Current Test Script: sdps.rb

```
#!/usr/bin/env ruby

module Sdps
end

libdir = File.expand_path(File.dirname(__FILE__))
unless $LOAD_PATH.include? libdir
  $LOAD_PATH.unshift libdir
end

sdpsdir = File.expand_path("../#{File.basename(__FILE__, '.rb')}", __FILE__)

libs = Dir[sdpsdir + '/*.rb']
libs.each do |lib|
  require lib
end
```

Current Test Script: exportrequest.rb

```
# Contains BMGT specific classes and methods calling bmgt custom code binaries
module Sdps
  module Bmgt

   # custom code application functions
     class Sdps::Bmgt::ExportRequest

      def initialize(mode)
        @mode = mode
      end

      def manual_granule_export_for_collection?(collection_esdt, options)
        begin
          cmdoptions = ''

          cmdoptions = options.join(' ') if options != nil
          cdir = "/usr/ecs/#{@mode}/CUSTOM/utilities"
```

```
    Dir.chdir("#{cdir}") do
      status = system("./EcBmBMGTManualStart #{@mode} -metg -c #{collection_esdt} #{cmdoptions}")
    end
  rescue =>e
    puts e.backtrace
    false
  end
end

def manual_export_for_granules?(granules, options)
  begin
    cmdoptions = ''

    cmdoptions = options.join(' ') if options != nil
    cdir = "/usr/ecs/#{@mode}/CUSTOM/utilities"
    Dir.chdir("#{cdir}") do
      status = system("./EcBmBMGTManualStart #{@mode} -metg -g #{granules} #{cmdoptions}")
    end
  rescue =>e
    puts e.backtrace
    false
  end
end

#passing additional command arguments as an array.
def manual_export_for_collection?(collectionid, options)
  begin
   cmdoptions = ''

   cmdoptions = options.join(' ') if options != nil

   cdir = "/usr/ecs/#{@mode}/CUSTOM/utilities"
   Dir.chdir("#{cdir}") do
    status = system("./EcBmBMGTManualStart #{@mode} -metc -c #{collectionid} #{cmdoptions}")
    end
  rescue =>e
```

```
          puts e.backtrace
          false
        end
      end

      #passing additional command arguments as an array.
      def manual_file_export?(file, type, options)
        begin
         cmdoptions = ''

         cmdoptions = options.join(' ') if options != nil

         cdir = "/usr/ecs/#{@mode}/CUSTOM/utilities"
         Dir.chdir("#{cdir}") do
          status = system("./EcBmBMGTManualStart #{@mode} #{type} -cf #{file} #{cmdoptions}")
          end
        rescue =>e
          puts e.backtrace
          false
        end
      end

      def manual_time_export?(type, starttime, endtime, options)

        begin
         cmdoptions = ''

         cmdoptions = options.join(' ') if options != nil

         cdir = "/usr/ecs/#{@mode}/CUSTOM/utilities"
         Dir.chdir("#{cdir}") do
          status = system("./EcBmBMGTManualStart #{@mode} #{type} -st '#{starttime}' -et '#{endtime}'
#{cmdoptions}")
          end
        rescue =>e
          puts e.backtrace
```

```
          false
        end

      end


    end # class ExportRequest
  end # module Bmgt
end # module Sdps
```

