRunnerLog( ELinfo, "Total L3 Framework elapsed time(secs) ")
                     << (double)(finish - fwkTime)/CLOCKS_PER_SEC << endmsg;

       return true;
}
//-------------------------------------------------------
 bool ScriptRunner::d_itc_Info(_ITC_EVENT_NAMESPACE::Event_Header &hd) {

       hd = *It->getEventHeader();

       return true;
}


//-----------------------------------------
bool ScriptRunner::d_trigInfo(const CoorInput &x) {
    bool ret = false;
        ret = Dispatch(x);
        return ret;
}

//-----------------
bool ScriptRunner::Dispatch(const CoorInput &_input) {

   l3string _coorline = _input.CoorString();
   l3vector<l3string> _vtmp;
   l3map<int, ClientDef *>::iterator m_iter;
   l3vector<l3string>::const_iterator v_iter;


   int words = decomposeString( _coorline, _vtmp);
   if(words == 0)
     {
       _ScriptRunnerLog(ELinfo,
                          "ScriptRunner was passed and empty string")
                            << endmsg;
       return true;
     }
//------Init
   if(_vtmp[0] == "init" ) {
      //first, clear all clients
      if( ! _currentSt.empty() ) {
         //drop all l2bits and reset
         std::bitset<l2base::MAX_NUM_L2_BITS> _all_bits;
         _all_bits.set();
         It->DropL2Bits(_all_bits);
         It->DropL2MonInfo( _all_bits);
         It->setMonMap();  // this will erase it
```

```
        //reset l1,l2 maps
        _l1map.clear();
        _l2map.clear();
        It->UpdateL2Map(_l2map);
        _currentSt.clear();
        It->UpdateL3Map(_currentSt);
        if(!this->rebuildStreamIDMap()){return false;}
      }
      //now, do the init
      if( ! is_init )
          return d_constructor();
      else {
           if( d_destructor() ) {
               if( d_constructor() ){
                   _ScriptRunnerLog( ELinfo,
                            "ScriptRunner reset after an INIT directive")
                  << endmsg;
                   return true;

               } else {
                  ErrorObj maperror( ELsevere,
                                  " Failed to reset ScriptRunner after INIT! ");
                   maperror << endmsg;
                      //  throw it!!
                  //ZMthrow( L3ZMxel (maperror));
                  return false;   }

            } else {
                ErrorObj maperror( ELsevere,
                                  " Failed to destroy ScriptRunner after INIT! ");
                   maperror << endmsg;
                      //  throw it!!
                  //ZMthrow( L3ZMxel (maperror));
                  return false;
            }
         }
//------Clear_client
   } else if( _vtmp[0] == "clear_client" ) {

      int _client  = atoi(_vtmp[1].c_str());
           //check if it's defined
       m_iter = _currentSt.find(_client);
       if( m_iter == _currentSt.end() ) {
            ErrorObj maperror( ELerror, " Cant Clear -> Client not defined! ");
            maperror << _client << endmsg;
               //  throw it!!
           //    ZMthrow( L3ZMxel (maperror));
           //    return false;
                        return true;
       }


      //deactivate bits for this client
      if( ( m_iter = _currentSt.find(_client)) != _currentSt.end() ){

         if( _currentSt[_client]->is_running ){
              _currentSt[_client]->is_running =false;
              It->EndRun(_currentSt[_client]->_runL2bits);
         }
         //drop all l2bits and reset
         It->DropL2Bits(_currentSt[_client]->_runL2bits);
         It->DropL2MonInfo( _currentSt[_client]->_runL2bits);
         It->setMonMap();  // this will erase it

         //reset l1,l2 maps
         int jj;
         for( jj=0; jj<l2base::MAX_NUM_L2_BITS ; jj++ ) {
           if(_currentSt[_client]->_runL2bits.test(jj) ) {
               int jj1= _currentSt[_client]->_l21mapbit[jj];
```



```
                _l1map.erase(jj1);
                _l2map.erase(jj);
            }
          }
          It->UpdateL2Map(_l2map);

           delete _currentSt[_client];
           _currentSt.erase(_client);
       }
       It->UpdateL3Map(_currentSt);
       if(!this->rebuildStreamIDMap()){return false;}
       return true;
//--------Set_client
   } else if( _vtmp[0] == "set_client" ) {

        int _client  = atoi(_vtmp[1].c_str());
            //check for an already defined client
        m_iter = _currentSt.find(_client);
        if( m_iter != _currentSt.end() ) {
             ErrorObj maperror( ELsevere, " FATAL -> Client already defined! ");
             maperror << _client << endmsg;
                //  throw it!!
               // ZMthrow( L3ZMxel (maperror));
                return false;
            }
        _currentSt[_client] = new ClientDef();
        return true;
//------l1bit
   } else if( _vtmp[0] == "l1bit" ) {

       int _l1bit = atoi( _vtmp[1].c_str() );
       l3string l1bit_st= _vtmp[2];
       _l1map[_l1bit]=l1bit_st;

       // crate list parsing...
       if(words>2)
         {
           l3string l1crates_st;
           for(int i=3;i<words;i++)
             {
               l1crates_st+=" ";
               l1crates_st+= _vtmp[i];
             }
           std::cout << "Adding crate list:" << _l1bit << ":" <<
             l1crates_st << std::endl;
           _l1CrateListMap[_l1bit] = l1crates_st;
         }
       else
         {
           std::cout << "no crate list associated with l1-bit" << _l1bit
                     << std::endl;
           _ScriptRunnerLog(ELwarning, "no crate list associated wiht l1bit")
             << _l1bit << endmsg;
           return false;
         }
       return true;
//------l2bit
   } else if( _vtmp[0] == "l2bit" ) {

       int _l2bit = atoi( _vtmp[1].c_str() );
       l3string l2bit_st= _vtmp[2];
       _l2map[_l2bit]=l2bit_st;
       return true;

//---------Runinfo
   } else if( _vtmp[0]== "runinfo" ) {
       int _client  = atoi(_vtmp[1].c_str());

       //get the client for this run - FATAL if not;
```

```
      m_iter = _currentSt.find(_client);
      if( m_iter == _currentSt.end() ) {

            ErrorObj maperror( ELsevere, " Failed -> no Client! ");
            maperror << _client << endmsg;
            //  throw it!!
            //ZMthrow( L3ZMxel (maperror));

            return false;
     }

      _currentSt[_client]->_runnb =
        atoi(  _vtmp[2].c_str()  );
      return true;

//-------Define_trigger
   } else if( _vtmp[0]== "define_trigger" ) {

      int _l3 = atoi( _vtmp[1].c_str() );
      int _l1 = atoi( _vtmp[3].c_str() );
      int _l2 = atoi( _vtmp[4].c_str() );
      int _client = atoi( _vtmp[2].c_str() );

      //get the client for this run - FATAL if not;
      m_iter = _currentSt.find(_client);
      if( m_iter == _currentSt.end() ) {

            ErrorObj maperror( ELsevere, " Failed to Define Trigger -> no Client! ");
            maperror << _client << endmsg;
            //  throw it!!
            //ZMthrow( L3ZMxel (maperror));
            return false;
     }
      _currentSt[_client]->_runL2bits.set(_l2);
      _currentSt[_client]->_l21mapbit[_l2]=_l1;
      _currentSt[_client]->_l3map[_l3] = _vtmp[5];

      _currentSt[_client]->is_l3map_defined = true;
      _currentSt[_client]->is_pending = true;

      return true;

//-------Trigger_list
   } else if( _vtmp[0] == "trigger_list" ||
                _vtmp[0] == "trigger_text" ) {

       // check if SR was initialized
       if( ! is_init ) {
          // Set up ErrorLogging
          _ScriptRunnerLog.setPackage("ScriptRunner");
          L3ZMxel::setLogger( ZMexErrorLog(_ScriptRunnerLog) );

          ErrorObj maperror( ELsevere, "Fatal - SR not initialized! ");
                   maperror << endmsg;
            //  throw it!!
           //ZMthrow( L3ZMxel (maperror));
           return false;
      }