Current Test Script: xml.rb

```
# xml functions specific to BMGT
module Sdps
  module Bmgt

    require 'nokogiri'
    require 'sdps/tcpproxylogger'

    class Sdps::Bmgt::Xml

      def granule_xml_valid?(granule_xml)
        #result = system "/tools/libxml2-2.9.1/bin/xmllint --noout --path
/tools/common/test/BE_82_01/echo10/schema --schema Granule.xsd #{granule_xml}"
        #modified to use Nokogiri methods
        xml_doc = Nokogiri::XML(granule_xml)
        xsd = Nokogiri::XML::Schema(File.open(ECHO10_GRANULE_SCHEMA))
        err = xsd.validate(xml_doc)
        err.length == 0
      end

      def granule_xml_contains?(granule_xml, xpath_expr, value)
        rv = false;
        xml_doc = Nokogiri::XML(granule_xml)
        con = xml_doc.xpath(xpath_expr).to_s
        if(con.include?(value))
          rv = true
```

```
        end
      end

      def granule_xml_getvalue(granule_xml, xpath_expr)
        xml_doc = Nokogiri::XML(granule_xml)
        xml_doc.xpath(xpath_expr).to_s
      end

      # timestamp = Time.now.strftime('%Y-%m-%dT%H:%M:%S.%6N')
      def http_put_for_granule?(log_pathname, requesttime, granuleid)
        begin
          #check if request time is in the time format used in tcp.log
          request_time = DateTime.parse(requesttime).strftime('%Y-%m-%dT%H:%M:%S.%6N')
          entries = Sdps::TcpProxyLogger.merge_requests(Sdps::TcpProxyLogger.parse_file(log_pathname))
          log_entry = entries.select {|x| x.timestamp > request_time and x.put? and
x.request.split("\r\n\r\n")[1].include?("#{granuleid}") }[0]
          if log_entry.nil?
              false
          else
              true
          end
        rescue =>e
          puts e.backtrace
          false
        end
      end

      def http_verification_put_for_granule?(log_pathname, requesttime, granuleid)
        begin
          #check if request time is in the time format used in tcp.log
          request_time = DateTime.parse(requesttime).strftime('%Y-%m-%dT%H:%M:%S.%6N')
          entries = Sdps::TcpProxyLogger.merge_requests(Sdps::TcpProxyLogger.parse_file(log_pathname))
          log_entry = entries.select {|x| x.timestamp > request_time and x.put? and
x.request.split("\r\n\r\n")[1].include?("#{granuleid}") }[0]
          if log_entry.nil?
              false
```

```
          else
              log_entry.to_s.include?("_diff=true")
          end
        rescue =>e
          puts e.backtrace
          false
        end
      end

      def http_delete_for_granule?(log_pathname, requesttime, granuleid)
        begin
          #check if request time is in the time format used in tcp.log
          request_time = DateTime.parse(requesttime).strftime('%Y-%m-%dT%H:%M:%S.%6N')
          entries = Sdps::TcpProxyLogger.merge_requests(Sdps::TcpProxyLogger.parse_file(log_pathname))
          log_entry = entries.select {|x| x.timestamp > request_time and x.delete? and
x.request.split("\r\n\r\n")[0].include?("#{granuleid}") }[0]
          if log_entry.nil?
              false
          else
              true
          end
        rescue =>e
          puts e.backtrace
          false
        end
      end

    # timestamp = Time.now.strftime('%Y-%m-%dT%H:%M:%S.%6N')
      def get_granule_xml(log_pathname, requesttime, granuleid)
        begin

          request_time = DateTime.parse(requesttime).strftime('%Y-%m-%dT%H:%M:%S.%6N')
          entries = Sdps::TcpProxyLogger.merge_requests(Sdps::TcpProxyLogger.parse_file(log_pathname))

          log_entry = entries.select {|x| x.timestamp > request_time and x.put? and
x.request.split("\r\n\r\n")[1].include?("#{granuleid}") }[0]
```

```
          if log_entry.nil?
               nil
          else
              metadata = log_entry.request.split("\r\n\r\n")[1]
              unless(metadata.include?("<\/Granule>"))
                ctag = entries.select {|x| x.timestamp > request_time and
x.request.to_s.include?("<\/Granule>") }[0]
                metadata = metadata.to_s + ctag.request.to_s
              end
              return metadata
          end
        rescue =>e
          puts e.backtrace
          nil
        end
      end

      def collection_xml_valid?(collection_xml)
        #modified to use Nokogiri methods
        xml_doc = Nokogiri::XML(collection_xml)
        xsd = Nokogiri::XML::Schema(File.open(ECHO10_COLLECTION_SCHEMA))
        xsd.validate(xml_doc).each do |error|
          puts error.message
        end
        xsd.validate(xml_doc).length == 0
      end

      def collection_xml_contains?(collection_xml, xpath_expr, value)
        xml_doc = Nokogiri::XML(collection_xml)
        xml_doc.xpath(xpath_expr).to_s == value
      end

       # timestamp = Time.now.strftime('%Y-%m-%dT%H:%M:%S.%6N')
      def http_put_for_collection?(log_pathname, requesttime, dataset)
        begin
          #status = false
```

```
          request_time = DateTime.parse(requesttime).strftime('%Y-%m-%dT%H:%M:%S.%6N')
          escDataset = ERB::Util.url_encode(dataset)

          entries = Sdps::TcpProxyLogger.merge_requests(Sdps::TcpProxyLogger.parse_file(log_pathname))
          log_entry = entries.select {|x| x.timestamp > request_time and x.put? and
x.request.split("\r\n\r\n")[0].include?("#{escDataset}") }[0]
          if log_entry.nil?
              false
          else
              true
          end

        rescue =>e
          puts e.backtrace
          false
        end
      end

      def http_verification_put_for_collection?(log_pathname, requesttime, dataset)
        begin
          #status = false
          request_time = DateTime.parse(requesttime).strftime('%Y-%m-%dT%H:%M:%S.%6N')
          escDataset = ERB::Util.url_encode(dataset)

          entries = Sdps::TcpProxyLogger.merge_requests(Sdps::TcpProxyLogger.parse_file(log_pathname))
          log_entry = entries.select {|x| x.timestamp > request_time and x.put? and
x.request.split("\r\n\r\n")[0].include?("#{escDataset}") }[0]
          if log_entry.nil?
            false
          else
            log_entry.to_s.include?("_diff=true")
          end

        rescue =>e
          puts e.backtrace
          false
```

```
        end
      end

      def http_delete_for_collection?(log_pathname, requesttime, dataset)
        begin
          status = false
          escDataset = ERB::Util.url_encode(dataset)

          request_time = DateTime.parse(requesttime).strftime('%Y-%m-%dT%H:%M:%S.%6N')

          entries = Sdps::TcpProxyLogger.merge_requests(Sdps::TcpProxyLogger.parse_file(log_pathname))
          log_entry = entries.select {|x| x.timestamp > request_time and x.delete? and
x.request.split("\r\n\r\n")[0].include?("#{escDataset}") }[0]
          if log_entry.nil?
              false
          else
              true
          end
        rescue =>e
          puts e.backtrace
          false
        end
      end
      # timestamp = Time.now.strftime('%Y-%m-%dT%H:%M:%S.%6N')
      def get_collection_xml(log_pathname, requesttime, coll)
        begin
          request_time = DateTime.parse(requesttime).strftime('%Y-%m-%dT%H:%M:%S.%6N')
          #xml_arr = Array.new
          shortName = coll.shortname
          #changed line 192 due to NCR 8052154 causing the bmgt tests to fail; Georgia Sandoval
          vId = "%03d" % coll.versionid #converts the version id from int to 3-char string (i.e. 7 => "007")
          entries = Sdps::TcpProxyLogger.merge_requests(Sdps::TcpProxyLogger.parse_file(log_pathname))

          log_entry = entries.select {|x| x.timestamp > request_time and x.put? and
x.request.split("\r\n\r\n")[1].match(/<Collection><ShortName>#{shortName}<\/ShortName><VersionId>#{vId}/) !=
nil}[0]
```

```
          if log_entry.nil?
                nil
          else
               metadata = log_entry.request.split("\r\n\r\n")[1]
          end

        rescue =>e
        puts e.backtrace
        nil
        end
      end
    end # class Xml

  end #module Bmgt

end #module Sdps

Current Test Script: bmgtdb.rb
module Sdps
  module Bmgt

  #queries to bg tables
    class Sdps::Bmgt::BmgtDb

      def initialize(dbConn)
        @conn = dbConn
      end

      # the following queries relate to bg_collection_configuration
      def get_collection_granule_export( collectionid)
        begin
          @conn.exec("select collectionexportflag, granuleexportflag from bg_collection_configuration where
collectionid = #{collectionid}")[0]
        rescue =>e
          puts e.backtrace
          nil
```

```
      end
    end

    def insert_collection_into_collectionconfig( collectionid)
      begin
        rs = @conn.exec("insert into bg_collection_configuration(collectionid, shortname, versionid, longname,
            granuleexportflag, collectionexportflag, difid, maxgranulestoverify)
            select collectionid, shortname, versionid, longname, 'N', 'Y', shortname,  5000 from amcollection
             where collectionid = #{collectionid}")
        if rs != nil and rs.count > 0
          true
        else
          false
        end
      rescue =>e
        puts e.backtrace
        false
      end
    end

    def enable_collection_granule_export( collectionid)
      begin
        rs = @conn.exec("update bg_collection_configuration set collectionexportflag = 'Y', granuleexportflag
= 'Y' where collectionid = #{collectionid} returning 1")
        if rs != nil and rs.count > 0
          true
        else
          false
        end
    rescue =>e
        puts e.backtrace
        false
      end
    end

    def enable_collection_export( collectionid)
```

```
        begin
          rs = @conn.exec("update bg_collection_configuration set collectionexportflag = 'Y' where collectionid
= #{collectionid} returning 1")
          if rs != nil and rs.count > 0
            true
          else
            false
          end
      rescue =>e
          puts e.backtrace
          false
        end
      end

      def enable_collection_export_by_shortname( shortname, version)
        begin
          rs = @conn.exec("update bg_collection_configuration set collectionexportflag = 'Y' where shortname =
#{shortname} and versionid = #{version} returning 1")
          if rs != nil and rs.count > 0
            true
          else
            false
          end
      rescue =>e
          puts e.backtrace
          false
        end
      end

      def enable_opendap_url_export( collectionid)
        begin
          rs = @conn.exec("update bg_collection_configuration set opendapurlexportflag = 'Y' where collectionid
= #{collectionid} returning 1")
          if rs != nil and rs.count > 0
            true
          else
```

```
            false
          end
        rescue =>e
          puts e.backtrace
          false
        end
      end

      def disable_collection_granule_export( collectionid)
        begin
          rs = @conn.exec("update bg_collection_configuration set collectionexportflag = 'N', granuleexportflag
= 'N' where collectionid = #{collectionid} returning 1")
          if rs != nil and rs.count > 0
            true
          else
            false
          end
      rescue =>e
          puts e.backtrace
          false
        end
      end

      def collection_export_enabled?( collectionid)
        begin
          rs = @conn.exec("select collectionexportflag,granuleexportflag from bg_collection_configuration where
collectionid = #{collectionid}")
          if rs != nil and rs.count > 0
            collectionexportflag = rs[0]['collectionexportflag']
            granuleexportflag = rs[0]['granuleexportflag']
            if collectionexportflag == 'Y' and granuleexportflag == 'Y'
              true
            else
              false
            end
          else
```

```
            puts "CollectionId  #{collectionid} does not exist  in collection configuration."
            false
          end

        rescue =>e
          puts e.backtrace
          false
        end
      end

      def collection_export_enabled_by_shortname?( shortname, version)
        begin
          rs = @conn.exec("select collectionexportflag,granuleexportflag from bg_collection_configuration where
shortname = '#{shortname}' and versionid = #{version}")
          if rs != nil and rs.count > 0
            collectionexportflag = rs[0]['collectionexportflag']
            granuleexportflag = rs[0]['granuleexportflag']
            if collectionexportflag == 'Y' and granuleexportflag == 'Y'
              true
            else
              false
            end
          else
            puts "Collection #{shortname} #{version} does not exist in collection configuration."
            false
          end

        rescue =>e
          puts e.backtrace
          false
        end
      end
      def get_collectionid_from_coll_config(shortname, versionid)
        begin
          rs = @conn.exec("select collectionid from bg_collection_configuration where shortname = '#{shortname}'
and versionid = #{versionid}")
```

```
          if rs != nil and rs.count > 0
            cid = rs[0]['collectionid']
            cid
          else
            nil
          end
        rescue =>e
          puts e.backtrace
          nil
        end

      end
      #get collection export status for a collection
      def get_collection_export_request_status(request_time, collectionid)
        begin
          rs = @conn.exec("select status from bg_export_request where collectionid = #{collectionid} and
enqueuetime > '#{request_time}'")
          if rs != nil and rs.count > 0
            status = rs[0]['status']
            status
          else
            nil
          end
        rescue =>e
          puts e.backtrace
          nil
        end
      end

      #get collection export status for a collection and an export queue
      def get_coll_export_request_status(request_time, collectionid, exportqueue)
        begin
          rs = @conn.exec("select status from bg_export_request where collectionid = #{collectionid} and
itemtype = 'CL' and exportqueue = '#{exportqueue}' and enqueuetime > '#{request_time}'")
          if rs != nil and rs.count > 0
            status = rs[0]['status']
```

```
    else
      nil
    end
  rescue =>e
    puts e.backtrace
    nil
  end
end

#get granule export status hash for all the active granules in a collection
def get_gran_export_request_statushash(request_time, collectionid, exportqueue)
  begin
    rv = 0
    statushash = Hash.new()
    aimdb = Sdps::AimDb.new(@conn)
    granules = aimdb.find_notdeleted_granuleids_by_collectionid(collectionid)
    count = granules.length
    if granules != nil
      granules.each do |granid|
        status = get_gran_export_request_status(request_time,granid,exportqueue)
        if status == nil
          rv = nil
        else
          statushash[granid] = status
        end
      end
      ##if any active granule doesn't make it to the bg_export_request table, then return nil
      if rv == nil
        nil
      else
        statushash
      end
    else
      nil
    end
  rescue =>e
```

```
          puts e.backtrace
          nil
        end
      end

      #get granule export status for a granule
      def get_gran_export_request_status(request_time,granuleid,exportqueue)
        begin
          rs = @conn.exec("select status from bg_export_request where granuleid = #{granuleid} and exportqueue =
'#{exportqueue}'and enqueuetime > '#{request_time}'")
          if rs != nil and rs.count > 0
            status = rs[0]['status']
            status
          else
            nil
          end
        rescue =>e
          puts e.backtrace
          nil
        end
      end

      def get_granules_by_time(collectionid, criteria, limit )
        begin
          granuleids = Array.new
          query = "select granuleid from amgranule where collectionid = #{collectionid } order by #{criteria}
limit #{limit}"

          rs = @conn.exec(query)
          if rs != nil and rs.count > 0
            rs.each do |row|
              id = row['granuleid']
              granuleids.push(id)
            end
            return granuleids
          end
```

```
        rescue =>e
          puts e.backtrace
          nil
        end
      end

      def get_gran_url_base()
        begin
          rs = @conn.exec("select propertyvalue from bg_configuration_property where propertyname =
'bmgt.granule.url.base'")
          if rs != nil and rs.count > 0
            urlbase = rs[0]['propertyvalue']
            urlbase
          else
            nil
          end
        rescue =>e
          puts e.backtrace
          nil
        end
      end

    end # class BmgtDb
  end # module  Bmgt
end # module Sdps
Current Test Script: bmgtxml.rb
# xml functions specific to BMGT
module Sdps
  module Bmgt

    require 'nokogiri'
    require 'sdps/tcpproxylogger'

    class Sdps::Bmgt::Xml

      def granule_xml_valid?(granule_xml)
```

```
        #result = system "/tools/libxml2-2.9.1/bin/xmllint --noout --path
/tools/common/test/BE_82_01/echo10/schema --schema Granule.xsd #{granule_xml}"
        #modified to use Nokogiri methods
        xml_doc = Nokogiri::XML(granule_xml)
        xsd = Nokogiri::XML::Schema(File.open(ECHO10_GRANULE_SCHEMA))
        xsd.validate(xml_doc).length == 0
      end

      def granule_xml_contains?(granule_xml, xpath_expr, value)
        xml_doc = Nokogiri::XML(granule_xml)
        xml_doc.xpath(xpath_expr).to_s == value
      end

      def granule_xml_getvalue(granule_xml, xpath_expr)
        xml_doc = Nokogiri::XML(granule_xml)
        xml_doc.xpath(xpath_expr).to_s
      end

      # timestamp = Time.now.strftime('%Y-%m-%dT%H:%M:%S.%6N')
      def http_put_for_granule?(log_pathname, requesttime, granuleid)
        begin
          #check if request time is in the time format used in tcp.log
          request_time = DateTime.parse(requesttime).strftime('%Y-%m-%dT%H:%M:%S.%6N')
          entries = Sdps::TcpProxyLogger.merge_requests(Sdps::TcpProxyLogger.parse_file(log_pathname))
          log_entry = entries.select {|x| x.timestamp > request_time and x.put? and
x.request.split("\r\n\r\n")[1].include?("#{granuleid}") }[0]
          if log_entry.nil?
              false
          else
              true
          end
        rescue =>e
          puts e.backtrace
          false
        end
      end
```

```
      def http_verification_put_for_granule?(log_pathname, requesttime, granuleid)
        begin
          #check if request time is in the time format used in tcp.log
          request_time = DateTime.parse(requesttime).strftime('%Y-%m-%dT%H:%M:%S.%6N')
          entries = Sdps::TcpProxyLogger.merge_requests(Sdps::TcpProxyLogger.parse_file(log_pathname))
          log_entry = entries.select {|x| x.timestamp > request_time and x.put? and
x.request.split("\r\n\r\n")[1].include?("#{granuleid}") }[0]
          if log_entry.nil?
              false
          else
              log_entry.to_s.include?("_diff=true")
          end
        rescue =>e
          puts e.backtrace
          false
        end
      end

      def http_delete_for_granule?(log_pathname, requesttime, granuleid)
        begin
          #check if request time is in the time format used in tcp.log
          request_time = DateTime.parse(requesttime).strftime('%Y-%m-%dT%H:%M:%S.%6N')
          entries = Sdps::TcpProxyLogger.merge_requests(Sdps::TcpProxyLogger.parse_file(log_pathname))
          log_entry = entries.select {|x| x.timestamp > request_time and x.delete? and
x.request.split("\r\n\r\n")[0].include?("#{granuleid}") }[0]
          if log_entry.nil?
              false
          else
              true
          end
        rescue =>e
          puts e.backtrace
          false
        end
      end
```

```
    # timestamp = Time.now.strftime('%Y-%m-%dT%H:%M:%S.%6N')
      def get_granule_xml(log_pathname, requesttime, granuleid)
        begin
          request_time = DateTime.parse(requesttime).strftime('%Y-%m-%dT%H:%M:%S.%6N')
          entries = Sdps::TcpProxyLogger.merge_requests(Sdps::TcpProxyLogger.parse_file(log_pathname))
          log_entry = entries.select {|x| x.timestamp > request_time and x.put? and
x.request.split("\r\n\r\n")[1].include?("#{granuleid}") }[0]
           if log_entry.nil?
                nil
           else
               metadata = log_entry.request.split("\r\n\r\n")[1]
           end
        rescue =>e
          puts e.backtrace
          nil
        end
      end

      def collection_xml_valid?(collection_xml)
        #modified to use Nokogiri methods
        xml_doc = Nokogiri::XML(collection_xml)
        xsd = Nokogiri::XML::Schema(File.open(ECHO10_COLLECTION_SCHEMA))
        xsd.validate(xml_doc).length == 0
      end

      def collection_xml_contains?(collection_xml, xpath_expr, value)
        xml_doc = Nokogiri::XML(collection_xml)
        xml_doc.xpath(xpath_expr).to_s == value
      end

       # timestamp = Time.now.strftime('%Y-%m-%dT%H:%M:%S.%6N')
      def http_put_for_collection?(log_pathname, requesttime, dataset)
        begin
          #status = false
          request_time = DateTime.parse(requesttime).strftime('%Y-%m-%dT%H:%M:%S.%6N')
```

```
          escDataset = ERB::Util.url_encode(dataset)

          entries = Sdps::TcpProxyLogger.merge_requests(Sdps::TcpProxyLogger.parse_file(log_pathname))
          log_entry = entries.select {|x| x.timestamp > request_time and x.put? and
x.request.split("\r\n\r\n")[0].include?("#{escDataset}") }[0]
          if log_entry.nil?
              false
          else
              true
          end

        rescue =>e
          puts e.backtrace
          false
        end
      end

      def http_verification_put_for_collection?(log_pathname, requesttime, dataset)
        begin
          #status = false
          request_time = DateTime.parse(requesttime).strftime('%Y-%m-%dT%H:%M:%S.%6N')
          escDataset = ERB::Util.url_encode(dataset)

          entries = Sdps::TcpProxyLogger.merge_requests(Sdps::TcpProxyLogger.parse_file(log_pathname))
          log_entry = entries.select {|x| x.timestamp > request_time and x.put? and
x.request.split("\r\n\r\n")[0].include?("#{escDataset}") }[0]
          if log_entry.nil?
            false
          else
            log_entry.to_s.include?("_diff=true")
          end

        rescue =>e
          puts e.backtrace
          false
        end
```

```
      end

      def http_delete_for_collection?(log_pathname, requesttime, dataset)
        begin
          status = false
          escDataset = ERB::Util.url_encode(dataset)

          request_time = DateTime.parse(requesttime).strftime('%Y-%m-%dT%H:%M:%S.%6N')

          entries = Sdps::TcpProxyLogger.merge_requests(Sdps::TcpProxyLogger.parse_file(log_pathname))
          log_entry = entries.select {|x| x.timestamp > request_time and x.delete? and
x.request.split("\r\n\r\n")[0].include?("#{escDataset}") }[0]
          if log_entry.nil?
              false
          else
              true
          end
        rescue =>e
          puts e.backtrace
          false
        end
      end
      # timestamp = Time.now.strftime('%Y-%m-%dT%H:%M:%S.%6N')
      def get_collection_xml(log_pathname, requesttime, coll)
        begin
          request_time = DateTime.parse(requesttime).strftime('%Y-%m-%dT%H:%M:%S.%6N')
          #xml_arr = Array.new
          shortName = coll.shortname
          vId = coll.versionid

          entries = Sdps::TcpProxyLogger.merge_requests(Sdps::TcpProxyLogger.parse_file(log_pathname))
          log_entry = entries.select {|x| x.timestamp > request_time and x.put? and
x.request.split("\r\n\r\n")[1].match(/<Collection><ShortName>#{shortName}<\/ShortName><VersionId>#{vId}/) !=
nil}[0]
          if log_entry.nil?
                nil
```

```
          else
              metadata = log_entry.request.split("\r\n\r\n")[1]
         end

        rescue =>e
        puts e.backtrace
        nil
        end
      end
    end # class Xml

  end #module Bmgt

end #module Sdps

Current Test Script: ems.rb
require 'sdps/ems/emsutils'

module Sdps
  module EMS
    DB_SUBSYSTEM = 'aim'
  end
end
Current Test Script: ivt.rb

module Sdps
  class InventoryValidation
      #include RSpec::Matchers

      attr_reader :conn, :mode, :dplhost, :pollinglocation

      def initialize(conn, mode)
         @conn = conn
         @mode = mode
         @dplhost = Conf.get_property('ivt','hostname')
         @pollinglocation = Conf.get_property("#{@mode.downcase}_ingest", "polling_location")
```

```
      end

      def GetOutputDir()
        #note: system call returns true/false status; `` call returns the command output
        outputdir=`ssh -q #{@dplhost} 'cat /usr/ecs/#{@mode}/CUSTOM/cfg/EcDlInventoryValidationTool.CFG | grep
VALIDATION_OUTPUT_DIR | sed -e 's/VALIDATION_OUTPUT_DIR=//g'`
        outputdir.delete("\n") + "/autotest/"
      end

      def RunIVT()
         rv = true
         #First remove all the files in the output directory
         begin
           #First remove all the output files in the output directory if exists
           dir = GetOutputDir()
           files = dir + "*"
           puts "files are: #{files}"

           #Got to make sure there are files to remove, otherwise it will go to rescue block
           #Note: The directory here is accessible to all machines here.
           if File.directory?(dir) == true and Dir.entries(dir) != [".", ".."]
             rv = system("ssh -q #{@dplhost} 'rm -f #{files}'")
           else
             puts "Directory #{dir} does not exist or no files in it - nothing to remove"
           end

           #Then remove the log file in the log directory if exists
           logfile = "/usr/ecs/#{@mode}/CUSTOM/logs/EcDlInventoryValidationTool.log"
           #can't use File.exists? because it's a remote file
           #if File.exists?(logfile)
           rv = system("ssh -q #{@dplhost} 'ls #{logfile}'")
           if rv == true
             rv = system("ssh -q #{@dplhost} 'rm -f #{logfile}'")
           else
             puts "#{logfile} does not exist - nothing to remove"
           end
```

```
          #Then run the IVT
          status = system("ssh -q #{@dplhost}
'/usr/ecs/#{@mode}/CUSTOM/utilities/EcDlInventoryValidationTool.pl #{@mode} --outputdir autotest'")
          #Note: IVT returns exitstatus =2 whenever discrepancies are found
          #which is not good. When IVT functions normally, it should return 0
          #$?.exitstatus shows the exitstatus
          puts "exitstatus is #{$?.exitstatus}"
          if $?.exitstatus != 0 and $?.exitstatus != 2
            rv = false
          end
        rescue =>e
          puts "error occured"
          puts e.backtrace
          rv = false
        end
        rv
      end

      def IsGranuleReported(granuleid, reportfilename)
        #Check to see if granuleid is included in the reportfile
        #note: reportfile is the filename only, which resides in the outputdir
        rv = true
        begin
          file = GetOutputDir() + "#{reportfilename}*"

          puts "file is #{file}"
          #Make sure the file exists before grep the granuleid
          #Since the file contains wildcard, even though there is only one
          #exists, can't use File.exists(file) funtion
          #if File.exists?(file)
          if !Dir.glob(file).empty?
            puts "file #{file} exists"
            grep = `ssh -q #{@dplhost} 'grep #{granuleid} #{file}'`
            grep = grep.delete("\n")
            #puts "grep is #{grep}, granuleid is #{granuleid}"
```

```
      if grep != granuleid.to_s
        rv = false
      end
    else
      puts "file #{file} doesn't exist"
      rv = false
    end
  rescue =>e
    puts "error caught"
    puts e.backtrace
    rv = false
  end
  rv
end

def IsGranuleLogged(granuleid)
  rv = true
  begin
    logfile = "/usr/ecs/#{@mode}/CUSTOM/logs/EcDlInventoryValidationTool.log"
    rv = system("ssh -q #{@dplhost} 'ls #{logfile}'")
    if rv == true
      grep = `ssh -q #{@dplhost} 'grep #{granuleid} #{logfile}'`
      if grep == ""
        puts "#{granuleid} is not logged"
        rv = false
      end
    else
       puts "#{logfile} does not exist"
       rv = false
    end
  rescue =>e
    puts "error caught"
    puts e.backtrace
    rv = false
  end
  rv
```

```
end

def GetGranules(query_findgran, query_modifygran, num_grans_needed)
  #Find existing granules using query_findgran.
  #if not found, update some granules to get the qualified granules using  query_modifygran

  @granules = Array.new
  rv = true
  begin
    rs = @conn.exec ("#{query_findgran}")
    if rs != nil and rs.count == #{num_grans_needed}
      rs.each do |row|
        @granules.push(row['GranuleId'])
      end
    else
      puts "Can't find existing qualified granules, update some granules""
      rs = @conn.exec ("#{query_modifygran}")
      if rs != nil and rs.count == #{num_grans_needed}
         rs.each do |row|
           @granules.push(row['GranuleId'])
         end
      else
        puts "Can't find qualified granules in the system"
        rv = false
      end
    end
  rescue =>e
    puts e.backtrace
    rv = false
  end

  if rv == true
    @granules
  else
    nil
  end
```

```
end

def Get_CheckInventoryB_Granules(num_grans_needed)
  #First make sure no granules with inconsistent EcsFlag in AmGranule and in AmCollection
  #Identify granules that are registered but not archived
  queryB1 = "select a.GranuleId
             from AmGranule a left join DlBcpGransToDelete b on a.GranuleId = b.granuleId
             where a.RegistrationTime is not null -- in DPL
             and b.granuleId is null -- do not return granules that are in DlBcpGransToDelete
             and a.ArchiveTime is null -- not archived in AIM
             and a.EcsFlag = 'Y' limit #{num_grans_needed}"

  #if query1 doesn't return enough existing granules, update some granules
  queryB2 = "update AmGranule
             set    ArchiveTime = null
             and    granuleid in (select granuleid
                                  from   AmGranules
                                  where  RegistrationTime is not null
                                  and    EcsFlag = 'Y'
                                  and    granuleid not in (select granuleid from DlBcpGransToDelete)
                                  limit #{num_grans_needed})
             returning granuleid"

    rs = @conn.exec("select 1 from AmCollection c join AmGranule g
                     on g.CollectionId = c.CollectionId where g.EcsFlag <> c.ECSFlag ")
    if rs == nil or rs.count > 0
      nil
    else
      GetGranules(queryB1,queryB2,num_grans_needed)
    end
end

def Get_CheckInventoryF_Granules()
  #  Identify granules in public Datapool with deleteEffectiveDate != NULL or
  #  DeleteFromArchive in ('Y','H')
```

```
#  or UnpublishFlag = "Y" in DsMdGranuleRestriction table.

@granulesF = Array.new
rv = true

begin
  #First try to find public granules that are logically deleted, DeleteFromArchive is 'Y'
  #DeleteFromArchive is 'H' and being in DsMdGranuleRestriction with UnpublishFlag
  #being 'Y',respectively. And a public granule that's eligible to be public

  #If not found, then find at least 5 granules that are public; and eligible to be public:
  #i.e. deleteEffectiveDate is null; DeleteFromArchive is not in ('Y','H');
  #not exists in DsMdGranuleRestriction table, then update them to be the granules above.

  basequery0 = "select granuleid, shortname,isorderonly from amgranule
                where  (IsOrderOnly is null or IsOrderOnly = 'B')
                and    deleteEffectiveDate is null
                and    DeleteFromArchive not in ('Y', 'H')
                and    granuleid not in (select granuleId
                                         from   DsMdGranuleRestriction)
                and    shortname not in ('PH', 'QA', 'DAP', 'HDF_MAP')
                and    collectionid in (select collectionid from bg_collection_configuration)"


  basequery1 = "select granuleid, shortname,isorderonly  from amgranule
                where  (IsOrderOnly is null or IsOrderOnly = 'B')
                and    deleteEffectiveDate is null
                and    DeleteFromArchive = 'Y'
                and    granuleid not in (select granuleId
                                         from   DsMdGranuleRestriction)
                and    shortname not in ('PH', 'QA', 'DAP', 'HDF_MAP')
                and    collectionid in (select collectionid from bg_collection_configuration)"


  basequery2 = "select granuleid, shortname,isorderonly from amgranule
                where  (IsOrderOnly is null or IsOrderOnly = 'B')
```

```
                and    deleteEffectiveDate is null
                and    DeleteFromArchive = 'H'
                and    granuleid not in (select granuleId
                                         from   DsMdGranuleRestriction)
                and    shortname not in ('PH', 'QA', 'DAP', 'HDF_MAP')
                and    collectionid in (select collectionid from bg_collection_configuration)"

basequery3 = "select granuleid, shortname, isorderonly from amgranule
                where  (IsOrderOnly is null or IsOrderOnly = 'B')
                and    deleteEffectiveDate is null
                and    DeleteFromArchive not in ('Y', 'H')
                and    granuleid in (select granuleId
                                     from   DsMdGranuleRestriction
                                     where  UnpublishFlag = 'Y')
                and    shortname not in ('PH', 'QA', 'DAP', 'HDF_MAP')
                and    collectionid in (select collectionid from bg_collection_configuration)"

basequery4 = "select granuleid, shortname, isorderonly from amgranule
                where  (IsOrderOnly is null or IsOrderOnly = 'B')
                and    deleteEffectiveDate is null
                and    DeleteFromArchive not in ('Y', 'H')
                and    granuleid not in (select granuleId
                                         from   DsMdGranuleRestriction)
                and    shortname not in ('PH', 'QA', 'DAP', 'HDF_MAP')
                and    collectionid in (select collectionid from bg_collection_configuration)"

granuleid0 = FindOneGranuleF(0, basequery0)
granuleid1 = FindOneGranuleF(1, basequery1)
granuleid2 = FindOneGranuleF(2, basequery2)
granuleid3 = FindOneGranuleF(3, basequery3)
granuleid4 = FindOneGranuleF(4, basequery4)

if granuleid0 == nil or granuleid1 == nil or granuleid2 == nil or granuleid3 == nil or granuleid4 ==
nil
   puts "Couldn't find 5 granules in the system"
   rv = false
```

```
    else
      puts "Five granules are: #{granuleid0},#{granuleid1},#{granuleid2},#{granuleid3},#{granuleid4}"
      @granulesF.push(granuleid0)
      @granulesF.push(granuleid1)
      @granulesF.push(granuleid2)
      @granulesF.push(granuleid3)
      @granulesF.push(granuleid4)
    end

  rescue =>e
    puts e.backtrace
    rv = false
  end

  if rv = true
    @granulesF
  else
    nil
  end
end


def FindOneGranuleF(granCategory, granBaseQuery)
  #Keep running the granBaseQuery until finding one granule
  #that have all the files exist(required by unpublishing);
  #Or until the granBaseQuery doesn't return any new granules
  #Then find an eligible public granule to update to the desired one
  #based on the granCategory

  begin
    found = false
    #modify is set to true when looking for a public granule to modify
    modify = false
    granuleid = nil
    shortname = nil
```

```
while ( !found  )
  #Run the granBaseQuery to find a granuleid
  puts "granBaseQuery is: #{granBaseQuery}"
  rs = @conn.exec("#{granBaseQuery}" + " limit 1")
  if rs != nil and rs.count == 1
    granuleid = rs[0]['granuleid']
    shortname = rs[0]['shortname']
    isorderonly = rs[0]['isorderonly']

    if GranuleFilesExist(shortname, granuleid)
      puts "granule files exist for granule: #{granuleid},#{shortname},#{isorderonly}"
      found = true
    else
       puts "granule files do not exist for granule: #{granuleid}, #{shortname}"
      granBaseQuery = granBaseQuery + " and granuleid != #{granuleid}"
    end
  else
    #if can't find a public granule to modify, then break out
    if modify
      puts "Can't get a public granule to modify either"
      granuleid = nil
      break
    end

    puts "can't find existing qualified granule. get a public granule to modify"
    granBaseQuery = "select granuleid, shortname,isorderonly from amgranule
                 where  (IsOrderOnly is null or IsOrderOnly = 'B')
                 and    deleteEffectiveDate is null
                 and    DeleteFromArchive not in ('Y', 'H')
                 and    granuleid not in (select granuleId
                                          from   DsMdGranuleRestriction)
                 and    shortname not in ('PH', 'QA', 'DAP', 'HDF_MAP')
                 and    collectionid in (select collectionid from bg_collection_configuration)"
    modify = true
  end
end
```

```
          if granuleid != nil and modify
            case granCategory
              when 0
                granUpdateQuery="update AmGranule set deleteEffectiveDate = localtimestamp where granuleid =
#{granuleid} returning 1"

              when 1
                granUpdateQuery="update AmGranule set DeleteFromArchive = 'Y' where granuleid = #{granuleid}
returning 1"

              when 2
                granUpdateQuery="update AmGranule set DeleteFromArchive = 'H' where granuleid = #{granuleid}
returning 1"

              when 3
                granUpdateQuery="insert into DsMdGranuleRestriction (granuleId, RestrictionFlag, UnpublishFlag)
select #{granuleid}, restrictionflag, 'Y' from DsMdRestrictionFlag limit 1 returning 1"
              else
                 granUpdateQuery=nil
            end

            #run the update query
            if granUpdateQuery != nil
              puts "update query is: #{granUpdateQuery}"
              rs = @conn.exec("#{granUpdateQuery}")
              if rs != nil and rs.count == 1
                puts "update is successful for granule: #{granuleid}"
              else
                puts "update failed for granule: #{granuleid}"
                granuleid = nil
              end
            end
          end
        rescue =>e
          puts e.backtrace
```

```
    granuleid = nil
  end

  granuleid
end

def GranuleFilesExist(shortname, granuleid)
  #Make sure the granule's files exist, if not, run the granBaseQuery
  #again to get a different granule.
  begin
    rv = true
    puts "select absfilepath from ProcGetGrFiles('#{shortname}',#{granuleid})"
    rs = @conn.exec("select absfilepath from ProcGetGrFiles('#{shortname}',#{granuleid})")
    if rs != nil and rs.count > 0
      rs.each do |row|
        file = row['absfilepath']
        if !File.file?("#{file}")
          puts "file doesn't exist: #{file}"
          rv = false
          break
        end
      end
    else
      rv = false
    end
  rescue =>e
    puts "ERROR HAPPENED in GranuleFilesExist"
    puts e.backtrace
    rv = false
  end

  rv
end

def Get_CheckInventoryG_Granules()
  #  ECS granules that in public collections, ARE eligible to be public per S-DPL-08050,
```

```
    #  and for which no more recent replacement granules are public, but that are currently
    #  in the hidden DataPool

  end

  def Get_CheckInventoryI_Granules(num_grans_needed)
    # Hidden, not logically deleted browse, associated with non logically deleted and
    # not DFAed SC

    queryI1 = "select BrowseId
               from   AmBrowse b
               where  b.IsOrderOnly in ('H', 'Y')
               and    b.deleteEffectiveDate is null
               and exists (
                           select 1
                           from AmGranule s join AmBrowseGranuleXref x on s.GranuleId = x.GranuleId
                           where b.BrowseId = x.BrowseId -- correlate
                           and  s.deleteEffectiveDate is null
                           and  s.DeleteFromArchive = 'N'
                          )
               limit #{num_grans_needed}"


    queryI2 = "update AmBrowse
               set    isOrderOnly = 'H'
               where  browseid in (select BrowseId
                                   from   AmBrowse b
                                   where  b.IsOrderOnly is null
                                   and    b.deleteEffectiveDate is null
                                   and exists (
                                               select 1
                                               from AmGranule s join AmBrowseGranuleXref x on s.GranuleId =
x.GranuleId
                                               where b.BrowseId = x.BrowseId -- correlate
                                               and  s.deleteEffectiveDate is null
                                               and  s.DeleteFromArchive = 'N'
```

```
                                        )
                                 limit #{num_grans_needed})"


  GetGranules(queryI1, queryI2, num_grans_needed)
end

def Get_CheckInventoryJ_Granules(num_grans_needed)
  # Public DPL Browse Granules is not associated with any Science granules
  queryJ1 = "select BrowseId
             from   AmBrowse b
             where  (b.IsOrderOnly is null or b.IsOrderOnly = 'B')
             and    not exists ( -- but no an associated science granule
                                 select 1
                                 from AmBrowseGranuleXref x
                                 where b.BrowseId = x.BrowseId -- correlate
                                )
             limit #{num_grans_needed}"

  queryJ2 = "delete
             from   AmBrowseGranuleXref
             where  browseid in (select BrowseId
                                 from   AmBrowse
                                 where  (IsOrderOnly is null or IsOrderOnly = 'B')
                                 --should exclude the logically deleted browse in IVT
                                 --and deleteEffectiveDate is null
                                 limit  #{num_grans_needed}
                                )"

  GetGranules(queryJ1,queryJ2,num_grans_needed)
end

def Get_CheckInventoryK_A_Granules(num_grans_needed)
  # First Find public granules with hidden path
  queryKA1 = "select g.granulid
              from   AmGranule g join AmDataFile d on g.GranuleId = d.GranuleId
```

```
                    where  (g.isorderonly is null or g.isorderonly = 'B')
                    and    d.DirectoryPath not like ('/' || esdt(g.ShortName, g.VersionId) || '/%')
                    limit  #{num_grans_needed}"

        queryKA2 = "update AmGranule
                    set    isorderonly = null
                    where  granuleid in (select g.granuleid
                                         from   AmGranule g join AmDataFile d on g.GranuleId = d.GranuleId
                                         and    g.isorderonly = 'H'
                                         and    d.DirectoryPath not like ('/' || esdt(g.ShortName, g.VersionId)
|| '/%')
                                         limit  #{num_grans_needed})
                    returning granuleid"

        GetGranules(queryKA1,queryKA2,num_grans_needed)
      end

      def Get_CheckInventoryK_B_Granules(num_grans_needed)
        # Then Find hidden granules in public path
        queryKB1 = "select g.granulid
                    from   AmGranule g join AmDataFile d on g.GranuleId = d.GranuleId
                    where  g.isorderonly in ('H', 'Y')
                    and    d.DirectoryPath like ('/' || esdt(g.ShortName, g.VersionId) || '/%')
                    limit  #{num_grans_needed}"

        queryKB2 = "update AmGranule
                    set    isorderonly = 'H'
                    where  granuleid in (select g.granuleid
                                         from   AmGranule g join AmDataFile d on g.GranuleId = d.GranuleId
                                         and    g.isorderonly is null or g.isorderonly = 'B'
                                         and    d.DirectoryPath like ('/' || esdt(g.ShortName, g.VersionId) ||
'/%')
                                         limit  #{num_grans_needed})
                    returning granuleid"

        GetGranules(queryKB1,queryKB2,num_grans_needed)
```

```
      end

      def Get_CheckInventoryL_Granules(ingest, data)
        #First check if the two granules we are looking for are already in the system.
        #If so, get them, otherwise, ingest them (

        @queries = Array.new
        @queries.push("update AmGranule set versionnumber = 1 where granuleid in (select granuleid from
AmDataFile where onlinefilename = 'MOD29P1D.A2011060.h05v28.005.2011061081202_DP_7F_01_200_L1.hdf') and
isorderonly in ('H', 'Y') returning granuleid")
        @queries.push("update AmGranule set versionnumber = 1 where granuleid in (select granuleid from
AmDataFile where onlinefilename = 'MOD29P1D.A2011060.h05v29.005.2011061081110_DP_7F_01_200_L2.hdf') and
isorderonly in ('H', 'Y') returning granuleid")
        #@queries.push("select granuleid from AmDataFile where onlinefilename =
'MOD29P1D.A2011060.h05v28.005.2011066081202_DP_7F_01_200_L3.hdf'")
        #@queries.push("select granuleid from AmDataFile where onlinefilename =
'MOD29P1D.A2011060.h05v29.005.2011066081110_DP_7F_01_200_L4.hdf'")

        @granules = Array.new
        rv = true
        begin
          @queries.each {|query|
            rs = @conn.exec ("#{query}")
            if rs != nil and rs.count==1
              rs.each do |row|
                @granules.push(row['granuleid'])
              end
            end
          }
          if @granules.length != 2
            puts "test data are not ready, ingest the granules for S-1l"
            rv = false
          end
        rescue =>e
          puts e.backtrace
          rv = false
```

```
        end

        if rv == true
          @granules
        else
          nil
        end
      end
  end
end
```