      //Here we cache the trigger_list directive, but the client must exist
      int _client  = atoi(_vtmp[1].c_str());
           //check if it's defined
       m_iter = _currentSt.find(_client);
       if( m_iter == _currentSt.end() ) {
            ErrorObj maperror( ELsevere, " Can't assign triggerlist -> Client not defined! ");
            maperror << _client << endmsg;
               //  throw it!!
               //ZMthrow( L3ZMxel (maperror));
               return false;
```



```
       }

      if(_coorline.find(".lst") == std::string::npos )
           getListString(_coorline, _currentSt[_client]->_triggerlist);
      else
           _currentSt[_client]->_triggerlist = _vtmp[2];
      return true;

 //--------------------- configure
  } else if( _vtmp[0] == "configure" ) {

    // first step .. create the crate list
    l3map<int,l3string>::const_iterator crateMapItr;
    _crateList.clearCrates();
    for(crateMapItr = _l1CrateListMap.begin();
        crateMapItr != _l1CrateListMap.end();
        crateMapItr++)
      {
        _crateList.addCrates((*crateMapItr).second);
      }
    std::cout << "Building Crate list : " << std::endl;
    _crateList.dumpList();
     int nb_pending = 0;
      //loop over all clients and look for 'pending' ones
      for( m_iter=_currentSt.begin(); m_iter != _currentSt.end(); m_iter++){
        if( m_iter->second->is_pending) {
           nb_pending++;
           m_iter->second->is_pending = false;
           m_iter->second->is_configured = true;
           int _thisclient = m_iter->first;
           // First, resume trigger_list
           It->UpdateL2Map(_l2map);
           It->UpdateL3Map(_currentSt);

           //do triggerlist parsing
           try{
             l3vector<int> tmpvec = _crateList.getCrateList();
             It->setCrateList(tmpvec);
              if( !It->AddL2bits(_currentSt[_thisclient]->_triggerlist) )
                    return false;
           }

           catch( L3ZMxel &Z) {
              if( Z.errorObject().xid().severity > ELerror ) {
                // rethrow the exception. This will be caught by the Supervisor
                //ZMthrow( Z ) ;
                return false;
              } else {
                _ScriptRunnerLog(ELerror, "AddL2Bits")
                << "ScriptRunner caught an exception with severity < 'severe'" << endmsg;
                 return false;
              }
           }
           catch (edm::XRCP ERROR ) {
              std::cout<< "ScriptRunner caught a RCP exception : \n" << ERROR.message() << s
td::endl;
             // ZMthrow( ERROR );
              return false;
           }
           catch(...) {
             _ScriptRunnerLog(ELerror,"AddL2Bits")
                <<"ScriptRunner caught an unknown exception " << endmsg;
                return false;
           }

           It->setMonMap(_currentSt[_thisclient]->_runnb,
                                 _currentSt[_thisclient]->_runL2bits);
        }
      }
```

```
      if( nb_pending == 0 ) {
        //here configure didn't configure any client
          ErrorObj maperror( ELsevere, "Fatal - Configure: no clients to configure");
          maperror << endmsg;
          //  throw it!!
           //ZMthrow( L3ZMxel (maperror));
         return false;
      }

       return true;

//-----Start_run
   } else if( _vtmp[0] == "start_run" ) {
      int _thisrun = atoi (_vtmp[1].c_str() );

      //get the client for this run - FATAL if not;
      int _thisclient = -1;
      for(m_iter = _currentSt.begin(); m_iter != _currentSt.end(); m_iter++) {
         if( m_iter->second->_runnb == _thisrun )
             _thisclient = m_iter->first;
      }

      if( _thisclient < 0 ) {

            ErrorObj maperror( ELsevere, " Failed to Begin Run -> no Client! ");
            maperror << _thisrun << endmsg;
                    //  throw it!!
            //ZMthrow( L3ZMxel (maperror));
            return false;
      }
      if( _currentSt[_thisclient]->is_running) {
            ErrorObj maperror( ELsevere, " FATAL - Run already started ");
            maperror << _thisrun << endmsg;
                    //  throw it!!
            //ZMthrow( L3ZMxel (maperror));
            return false ;
      }