Current Test Script: errors.rb

```
module Sdps
  ModeMissingError = Class.new(RuntimeError)
  NotImplementedYet = Class.new(RuntimeError)
  GranuleNotFoundError = Class.new(RuntimeError)
end
```

Current Test Script: util.rb

```
require 'securerandom'
require 'time'

module Sdps
  module Util
    @@default_timeout_seconds = 120
    @@default_interval_seconds = 2

    class << self

      # Silences warnings while executing a block.
      def silence_warnings
        begin
          old_verbose, $VERBOSE = $VERBOSE, nil
          yield
        ensure
          $VERBOSE = old_verbose
```

```
        end
      end

      # Generates a random 32-character hex string.
      def token
        SecureRandom.hex(16)
      end

      # Repeat a block at fixed intervals until it returns a true value
      # (i.e., not false or nil) or the elapsed time exceeds the
      # timeout.
      def poll(timeout_seconds=@@default_timeout_seconds, interval_seconds=@@default_interval_seconds)
        timeout = Time.now + timeout_seconds
        result = yield
        until result or Time.now > timeout
          sleep interval_seconds
          result = yield
        end
        result
      end

    end
  end
end
```

Current Test Script: xml.rb

```
# generic xml functions
module Sdps
   module Xml
      require 'nokogiri'

        def xml_valid?(schema_file, xmldata)
            xml_doc = Nokogiri::XML(xmldata)
            xsd = Nokogiri::XML::Schema(File.open(schema_file))
            xsd.validate(xml_doc).length == 0
        end
```

```
def xmlfile_valid?(schema_file, xmlfile)
    xml_doc = Nokogiri::XML(File.read(xml_file))
    xsd = Nokogiri::XML::Schema(File.open(schema_file))
    xsd.validate(xml_doc).length == 0
end

def get_value(xml_file, xpath_expr)
     #puts xml_file
     #puts xpath_expr
     begin
        puts xpath_expr
        xml_doc = Nokogiri::XML(File.read(xml_file))
        #xml_doc.remove_namespaces!
        xml_doc.xpath(xpath_expr).to_s
     rescue=> e
         puts "exception:" + e.to_s
         puts e.backtrace
     end
end

def contains?(xml_file, xpath_expr)
    xml_doc = Nokogiri::XML(File.read(xml_file))
    #xml_doc.remove_namespaces!
    if xml_doc.xpath(xpath_expr).nil?
       false
    else
       if xml_doc.xpath(xpath_expr).to_s.nil?
          false
       else
          if xml_doc.xpath(xpath_expr).to_s.empty?
            false
          else
            true
          end
       end
    end
```

```
        end

        def has_value?(xml_file, xpath_expr, value)
            xml_doc = Nokogiri::XML(xml_file)
            xml_doc.remove_namespaces!
            xml_doc.xpath(xpath_expr).to_s == value
        end

    end
end
```

Current Test Script: tcpproxylogger.rb

```
require 'logger'
require 'sdps/tcpproxylogger/timestamp'
require 'sdps/tcpproxylogger/logentry'

# The TcpProxyLogger can parse logs that it writes.  The resulting list
# of LogEntry objects can be searched for requests on or after a
# specified time, for example.

# t0 = Time.new
# # ...export BMGT requests...
# log = Sdps::TcpProxyLogger.parse_file('/path/to/tcp.log')
# requests = log.select{|x| x.time > t0 and x.request?}

module Sdps
  class TcpProxyLogger < Logger

    include Timestamp

    class << self
      # Parse an entire log written by the TcpProxyLogger.
      # Returns a list of LogEntrys.
      def parse_file(log_filename)
        LogEntry.parse_file(log_filename)
      end
```

```
  # Combines request header and body that were sent separately.
  # I.e., one log entry has the request header and a later log entry
  # has only the body.
  def merge_requests(log_entries)
    LogEntry.merge_requests(log_entries)
  end
end

#def initialize(*args)
#  super
#  self.level = INFO
#  self.formatter = proc { |severity, datetime, progname, data|
#    hash = {timestamp:timestamp(datetime), severity: severity}
#    if data.kind_of? Exception
#      hash[:msg] = data.to_s
#      hash[:exception] = data.class
#      hash[:backtrace] = data.backtrace
#    elsif data.kind_of? Hash
#      data.each do |k,v|
#        if v.kind_of? Exception
#          hash[:msg] = v.to_s
#          hash[:exception] = v.class
#          hash[:backtrace] = v.backtrace
#        else
#          hash[k] = v
#        end
#      end
#    else
#      hash[:msg] = data.to_s
#    end
#    hash.to_s + "\n"
#  }
#end

#def request(uuid, data, msg=nil)
#  tcp(:request, uuid, data, msg)
```

```
    #end

    #def response(uuid, data, msg=nil)
    #  tcp(:response, uuid, data, msg)
    #end

    #def tcp(direction, uuid, data, msg)
    #  hash = {uuid:uuid, direction => data}
    #  hash[:msg] = msg if msg
    #  self.info(hash)
    #end

  end
end
Current Test Script: qauu.rb
  module Sdps
    require 'sdps/qauu/qauudb'
    require 'sdps/qauu/qarequest'
    require 'sdps/xml'

    class QAUU
        include Sdps::Xml

       attr_reader :mode, :qauudb,:qauuhost,:qarequest, :site_configured, :sitename, :shortname

       QAUU_PROPERTIES_FILE = "/usr/ecs/CUSTOM/#{@mode}/"

       def initialize(mode, db_conn)
         @mode = mode
         @qauudb = Sdps::QAUU::QAUUDb.new(db_conn)
         @qauuhost = Conf.get_property('qauu','hostname')
       end

       def get_configured_site()

       end
```

```
        def configure_site(shortname, site, email)
              status = @qauudb.add_collection_to_site(shortname, site)
              @sitename = site
              if status == true
                   @site_configured = update_properties_file(sitename, email)
              else
                   @site_configured = false
              end
        end

        def update_properties_file(sitename, email)
            #prop1 = "#{sitename}_EMAIL_FROM_ADDRESSES=#{email}"
            #prop2 = "#{sitename}__EMAIL_REPLY_ADDRESS=#{email}"
            #prop3 = "#{sitename}_NOTIFICATION_ON_SUCCESS=N"

            prop1 = "#{sitename}_EMAIL_FROM_ADDRESSES"
            prop2 = "#{sitename}_EMAIL_REPLY_ADDRESS"
            prop3 = "#{sitename}_NOTIFICATION_ON_SUCCESS"

            status1= update_property(prop1)
            status2 = update_property(prop2)
            status3 = update_property(prop3)
            status1 and status2 and status3
        end

        def get_property(prop)
             puts prop
             grep = `ssh -q #{@qauuhost} 'grep #{prop}
/usr/ecs/#{@mode}/CUSTOM/cfg/EcDsAmQaUpdateUtility.properties'`
             puts grep
             grep
        end

        def update_property(prop)
             #grep = `ssh -q #{@qauuhost} 'grep #{prop}
```

```
/usr/ecs/#{@mode}/CUSTOM/cfg/EcDsAmQaUpdateUtility.properties'`
            grep = get_property(prop)
            if grep.nil? or grep.empty?
               #add property and grep if it added
               puts "adding property - " + prop
               grep = `ssh -q #{@qauuhost} 'echo #{prop} >>
/usr/ecs/#{@mode}/CUSTOM/cfg/EcDsAmQaUpdateUtility.properties; grep #{prop}
/usr/ecs/#{@mode}/CUSTOM/cfg/EcDsAmQaUpdateUtility.properties'`
           end
           !grep.nil? and !grep.empty?
       end

       def get_valid_science_quality_flags()
          begin
            @qauuhost = Conf.get_property('qauu','hostname')
            valid_science_flags = `ssh -q #{@qauuhost} 'grep VALID_SCIENCE_QA_FLAGS
/usr/ecs/#{@mode}/CUSTOM/cfg/EcDsAmQaUpdateUtility.properties'`
            if !valid_science_flags.nil?  and !valid_science_flags.empty?
                 puts valid_science_flags
                 valid_science_flags.split("=")[1].split(",")
            end
          rescue =>e
            puts e.backtrace
            nil
          end
        end

        def get_measured_parameter_science_quality_flag(metadatafile, parameter_name)
           measured_parameter_science_quality_flag = get_value(metadatafile,
"//MeasuredParameter/MeasuredParameterContainer[ParameterName='#{parameter_name}']/QAFlags/ScienceQualityFlag/te
xt()")
        end

        def get_measured_parameter(metadatafile)
            measured_parameter = get_value(metadatafile,
"//MeasuredParameter/MeasuredParameterContainer/ParameterName/text()")
```

```
        end

        def configure_request(versionid, description)
              @qarequest = QaRequest.new(@mode, @sitename, @shortname, versionid)
              @qarequest.initialize_qa_request_file(description)
        end

        #TODO : need a better way to do this!
        def create_science_qa_request_with_granule_ur(granuleid, measured_parameter_name, qa_flag_value,
qa_flag_explanation)
             update_buffer = @qarequest.create_science_qa_request_with_granule_ur(granuleid,
measured_parameter_name, qa_flag_value, qa_flag_explanation)

             #grep QA_REQUEST_DIR
             grep_result = `ssh -q #{@qauuhost} 'grep QA_REQUEST_DIR
/usr/ecs/#{@mode}/CUSTOM/cfg/EcDsAmQaUpdateUtility.properties'`
             puts "grep result="
             puts grep_result
             if !grep_result.nil? and !grep_result.empty?
                     qa_request_dir = grep_result.split("=")[1].chomp
                     puts "qa_request_dir=" + qa_request_dir
                      #copy buffer into request file
                     #qa_request_file = "#{qa_request_dir}/#{@qarequest.qa_request_filename}"
                     qa_request_file = "#{qa_request_dir}/#{@qarequest.qa_request_filename}"
                     temp_file = "#{qa_request_file}.tmp"
                     puts "temp_file = #{temp_file}"
                     puts "qa_request_file= #{qa_request_file}"
                     puts "qa_request_file = #{@qarequest.qa_request_filename}"
                     status = `ssh -q #{@qauuhost} 'echo #{update_buffer[0]} > #{qa_request_file}; echo
#{update_buffer[1]} >> #{qa_request_file}; echo #{update_buffer[2]}  > #{temp_file}; cat #{temp_file}  | sed "s/
\+/\t/g" >> #{qa_request_file}; echo #{update_buffer[3]} >> #{qa_request_file}'`
                     @qarequest.qa_request_filename
             else
                     nil
             end
       end
```

```
       def qa_update(request_file)
           begin
             puts "qauu-requestfile:" +  request_file

             status = system("ssh -q #{@qauuhost} 'cd /usr/ecs/#{mode}/CUSTOM/utilities;
/usr/ecs/#{@mode}/CUSTOM/utilities/EcDsAmQAUUStart #{@mode} -file #{request_file} -noprompt'")
             puts status
             true
           rescue =>e
             puts e.backtrace
             false
           end
        end

         #TODO: check logs for success or errors
        def check_log()

        end

    end # class QAUU
  end # module Sdps
Current Test Script: collection.rb
##################################################################################
#Queries specific to AmCollection table
#The module is included in AimDb class
###################################################################################
module Sdps

  module Collection

    #find collection queries
    COLLECTION_BY_SHORTNAME_VERSIONID_SQL = "
      select *
      from amcollection
```

```
      where shortname = $1
      and versionid = $2
    "
    def find_collection_by_shortname_versionid(shortname, versionid)
      find_collections(COLLECTION_BY_SHORTNAME_VERSIONID_SQL, shortname, versionid)[0]
    end

    COLLECTION_BY_COLLECTIONID_SQL = "
      select *
      from amcollection
      where collectionid = $1
    "
    def find_collection_by_collectionid(collectionid)
      find_collections(COLLECTION_BY_COLLECTIONID_SQL, collectionid)[0]
    end

    COLLECTION_INSTALLED_SQL = "
      select esdtstate
      from DsGeEsdtConfiguredType
      where configuredname = $1
      and versionid = $2
    "

    def is_collection_installed?(shortname, versionid)
      rv = false
      rs = do_query(COLLECTION_INSTALLED_SQL,shortname, versionid)
      if rs.count > 0
        if(rs[0]['esdtstate'] == 'installed')
          rv = true
        end
      end
      return rv
    end
  end
end
```

Current Test Script: amgranule.rb

```
# Represents a row from amgranule

require 'sdps/dbrecord'
require 'sdps/aimdbextras'

module Sdps
  class AimDb
    class AmGranule < DbRecord
      include AimDbExtras

      db_attr :granuleid, "bigint"
      db_attr :collectionid, "bigint"
      db_attr :shortname, "character varying(8)"
      db_attr :versionid, "smallint"
      db_attr :ecsflag, "character(1)"
      db_attr :granulestate, "character(2)"
      db_attr :registrationtime, "timestamp without time zone"
      db_attr :archivetime, "timestamp without time zone"
      db_attr :publishtime, "timestamp without time zone"
      db_attr :lastupdate, "timestamp without time zone"
      db_attr :deletefromarchive, "character(1)"
      db_attr :deleteeffectivedate, "timestamp without time zone"
      db_attr :externalid, "character varying(80)"
      db_attr :isorderonly, "character(1)"
      db_attr :versionnumber, "integer"
      db_attr :archiveneededflag, "character(1)"
      db_attr :pgeversion, "character varying(10)"
      db_attr :productiondatetime, "timestamp without time zone"
      db_attr :rangebeginningdate, "date"
      db_attr :rangebeginningtime, "character varying(20)"
      db_attr :rangeendingdate, "date"
      db_attr :rangeendingtime, "character varying(20)"
      db_attr :calendardate, "date"
      db_attr :timeofday, "character varying(20)"
      db_attr :beginningdatetime, "timestamp without time zone"
```

```
      db_attr :endingdatetime, "timestamp without time zone"
      db_attr :daynightflag, "character(5)"
      db_attr :sizembecsdatagranule, "double precision"
      db_attr :localgranuleid, "character varying(80)"
      db_attr :localversionid, "character varying(60)"
      db_attr :radiometricdbversion, "character varying(255)"
      db_attr :reprocessingplanned, "character varying(45)"
      db_attr :reprocessingactual, "character varying(20)"
      db_attr :geometricdbversion, "character varying(255)"
      db_attr :dapid, "character varying(12)"
      db_attr :dapinsertdate, "timestamp without time zone"

    end
  end
end
```

Current Test Script: dsmdrestrictionflag.rb

```
module Sdps

    module DsMdRestrictionFlag

    GET_RESTRICTION_SQL = "select restrictionflag, description  from dsmdrestrictionflag where restrictionflag =
$1"
    ADD_RESTRICTION_SQL = "insert into dsmdrestrictionflag values ($1, $2) returning restrictionflag"
    UPDATE_RESTRICTION_SQL = "update dsmdrestrictionflag set description = $2 where restrictionflag = $1
returning restrictionflag"
    DELETE_RESTRICTION_SQL = "delete from dsmdrestrictionflag where restrictionflag = $1"


    def get_restriction (restrictionflag)
      rs = do_query(GET_RESTRICTION_SQL, restrictionflag)
      if rs.count > 0
        rs[0]['restrictionflag']
      else
        nil
      end
```

```
    end

    def add_restriction (restrictionflag, description)
      if get_restriction(restrictionflag).nil?
         rs = do_query(ADD_RESTRICTION_SQL, restrictionflag, description)
      else
         rs = do_query(UPDATE_RESTRICTION_SQL, restrictionflag, description)
      end
      if rs.count > 0
        true
      else
        false
      end
    end

    def delete_restriction (restrictionflag)
      if get_restriction(restrictionflag) == true
          do_query(DELETE_RESTRICTION_SQL, restrictionflag)
      end
      true
    end

    end # class DsMdRestrictionFlag

end #module Sdps
Current Test Script: aim_db_granule_spec.rb
#Note:  These are using the live DB and will only work as expected if the data is in this state
# perhaps in the future it should be updated to have a mock db_conn which returns the data...

require_relative '../../aim_db'
require_relative '../../../db_utils'
require '/usr/ecs/DEV06/CUSTOM/lib/COM/ruby/pgx'


describe Sdps::AimDb::Granule do
  before(:all) do
```

```
  @mode = ENV['MODE']
  @mode.should_not be_nil
  @db_subsystem = 'aim'
  @db_conn = DBUtils::DBConn.new(@db_subsystem, @mode)
  @db_conn.conn.should_not be_nil

  #Again, this test is geared against the following granule.  If you want to use a different
  #granule w/o rewriting the test, just update the fields below
  @granuleid = "68035"
  @shortname = "AE_SI12"
  @versionid = "2"
  @registrationtime = "2/2/2010 7:42:25 AM"
  @localgranuleid = "AMSR_E_L3_SeaIce12km_V12_20030105.hdf"

  #granuleid with qa/ph
  @granuleid2 = "70343"
end

 before(:each) do
  @granule = Sdps::AimDb::Granule.new(@db_conn.conn)
end

describe"#new" do
  it "takes one argument and creates a class of aim_db" do
    @granule.should be_an_instance_of Sdps::AimDb::Granule
  end
end

it "finds a granule by granule id" do
  @granule.find_granule_by_id(@granuleid)
  @granule.granuleid.should eql @granuleid
  @granule.installed.should eql true
  @granule.shortname.should eql @shortname
  @granule.versionid.should eql @versionid
end
```

```
  it "tells you a granule isn't installed if it isn't found" do
    @granule.find_granule_by_id("0")
    @granule.installed.should eql false
  end

   it "finds a granule by version number and shortname" do
    @granule.find_granule_by_shortname_version(@shortname,  @versionid)
    @granule.installed.should eql true
    @granule.shortname.should eql @shortname
    @granule.versionid.should eql @versionid
  end

    it "finds a granule by localgranuleid, version number and shortname" do
    @granule.find_granule_by_localgranuleid(@localgranuleid, @shortname, @versionid)
    @granule.installed.should eql true
    @granule.granuleid.should eql @granuleid
  end

  it "finds a granule by time, localgranuleid, version number and shortname" do
    @granule.find_granule_by_localgranuleid(@localgranuleid, @shortname, @versionid, @registrationtime)
    @granule.installed.should eql true
    @granule.granuleid.should eql @granuleid
  end

  it "tells you a granule is not installed if it doesn't match time, localgranuleid, version number and
shortname" do
    @granule.find_granule_by_localgranuleid(@localgranuleid, @shortname, @versionid, "2/2/3010 7:42:25 AM")
    @granule.installed.should eql false
    @granule.granuleid.should eql nil
  end

   it "finds a granule by time, grfilename, version number and shortname" do
    @granule.find_granule_by_grfilename(@localgranuleid, @shortname, @versionid, @registrationtime)
    @granule.installed.should eql true
    @granule.granuleid.should eql @granuleid
  end
```

```
   it "tells you a granule is not installed if it doesn't match time, localgranuleid, version number and
shortname" do
    @granule.find_granule_by_grfilename(@localgranuleid, @shortname, @versionid, "2/2/3010 7:42:25 AM")
    @granule.installed.should eql false
    @granule.granuleid.should eql nil
  end

  it "finds associated granules" do
    assoc_granules = @granule.find_associated_granules(@granuleid2)
    assoc_granules.should_not eql nil
    assoc_granules["ph"].size.should > 0
    assoc_granules["ph"][0].should_not eql nil
    assoc_granules["qa"].size.should > 0
    assoc_granules["qa"][0].should_not eql nil
    assoc_granules["browse"].size.should eql 0
    assoc_granules["hdf"].size.should eql 0
  end
end

Current Test Script: aim_db_collection_spec.rb
#Note:  These are using the live DB and will only work as expected if the data is in this state
# perhaps in the future it should be updated to have a mock db_conn which returns the data...

require_relative '../../aim_db'
require_relative '../../../db_utils'
require '/usr/ecs/DEV06/CUSTOM/lib/COM/ruby/pgx'

describe Sdps::AimDb::Collection do
  before(:all) do
    @mode = ENV['MODE']
    @mode.should_not be_nil
    @db_subsystem = 'aim'
    @db_conn = DBUtils::DBConn.new(@db_subsystem, @mode)
    @db_conn.conn.should_not be_nil
```

```
  #This test is geared against the following collection.  If you want to use a different
  #collection w/o rewriting the test, just update the fields below
  @collectionid = "71170"
  @shortname = "NISE"
  @versionid = "2"
  @insertTime = "10/2/2013 10:56:26 AM"

  #collection with multiple granules
  @collectionid2 = "50084"

end

before(:each) do
  @collection = Sdps::AimDb::Collection.new(@db_conn.conn)
end

it "finds a collection by collection id" do
  @collection.find_collection_by_id(@collectionid)
  @collection.installed.should eql true
  @collection.collectionid.should eql @collectionid
  @collection.shortname.should eql @shortname
  @collection.versionid.should eql @versionid
  @collection.collectionexportflag.should eql 'Y'
  @collection.collectionexportflag.should eql 'Y'
end

 it "finds a collection by shortname and version" do
  @collection.find_collection_by_shortname_version(@shortname, @versionid)
  @collection.installed.should eql true
  @collection.shortname.should eql @shortname
  @collection.versionid.should eql @versionid
end

it "finds a collection by time, shortname and version" do
  @collection.find_collection_by_shortname_version(@shortname, @versionid, @insertTime)
  @collection.installed.should eql true
```

```
    @collection.shortname.should eql @shortname
    @collection.versionid.should eql @versionid
  end

   it "tells you a collection isn't installed if it doesn't meet the time, shortname and version" do
    granules = @collection.get_active_granules_in_collection(@collectionid2)
    granules.should_not eql nil
    granules.length.should > 0


  end

end


Current Test Script: granule.rb
##########################################################################################
############
# Queries specific to AmGranule table
# this module is included in AimDb class
##########################################################################################
############
module Sdps

  module Granule

    #queries on amgranule and related tables
    GRANULE_DATAFILE_SQL = "select getdatafile(granuleid) as filename from amgranule where granuleid = $1"
    def get_granule_datafile(granuleid)
        rs = do_query(GRANULE_DATAFILE_SQL, granuleid)
        if  rs.count > 0
          rs[0]['filename']
        else
           nil
        end
    end
```

```
    #queries on amgranule and related tables
    ONLINE_FILENAME_SQL = "select filesystempath || groupid || directorypath || onlinefilename as
online_filename
                                  from dlfilesystems d, amcollection c,amdatafile m, amgranule g where
m.granuleid = $1
                                  and m.granuleid = g.granuleid and g.shortname = c.shortname and g.versionid =
c.versionid
                                  and c.filesystemlabel = d.filesystemlabel"
    def get_online_filename(granuleid)

      rs = do_query(ONLINE_FILENAME_SQL,granuleid)
      if rs.count > 0
        rs[0]['online_filename']
      else
        nil
      end
    end

    ONLINE_METADATA_FILENAME_SQL = "select filesystempath || groupid || onlinemetdirectorypath ||
onlinemetfilename as online_metadata_filename
                                            from dlfilesystems d, amcollection c,ammetadatafile m, amgranule g
where m.granuleid = $1
                                            and m.granuleid = g.granuleid and g.shortname = c.shortname and
g.versionid = c.versionid
                                            and c.filesystemlabel = d.filesystemlabel"
    def get_online_metadata_filename(granuleid)
      rs = do_query(ONLINE_METADATA_FILENAME_SQL,granuleid)
      if rs.count > 0
        rs[0]['online_metadata_filename']
      else
        nil
      end
    end

    GRANULE_METADATA_FILENAME_SQL = "select p.path || '/' || f.archivemetfilename as metadata_filename from
```

```
ammetadatafile f
                                                join dsmdxmlpath p on f.archivepathid = p.archivepathid where
f.granuleid = $1"
    def get_granule_metadata_file(granuleid)
      rs = do_query(GRANULE_METADATA_FILENAME_SQL,granuleid)
      if rs.count > 0
        rs[0]['metadata_filename']
      else
        nil
      end
    end

    ABSOLUTE_FILEPATHNAME_SQL = "select absolutefilesystempath || groupid || directorypath || onlinefilename as
filepathname
                                  from dlfilesystems d, amcollection c,amdatafile m, amgranule g where
m.granuleid = $1
                                  and m.granuleid = g.granuleid and g.shortname = c.shortname and g.versionid =
c.versionid
                                  and c.filesystemlabel = d.filesystemlabel"
    def get_granule_filename(granuleid)

      rs = do_query(ABSOLUTE_FILEPATHNAME_SQL,granuleid)
      if rs.count > 0
        rs[0]['filepathname']
      else
        nil
      end
    end

    def granule_published?(granuleid)
         published_granule = find_granule_by_granuleid(granuleid)
         if published_granule.nil?
            raise Sdps::GranuleNotFoundError.new('The granuleid does not exist in the amgranule table')
         elsif published_granule.isorderonly != 'H' and (published_granule.publishtime != nil and
published_granule.publishtime.to_s().length > 0)
           true
```

```
     else
       false
     end
end

def granule_deleted?(granuleid)
     granule = find_granule_by_granuleid(granuleid)
     if granule.nil?
        raise Sdps::GranuleNotFoundError.new('The granuleid does not exist in the amgranule table')
     elsif granule.deletefromarchive == 'Y' or granule.deleteeffectivedate != nil
       true
     else
       false
     end
end


#amgranule updates

HIDE_GRANULE_SQL = "update amgranule set deletefromarchive = 'H' where granuleid = $1 returning granuleid"
def hide_granule(granuleid)
  rs = do_query(HIDE_GRANULE_SQL,granuleid)
  if rs.count > 0
    true
  else
    false
  end
end

UNHIDE_GRANULE_SQL = "update amgranule set deletefromarchive = 'N' where granuleid = $1 returning granuleid"
def unhide_granule(granuleid)
  rs = do_query(UNHIDE_GRANULE_SQL, granuleid)
  if rs.count > 0
    true
  else
    false
```

```
      end
    end

  end
end
```