      if( _currentSt[_thisclient]->is_l3map_defined ) {
          if(_currentSt[_thisclient]->_triggerlist =="") {
             ErrorObj maperror( ELsevere, " Failed to Begin Run -> no Trigger List! ");
             maperror << _thisclient << endmsg;
                    //  throw it!!
                 //ZMthrow( L3ZMxel (maperror));
                 return false;
          }

         if( ! _currentSt[_thisclient]->is_configured) {
            ErrorObj maperror( ELsevere, " FATAL - Run/Client not configured ");
            maperror << _thisrun << "/" <<_thisclient << endmsg;
                    //  throw it!!
            //ZMthrow( L3ZMxel (maperror));
            return false ;
         }


      } else {
          //TEST MODE
          //if we get here, we assume a 'daq_test mode'
          // activate all bits and map l1<->l2 one to one
          _ScriptRunnerLog (ELwarning, "Entering Test Mode ")
                      << endmsg;
          for( l3map<int, l3string>::const_iterator p=
                      _l1map.begin(); p!= _l1map.end(); p++) {
                 _currentSt[_thisclient]->_runL2bits.set(p->first);
               _currentSt[_thisclient]->_l21mapbit[p->first]=p->first;
               _currentSt[_thisclient]->_l3map[p->first] = "daq_test";
```



```
            }
            It->UpdateL2Map(_l2map);
            It->UpdateL3Map(_currentSt);
            It->setupTest();

            //END TEST MODE
      }


      //l3run_config initialization
        It->ToolsRunInit();   //Hey, we don't need to call only Tools belonging
to this run?? Argh!

       // mark it as "running"
      _currentSt[_thisclient]->is_running=true;
      // rebuild stream-ID map
      if(!this->rebuildStreamIDMap()) return false;
      //build l1 active bits mask
       _currentSt[_thisclient]->_runL1bits.reset();
      for( v_iter = _vtmp.begin()+2; v_iter != _vtmp.end() ; v_iter++) {

          int _thisl1 = atoi( (*v_iter).c_str() );
          _currentSt[_thisclient]->_runL1bits.set(_thisl1);
       }

       // now build the l2 active bits mask for this run
       if(  _currentSt[_thisclient]->fillRunL2Bits() == 0 ) {
              _ScriptRunnerLog( ELinfo," No L2bits for this run")
              << _thisrun << endmsg;

       } else {

          if( ! It->BeginRun(_currentSt[_thisclient]->_runL2bits) ){
            ErrorObj maperror( ELsevere, " Failed to Begin Run! ");
            maperror << _thisrun << endmsg;
            //  throw it!!
            //ZMthrow( L3ZMxel (maperror));
            return false;
          }
       }

       //notify the servers this node id
       sendLumInfo->SendIt(4,It->GetLumIndex(), "BofR", 5);

       return true;

 //-------------------- stop_run
   } else if( _vtmp[0] == "stop_run" ) {
      int _thisrun = atoi (_vtmp[1].c_str() );

      //get the client for this run - FATAL if not;
      int _thisclient = -1;
      for(m_iter = _currentSt.begin(); m_iter != _currentSt.end(); m_iter++) {
         if( m_iter->second->_runnb == _thisrun )
             _thisclient = m_iter->first;
      }

      if( _thisclient < 0 ) {

            ErrorObj maperror( ELsevere, " Failed to Stop Run -> no Client! ");
            maperror << _thisrun << endmsg;
            //  throw it!!
           // ZMthrow( L3ZMxel (maperror));
            return false;
      }
      _currentSt[_thisclient]->is_running=false;
      // rebuild streamID map
      if(!this->rebuildStreamIDMap()) return false;
      //
```

```
      if( ! It->EndRun(_currentSt[_thisclient]->_runL2bits) ) {
            ErrorObj maperror( ELsevere, " Failed to Stop Run! ");
            maperror << _thisrun << endmsg;
            //  throw it!!
            //ZMthrow( L3ZMxel (maperror));
            return false;
      }
      // should we issue a job_summary to ExternIterator?