Current Test Script: ambrowse.rb

```
# Represents a row from ambrowse

require 'sdps/dbrecord'

module Sdps
  class AimDb
    class AmBrowse < DbRecord

      db_attr :browseid, "bigint"
      db_attr :externalid, "character varying(80)"
      db_attr :misbrid, "bigint"
      db_attr :granulestate, "character(2)"
      db_attr :registrationtime, "timestamp without time zone"
      db_attr :archivetime, "timestamp without time zone"
      db_attr :browsesize, "double precision"
      db_attr :lastupdate, "timestamp without time zone"
      db_attr :deleteeffectivedate, "timestamp without time zone"
      db_attr :archiveneededflag, "character(1)"
      db_attr :publishtime, "timestamp without time zone"
      db_attr :isorderonly, "character(1)"
      db_attr :versionnumber, "integer"
      db_attr :browsedescription, "character varying(255)"
      db_attr :browseproductiondatetime, "timestamp without time zone"

    end
  end
end
```

Current Test Script: dsmdgranulerestriction.rb

```
module Sdps

    module DsMdGranuleRestriction

       IS_GRANULE_RESTRICTED_SQL = "select restrictionflag from dsmdgranulerestriction where granuleid = $1"
       GET_GRANULE_RESTRICTED_SQL = "select restrictionflag from dsmdgranulerestriction where granuleid = $1"
       RESTRICT_GRANULE_SQL = "insert into dsmdgranulerestriction values ($1, $2) returning granuleid"
       UNRESTRICT_GRANULE_SQL = "delete from dsmdgranulerestriction where granuleid = $1"


       def granule_restricted?(granuleid)
          rs = do_query(IS_GRANULE_RESTRICTED_SQL, granuleid)
          if rs.count > 0
             true
          else
             false
          end
       end

       def get_granule_restriction(granuleid)
           rs = do_query(GET_GRANULE_RESTRICTED_SQL, granuleid)
           if rs.count > 0
               rs[0]['restrictionflag']
           else
               nil
           end
       end

       def restrict_granule(granuleid, restrictionflag)
           rs = do_query(RESTRICT_GRANULE_SQL, granuleid, restrictionflag)
           if rs.count > 0
              true
           else
              false
           end
       end
```

```
       def unrestrict_granule(granuleid)
          if granule_restricted?(granuleid) == true
             rs = do_query(UNRESTRICT_GRANULE_SQL, granuleid)
         end
         true
       end

    end
end
```

Current Test Script: dlfilesystems.rb

```
module Sdps
     module DlFileSystems

            # can be updated to get a whole row
           GET_ACTIVE_FILESYSTEMS_SQL = "select filesystemlabel from dlfilesystems where ingeststatus = 'ACTIVE'
 and filesystemlabel not like '%HDF%'"
           GET_ABSOLUTEFILESYSTEMPATH_SQL = "select absolutefilesystempath from dlfilesystems where
filesystemlabel = $1"
           GET_FILESYSTEMPATH_SQL = "select filesystempath from dlfilesystems where filesystemlabel = $1"
           def get_active_filesystems ()
                 rs = do_query(GET_ACTIVE_FILESYSTEMS_SQL)
                 if rs.count > 0
                       Array filesystems = Array.new()
                       rs.each_row  {|row|
                            filesystems.push(row[0])
                       }
                       filesystems
                 else
                        nil
                 end
           end

           def get_absolutefilesystempath(filesystemlabel)
                  rs = do_query(GET_ABSOLUTEFILESYSTEMPATH_SQL, filesystemlabel)
```

```
                if rs.count > 0
                          absolutefilesystempath = rs[0]['absolutefilesystempath']
                          absolutefilesystempath
                else
                      nil
                end
          end

          def get_filesystempath(filesystemlabel)
                rs = do_query(GET_FILESYSTEMPATH_SQL, filesystemlabel)
                if rs.count > 0
                          filesystempath = rs[0]['filesystempath']
                          filesystempath
                else
                      nil
                end
          end
     end
end
```

Current Test Script: amcollection.rb

```
# Represents a row in AmCollection.

require 'sdps/dbrecord'
require 'sdps/aimdbextras'

module Sdps
  class AimDb
    class AmCollection < Sdps::DbRecord
      include Sdps::AimDbExtras

      db_attr :collectionid, "bigint"
      db_attr :shortname, "character varying(8)"
      db_attr :longname, "character varying(80)"
      db_attr :versionid, "smallint"
      db_attr :collectiondescription, "character varying(255)"
      db_attr :revisiondate, "timestamp without time zone"
```

```
db_attr :type, "character varying(30)"
db_attr :subtype, "character varying(30)"
db_attr :groupid, "character(12)"
db_attr :filesystemlabel, "character varying(10)"
db_attr :spatialsearchtype, "character varying(40)"
db_attr :inserttime, "timestamp without time zone"
db_attr :lastupdate, "timestamp without time zone"
db_attr :lastcollectionmovetime, "timestamp without time zone"
db_attr :publicretentiontime, "integer"
db_attr :hiddenretentiontime, "integer"
db_attr :bulkexportedflag, "character(1)"
db_attr :convertenabledflag, "character(1)"
db_attr :cloudcoversourceid, "bigint"
db_attr :daynightflag, "character(1)"
db_attr :globalflag, "character(1)"
db_attr :exclusionscflag, "character(1)"
db_attr :publishbydefaultflag, "character(1)"
db_attr :allowpublishflag, "character(1)"
db_attr :moveflag, "character(1)"
db_attr :orderonlyfslabel, "character varying(10)"
db_attr :orderonlysndirname, "character varying(40)"
db_attr :replacementonflag, "character(1)"
db_attr :ecsflag, "character(1)"
db_attr :orderviewphflag, "character(1)"
db_attr :orderviewqaflag, "character(1)"
db_attr :orderbrowseflag, "character(1)"
db_attr :twentyfourhourflag, "character(1)"
db_attr :nominalcoveragerule, "character varying(40)"
db_attr :publicpriority, "smallint"
db_attr :gridlevel, "smallint"
db_attr :qualitysummary, "character varying(255)"
db_attr :themepsaname, "character varying(20)"
db_attr :processingcenter, "character varying(20)"
db_attr :archivecenter, "character varying(20)"
db_attr :processinglevelid, "character(6)"
db_attr :sciencemimetypeid, "integer"
```

```
      db_attr :browsemimetypeid, "integer"
      db_attr :browseonlinemimetypeid, "integer"
      db_attr :metadatamimetypeid, "integer"
      db_attr :ingestcreatemapflag, "character(1)"
      db_attr :orderviewhdf4mapflag, "character(1)"
      db_attr :filedataformatid, "integer"
      db_attr :duplicategranruleno, "smallint"
      db_attr :datamodeltype, "character varying(12)"

    end
  end
end
```

Current Test Script: publishing.rb

```
  module Sdps
    class Publishing
      attr_reader :mode

      def initialize(mode)
         @mode = mode
      end

      def publish_granule(granuleid)
         begin
           @dplhost = Conf.get_property('publish','hostname')
           puts "in publish granule" + granuleid.to_s()
           puts "dplhost= " + @dplhost
           puts "mode= " + @mode
           status = system("ssh -q #{@dplhost} '/usr/ecs/#{@mode}/CUSTOM/utilities/EcDlPublishUtilityStart
#{@mode} -ecs -g #{granuleid}'")
           puts status
           true
         rescue =>e
           puts e.backtrace
           false
         end
      end
```

```
      def publish_granules(granulefile)
         begin
           @dplhost = Conf.get_property('publish','hostname')
           puts "in publish granules in" + granulefile.to_s()
           puts "dplhost= " + @dplhost
           puts "mode= " + @mode
           status = system("ssh -q #{@dplhost} '/usr/ecs/#{@mode}/CUSTOM/utilities/EcDlPublishUtilityStart
#{@mode} -ecs -f #{granulefile}'")
           status
         rescue =>e
           puts e.backtrace
           false
         end
      end

      def get_gran_publishing_status (conn, request_time, granuleid)
         begin
           rs = conn.exec("select status from DlInsertActionQueue where ecsid = #{granuleid} and enqueuetime >
'#{request_time}'")
           if rs != nil and rs.count > 0
              status = rs[0]['status']
              status
           else
             nil
           end
           rescue =>e
             puts e.backtrace
             false
         end
      end

      def gran_publishing_successful (conn, request_time, granuleid)
         #poll and wait
         status = Sdps::Util.poll(180, 15){
            result = get_gran_publishing_status(conn, request_time, granuleid)
```

```
        if (result != 'FAILED' and result != 'COMPLETE' and result != 'CANCELED')
          nil
        else
          result
        end
      }
      if status != 'COMPLETE'
        false
      else
        true
      end
    end

    def granules_publishing_successful (conn, request_time, granulefile)
      rv = true
      granules = Array.new
      file = File.new(granulefile, "r")
      while (line = file.gets)
        line = line.delete(" ").delete("\n")
        if (line != "")
          granules.push(line)
        end
      end
      granules.each do | granuleid |
        if gran_publishing_successful(conn, request_time, granuleid) != true
          rv = false
        end
      end
      rv
    end

    def unpublish_granule(granuleid)
      begin
        @dplhost = Conf.get_property('publish','hostname')
        status = system("ssh -q #{@dplhost} 'cd /usr/ecs/#{@mode}/CUSTOM/utilities; ./EcDlUnpublishStart.pl
-mode #{@mode} -g #{granuleid}'")
```

```
            puts status
            true
         rescue =>e
           puts e.backtrace
           false
         end
      end

      def unpublish_granules(granulefile)
         begin
            @dplhost = Conf.get_property('publish','hostname')
            status = system("ssh -q #{@dplhost} 'cd /usr/ecs/#{@mode}/CUSTOM/utilities; ./EcDlUnpublishStart.pl
-mode #{@mode} -f #{granulefile}'")
            status
         rescue =>e
           puts e.backtrace
           false
         end
      end


       #TODO: check logs for success or errors
      def check_log()

      end
    end # class Publishing
  end # module Sdps
Current Test Script: ingest.rb

require 'fileutils'
require 'sdps/util'
require 'sdps/ingest/db'
require 'sdps/ingest/request'

module Sdps
  class IngestFailedError < RuntimeError; end
```

```
class Ingest

  class << self
    def build(pgconn)
      Ingest.new(Ingest::Db.new(pgconn))
    end
  end

  def initialize(ingestdb)
    @db = ingestdb
  end

  # Copies a PDR to a polling directory and returns a Request.
  def submit(pollingdir, pdrdir, pdrname)
    tag = Sdps::Util.token
    srcpathname = File.join(pdrdir, pdrname)
    destname = "#{tag}__#{pdrname}"
    destpathname = File.join(pollingdir, destname)
    FileUtils.cp(srcpathname, destpathname)
    Request.new(@db, destname)
  end

  # Submits a PDR and waits for the ingest request to complete.
  # Returns a list of granule IDs if successful.
  # Otherwise, raises an IngestFailedError.
  def submit_and_wait(pollingdir, pdrdir, pdrname)
    request = submit(pollingdir, pdrdir, pdrname)
    Util.poll do
      if request.successful?
        request.granule_ids
      elsif request.paused?
        # TODO abort after number of retries
        request.resume
      elsif request.stopped?
        raise IngestFailedError.new("Request stopped for PDR: #{request.filename}")
```

```
        end
      end
    end

  end
end
```

Current Test Script: movecollection.rb

```
  module Sdps
    class MoveCollection
      attr_reader :mode

      def initialize(mode)
         @mode = mode
      end

      def move_collection(collection, target_file_system)
         begin
           @dplhost = Conf.get_property('movecollectionutility','hostname')
           puts "in move collection " +  collection.subtype
           puts "dplhost= " + @dplhost
           puts "mode= " + @mode

           status = `ssh -q #{@dplhost} 'cd /usr/ecs/#{@mode}/CUSTOM/utilities;
/usr/ecs/#{@mode}/CUSTOM/utilities/EcDlMoveCollection.pl #{@mode} -shortname #{collection.shortname} -versionid
#{collection.versionid} -sourcefs #{collection.filesystemlabel} -targetfs #{target_file_system} -verbose'`
           puts status
           true
         rescue =>e
           puts e.backtrace
           false
         end
      end

      def get_files_in_filesystem(collection, absolutefilesystempath)
           @dplhost = Conf.get_property('movecollectionutility','hostname')
           puts "in get files collection " +  collection.subtype
```

```
            cmd = "ssh -q -n #{@dplhost}  'ls  -R
#{absolutefilesystempath}#{collection.groupid.strip}/#{collection.subtype}/*'"
            puts cmd
            listing = `ssh -q -n #{@dplhost}  'ls  -R
#{absolutefilesystempath}#{collection.groupid.strip}/#{collection.subtype}/*'`
            puts listing

            #separate the folders and files into a hash
            arr = listing.split("\n")
            puts arr
            dirlisting = arr
      end

      def compare_files(source_listing, dest_listing)

      end

       #TODO: check logs for success or errors
      def check_log()

      end

    end # class MoveCollection
  end # module Sdps
Current Test Script: bmgt.rb
require 'sdps/bmgt/bmgtdb'
require 'sdps/bmgt/exportrequest'
require 'sdps/bmgt/xml'

module Sdps
  module Bmgt

  #BMGT specific constants
  ECHO10_GRANULE_SCHEMA = '/tools/common/test/BE_82_01/echo10/schema/Granule.xsd'
  ECHO10_COLLECTION_SCHEMA = '/tools/common/test/BE_82_01/echo10/schema/Collection.xsd'
  DB_SUBSYSTEM = 'aim'
```

```
  end # module  Bmgt
end # module Sdps
```

Current Test Script: timestamp.rb

```
require 'time'

module Sdps
  class TcpProxyLogger < Logger
    module Timestamp

      def Timestamp.included klass
        klass.extend Timestamp
      end

      def timestamp(datetime = Time.now)
        datetime.strftime("%Y-%m-%dT%H:%M:%S.%6N")
      end

    end
  end
end
```

Current Test Script: logentry.rb

```
require 'pry'

require 'logger'
require 'time'

module Sdps
  class TcpProxyLogger < Logger
    class LogEntry

      class << self
        # Parse a single line written by the TcpProxyLogger.
        # Returns a new LogEntry.
        def parse(line)
```

```
    LogEntry.new(eval(line))
  end

  # Parse an entire log written by the TcpProxyLogger.
  # Returns a list of LogEntrys.
  #
  # TODO Combine successive request bodies when a request is
  # split.  E.g., One request has headers, including a large
  # "Content-Length", but no body.  The body is in a separate
  # request.
  def parse_file(log_filename)
    open(log_filename) do |log|
      log.map { |line| parse(line) }
    end
  end

  def merge_requests(log_entries)
    # for each request with a large Content-Length and no body
    #   find the next request with the same uuid
    #   append the request body to the parent request
    #   remove the headless request from the list
    body_missing = log_entries.select{|x| x.put? and not x.has_body?}
    head_missing = log_entries.select{|x| x.request? and not x.has_head?}
    body_missing.each do |entry|
      body = head_missing.select { |x|
        x.uuid == entry.uuid and x.timestamp > entry.timestamp
      }[0]
      if body
        entry.instance_variable_set(:@request,  entry.request + body.request)
        body.instance_variable_set(:@request, :merged)
        entry.instance_variable_set(:@merged,  true)
      end
    end
    log_entries.reject{|x| x.request? and x.request == :merged}
  end
end
```

```
attr_reader :timestamp, :severity, :uuid, :request, :response, :msg, :merged

def initialize(hash)
  # timestamp = severity = uuid = request = response = msg = nil
  hash.each do |key, value|
    instance_variable_set("@#{key}", value)
  end
end

def request?
  request and true
end

def response?
  response and true
end

def put?
  request and request.start_with?('PUT ')
end

def post?
  request and request.start_with?('POST ')
end

def delete?
  request and request.start_with?('DELETE ')
end

def get?
  request and request.start_with?('GET ')
end

def has_head?
  return response.start_with?('HTTP/') if response?
```

```
        first_word = request.split(' ', 2)[0]
        request and %w(PUT POST DELETE GET).include?(first_word)
      end

      def has_body?
        head, body = request.split("\r\n\r\n", 2)
        headers = head.split("\r\n")
        content_length = headers.select{|x| x.start_with?('Content-Length:')}[0]
        if content_length
          (content_length.split(': ')[1].to_i > 1) and (body.length > 1)
        else
          true
        end
      end

      def time
        Time.strptime(timestamp, "%Y-%m-%dT%H:%M:%S.%N")
      end

      def to_s
        result = ["#<LogEntry timestamp=#{timestamp.inspect}"]
        result << "uuid=#{uuid.inspect}"
        result << "severity=#{severity.inspect}"
        if request?
          result << "request=#{request.inspect}"
        elsif response?
          result << "response=#{response.inspect}"
        end
        result.join(" ") + ">"
      end

    end
  end
end
Current Test Script: sotestdatagranule.rb
```

```
module Sdps
  class SoTestData
    class SoTestDataGranule < Sdps::SoTestData::SoTestDataItem
      attr_reader :localgranuleid, :scgrfilenames

      def initialize(pathname,shortname,versionid,longname)
        super(pathname,shortname,versionid,longname)
        #SoTestDataItem can be a granule Item, or a collection item.
        #in case of granule item, pathname should be a PDR name with absolute path
        #because each pdr directory can contain multiple PDR files.

        @localgranuleid = nil
        @scgrfilenames = Array.new()

        if pathname =~ /PDR$/
          get_lgid_grfiles()
        else
          puts "Pathname doesn't contain PDR file, so no localgranuleid and granulefiles can be extracted"
        end
      end

      def get_lgid_grfiles ()
        #given the PDR name with absolute path, we could get either localGranuleId and a single science granule
file
        #or a nil localgranuleid and a list of scgrfilenames.

        @objvalpairs = Hash.new()
        begin
          file = File.new(@pathname, "r")

          while (line = file.gets)
            #read one block of OBJECT = FILE_SPEC/END_OBJECT = FILE_SPEC at a time

            if line =~ /FILE_SPEC/
              while ( line = file.gets and line !~ /FILE_SPEC/)
                @objvalpairs[line.split("=")[0].delete(" ").delete("\t")]=line.split("=")[1].delete("
```

```
").delete(";").delete("\n")
              end

              #if FILE_TYPE in ("HDF", "HDR-EDOS" and "SCIENCE"), put FILE_ID in @scgrfilenames
              #if FILE_TYPE is METADATA, put DIRECTORY_ID/FILE_ID in @metadatafile
              if @objvalpairs["FILE_TYPE"] =~ /SCIENCE|HDF|HDF-EDOS/
                @scgrfilenames.push(@objvalpairs["FILE_ID"])
                #puts "+++++++++++++SC file is: #{@objvalpairs["FILE_ID"]}"
              elsif @objvalpairs["FILE_TYPE"] =~ /METADATA/
                @metadatafile = @objvalpairs["DIRECTORY_ID"] + "/" + @objvalpairs["FILE_ID"]
                #puts "+++++++++++++metadatafile is: #{@metadatafile}"
              end
            end
          end
          file.close

          #if single file case, find the localgranuleid from metadata file.
          if @scgrfilenames.length == 1
            file = File.new(@metadatafile, "r")
            while (line = file.gets and @localgranuleid == nil)
                if line =~ /LOCALGRANULEID/
                  while (line = file.gets and line !~ /LOCALGRANULEID/)
                    if line =~ /VALUE/
                      @localgranuleid = line.split("=")[1].delete(" ").delete(";").delete("\"").delete("\n")
                      #puts "++++++++++++++localgranuleid is #{@localgranuleid}"
                    end
                  end
                end
            end
            file.close
          end
        rescue => err
          puts "Exception: #{err}"
          err
        end
      end
```

```
    end
  end
end
```

Current Test Script: sotestdataisocollection.rb

```
module Sdps
  class SoTestData
    class SoTestDataIsoCollection < Sdps::SoTestData::SoTestDataItem
      attr_reader :iso_collection_descriptor_file, :iso_collection_xml_file, :iso_collection_xpath_file,
:iso_granule_xpath_file

      def initialize(pathname,shortname,versionid,longname)
        super(pathname,shortname,versionid,longname)
        get_iso_collection_data()
      end

      def get_iso_collection_data()
        #get '0' padded version
        paddedversionid = @versionid.to_s().rjust(3,'0')

        base = @pathname + "/DsESDTSm" + @shortname + "."
        @iso_collection_descriptor_file = base + paddedversionid + ".desc"
        @iso_collection_xml_file = base + paddedversionid + ".series.xml"
        @iso_collection_xpath_file = base + paddedversionid + ".series.xpath"
        @iso_granule_xpath_file = base + paddedversionid + ".dataset.xpath"
      end

    end
  end
end
```

Current Test Script: sotestdataitem.rb

```
module Sdps
  class SoTestData
    class SoTestDataItem
```

```
      attr_reader :pathname, :shortname, :versionid, :longname

      def initialize(pathname,shortname,versionid,longname)
        @pathname = pathname
        @shortname = shortname
        @versionid = versionid
        @longname = longname
      end

      def esdt
          "%s.%03d" % [shortname, versionid]
      end

      def datasetid
      "%s V%03d" % [longname, versionid]
    end
    end
  end
end
```

Current Test Script: aimdb.rb

```
require 'sdps/aimdb/amcollection'
require 'sdps/aimdb/amgranule'
require 'sdps/aimdb/ambrowse'
require 'sdps/aimdb/dsmdgranulerestriction'
require 'sdps/aimdb/dsmdrestrictionflag'
require 'sdps/aimdb/dlfilesystems'
require 'sdps/aimdb/collection'
require 'sdps/aimdb/granule'

module Sdps
  class AimDb
       include Sdps::DsMdGranuleRestriction
       include Sdps::DsMdRestrictionFlag
       include Sdps::DlFileSystems
       include Sdps::Collection
```

```
   include Sdps::Granule

def initialize(dbconn)
  @dbconn = dbconn
end

#granule queries
NOTDELETED_GRANULES_RANDOM_BY_ESDT_SQL = "
  select *
  from amgranule
  where shortname = $1
  and versionid = $2
  and deleteeffectivedate is null
  and deletefromarchive = 'N'
  order by random()
  LIMIT 10
"
def find_notdeleted_granules_by_esdt(esdt)
    shortname = esdt.split(".")[0]
    versionid = esdt.split(".")[1].to_i
    query_result = find_granules(NOTDELETED_GRANULES_RANDOM_BY_ESDT_SQL, shortname, versionid)
  #query_result.values.flatten.map(&:to_i)
end


NOTDELETED_GRANULEIDS_SQL = "
  select granuleid
  from amgranule
  where collectionid = $1
  and deleteeffectivedate is null
  and deletefromarchive = 'N'
"
def find_notdeleted_granuleids_by_collectionid(collectionid)
  query_result = @dbconn.exec_params(NOTDELETED_GRANULEIDS_SQL, [collectionid])
  query_result.values.flatten.map(&:to_i)
end
```

```
#granule queries
DELETED_GRANULEIDS_SQL = "
  select granuleid
  from amgranule
  where collectionid = $1
  and (deleteeffectivedate is  not null
  or deletefromarchive = 'Y')
"
def find_deleted_granuleids_by_collectionid(collectionid)
  query_result = @dbconn.exec_params(DELETED_GRANULEIDS_SQL, [collectionid])
  query_result.values.flatten.map(&:to_i)
end

#find granule queries
GRANULE_BY_GRANULEID_SQL = "
  select *
  from amgranule
  where granuleid = $1
"
def find_granule_by_granuleid(granuleid)
  find_granules(GRANULE_BY_GRANULEID_SQL, granuleid)[0]
end

GRANULES_BY_SHORTNAME_VERSIONID_SQL = "
  select *
  from amgranule
  where shortname = $1
  and versionid = $2
  LIMIT 10
"
def find_granules_by_shortname_versionid(shortname, versionid)
  find_granules(GRANULES_BY_SHORTNAME_VERSIONID_SQL, shortname, versionid)
end

def find_granules_by_esdt(esdt)
```

```
    shortname = esdt.split(".")[0]
    versionid = esdt.split(".")[1]
   find_granules_by_shortname_versionid( shortname, versionid)
end



GRANULES_BY_LOCALGRANULEID_SQL = "
  select *
  from amgranule
  where localgranuleid = $1
"
def find_granules_by_localgranuleid(localgranuleid)
  find_granules(GRANULES_BY_LOCALGRANULEID_SQL, localgranuleid)
end

GRANULES_BY_GRFILENAME_SQL = "
  select *
  from amgranule gr
  join AmDataFile df
  on gr.granuleid = df.granuleid
  and df.userdatafile = $1
"
def find_granules_by_grfilename(grfilename)
  find_granules(GRANULES_BY_GRFILENAME_SQL, grfilename)
end

BROWSE_BY_BROWSEID_SQL = "
 select *
  from ambrowse
  where browseid = $1
 "

def find_browse_by_browseid(browseid)
  find_many_items(AmBrowse, BROWSE_BY_BROWSEID_SQL, browseid)[0]
end
```

```
BROWSE_BY_SCIENCEID_SQL = "
  select *
  from ambrowse
  where browseid in (
    select browseid
    from ambrowsegranulexref
    where granuleid = $1
  )
"
def find_associated_browse_by_granuleid(scienceid)
  find_many_items(AmBrowse, BROWSE_BY_SCIENCEID_SQL, scienceid)
end

ASSOCIATED_PH_SQL = "
  select *
  from amgranule
  where granuleid in (
    select phid
    from amphgranulexref
    where scienceid = $1
  )
"
def find_associated_ph(scienceid)
  find_granules(ASSOCIATED_PH_SQL, scienceid)
end

ASSOCIATED_QA_SQL = "
  select *
  from amgranule
  where granuleid in (
    select qaid
    from amqagranulexref
    where scienceid = $1
  )
"
```

```
def find_associated_qa(scienceid)
  find_granules(ASSOCIATED_QA_SQL, scienceid)
end

ASSOCIATED_HDFMAP_SQL = "
  select *
  from amgranule
  where granuleid in (
    select hdfmapid
    from amhdfmapgranulexref
    where scienceid = $1
  )
"
def find_associated_hdfmap(scienceid)
  find_granules(ASSOCIATED_HDFMAP_SQL, scienceid)
end

def find_collections(sql, *params)
  find_many_items(AmCollection, sql, *params)
end

def find_granules(sql, *params)
  find_many_items(AmGranule, sql, *params)
end

def find_many_items(item_class, sql, *params)
  @dbconn.exec_params(sql, params).map{|tuple| item_class.new(tuple)}
end

def do_query(sql, *params)
  @dbconn.exec_params(sql, params)
end



#ambrowse updates
```

```
UNLINK_BROWSE_SQL = "delete from AmBrowseGranuleXref where granuleid = $1"
def unlink_browse (granuleid)
  rs = do_query(UNLINK_BROWSE_SQL, granuleid)
  # TODO: should delete the browse from ambrowse table!
  true
end

BROWSE_SQL = "select g.granuleid
              from amgranule g
              join ambrowsegranulexref bx
              on g.granuleid = bx.granuleid
              where g.collectionid = $1
              limit $2;"
def get_granules_with_browse(collectionid, limit)

  rs = do_query(BROWSE_SQL, collectionid, limit)
  if rs != nil and rs.count > 0
    granuleids = Array.new
    rs.each do |row|
      granuleids.push(row['granuleid'])
    end
    return granuleids
  else
    return nil
  end
end

PUBLIC_GRANULES = "select granuleid
              from amgranule
              where IsOrderOnly is null and
              collectionid = $1;"
def get_public_granules(collectionid)
  rs = do_query(PUBLIC_GRANULES, collectionid)

  if rs != nil and rs.count > 0
    granuleids = Array.new
```

```
        rs.each do |row|
          granuleids.push(row['granuleid'])
        end
        return granuleids
      else
        return nil
      end
    end

    ADD_GRANULE_RESTRICTION = "insert into dsmdgranulerestriction values ($1, 255)"
    GRANULE_RESTRICTION_Exists = "select * from dsmdgranulerestriction where granuleid = $1"
    Update_GRANULE_RESTRICTION = "update dsmdgranulerestriction set restrictionflag = 255 where granuleid = $1"


    def add_granule_restriction (granuleid)
      rs = do_query(GRANULE_RESTRICTION_Exists, granuleid)
      if(rs != nil and rs.count > 0)
        rs = do_query(Update_GRANULE_RESTRICTION, granuleid)
      else
        rs = do_query(ADD_GRANULE_RESTRICTION, granuleid)
      end
      true
    end

  end
end
Current Test Script: time.rb
class Time
  def to_s
    strftime("%Y-%m-%d %H:%M:%S.%6N").sub(/\.?0+$/, '')
  end
end
Current Test Script: conf.rb
#!/usr/bin/env ruby

require 'yaml'
```

```
require 'ostruct'

module Conf

  class  << self

   @@properties = nil

    def load(file)

      if(@@properties.nil?)
        config = YAML.load_file(file)
        @@properties = parse(config)
      end #END if @@properties is nil.
    rescue =>e
      puts e.backtrace

      @@properties

    end

    def parse(hash=nil)
      table = Hash.new

      if hash
        hash.each do |k,v|
          table[k.to_sym] = (v.is_a?(Hash) ? get_OStruct(v) : v)
        end
      end
      return OpenStruct.new(table)

    end

    def get_OStruct(hash = nil)
      if hash
        table = {}
```

```
        hash.each do |k,v|
          table[k.to_sym] = (v.is_a?(Hash) ? get_OStruct(v) : v)
        end
        return OpenStruct.new(table)
      end
    end

    def get_property(subsys, name)
      value =  @@properties.marshal_dump[subsys.to_sym].marshal_dump[name.to_sym]
    end

  end

end
```

Current Test Script: om.rb

```
require 'sdps/om/omdb'
require 'sdps/om/omrequest'
require 'sdps/om/omgranule'
require 'sdps/om/omserver'
module Sdps
  module Om

    OM_SERVER_HOST = 'f5oml01v'
    OM_CONFIG_FILE = 'EcOmOrderManager.CFG'
    DB_SUBSYSTEM = 'oms'

  end
end
```

Current Test Script: emsutils.rb

```
require 'sdps/db'

module Sdps
  module EMS

    DB_SUBSYSTEM = 'aim'
```

```
class << self

  # EcDbEMSdataExtractor.pl -m <MODE> -s "<start date>" -e "<end date>" -v -i -x <DATASET> -o
  def EMS_data_extractor( mode, startDate, endDate, verbose, initial, dataSet, override )

    # build path using provided mode
    ems_data_extractor = "/usr/ecs/#{mode}/CUSTOM/utilities/EcDbEMSdataExtractor.pl"

    if !File.file?( ems_data_extractor )
      fail( ArgumentError.new( "File does not exist (#{ems_data_extractor})" ) )
    end

    # go through each of the parameters and determine
    s = startDate ? "-s \"#{startDate}\"" : ""
    e = endDate   ? "-e \"#{endDate}\"" : ""
    v = verbose   ? "-v" : ""
    i = initial   ? "-i" : ""
    d = dataSet   ? "-x \"#{dataSet}\"" : ""
    o = override  ? "-o" : ""

    # construct the command
    command = "#{ems_data_extractor} -m #{mode} #{s} #{e} #{v} #{i} #{d} #{o}"

    system( command ) #return results of the command
  end

  def EMS_find_record_in_log( logFile, mode, startDate, endDate, dataSet )
    if !File.file?( logFile )
      fail( ArgumentError.new( "Log file '#{logFile}' does not exist" ) )
    else
      file = IO.read( logFile ) # put strings making up logFile into "file"
    end

    file.foreach { |line|   # look for entry in log
      if line =~
```

```
/MODE:\s+#{mode}\s+STARTDATE:\s+#{startDate}\s+ENDDATE:\s+#{endDate}\s+EXTRACTTYPE:\s+#{dataSet}/
            true                        # return true if found
          end
        }
        fail( ArgumentError.new( "Record not found in log" ) )
      end


      ###
      def EMS_find_record_in_flatfile( flatFile, recordHash )

        # open flat file
        if !File.file?( flatFile )
          fail( ArgumentError.new( "File does not exist (#{flatFile})" ) )
        else
          file = IO.read( logFile ) # put strings making up logFile into "file"
        end

        # for every line, place line contents in a record
        file.foreach { |line|   # compare record from line to record hash
          if line =~ /(\w+)|&|(\w+)|&|(\w+)|&|(\w+)|&|(\w+)|&|(\w+)|&|(\w+)|&|(\w+)|&|(\w+)|&|(\w+)|&|(\w+)/

          end
        }
        fail( ArgumentError.new( "Record not found in file" ) )

      end
      ###



      def EMS_find_record_in_table( table, recordHash )
        recordCount = 0
        db_connection = EMS_database_connect() # create connection

        # construct query
```

```
  query = "SELECT * FROM #{table} WHERE "

  recordHash.each{ |key, value|

    if( value != nil )
      query << "#{key} = '#{value}'"
    end

    recordCount++

    if( recordCount < recordHash.length )
      query << " AND "
    else
      query << ";"
    end
  }

  # run query to search for record
  result = db_connection.exec( query )

  if( result == nil )
    fail( ArgumentError.new( "Record not found in table" ) )
  end
  result
end

def EMS_get_date_list( startDate, endDate )

  modifiedStartDate = nil
  modifiedEndDate   = nil
  count = 0
  days = []

  monthHash = { :Jan => "01",
                :Feb => "02",
                :Mar => "03",
```

```
              :Apr => "04",
              :May => "05",
              :Jun => "06",
              :Jul => "07",
              :Aug => "08",
              :Sep => "09",
              :Oct => "10",
              :Nov => "11",
              :Dec => "12" }

if( startDate =~ /(\w+)\s+(\w+)\s+(\w+)/ )
  dateArray = @startDate.match( /(\w+)\s+(\d+)\s+(\d+)/ )
  monthSymbol = dateArray[1].to_sym
  modifiedStartDate = "#{dateArray[3]}#{monthHash[monthSymbol]}#{dateArray[2]}"
else
  fail( ArgumentError.new( "Start Date format error" ) )
end

if( endDate =~ /(\w+)\s+(\w+)\s+(\w+)/ )
  dateArray = @endDate.match( /(\w+)\s+(\d+)\s+(\d+)/ )
  monthSymbol = dateArray[1].to_sym
  modifiedEndDate = "#{dateArray[3]}#{monthHash[monthSymbol]}#{dateArray[2]}"
else
  fail( ArgumentError.new( "End Date format error" ) )
end

currentDate = modifiedEndDate
timeIndex = Time.new( dateArray[3], monthHash[monthSymbol], dateArray[2] )

while( currentDate >= modifiedStartDate )
  days.push( currentDate )
  timeIndex = timeIndex - 86400 # subtract a day
  currentDate = "#{timeIndex.year}#{timeIndex.strftime("%m")}#{timeIndex.strftime("%d")}"
end

return days
```

```
      end

      def EMS_one_record_per_line( datasetFile )
        fail( ArgumentError.new( "More than one record found per line" ) )
      end

      def EMS_each_record_delimited( datasetFile )
        fail( ArgumentError.new( "Records are not delimited" ) )
      end

      def EMS_get_cfg( cfgFile, field )                 # look for <field> =<result> in cfgFile
        file = File.new( cfgFile )                      # put strings making up cfgFile into "file"
        line = file.gets
        result = nil
        while line != nil
          result = /#{field} = (\S+)/.match( line )     # look for match field line in "file"
          if result != nil
            return result[1]
          end
          line = file.gets
        end
        fail( ArgumentError.new( "Field not found in file" ) )
      end

      def EMS_database_connect( subsystem=DB_SUBSYSTEM )
        Db.connect( subsystem )
      end

    end

  end
end
Current Test Script: mode_admin_lib.rb
#!/usr/bin/env ruby
```

```
##################################################
# get list of configured modes
##################################################
def get_configured_modes
  File.open("/home/cmshared/LAB/LinuxStart/LINUXMODES", 'r') do |f1|
    while line = f1.gets
      mode1, baseline, host =line.split(",")
      if mode1 !~/^ *#/
        @modes.push(mode1)
      end
    end
  end
end

##################################################
# is_controlled_mode returns true if mode is not
# run from clearcase.
##################################################
def is_controlled_mode(mode)
  controlled_modes = %w(OPS TS2 TS3)
  controlled_modes.include?(mode.upcase)
end

##################################################
# get configured baseline view for a given mode
##################################################
def get_default_view(mode)
  view = "no_view_defined"
  if is_controlled_mode(mode)
    view = "noview"
  elsif
    File.open("/home/cmshared/LAB/LinuxStart/LINUXMODES", 'r') do |f1|
      while line = f1.gets
        mode1, baseline, host =line.split(",")
        if mode1 == mode
```

```
          if baseline.strip == 'RELB'
            view = "current_test"
          elsif
            view = baseline+"_linux"
          end
        end
      end
    end
  end
  if !@verbose.nil?
    puts "\nview for #{mode} is #{view}\n\n"
  end
  view
end

##########################################################
# Start Mode uses server start scripts to start the mode
# view is an optional parameter, if missing the view start
# with its default from the configuration files
##########################################################
def start_mode(mode, view = nil)
  if view.nil?
    view = get_default_view(mode)
  end
  File.open("/home/cmshared/LAB/LinuxStart/"+mode, 'r') do |f1|
    while line = f1.gets
      host, script_prefix, mode1, start_args, def_view = line.split(":")
      if host.include?('#')
        next
      end
      if !@verbose.nil?
        puts "Executing #{script_prefix}Start for #{mode} from #{view} on #{host}\n"
      end
      if script_prefix == 'BmgtProxyServer'
         buffer = `ssh -q  #{host} '/tools/common/test/BE_82_01/proxy/proxy_start #{mode}'`
      else
```

```
        if view =="noview"
          buffer = `ssh -q  #{host} '/usr/ecs/#{mode}/CUSTOM/utilities/#{script_prefix}Start #{mode}
#{start_args}'`
        elsif
          buffer = `ssh -q  #{host} '/usr/atria/bin/cleartool setview -exec
"/usr/ecs/#{mode}/CUSTOM/utilities/#{script_prefix}Start #{mode} #{start_args}" #{view}'`;
        end
      end
      if !@verbose.nil?
        puts "#{buffer}==========================================================================\n"
      end
    end
  end
end

##########################################################
# Stop Mode uses server stop scripts to shutdown the given mode
##########################################################
def stop_mode(mode)
  File.open("/home/cmshared/LAB/LinuxStart/"+mode, 'r') do |f1|
    while line = f1.gets
      host, script_prefix, mode1, start_args, def_view = line.split(":")
      if host.include?('#')
        next
      end
      if !@verbose.nil?
        puts  "Executing #{script_prefix}Stop for #{mode} on #{host}\n";
      end
      if script_prefix == "EcNbDriver"
        buffer = `ssh -q -n #{host} '/home/cmshared/bin/killsss.csh #{mode}'`
      elsif script_prefix == "BmgtProxyServer"
        buffer = `ssh -q -n #{host} '/tools/common/test/BE_82_01/proxy/proxy_stop #{mode}'`
      else
        buffer = `ssh -q -n #{host} '/usr/ecs/#{mode}/CUSTOM/utilities/#{script_prefix}Stop #{mode}'`
      end
      if !@verbose.nil?
```

```
        puts "#{buffer}-------------------------------------------------\n"
      end
    end
  end
end


#########################################################
# Start a subsystem for a given mode
# a limited number of subsystems are supported at this time:
# BMGT, INS, OMS, SSS
# the view parameter is optional
#########################################################
def start_subsys(subsys, mode, view = nil)
  if mode.nil?
    puts "Error: Missing Subsystem or Mode Parameter"
    exit
  elsif view.nil?
    view = get_default_view(mode)
  end
  case subsys
  when 'BMGT'
    component = 'EcBm'
  when 'INS'
    component = 'EcDlIn'
  when 'OMS'
    component = 'EcOm'
  when 'SSS'
    component = 'EcNb'
  else
    puts "Error: unknown subsystem parameter"
    exit
  end
  File.open("/home/cmshared/LAB/LinuxStart/"+mode, 'r') do |f1|
    while line = f1.gets
      host, script_prefix, mode1, start_args, def_view = line.split(":")
```

```
      if host.include?('#')
        next
      end
      if script_prefix.include?(component)
        if !@verbose.nil?
          puts "Executing #{script_prefix}Start for #{mode} from #{view} on #{host}\n"
        end
        if script_prefix == 'BmgtProxyServer'
           buffer = `ssh -q  #{host} '/tools/common/test/BE_82_01/proxy/proxy_start #{mode}'`
        else
          if view =="noview"
            buffer = `ssh -q  #{host} '/usr/ecs/#{mode}/CUSTOM/utilities/#{script_prefix}Start #{mode}
#{start_args}'`
          elsif
            buffer = `ssh -q  #{host} '/usr/atria/bin/cleartool setview -exec
"/usr/ecs/#{mode}/CUSTOM/utilities/#{script_prefix}Start #{mode} #{start_args}" #{view}'`;
          end
        end
        if !@verbose.nil?
          puts "#{buffer}==========================================================================\n"
        end
      end
    end
  end
end

##########################################################
# Stop subsystem for a given subsystem and mode
# a limited number of subsystems are supported at this time:
# BMGT, INS, OMS, SSS
#########################################################
def stop_subsys(subsys, mode)
  if mode.nil?
    puts "Error: Missing Subsystem or Mode Parameter"
    exit
  end
```

```
case subsys
when 'BMGT'
  component = 'EcBm'
when 'INS'
  component = 'EcDlIn'
when 'OMS'
  component = 'EcOm'
when 'SSS'
  component = 'EcNb'
else
  puts "Error: unknown subsystem parameter"
  exit
end
File.open("/home/cmshared/LAB/LinuxStart/"+mode, 'r') do |f1|
  while line = f1.gets
    host, script_prefix, mode1, start_args, def_view = line.split(":")
    if host.include?('#')
      next
    end
    if script_prefix.include?(component)
      if !@verbose.nil?
        puts  "Executing #{script_prefix}Stop for #{mode} on #{host}\n";
      end
      if script_prefix == "EcNbDriver"
        buffer = `ssh -q -n #{host} '/home/cmshared/bin/killsss.csh #{mode}'`
      elsif script_prefix == "BmgtProxyServer"
        buffer = `ssh -q -n #{host} '/tools/common/test/BE_82_01/proxy/proxy_stop #{mode}'`
      else
        buffer = `ssh -q -n #{host} '/usr/ecs/#{mode}/CUSTOM/utilities/#{script_prefix}Stop #{mode}'`
      end
      if !@verbose.nil?
        puts "#{buffer}--------------------------------------------------\n"
      end
    end
  end
end
```

```
end


##########################################################
# remote process query for mode_status
##########################################################
def remote_process_query (mode,host,script_prefix)
  status = false;
  buffer = `ssh -q -n #{host} 'ps auxwww  | egrep #{script_prefix} | grep #{mode} | grep -v grep'`
  if !buffer.empty?
    if !@verbose.nil?
      puts "#{script_prefix} for #{mode} on #{host} is [UP]"
    end
    status = true
  else
    if !@verbose.nil?
      puts "#{script_prefix} for #{mode} on #{host} is [DOWN]!"
    end
    status = false
  end
  status
end

##########################################################
# Mode Status returns the number of servers that are down
# for a given mode
##########################################################
def mode_status(mode)
  @offline_servers = []
  File.open("/home/cmshared/LAB/LinuxStart/"+mode, 'r') do |f1|
    while line = f1.gets
      host, script_prefix, mode1, start_args, def_view = line.split(":")
      host.tr('^A-Za-z0-9', '')
      case script_prefix
      when 'EcDlChksumLookupSrvr'
        if !remote_process_query(mode1, host, 'EcDlChecksumLookupServer')
```

```
    @offline_servers.push(script_prefix)
  end
when 'EcAmMGS'
  if !remote_process_query(mode1, host, 'EcDsAmMapGenerationServer')
    @offline_servers.push(script_prefix)
  end
when 'EcDlCVS'
  if !remote_process_query(mode1, host, 'EcDlChecksumServer')
    @offline_servers.push(script_prefix)
  end
when 'EcBmBMGTApp'
  if mode1=='OPS'
    mode1 = 'OPS/CUSTOM'
  end
  if !remote_process_query(mode1, host, 'EcBmDispatcher')
    @offline_servers.push('EcBmDispatcher')
  end
  if !remote_process_query(mode1, host, 'EcBmAuto')
    @offline_servers.push('EcBmAuto')
  end
  if !remote_process_query(mode1, host, 'EcBmMonitor')
    @offline_servers.push('EcBmMonitor')
  end
when 'EcNbDriver'
  if !remote_process_query(mode1, host, 'EcNbActionDriver')
    @offline_servers.push('EcNbActionDriver')
  end
  if !remote_process_query(mode1, host, 'EcNbSubscribedEventDriver')
    @offline_servers.push('EcNbSubscribedEventDriver')
  end
  if !remote_process_query(mode1, host, 'EcNbRecoverDriver')
    @offline_servers.push('EcNbRecoverDriver')
  end
  if !remote_process_query(mode1, host, 'EcNbDeleteRequestDriver')
    @offline_servers.push('EcNbDeleteRequestDriver' )
  end
```

```
      when 'BmgtProxyServer'
        if !remote_process_query(mode1, host, 'proxy')
          @offline_servers.push(script_prefix)
        end
      else
        if !remote_process_query(mode1, host, script_prefix)
          @offline_servers.push(script_prefix)
        end
      end
    end
  end
  @offline_servers
end

##########################################################
# Subsystem Status returns the number of servers that are down
# for a given mode
# a limited number of subsystems are supported at this time:
# BMGT, INS, OMS, SSS
##########################################################
def subsys_status(subsys, mode)
  if mode.nil?
    puts "Error: Missing Subsystem or Mode Parameter"
    exit
  end
  case subsys
  when 'BMGT'
    component = 'EcBm'
  when 'INS'
    component = 'EcDlIn'
  when 'OMS'
    component = 'EcOm'
  when 'SSS'
    component = 'EcNb'
  else
    puts "Error: unknown subsystem parameter"
```

```
  exit
end
@offline_servers = []
File.open("/home/cmshared/LAB/LinuxStart/"+mode, 'r') do |f1|
  while line = f1.gets
    host, script_prefix, mode1, start_args, def_view = line.split(":")
    host.tr('^A-Za-z0-9', '')
    if script_prefix.include?(component)
      case script_prefix
      when 'EcDlChksumLookupSrvr'
        if !remote_process_query(mode1, host, 'EcDlChecksumLookupServer')
          @offline_servers.push(script_prefix)
        end
      when 'EcAmMGS'
        if !remote_process_query(mode1, host, 'EcDsAmMapGenerationServer')
          @offline_servers.push(script_prefix)
        end
      when 'EcDlCVS'
        if !remote_process_query(mode1, host, 'EcDlChecksumServer')
          @offline_servers.push(script_prefix)
        end
      when 'EcBmBMGTApp'
        if mode1=='OPS'
          mode1 = 'OPS/CUSTOM'
        end
        if !remote_process_query(mode1, host, 'EcBmDispatcher')
          @offline_servers.push('EcBmDispatcher')
        end
        if !remote_process_query(mode1, host, 'EcBmAuto')
          @offline_servers.push('EcBmAuto')
        end
        if !remote_process_query(mode1, host, 'EcBmMonitor')
          @offline_servers.push('EcBmMonitor')
        end
      when 'EcNbDriver'
        if !remote_process_query(mode1, host, 'EcNbActionDriver')
```

```
            @offline_servers.push('EcNbActionDriver')
          end
          if !remote_process_query(mode1, host, 'EcNbSubscribedEventDriver')
            @offline_servers.push('EcNbSubscribedEventDriver')
          end
          if !remote_process_query(mode1, host, 'EcNbRecoverDriver')
            @offline_servers.push('EcNbRecoverDriver')
          end
          if !remote_process_query(mode1, host, 'EcNbDeleteRequestDriver')
            @offline_servers.push('EcNbDeleteRequestDriver' )
          end
        when 'BmgtProxyServer'
          if !remote_process_query(mode1, host, 'proxy')
            @offline_servers.push(script_prefix)
          end
        else
          if !remote_process_query(mode1, host, script_prefix)
            @offline_servers.push(script_prefix)
          end
        end
      end
    end
  end
  @offline_servers
end

Current Test Script: request.rb

# ecs=> select * from invalrequeststate ;
#  requeststatekey |    requeststate
# -----------------+---------------------
#                1 | New
#                2 | Active
#                3 | Partially_Suspended
#                4 | Suspended
#                5 | Cancelling
```

```
#                6 | Resuming
#                7 | Successful
#                8 | Cancelled
#                9 | Failed
#               10 | Partial_Failure
#               11 | Terminated
#               12 | Cleaned_PDRFile
#               14 | Partially_Cancelled
#               15 | Validated
#               19 | Suspending
# (15 rows)

module Sdps
  class Ingest
    class Request

      attr_reader :filename

      def initialize(ingestdb, filename)
        @db = ingestdb
        @filename = filename
        @requestid = nil
      end

      def requestid
        @requestid ||= @db.requestid(@filename)
      end

      # Returns the ingest request's status.
      def status
        @db.request_status(requestid) if requestid
      end

      # Returns a, possibly empty, list of granule IDs, if the request
      # was successful.
      def granuleids
```

```
  @db.granuleids(requestid) if requestid
end

# Returns a, possibly empty, list of browse IDs, if the request
# was successful.
def browseids
  @db.browseids(requestid) if requestid
end

# True if the request is successful.
def successful?
  :Successful == status
end

# True if the request partially or completely suspended.
def suspended?
  [:Partially_Suspended, :Suspended, :Suspending].include?(status)
end

def stopped?
  [:Successful, :Cancelled, :Failed, :Partial_Failure,
   :Terminated, :Partially_Cancelled].include?(status)
end

# Restarts a paused request.
def restart
  @db.restart_request(requestid) if requestid
end

# Resumes a paused request.
def resume
  @db.resume_request(requestid) if requestid
end

# Cancels a paused request.
def cancel
```

```
        @db.cancel_request(requestid) if requestid
      end

    end
  end
end
```