        //yes!!
        //size_t buffer_size;
        //d_mon_Info(_monInfo, buffer_size);
        //sendMonInfo->SendIt(It->getMonId(),_monInfo, buffer_size);

      return true;


   } else if( _vtmp[0] == "stream" ) {
     //     return  L3Streaming::instance()->parseCoorMsg(_coorline);
     UINT id = atoi(_vtmp[1].c_str());
     int client = atoi(_vtmp[2].c_str());
     l3string streamName = _vtmp[3];

     //     std::cout << "parsing stream directive: "
     //        << "client No. " << client << " "
     //        << streamName << " "
     //        <<id << std::endl;
     m_iter = _currentSt.find(client);
     if(m_iter==_currentSt.end())
       {
         _ScriptRunnerLog(ELerror, "Unknown client: ")
           << client << " when parsing stream directive" << endmsg;
         return false;
       }
     // are streamName and streamID valid:
     else if(!L3Streaming::instance()->validateStreamID(streamName,id))
       {
         _ScriptRunnerLog(ELerror, "Invalid stream assignment ")
           << streamName << " " << id << endmsg;
         return false;
       }
     else
       {
         m_iter->second->addStreamDef(streamName,id);
       }
     return true;

   //-------Others not yet implemented or not yet understood
   } else if( _vtmp[0] == "process") {
       return true; // this is a (l3f) private off-line message, which is caught
 by
         // CoorMgr
   } else {
        // unknown!
       _ScriptRunnerLog (ELwarning, "Unknown directive ")
          << _vtmp[0] << endmsg;

       return false;
   }

}
//---------------
ScriptRunner::~ScriptRunner()
{
   for(l3map<int, ClientDef *>::iterator it_map
       =_currentSt.begin(); it_map != _currentSt.end(); it_map++){
        delete it_map->second;
   }
```



```
   _currentSt.clear();

   d_destructor();
}

//------------
int decomposeString(l3string &str_line, l3vector<l3string> &output)
{
  std::istringstream ist(str_line);
  l3string word;
  while(ist>>word)
    {
      //      std::cout << "found : " << word << std::endl;
      output.push_back(word);
    }
  return output.size();
}
//------------
  void ScriptRunner::getListString(l3string &str_line, l3string &trigString) {

   // first, get rid of trailing blanks ( thanks to Carmem's parser )
   typedef l3string::iterator si;
   si pi=str_line.begin();
   l3string newch;

   int j=0;
   bool space=0;
   pi=str_line.begin();

   while (pi!=str_line.end()){
      if (isspace(*pi)){
         space=1;
         ++pi;
         while (isspace(*pi)&&pi!=str_line.end())
                ++pi;
         if (pi==str_line.end())
                --pi;
      }
      if (space){
        newch.append(" ");
        space=0;
      }
      newch.append(pi,1);
      ++pi;
   }
   if (newch.end()!= " ")
         newch.append(" ");

   str_line=newch;

   int ii, start=0;
   for( int kk=0; kk<2; kk++) {
     ii = str_line.find(" ", start);
     start = ii+1;
   }
   trigString = str_line.substr(start,str_line.size()-start);

}

bool ScriptRunner::rebuildStreamIDMap()
{
  // std::cout << "ScriptRunner::rebuildStreamIDMap" << std::endl;

  l3map<int, ClientDef *>::const_iterator clientMapIter;
  l3map<l3string,l3vector<UINT> >::const_iterator streamMapIter;

  L3Streaming::instance()->clearStreamIDMap();

  for(clientMapIter = _currentSt.begin();
```

```
      clientMapIter != _currentSt.end();
      clientMapIter++)
    {
      ClientDef *clientPtr = clientMapIter->second;
      if(clientPtr->is_running)
        {
          if(!L3Streaming::instance()->addStreamIDMap(clientPtr->_stream_id_map)
)
            {
              _ScriptRunnerLog(ELsevere, "Failed to rebuild stream ID map");
                return false;
            }
        }

    }
  return true;
}
```