Current Test Script: db.rb

```
require 'sdps/errors'

module Sdps
  # This is an adapter that insulates the Ingest class from the
  # database.
  class Ingest
    class Db

      def initialize(dbconn)
        @dbconn = dbconn
      end

      REQUESTID_SQL = "select requestid
        from indplingestpdr
        where pdrfilename like '%/' || $1::varchar"

      HISTORIC_REQUESTID_SQL = "select requestid
        from inhistoricrequest
        where danfilename like '%/' || $1::varchar"

      # Returns a, possibly empty, list of requestids for the PDR
      # filename.
      def requestid(pdrfilename)
        r1 = @dbconn.exec_params(REQUESTID_SQL, [pdrfilename])
        r2 = @dbconn.exec_params(HISTORIC_REQUESTID_SQL, [pdrfilename])
        result = (r1.values + r2.values).flatten.map(&:to_i)[0]
        r1.clear
        r2.clear
        result
```

```
end

REQUEST_STATUS_SQL = "select requeststatus
  from indplingestpdr
  where requestid = $1::int"

HISTORIC_REQUEST_STATUS_SQL = "select v.requeststate
  from inhistoricrequest h
  join invalrequeststate v
  on h.requeststatekey = v.requeststatekey
  where h.requestid = $1::int"

def request_status(requestid)
  dbresult = @dbconn.exec_params(REQUEST_STATUS_SQL, [requestid])
  if dbresult.values.empty?
    dbresult.clear
    dbresult = @dbconn.exec_params(HISTORIC_REQUEST_STATUS_SQL, [requestid])
  end
  result = dbresult.values[0] && dbresult.values[0][0].to_sym
  dbresult.clear
  result
end

GRANULEIDS_SQL = "
  select ecsgranuleid
  from indplingestgranule
  where requestid = $1
  and datatype != 'Browse'"

HISTORIC_GRANULEIDS_SQL = "
  select ecsgranuleid
  from inhistoricgranule
  where requestid = $1
  and datatype != 'Browse'"

def granuleids(requestid)
```

```
  r1 = @dbconn.exec_params(GRANULEIDS_SQL, [requestid])
  r2 = @dbconn.exec_params(HISTORIC_GRANULEIDS_SQL, [requestid])
  result = (r1.values + r2.values).flatten.map(&:to_i)
  r1.clear
  r2.clear
  result
end

BROWSEIDS_SQL = "
  select ecsgranuleid
  from indplingestgranule
  where requestid = $1
  and datatype = 'Browse'"

HISTORIC_BROWSEIDS_SQL = "
  select ecsgranuleid
  from inhistoricgranule
  where requestid = $1
  and datatype = 'Browse'"

def browseids(requestid)
  r1 = @dbconn.exec_params(BROWSEIDS_SQL, [requestid])
  r2 = @dbconn.exec_params(HISTORIC_BROWSEIDS_SQL, [requestid])
  result = (r1.values + r2.values).flatten.map(&:to_i)
  r1.clear
  r2.clear
  result
end

def restart_request(token)
  # TODO call pg function to restart the ingest request
  # What does the GUI do?
  raise NotImplementedYet.new("#{self.class}##{__method__.to_s} is not yet implemented")
end

def resume_request(token)
```

```
        # TODO call pg function to resume the ingest request
        # What does the GUI do?
        raise NotImplementedYet.new("#{self.class}##{__method__.to_s} is not yet implemented")
      end

      def cancel_request(token)
        # TODO call pg function to cancel the ingest request
        # What does the GUI do?
        raise NotImplementedYet.new("#{self.class}##{__method__.to_s} is not yet implemented")
      end

    end
  end
end
```

Current Test Script: sotestdata.rb

```
require 'sdps/util'
require 'sdps/sotestdata/sotestdataitem'
require 'sdps/sotestdata/sotestdatagranule'
require 'sdps/sotestdata/sotestdataisocollection'

module Sdps
  class SoTestData
    attr_reader :data

    def initialize (conn)
      @conn = conn
      @data = Array.new
      @testgroup = Conf.get_property('global','testgroup')
    end

    #def initialize (conn, localgranuleid)
    #  @conn = conn
    #  @data = Array.new
    #  granules = @aimdb.find_granules_by_localgranuleid(localgranuleid)
      #granules.each do
```

```
      #   @collection = @aimdb.find_collection_by_shortname_versionid(shortname, versionid)
      #end
    #end

    def [] (criteriainfo)
      begin
        @criteriainfo = criteriainfo
        rs = @conn.exec("select pathname, shortname, versionid, longname from AutoTestSoTestDataInfo  where
criteriainfo like '#{@criteriainfo}' and testgroup = '#{@testgroup}' order by criteriainfo")

        if rs != nil and rs.count > 0
          rs.each do |row|
            pathname = row['pathname']
            shortname = row['shortname']
            versionid = row['versionid']
            longname = row['longname']
            #puts "def[] shortname, versionid: #{shortname}, #{versionid}"
            sotestdataitem = SoTestDataItem.new(pathname,shortname,versionid,longname)
            #shortname = sotestdataitem.shortname
            #puts "def[] shortname is #{shortname}"
            @data.push(sotestdataitem)
          end
          @data
        else
          nil
        end
      rescue =>e
        puts e.backtrace
        nil
      end
    end

    def get_unique_collection_esdts()
      @data.map(&:esdt).uniq
    end
```

```
    def get_unique_collection_data_items()
      @data.uniq{|item| item.esdt}
    end

    def get_so_test_data_granules(so_test_data_items)

      @so_test_data_granules = so_test_data_items.map { |so_test_data_item|
                            Sdps::SoTestData::SoTestDataGranule.new(
                            so_test_data_item.pathname,
                            so_test_data_item.shortname,
                            so_test_data_item.versionid,
                            so_test_data_item.longname)
                            }
    end

  end # class SoTestData
end  # module Sdps
```

Current Test Script: omgranule.rb

```
module Sdps
  module Om
    class Sdps::Om::OmGranule

    end
  end
end
```

Current Test Script: omserver.rb

```
module Sdps
  module Om
    class Sdps::Om::OmServer

      attr_reader :pid,:host,:mode,:calling_host,:db_config,:server_config

      def initialize(mode,dbconn)
        @host = OM_SERVER_HOST
        @mode = mode
```

```
        @config_file = "/usr/ecs/CUSTOM/#{@mode}/cfg/".OM_SERVER_CONFIG
        @omdb = Sdps::Om::OmDb.new(dbconn)
        init_calling_host()
        init_db_config_hash()
        init_server_config_hash()
        is_up?
      end

      def is_up?
        if @calling_host == @host
          @pid = `ps auxwww|grep #{@mode}|grep #{@server_config["Name"]}|awk '{print $2}'`
        else
          @pid = `ssh #{@mode} "ps auxwww|grep #{@mode}|grep #{@server_config["Name"]}"|awk '{print $2}'`
        end
        return @pid?true:false
      end

      private
      def init_db_config_hash
        @db_config = @omdb.get_configs()
      end

      def init_server_config_hash
        @server_config = Hash.new
        ret_array = `cat #{@config_file} | grep -v "^#" | grep "\S" | sed
's/^\s*\(\S*\)\s*=\s*\(\S*.*\)/\1\|\2/g'`
        ret_array.each do |line|
          values = line.split("|")
          @server_config[values[0]] = values[1]
        end
      end

      def init_calling_host
        @calling_host = `hostname`
      end
```

```
    end
  end
end
Current Test Script: omdb.rb
module Sdps
  module Om
    class Sdps::Om::OmDb

      def initialize(dbConn)
        @conn = dbConn
      end

      def do_query(sql, *params)
        @conn.exec_params(sql, params)
      end

      GET_CONFIGS_SQL = "select name,value from omconfig"
      def get_configs()
        rs = do_query(GET_CONFIGS_SQL)
        if rs != nil and rs.count > 0
          db_config_hash = Hash.new
          rs.each do |row|
            name = row['name']
            value = row['value']
            db_config_hash[name] = value
          end
          db_config_hash
        else
          #puts "no rows"
          nil
        end
      end
    end
  end
endCurrent Test Script: omrequest.rb
```

```
module Sdps
  module Om
    class Sdps::Om::OmRequest

    end
  end
end
Current Test Script: xru.rb
  module Sdps
    class XRU
      attr_reader :mode

      def initialize(mode)
         @mode = mode
      end

      def xml_replacement(xml_file)
         begin
           @dplhost = Conf.get_property('xru','hostname')
           puts "xru-xmlfile:" +  xml_file
           puts "dplhost= " + @dplhost
           puts "mode= " + @mode

           status = system("ssh -q #{@dplhost} '/usr/ecs/#{@mode}/CUSTOM/utilities/EcDsAmXruStart #{@mode}
-xmlfile #{xml_file}'")
           puts status
           true
         rescue =>e
           puts e.backtrace
           false
         end
      end

       #TODO: check logs for success or errors
      def check_log()
```

```
      end

    end # class XRU
  end # module Sdps
```

Current Test Script: aimdbextras.rb

```
# Enancements to AmCollection, AmGranule

module Sdps
  module AimDbExtras

    def esdt
      "%s.%03d" % [shortname, versionid]
    end

    def datasetid
      "%s V%03d" % [longname, versionid]
    end

  end
end
```

Current Test Script: dbrecord.rb

```
require 'time'

module Sdps
  class DbRecord

    class << self

      # NOTE Some times have usec, some do not
      # Time.strptime adds a timezone, which should be ignored
      STR_TO_TIME_NO_USEC = '%Y-%m-%d %H:%M:%S'
      STR_TO_TIME_USEC = STR_TO_TIME_NO_USEC + '.%N'
      TIME_TO_STR_NO_USEC = STR_TO_TIME_NO_USEC
      TIME_TO_STR_USEC = TIME_TO_STR_NO_USEC + '%6N'
```

```
# Use in a class definition to define a database field.
# Similar to attr_reader but requires a PostgreSQL type string
# (e.g., from psql: \d tablename).
def db_attr(attr_name, attr_type)
  case attr_type
  when 'bigint', 'smallint', 'integer'
    define_method(attr_name) {
      value = instance_variable_get("@#{attr_name}")
      value and value.to_i
    }
  when 'double precision'
    define_method(attr_name) {
      value = instance_variable_get("@#{attr_name}")
      value and value.to_f
    }
  when 'timestamp without time zone'
    define_method(attr_name) {
      str_value = instance_variable_get("@#{attr_name}")
      if str_value
        begin
          Time.strptime(str_value, STR_TO_TIME_USEC)
        rescue ArgumentError => e
          Time.strptime(str_value, STR_TO_TIME_NO_USEC)
        end
      end
    }
  when 'date'
    define_method(attr_name) do
      value = instance_variable_get("@#{attr_name}")
      value and Date.strptime(value)
    end
  else
    attr_reader attr_name
  end
end
```

```
    end

    # Takes a tuple from a PG query.
    # A tuple is an ordered list of Hashes: [{field1: val1}, ...]
    def initialize(tuple)
      tuple.each do |name, value|
        instance_variable_set("@#{name}".to_sym, value)
      end
    end

  end
end
```

Current Test Script: mode.rb

```
module Sdps

  class Sdps::Mode
    attr_reader :mode, :proxy_log_location, :proxy_up, :mock_echo_up, :stunnel_up, :dispatcher_up,
:autodriver_up, :monitor_up

    def initialize (mode)
      @mode = mode
      mode_checked_out?()
      #bmgt_subsystem_checked_out?()
      #get_proxy_log_location()
      # add other subsystem checks
    end

    def mode_checked_out?()
    # add check hosts
    # add check database
      true
    end

    def get_db_connection

        db_conn = Sdps::Db.connect('aim', @mode)
```

```
    db_conn
end

#BMGT
#need to be updated to work with tcp proxy and mock echo
def get_proxy_log_location()
  if @proxy_up == true
    @proxy_log_location = "/tools/common/test/BE_82_01/proxy/#{@mode}/logs/tcp.log"
  elsif @mock_echo_up == true
    @proxy_log_location = "/tools/common/test/BE_82_01/proxy/#{@mode}/logs/simplemockecho.log"
  end
end




def bmgt_subsystem_checked_out?()
  #check if proxy is running
  if proxy_up?() == true
    get_proxy_log_location()
  #check if bmgt  servers are running
    bmgtservers_up?()
  end

end

#needs to be updated with configurable parameters
PROXY_PORT = {
   "OPS" => 3000,
   "DEV02" => 3200,
   "DEV06" => 3600,
   "DEV07" => 3700,
   "DEV08" => 3800,
   "DEV09" => 3900,
   "TS3" => 3030
}
```

```
    def proxy_up?()
      @stunnel_up = up?("ps auxwww | grep -E '[s]tunnel
/tools/common/test/BE_82_01/proxy/#{@mode}/cfg/stunnel_client.cfg'")

      @proxy_up = up?("ps auxwww | grep -E '[r]uby [^ ]*/tcpproxy.rb #{PROXY_PORT[@mode]}'")


      if @proxy_up == false
         # check for mock echo
         @mock_echo_up = up?("ps auxwww | grep 'ruby [^ ]*/simplemockecho[.]rb #{PROXY_PORT[@mode]}'")
      end
      (@stunnel_up and @proxy_up) or (@mock_echo_up)
    end

    def bmgtservers_up?()
      @dispatcher_up = up?("ps auxwww | grep 'gov.nasa.eed.bmgt.dispatcher.Main #{@mode}' | grep -v grep")
      if @dispatcher_up == false
          puts "dispatcher is down"
      end

      @autodriver_up = up? ("ps auxwww | grep 'gov.nasa.eed.bmgt.auto.Main #{@mode}' | grep -v grep")
      if @autodriver_up == false
          puts "autodriver is down"
      end

      @monitor_up = up?("ps auxwww | grep 'gov.nasa.eed.monitor.Main #{@mode}' | grep -v grep")
      if @monitor_up == false
          puts "monitor is down"
      end

      dispatcher_up and autodriver_up and monitor_up
    end

    def up?(command)
         output =  system(command)
```

```
    end


    #Data Access
    def da_subsystem_checked_out?()
        # check if esi, egi, rqs, esir, gdalservice and hegservice are up
        true
    end
  end # class Mode

end # module Sdps
Current Test Script: da.rb
require 'sdps/da/dadb'
require 'sdps/da/gdal'
require 'sdps/da/heg'
require 'sdps/da/esi'
require 'sdps/da/egi'
require 'sdps/da/dautil'

module Sdps
  module Da

    #DA specific constants
    DB_SUBSYSTEM = 'aim'

    #global hashes
    FORMAT_EXTENSIONS =  { "GeoTIFF" => ".tif",
                           "NetCDF-3" => ".nc",
                           "NetCDF4-CF" => ".nc",
                            "NetCDF-4" => ".nc",
                           "ASCII" => ".ascii",
                           "KML" => ".kmz",
                           "HDF-EOS" => ".hdf",
                           "HDF-EOS5" => ".he5",
                           "PNG" => ".png",
                           "TABULAR_ASCII" => ".ascii"
```

```
    }

    #the epoch for GLAS and ICE SAT collections
    EPOCH = {"ICESAT2" => "2018-01-01T00:00:00.000000",
                       "GLAS" => "2000-01-01T00:00:00.000000"
    }

  end # module  Da
end # module Sdps
```

Current Test Script: db.rb

```
require 'sdps/errors'
require 'sdps/conf'
require 'logger'

PGX_RB = "/usr/ecs/#{ENV['MODE']}/CUSTOM/lib/COM/ruby/pgx.rb"

begin
  require PGX_RB
rescue LoadError => e
  if ENV['MODE'].nil?
    raise Sdps::ModeMissingError.new('The MODE environment variable must be set.')
  else
    raise RuntimeError.new("#{PGX_RB}: File not found.")
  end
end

module Sdps
  module Db
    INFO_LOGGER = Logger.new(STDOUT)
    INFO_LOGGER.level = Logger::INFO

    DEFAULT_OPTIONS = {
      port: 5432,
      host: 'f4db103',
      dbname: 'ecs',
```

```
      user: 'autotest',
      logger: INFO_LOGGER
    }

    class << self

      def connect(db_subsystem, ecs_mode=ENV['MODE'], options=DEFAULT_OPTIONS)
        port_mode = 'dbport_' + ecs_mode
        db_port = Conf.get_property('global',port_mode)
        if(ecs_mode == 'DEV01'||ecs_mode == 'DEV02'||ecs_mode == 'DEV06'||ecs_mode == 'DEV07'||ecs_mode ==
'TS2')
          db_host = Conf.get_property('global','dbhost_alt')
        else
          db_host = Conf.get_property('global','dbhost')
        end
        if(!db_port.nil?)
          options[:port] = db_port
        end
        if(!db_host.nil?)
          options[:host] = db_host
        end
        conn = new_pg_connection(db_subsystem, ecs_mode, options)
        if block_given?
          begin
            yield conn
          ensure
            conn && (not conn.finished?) && conn.finish
          end
        else
          at_exit do
            conn && (not conn.finished?) && conn.finish
          end
          conn
        end
      end
```

```
      # FIXME Get DB connection params from config
      # Note: password must be set in current user's ~/.pgpass
      def new_pg_connection(db_subsystem, ecs_mode, options)
        init = lambda { |con| con.exec ("select public.set_search_path('#{db_subsystem}', '#{ecs_mode}',
'#{options[:user]}')") }
        begin
          PGx::Connection.new(options.merge(connect_init: init))
        rescue => e
          puts e.backtrace
          raise e
        end
      end

      def do_query(sql, *params)
        @dbconn.exec_params(sql, params)
      end
    end
  end
end
```

Current Test Script: deletion.rb

```
  module Sdps
    class Deletion
      attr_reader :mode

        def initialize(current_mode)
           @mode = current_mode
        end

        #to be used only for deleting one or two test granules
        #for large granule deletion - use the pvc deletegranule utility
        def delete_granule(granuleid)
           begin
             delete_science_data_file(granuleid)
             delete_science_metadata_file(granuleid)
```

```
            delete_browse_data_file(granuleid)
            delete_science_browse_link(granuleid)
            delete_from_database(granuleid)
            true
          rescue =>e
             puts e.backtrace()
             false
          end

        end


        def delete_science_data_file(granuleid)

        end

        def delete_science_metadata_file(granuleid)

        end

        def delete_browse_data_file(granuleid)

        end

        def delete_science_browse_link(granuleid)

        end

        def delete_from_database(granuleid)

        end
    end
  end
```

Current Test Script: heg_1.rb

```
#heg and heg webservice methods
```

```
module Sdps
   module Da
      require 'nokogiri'
      require 'uri'

      require 'sdps/da/daxml'
      require 'sdps/da/dadb'

      class Sdps::Da::Heg
        RESPONSE_FILE_NOT_FOUND = "RESPONSE_FILE_NOT_FOUND"
        RESPONSE_FILE_PARSE_ERROR = "RESPONSE_FILE_PARSE_ERROR"
        RESPONSE_FILE_PARSE_EXCEPTION = "RESPONSE_FILE_PARSE_EXCEPTION"
        ESI_RESPONSE_SCHEMA = "http://eosdis.nasa.gov/esi/rsp/i
http://newsroom.gsfc.nasa.gov/esi/8.1/schemas/ESIAgentResponseInternal.xsd"

        SERVICE_NAME = "HEG"
         #HEG test for AE_Ocean.002
         #http://f5eil01v:22500/egi_TS2/request?FILE_IDS=46785&DATASET_ID=AMSR-E%2FAqua L2B Global Swath Ocean
Products derived from Wentz Algorithm
V002&CLIENT=ESI&EMAIL=sudha.murthy@nasa.gov&REQUEST_MODE=sync&INTERPOLATION=NN&FORMAT=HDF-EOS&PROJECTION=GEOGRAP
HIC&SUBAGENT_ID=HEG&BBOX=-50,-50,50,50&SUBSET_DATA_LAYERS=Swath1,Swath1%3AHigh_res_cloud,Swath1%3AHigh_res_cloud
%3ABand_1KM_RefSB,Swath1%3AHigh_res_cloud%3ABand_1KM_RefSB%3ANBandDim&

        def initialize(mode)
           @host  ="f5dpl01v"
           @port  = 22500
           @service ="HegService"
           @mode   = mode

           #@service_url = "http://#{@host}#{@port}/#{@service}_#{@mode}"
           if @mode == "OPS"
             @service_path = "/#{@service}/"
           else
              @service_path = "/#{@service}_#{@mode}/"
           end
```

```
           #need to remove the release ...or make it come from the yaml config
           @output_response_dir  = "/workingdata/TestResults/RelB/DataAccess/HegService/#{@mode}/response/"
           @output_results_dir  = "/workingdata/TestResults/Rel82/DataAccess/HegService#{@mode}/results/"
        end

        def build_query_params(granule_file, format, output_dir)
              query_params = Hash.new()
              query_params[:FORMAT] = format
              query_params[:FILE_URLS] = granule_file
              #query_params[:OUTPUT_DIR] = output_dir
              puts query_params
              query_params
        end

        def submit_request(granule, granule_file, format)
          output_filename = nil
          begin
            query_params = build_query_params(granule_file, format, @output_results_dir)
            puts query_params.to_a
            query_params_str = URI.encode_www_form(query_params.to_a)
            puts query_params_str
            # newuri = URI::HTTP.build({:host=>'www.google.com'})
             #puts newuri
             puts @host
             puts @port
             puts @service_path
             puts query_params_str
            @service_url = URI::HTTP.build({:host=>@host, :port=>@port, :path=>@service_path,
:query=>query_params_str})
            puts @service_url
            timestamp = Time.now.strftime('%Y_%m_%d_%H_%M_%S')

            output_filename =
"#{@output_response_dir}#{@service}_#{granule.esdt}_#{granule.granuleid}_#{format}.xml"
            #command = ("curl -# -XGET '#{@service_url}' > #{output_filename}")
            command = ("curl -# -XGET '#{@service_url}'")
```

```
    puts "command= #{command}"
    response = system (command)
  rescue=>e
    puts e.backtrace
    e.to_s
  end
end #submitRequest


def parse_response(response_xml, granuleid)
   begin
     @heg_xml = Nokogiri::XML(response_xml)
     if @heg_xml.nil?
       response = Response.new(RESPONSE_FILE_NOT_FOUND, granuleid)
     elsif @heg_xml.contains?("/Exception")
       status = @heg_xml.get_value('/Exception/Code')
       message = @heg_xml.get_value('/Exception')[0..198]
       #info = @gdal_xml.get_xml('/Exception/addlInfo')[0..198]
       #message = message1[0..198]
       #if info.nil? = false
       #   message = (message1 << info)[0..198]
       #end
       response = Sdps::Da::Heg::Response.new(status, message)
     elsif @gdal_xml.contains?("/agentResponse")
       status = @gdal_xml.get_value('/agentResponse/processInfo/message')
       #message = @gdal_xml.get_value('/agentResponse')
       #urls = @gdal_xml.get_value('/agentResponse/downloadUrls')
       response = Sdps::Da::Gdal::Response.new(status, xml_filename)
     else
        response = Sdps::Da::Gdal::Response.new(RESPONSE_FILE_PARSE_ERROR, @gdal_xml)
     end
   rescue=>e
     puts e.backtrace
     response = Sdps::Da::Gdal::Response.new(RESPONSE_FILE_PARSE_EXCEPTION, e.to_s)
   end
   response
```

```
end # parse_response


def get_heg_formats
     formats = `#{GDAL_TRANSLATE_FORMATS_CMD}`
     if formats.nil?
        puts "no formats"
        nil
     else
        arr = formats.split("\n  ")
        #p arr
        if arr.nil? or arr.size == 0
           puts "no valid formats"
           nil
        else
           # delete first row
           arr.delete_at(0)
           #p arr
           heg_formats = Hash[ arr.map { |v| [ v.split(" (")[0], v.split(" (")[1].split("): ")[1] ] } ]
        end
     end
end

def get_heg_info(dataset)

end

def run_heg_tool()


end


class Sdps::Da::Heg::Response
   attr_reader :status, :message
   def initialize (code, description)
```

```
            @status = code
            @message = description
          end
        end

      class Sdps::Da::Heg::Request
           attr_reader :url, :format, :options, :operand
      end




         end # class Heg
    end # module Da
end # module Sdps
Current Test Script: utils.rb
module Sdps
    module Da
        require 'fileutils'
        class Sdps::Da::Utils

            H5DUMP = "/tools/hdf5/bin/h5dump"
             LISTGEO_CMD = "/net/origin/devdata1/DataAccess/test/libgeotiff-1.4.0/bin/listgeo -d"
             PROJ4_CMD = "/tools/proj/bin/proj"

            def initialize()
                begin

                rescue =>e
                    puts e.backtrace
                    nil
                end
            end

            def get_corner_points(input_wsen_arr)
                min_x = input_wsen_arr[0].to_f
```

```
            min_y = input_wsen_arr[1].to_f
            max_x = input_wsen_arr[2].to_f
            max_y = input_wsen_arr[3].to_f

            corner_points_array = [[min_x,max_y], [max_x,max_y], [max_x, min_y], [min_x, min_y]]
            #check for any points at axes edges in the span
            #0, -90, 90, -180, +180

        end

        #WSEN array
        def get_max_bbox(corner_points_arr, proj4_param)
             m_x_arr = Array.new
             m_y_arr = Array.new
             #p corner_points_arr
             corner_points_arr.each_index { | index |
                 corner_point_x = corner_points_arr[index][0]
                 corner_point_y = corner_points_arr[index][1]
                 #p corner_point_x
                 #p corner_point_y

                #p proj4_param
                #p PROJ4_CMD

                 cmd = %Q[echo " #{corner_point_x}  #{corner_point_y}  " | #{PROJ4_CMD}  #{proj4_param} -f
'%0.9f']
                 #puts "cmd1=#{cmd}"
                 #corner_point_m = `echo "#{corner_point_x}  #{corner_point_y}" | #{PROJ4_CMD}
#{proj4_param} -f  '%0.9f'`
                 corner_point_m = `#{cmd}`
                 corner_point_m_arr = corner_point_m.strip.split()
                 x = corner_point_m_arr[0].strip.to_f
                 m_x_arr.push(x)
                 y = corner_point_m_arr[1].strip.to_f
                 m_y_arr.push(y)
             }
```

```
                p m_x_arr
                p m_y_arr
                p m_x_arr.min
                p m_y_arr.max
                p m_x_arr.max
                p m_y_arr.min
                max_bbox_m_arr = [[m_x_arr.min, m_y_arr.max], [m_x_arr.max, m_y_arr.min]]
                p max_bbox_m_arr
                max_bbox_arr = Array.new
                max_bbox_m_arr.each_index {|index|

                    corner_point_x = max_bbox_m_arr[index][0]
                    corner_point_y = max_bbox_m_arr[index][1]
                    #p corner_point_x
                    #p corner_point_y

                    cmd = %Q[echo " #{corner_point_x}  #{corner_point_y}  " | #{PROJ4_CMD}  #{proj4_param} -I
-f  '%0.9f']

                    #cmd = 'echo "#{corner_point_x}  #{corner_point_y}" | #{PROJ4_CMD}  #{proj4_param} -I -f
"%0.9f" '
                    #puts "cmd2=#{cmd}"
                    #corner_point_m = `echo "#{corner_point_x}  #{corner_point_y}" | #{PROJ4_CMD}
#{proj4_param} -I -f  '%0.9f'`
                   corner_point_m = `#{cmd}`
                   corner_point_m_arr = corner_point_m.strip.split()
                    x = corner_point_m_arr[0].strip.to_f

                    y = corner_point_m_arr[1].strip.to_f

                    max_bbox_arr.push([x,y])
                }
                p max_bbox_arr
                max_bbox_arr
            end
```

```
def get_modified_input_subset(input_wsen_arr, proj4_cmd)
    #get input corner points
    corner_points_arr = get_corner_points(input_wsen_arr)

   #convert input_array to meters and get max bbox
   max_bbox = get_max_bbox(corner_points_arr, proj4_cmd)
   #add axis coordinates if in the range
   #get modified input wsen
   #convert back to degrees
   p max_bbox

   modified_input_wsen = [max_bbox[0][0], max_bbox[1][1], max_bbox[1][0], max_bbox[0][1]]
   p modified_input_wsen
   modified_input_wsen
end


def spatial_extent_match?(input_wsen_arr, output_wsen_arr, delta)
    match = true
    input_wsen_arr.each_index { | index |
        if (approx?(input_wsen_arr[index].to_f, output_wsen_arr[index].to_f, 0.05) == false)
            match = false
        end
    }
    match
end

def approx?(float1, float2, delta)
    float1.abs - float2.abs < delta
end

def dms_to_degrees(d, m, dir)
    #puts "dms to degrees"
    #puts d
    #puts m
    #puts dir
```

```
    degrees = d + m/60
    case dir
    when "W"
        degrees = degrees * -1
        #puts degrees
    when "S"
        degrees = degrees * -1
        #puts degrees
    end
    #puts degrees
    degrees
end


def get_band_params(h5_file)
    begin
        contents = `'#{H5DUMP}' --contents '#{h5_file}' | grep dataset`
        contents1 = contents.split(" dataset   ")
        contents2 = Array.new
        contents1.each {  | band |
            if !band.nil? && !band.empty?
                contents2.push(band.strip)
            end
        }
        contents2
    rescue=> e
        puts "exception:" + e.to_s
        puts e.backtrace
    end
end

# ######################################################## h5dump -d
# /South_Polar_Projection/cell_tb_v_fore -y -w 1 -A 0 filename
# process first 10 fields
# ########################################################
def get_h5_data(h5_file, band)
```

```
    begin
        data_dump = `'#{H5DUMP}' -d '#{band}' -y -w 1 -A 0  '#{h5_file}' `
        data_dump_arr = data_dump.split("\n ")
        subset_parameter = data_dump_arr[0]
        # #puts subset_parameter
        if subset_parameter.include?(band)
            # #skip the next 3 bands
            data = data_dump_arr[4..14]
            data10 = remove_space_comma(data)
            puts "band=#{subset_parameter},  input file 10 fields = #{data10}"
            data10
        else
            puts "h5 data does not contain parameter"
            nil
        end
    rescue=> e
        puts "exception:" + e.to_s
        puts e.backtrace
        nil
    end
end

# ############################# process first 10 fields
# ##############################
def get_opendap_ascii_data(ascii_file, band)
    begin
        # #puts "ascii file=#{ascii_file}"
        band1 = band.gsub("/","_")
        # #puts "band=#{band1}"
        lines = IO.readlines(ascii_file)
        # #first line is filename
        filename = lines[0]
        # #puts "line 0 from ascii file = #{filename}"
        data_dump_arr = lines[1].split(",")
        subset_parameter = data_dump_arr[0]
        # #puts "subset parameter in ascii file =
```

```
        # #{subset_parameter}"
        if subset_parameter.include?(band1)
            data = data_dump_arr[1..11]
            data10 = remove_space_comma(data)
            puts "band=#{subset_parameter}, first 10 fields in ascii file = #{data10}"
            data10
        else
            puts " ascii output does not contain parameter"
            nil
        end
    rescue => e
        puts "exception:" + e.to_s
        puts e.backtrace
        nil
    end
end

# ############################## process first 10 fields of nc output
# ##############################
def get_opendap_ascii_data(nc_file, band)
    begin
        # #puts "nc file=#{nc_file}"
        band1 = band.gsub("/","_")
        # #puts "band=#{band1}"
        lines = IO.readlines(ascii_file)
        # #first line is filename
        filename = lines[0]
        # #puts "line 0 from ascii file = #{filename}"
        data_dump_arr = lines[1].split(",")
        subset_parameter = data_dump_arr[0]
        # #puts "subset parameter in ascii file =
        # #{subset_parameter}"
        if subset_parameter.include?(band1)
            data = data_dump_arr[1..11]
            data10 = remove_space_comma(data)
            puts "band=#{subset_parameter}, first 10 fields in ascii file = #{data10}"
```

```
            data10
        else
            puts " ascii output does not contain parameter"
            nil
        end
    rescue => e
        puts "exception:" + e.to_s
        puts e.backtrace
        nil
    end
end



def remove_space_comma(data)
    data1 = data.collect {|x | x.strip}
    data2 = data1.collect { |x| x.gsub(",","")}
end

def compare_numeric_elements(input, output)
    if input.length == output.length
        result = true
        input.each_index do | i |
            in_val = input[i]
            out_val = output[i]
            if (in_val != out_val)
                puts ("expected = #{in_val}, result= #{out_val}")
                result = false
            end
        end
        result
    else
        false
    end
end
```

```
def validate_ascii_output(test_h5, output_file, band)
    input_arr = get_h5_data(test_h5, band)

    output_arr = get_opendap_ascii_data(output_file, band)
    if input_arr.nil? or output_arr.nil?
        return nil
    else
        if input_arr.eql?(output_arr)
            return true
        else compare_numeric_elements(input_arr, output_arr)
        end
    end
end

def validate_netcdf3_output(test_h5, output_file, band)
    input_arr = get_h5_data(tes_h5, band)
    output_arr = get_opendap_netcdf_data(output_file, band)
    if input_arr.nil? or output_arr.nil?
        return nil
    else
        if input_arr.eql?(output_arr)
            return true
        else compare_numeric_elements(input_arr, output_arr)
        end
    end

end
def copy_with_path(src, dest_dir)
    #FileUtils.mkdir_p(File.dirname(dest))
    FileUtils.mkdir_p(dest_dir)
    FileUtils.cp(src, dest_dir)
end

class Point

    attr_accessor :x,:y
```

```
                def initialize(*args)
                    @y,@x=args
                end

            end

        end # class Utils
    end #module da
end #module Sdps
```

Current Test Script: egi.rb

```
# ##########################################################################################
# EGI
#
#
# ##########################################################################################
module Sdps
    module Da
        require 'nokogiri'
        require 'sdps/da/daxml'
        require 'sdps/da/dadb'
        require 'sdps/da/dautil'
        require 'sdps/da/service'

        class Sdps::Da::Egi  < Sdps::Da::Service
           attr_reader :egi_service_request_url, :egi_service_response
            def initialize(mode)
                super(mode, "egi")
                if @mode == "DEV07" or @mode == "OPS"
                    @host  ="f7eil01.edn.ecs.nasa.gov"
                else
                    @host  ="f5eil01.edn.ecs.nasa.gov"
                end
                #if mode == "OPS" or mode == "ops"
                  #  @service_url = "https://#{@host}/#{@service}"
```

```
                #  @request_url = "https://#{@host}/#{@service}/request"
                  @service_url = "https://#{@host}/#{mode.downcase}/#{@service}"
                  @request_url = "https://#{@host}/#{mode.downcase}/#{@service}/request"
                #  @service_url = "http://#{@host}:22500/#{@service}"
                #  @request_url = "http://#{@host}:22500/#{@service}/request"
              #else
                  puts "mode=#{mode}"
      puts "Service URL=#{@service_url}"
      puts "Request URL=#{@request_url}"
                #  @service_url = "https://#{@host}/#{@service}_#{mode}"
                #  @request_url = "https://#{@host}/#{@service}_#{mode}/request"
                #  @service_url = "https://#{@host}/#{mode}/#{@service}"
                #  @request_url = "https://#{@host}/#{mode}/#{@service}/request"
                #  @service_url = "http://#{@host}:22500/#{@service}_#{mode}"
                #  @request_url = "http://#{@host}:22500/#{@service}_#{mode}/request"
             # end
          end

          #David's request
          def submit_pa_request(mode, esdt, producer_granuleid, projection, format, projection_param,
band_parameter)
              begin

                 @egi_request = Sdps::Da::Egi::Request.new(@request_url, "", "", format, projection,
projection_param, band_parameter , "" , "", "" , 'N', producer_granuleid)
                 @egi_service_request_url = @egi_request.egi_request_url

                 puts "#{@egi_service_request_url}"

                 @lowercase_mode = "#{mode}".downcase

                 @egi_service_response = `/tools/libcurl/bin/curl -k -s -L -o
/ecs/formal/COMMON/automation/autotest/#{@lowercase_mode}.tif -J "#{@egi_service_request_url}"`
              rescue=>e
                  puts e.backtrace
                  e.to_s
```

```
                end
            end

            def submit_request(esdt, granuleid, subagent_id, format:"", projection:"", projection_param:"",
band_parameter:"" , bbox:"" , temporal:"", interpolation:"" , include_meta:"N")
                begin
                    file_ids = "SC:#{esdt}:#{granuleid}"
                    @egi_request = Sdps::Da::Egi::Request.new(@request_url, file_ids, subagent_id, format,
projection, projection_param, band_parameter , bbox , temporal, interpolation , include_meta,"")
                   @egi_service_request_url = @egi_request.egi_request_url
                   @egi_service_response = `curl -k -s -XGET '#{@egi_service_request_url}'`
                   #@egi_service_response = `curl -k -s -X POST '#{@egi_service_request_url}'`
                rescue=>e
                    puts e.backtrace
                    e.to_s
                end
            end

             def submit_sync_request(esdt, granuleid, subagent_id, format:"", projection:"",
projection_param:"", band_parameter:"" , bbox:"" , temporal:"", interpolation:"" , include_meta:"N")
                begin
                    file_ids = "SC:#{esdt}:#{granuleid}"
                    @egi_request = Sdps::Da::Egi::Request.new(@request_url, file_ids, subagent_id, format,
projection, projection_param, band_parameter , bbox , temporal, interpolation , include_meta,"")
                   @egi_service_request_url = @egi_request.egi_request_url
                   @egi_service_response = `curl -k -s -XGET '#{@egi_service_request_url}'`
                   #@egi_service_response = `curl -k -s -X POST '#{@egi_service_request_url}'`
                rescue=>e
                    puts e.backtrace
                    e.to_s
                end
            end

            def submit_post_request(esdt, granuleid, subagent_id, format:"", projection:"", projection_param:"",
band_parameter:"" , bbox:"" , temporal:"", interpolation:"" , include_meta:"N")
                begin
```

```
                file_ids = "SC:#{esdt}:#{granuleid}"
                @egi_request = Sdps::Da::Egi::PostRequest.new(@request_url, file_ids, subagent_id, format,
projection, projection_param, band_parameter , bbox , temporal, interpolation , include_meta,"")
               @egi_service_request_url = @egi_request.egi_request_url
               #@egi_service_response = `curl -k -s -XGET '#{@egi_service_request_url}'`
               puts "curl -k -s -X POST '#{@egi_service_request_url}'"
               @egi_service_response = `curl -k -s -X POST '#{@egi_service_request_url}'`
            rescue=>e
                puts e.backtrace
                e.to_s
            end
        end

        def parse_response(xml_buff)
             begin
                @egi_xml = Sdps::Da::DaXml.new(xml_buff)
                if @egi_xml.nil?
                    response = Sdps::Da::Egi::Response.new(RESPONSE_FILE_NOT_FOUND, "FAILURE", xml_buff)
                elsif @egi_xml.contains?("/Exception")
                    status = @egi_xml.get_value('/Exception/Code')
                    response = Sdps::Da::Egi::Response.new(status, "FAILURE",@egi_xml)
                elsif @egi_xml.contains?("/agentResponse")
                    status = @egi_xml.get_value('//status')
                    urls = @egi_xml.get_value('//downloadUrls')
                    response = Sdps::Da::Egi::Response.new(status, "SUCCESS", urls)
                   if !urls.nil? && !urls.empty?
                        response = Sdps::Da::Egi::Response.new(status, "SUCCESS", urls)
                    else
                        message = @egi_xml.get_value('//message')
                        response = Sdps::Da::Egi::Response.new(status, message, urls)
                    end
                else
                    response = Sdps::Da::Egi::Response.new(RESPONSE_FILE_PARSE_ERROR, "FAILURE", @egi_xml)
                end
            rescue=>e
                puts e.backtrace
```

```
                    response = Sdps::Da::Egi::Response.new(RESPONSE_FILE_PARSE_EXCEPTION, "FAILURE", e.to_s)
                end
            end # parse_response

            class Sdps::Da::Egi::Response < Sdps::Da::Service::Response

            end

            class Sdps::Da::Egi::Request < Sdps::Da::Service::Request
                attr_reader :egi_request_url
                def initialize(req_url, file_ids, subagent_id, format, projection, projection_param,
band_parameter , bbox , temporal, interpolation , include_meta,producer_granule_id)
                    super(subagent_id, format, projection, projection_param, band_parameter , bbox , temporal,
interpolation , include_meta)
                    #puts "#{@command_url}"
                    request_str = "#{req_url}"
                      if !file_ids.nil? and !file_ids.empty?
                           request_str = "#{request_str}?FILE_IDS=#{file_ids}"
                      elsif !producer_granule_id.nil? and !producer_granule_id.empty?
                           request_str = "#{request_str}?producer_granule_id=#{producer_granule_id}"
                      end
                      if !request_str.include?("REQUEST_MODE")
                           request_str = "#{request_str}&REQUEST_MODE=sync"
                      end
                    @egi_request_url ="#{request_str}#{@request_url}"
                    #puts "#{@egi_request_url}"
                end
           end

            class Sdps::Da::Egi::PostRequest < Sdps::Da::Service::Request
                attr_reader :egi_request_url
                def initialize(req_url, file_ids, subagent_id, format, projection, projection_param,
band_parameter , bbox , temporal, interpolation , include_meta,producer_granule_id)
                    super(subagent_id, format, projection, projection_param, band_parameter , bbox , temporal,
interpolation , include_meta)
                    #puts "#{@command_url}"
```

```
                    #request_str = "#{req_url}"
                     if !file_ids.nil? and !file_ids.empty?
                           request_str = "#{request_str}?FILE_IDS=#{file_ids}"
                     end
                     if !producer_granule_id.nil? and !producer_granule_id.empty?
                           request_str = "#{request_str}?producer_granule_id=#{producer_granule_id}"
                     end
                     if !request_str.include?("REQUEST_MODE")
                           request_str = "#{request_str}?REQUEST_MODE=async"
                     else
                        request_str =  request_str.gsub('sync','async')
                     end
                     @egi_request_url ="#{request_str}#{@request_url}"
                    puts "#{@egi_request_url}"
                end
            end
        end # class Egi
    end # module Da
end # module Sdps
Current Test Script: daxml.rb

# xml functions specific to Data Access
module Sdps
  module Da

    require 'nokogiri'
    require 'sdps/tcpproxylogger'

    class Sdps::Da::DaXml

        def initialize(xml_buffer)
            begin
              #@xml_doc = Nokogiri::XML(File.read(xml_file))
              @xml_doc = Nokogiri::XML(xml_buffer)
              @xml_doc.remove_namespaces!
            rescue =>e
```

```
      puts e.backtrace
      nil
    end
end

def get_value(xpath_expr)
     #puts xml_file
     #puts xpath_expr
     begin
        @xml_doc.xpath(xpath_expr).text()
     rescue=> e
         puts "exception:" + e.to_s
         puts e.backtrace
     end
end

def get_xml(xpath_expr)
     #puts xml_file
     #puts xpath_expr
     begin
        @xml_doc.xpath(xpath_expr).to_s
     rescue=> e
         puts "exception:" + e.to_s
         puts e.backtrace
     end
end

def contains?(xpath_expr)
    if @xml_doc.nil?
       false

    elsif @xml_doc.xpath(xpath_expr).nil?
       false
    elsif @xml_doc.xpath(xpath_expr).text().nil?
       false
    elsif @xml_doc.xpath(xpath_expr).text().empty?
```

```
            false
          else
            true
          end
      end

    end # class Xml

  end #module Bmgt

end #module Sdps

Current Test Script: dautil.rb
module Sdps
  module Da
    require 'fileutils'
    class Sdps::Da::DaUtil
      attr_reader :input_arr, :output_arr, :error_msg

      H5DUMP = "/tools/hdf5/bin/h5dump"
      LISTGEO_CMD = "/net/origin/devdata1/DataAccess/test/libgeotiff-1.4.0/bin/listgeo -d"
      PROJ4_CMD = "/tools/proj/bin/proj"
      NC4DUMP = "/tools/netcdf/bin/ncdump"
      NC3DUMP = "/tools/netcdf/bin/ncdump"
      H4DUMP = "/tools/hdf4/bin/ncdump"

      def initialize()
        begin
          @input_arr = Array.new
          @output_arr = Array.new
          @error_msg = ""
        rescue =>e
          puts e.backtrace
          nil
        end
      end
```

```
def netcdf_filetype(input_file)
  extension = File.extname(input_file)
  if extension.eql?(".nc")
    filetype = `'#{NC3DUMP}' -k  '#{input_file}'`
  end
end

# ########################################
#  Spatial subsetting
#
# ########################################
def get_corner_points(input_wsen_arr)
  min_x = input_wsen_arr[0].to_f
  min_y = input_wsen_arr[1].to_f
  max_x = input_wsen_arr[2].to_f
  max_y = input_wsen_arr[3].to_f

  corner_points_array = [[min_x,max_y], [max_x,max_y], [max_x, min_y], [min_x, min_y]]
  # #check for any points at axes edges in the span #0, -90, 90,
  # -180, +180

end

# #WSEN array
def get_max_bbox(corner_points_arr, proj4_param)
  m_x_arr = Array.new
  m_y_arr = Array.new
  # #p corner_points_arr
  corner_points_arr.each_index { | index |
    corner_point_x = corner_points_arr[index][0]
    corner_point_y = corner_points_arr[index][1]
    # #p corner_point_x #p corner_point_y

    # #p proj4_param #p PROJ4_CMD
```

```
  cmd = %Q[echo " #{corner_point_x}  #{corner_point_y}  " | #{PROJ4_CMD}  #{proj4_param} -f  '%0.9f']
  # #puts "cmd1=#{cmd}" #corner_point_m = `echo
  # "#{corner_point_x}  #{corner_point_y}" | #{PROJ4_CMD}
  # #{proj4_param} -f  '%0.9f'`
  corner_point_m = `#{cmd}`
  corner_point_m_arr = corner_point_m.strip.split()
  x = corner_point_m_arr[0].strip.to_f
  m_x_arr.push(x)
  y = corner_point_m_arr[1].strip.to_f
  m_y_arr.push(y)
}
# #p m_x_arr #p m_y_arr #p m_x_arr.min #p m_y_arr.max #p
# m_x_arr.max #p m_y_arr.min
max_bbox_m_arr = [[m_x_arr.min, m_y_arr.max], [m_x_arr.max, m_y_arr.min]]
# #p max_bbox_m_arr
max_bbox_arr = Array.new
max_bbox_m_arr.each_index {|index|

  corner_point_x = max_bbox_m_arr[index][0]
  corner_point_y = max_bbox_m_arr[index][1]
  # #p corner_point_x #p corner_point_y

  cmd = %Q[echo " #{corner_point_x}  #{corner_point_y}  " | #{PROJ4_CMD}  #{proj4_param} -I -f  '%0.9f']

  # #cmd = 'echo "#{corner_point_x}  #{corner_point_y}" |
  # #{PROJ4_CMD}  #{proj4_param} -I -f  "%0.9f" ' #puts
  # "cmd2=#{cmd}" #corner_point_m = `echo "#{corner_point_x}
  # #{corner_point_y}" | #{PROJ4_CMD}  #{proj4_param} -I -f
  # '%0.9f'`
  corner_point_m = `#{cmd}`
  corner_point_m_arr = corner_point_m.strip.split()
  x = corner_point_m_arr[0].strip.to_f

  y = corner_point_m_arr[1].strip.to_f

  max_bbox_arr.push([x,y])
```

```
  }
  # #p max_bbox_arr
  max_bbox_arr
end

def get_modified_input_subset(input_wsen_arr, proj4_cmd)
  # #get input corner points
  corner_points_arr = get_corner_points(input_wsen_arr)

  # #convert input_array to meters and get max bbox
  max_bbox = get_max_bbox(corner_points_arr, proj4_cmd)
  # #add axis coordinates if in the range #get modified input wsen
  # #convert back to degrees #p max_bbox

  modified_input_wsen = [max_bbox[0][0], max_bbox[1][1], max_bbox[1][0], max_bbox[0][1]]
  # #p modified_input_wsen
  modified_input_wsen
end


def spatial_extent_match?(input_wsen_arr, output_wsen_arr, delta)
  match = true
  input_wsen_arr.each_index { | index |
    if (approx?(input_wsen_arr[index].to_f, output_wsen_arr[index].to_f, 0.05) == false)
      match = false
    end
  }
  match
end

def approx?(float1, float2, delta)
  puts "delta=#{delta}"
  if(!float1.abs.zero?) && (!float2.abs.zero?)
    puts "abs1=#{(float1.abs - float2.abs) /float1.abs}"
    (float1.abs - float2.abs) /float1.abs < delta
  elsif(float1.abs.zero?)
```

```
    puts "abs2=#{(float2.abs - float1.abs)} "
    (float2.abs - float1.abs)  < delta
  elsif(float2.abs.zero?)
    puts "abs3=#{(float1.abs - float2.abs)}"
    (float1.abs - float2.abs).abs  < delta
  end
end

def dms_to_degrees(d, m, dir)
  # #puts "dms to degrees" #puts d #puts m #puts dir
  degrees = d + m/60
  case dir
  when "W"
    degrees = degrees * -1
    # #puts degrees
  when "S"
    degrees = degrees * -1
    # #puts degrees
  end
  # #puts degrees
  degrees
end

# ######################################################## get bands
# ##########################################################
def get_band_params(input_file)
  contents2 = Array.new
  begin
    puts "file=#{input_file}"
    extension = File.extname(input_file)
    if extension.casecmp(".h5").zero?
      contents2.clear
      contents = `'#{H5DUMP}' --contents '#{input_file}' | grep dataset`
      if contents.include?("HDFEOS/GRIDS")
        contents = `'#{H5DUMP}' --contents '#{input_file}' | grep "dataset    /HDFEOS/GRIDS"`
        if !contents.nil? and !contents.empty?
```

```
      #puts "contents= #{contents}"
      contents0 = contents.gsub("Data Fields/",'')
      #puts "contents0=#{contents0}"
      contents1 = contents0.split("dataset    /HDFEOS/GRIDS")
      #puts "contents1=#{contents1}"
      contents11 = contents1[1..-1]
      #puts "contents11=#{contents11}"
      contents11.each {  | band |
        #puts "#{band}"
        if !band.nil? && !band.empty?
          # #remove links in the band names
          band1 = band.split("->")
          contents2.push(band1[0].strip)
          #puts "#{band1}"
        end
      }
    end
  else
    contents1 = contents.split(" dataset   ")

    contents1.each {  | band |
      if !band.nil? && !band.empty?
        # #remove links in the band names
        band1 = band.split("->")
        contents2.push(band1[0].strip)
      end
    }
  end
elsif extension.eql?(".nc")
  # #check type of nc file
  contents2.clear
  contents = `'#{H5DUMP}' --contents '#{input_file}' | grep dataset`
  contents1 = contents.split(" dataset   ")

  contents1.each {  | band |
    if !band.nil? && !band.empty?
```

```
                # #remove links in the band names
                band1 = band.split("->")
                contents2.push(band1[0].strip)
              end
            }
          elsif extension.eql?(".hdf")
            contents2.clear
            contents =       `'#{H4DUMP}' -h  '#{input_file}' | grep -E
'(GridName=)|(DataFieldName=)|(SwathName=)'`
            contents1 = contents.gsub("\\t",' ').gsub("\\n",' ').gsub("\n", ' ').gsub('"',' ').gsub("\\", '
').split(",")
            group = ""
            contents1.each { |band|
              band1 = band.strip

              if band1.include?("GridName=")
                group = band1.split("GridName=")[1].strip
              elsif band1.include?("SwathName=")
                group = band1.split("SwathName=")[1].strip
              elsif band1.include?("DataFieldName")
                parameter = band1.split("DataFieldName=")[1].strip
                contents2.push("/#{group}/#{parameter}")
              end
            }
          elsif extension.eql?(".he5")
            contents2.clear
            contents = `'#{H5DUMP}' --contents '#{input_file}' | grep "dataset    /HDFEOS/GRIDS"`
            if !contents.nil? and !contents.empty?
              #puts "contents= #{contents}"
              contents0 = contents.gsub("Data Fields/",'')
              #puts "contents0=#{contents0}"
              contents1 = contents0.split("dataset    /HDFEOS/GRIDS")
              #puts "contents1=#{contents1}"
              contents11 = contents1[1..-1]
              #puts "contents11=#{contents11}"
              contents11.each {  | band |
```

```
      #puts "#{band}"
      if !band.nil? && !band.empty?
        # #remove links in the band names
        band1 = band.split("->")
        contents2.push(band1[0].strip)
        #puts "#{band1}"
      end
    }
  else
    contents = `'#{H5DUMP}' --contents '#{input_file}' | grep "dataset    /HDFEOS/SWATHS"`
    if !contents.nil? and !contents.empty?
      #puts "contents= #{contents}"
      contents0 = contents.gsub("Data Fields/",'')
      #puts "contents0=#{contents0}"
      contents1 = contents0.split("dataset    /HDFEOS/SWATHS")
      #puts "contents1=#{contents1}"
      contents11 = contents1[1..-1]
      #puts "contents11=#{contents11}"
      contents11.each {  | band |
        #puts "#{band}"
        if !band.nil? && !band.empty?
          # #remove links in the band names
          band1 = band.split("->")
          contents2.push(band1[0].strip)
          #puts "#{band1}"
        end
      }
    else
      puts "not handled by get_band_params"
    end
  end
else
  puts "extn= #{extension} - not handled by get_band_params"
end
#puts "contents2=#{contents2}"
contents2
```

```
  rescue=> e
    puts "exception:" + e.to_s
    puts e.backtrace
    contents2
  end
end


def get_zerolength_bands(input_file)
  contents2 = Array.new
  begin
    extension = File.extname(input_file)
    if extension.casecmp(".h5").zero?
      contents = `'#{H5DUMP}' --contents '#{input_file}' | grep dataset`
      contents1 = contents.split(" dataset   ")

      contents1.each {  | band |
        if !band.nil? && !band.empty?
          # #remove links in the band names
          band1 = band.split("->")
          band2 = band1[0].strip
          data_arr = get_h5_data_for_band(input_file,band2)
          if data_arr.nil? or data_arr.empty?
            contents2.push(band2.strip)
          end
        end
      }
    end
    #puts "contents2=#{contents2}"
    contents2
  rescue=> e
    puts "exception:" + e.to_s
    puts e.backtrace
    contents2
  end
end
```

```
def get_non_zerolength_bands(input_file)
  contents2 = Array.new
  begin
    extension = File.extname(input_file)
    if extension.casecmp(".h5").zero?
      contents = `'#{H5DUMP}' --contents '#{input_file}' | grep dataset`
      contents1 = contents.split(" dataset   ")

      contents1.each {  | band |
        if !band.nil? && !band.empty?
          # #remove links in the band names
          band1 = band.split("->")
          band2 = band1[0].strip
          data_arr = get_h5_data_for_band(input_file,band2)
          if !data_arr.nil? and !data_arr.empty?
            contents2.push(band2.strip)
          else
            puts "zerolength=#{band2}"
          end
        end
      }
    end
    #puts "contents2=#{contents2}"
    contents2
  rescue=> e
    puts "exception:" + e.to_s
    puts e.backtrace
    contents2
  end
end


# ################################################## h5 contains
# h5_output_contains_bands
#
```

```
# ####################################################
def output_contains_bands(output_file, input_bands)

  if input_bands.nil?
    true
  end
  output_bands = get_band_params(output_file)
  if output_bands.nil?
    false
  else
    compare_input_output_bands(output_bands, input_bands)
  end
end


# ################################################## h5 contains
# h5_output_contains_bands
#
# ####################################################
def compare_input_output_bands(input_bands,output_bands)

  if input_bands.nil?
    true
  elsif output_bands.nil?
    false
  else
    # #if input_bands.all? { |e| output_bands.include?(e)}
    missing_bands = input_bands - output_bands
    if missing_bands.empty?
      puts "input bands: #{input_bands.length},  output bands : #{output_bands.length} "
      extra_bands = output_bands - input_bands
      if !extra_bands.empty?
        puts "bands in output not in input! =#{extra_bands}"
      end
      true
    else
```

```
      #check is all input bands are in the output
      missingbands = Array.new
      input_bands.each{ | band |
        if !output_bands.include?(band)
          missingbands.push(band)
        end
      }
      if !missingbands.empty?
        @error_msg = "Missing Bands = #{missingbands}"
        false
      else
        true
      end
    end
  end
end


# ################################################## h5 contains
# h5_output_contains_bands - get_non_zerolength_bands
#
# ######################################################
def h5_output_contains_non_zerolength_bands(output_file, input_bands)
  if input_bands.nil?
    true
  end
  output_bands = get_non_zerolength_bands(output_file)
  if output_bands.nil?
    false
  else
    # #if input_bands.all? { |e| output_bands.include?(e)}
    missing_bands = input_bands - output_bands
    @error_msg = "Missing Bands = #{missing_bands}"
    if missing_bands.empty?
      puts "input bands: #{input_bands.length} equal to output bands : #{output_bands.length} "
      true
```

```
    else
      @error_msg = "Missing Bands = #{missing_bands}"
      false
    end
  end
end

# ################################################### h5 contains
# band
#
# ######################################################
def h5output_contains_band(output_file, band)
  output_bands = get_band_params(output_file)
  if output_bands.nil?
    false
  else
    output_bands.include?(band)
  end
end


# ##############################
#
# get bands from opendap ascii output
# ##############################
def opendap_ascii_contains_bands(ascii_file, input_bands)
  begin
    lines = IO.readlines(ascii_file)
    missing_bands = Array.new
    input_bands.each do |band|
      band1 = band.gsub("/","_")[1..-1]
      # #puts "band=#{band1}" #exclude leading underscore
      select_line = lines.select {|line| line.include?(band1)}
      if !select_line.nil? && !select_line.empty?
        # #puts "#{select_line}"
        filename = lines[0]
```

```
        # #puts "line 0 from ascii file = #{filename}"
        data_dump_arr = select_line[0].split(",")
        subset_parameter = data_dump_arr[0]
        # #puts "subset parameter in ascii file =
        # #{subset_parameter}"
        if subset_parameter.include?(band1)
          true
        else
          puts " ascii output does not contain band1 :#{band1}"
          missing_bands.push(band)
          false
        end
      else
        puts " ascii output does not contain band :#{band}"
        missing_bands.push(band)
        false
      end
      if missing_bands.empty?
        true
      else
        @error_msg = "#{ascii_file} has missing bands - #{missing_bands.to_s}"
        false
      end
    end
  rescue => e
    puts "exception:" + e.to_s
    puts e.backtrace
    false
  end
end

# ##############################
#
# get bands from opendap ascii output
# ##############################
def opendap_ascii_contains_band(ascii_file, band)
```

```
  begin
    # #puts band #puts "ascii file=#{ascii_file}"
    band1 = band.gsub("/","_")[1..-1]
    # #puts "band=#{band1}"
    lines = IO.readlines(ascii_file)
    # #exclude leading underscore
    select_line = lines.select {|line| line.include?(band1)}
    if !select_line.nil? && !select_line.empty?
      # #puts "#{select_line}"
      filename = lines[0]
      # #puts "line 0 from ascii file = #{filename}"
      data_dump_arr = select_line[0].split(",")
      subset_parameter = data_dump_arr[0]
      # #puts "subset parameter in ascii file =
      # #{subset_parameter}"
      if subset_parameter.include?(band1)
        true
      else
        puts " ascii output does not contain band1 :#{band1}"
        false
      end
    else
      puts " ascii output does not contain band :#{band}"
      false
    end
  rescue => e
    puts "exception:" + e.to_s
    puts e.backtrace
    false
  end
end

# ############################## get bands from opendap netcdf
#
# ##############################
def opendap_netcdf3_contains_bands(nc_file, input_bands)
```

```
begin
  puts "nc file=#{nc_file}"
  missing_bands = Array.new
  input_bands.each do |band|
    if band[0] == '/'
      band = band[1..-1]
    end
    puts "band=#{band}"
    band1 = band.gsub("/","_")
    puts "band1=#{band1}"
    data = `#{NC3DUMP} #{nc_file} | grep '#{band1} '`
    if !data.nil? and !data.empty?
      data_dump_arr = data.split("=")
      subset_parameter = data_dump_arr[0]
      puts "subset parameter in nc file = #{subset_parameter}"
      if subset_parameter.include?(band1)
        true
      else
        puts " nc3 output does not contain parameter - #{band}"
        missing_bands.push(band)
        false
      end
    else
      puts " nc3 output does not contain parameter - #{band}"
      missing_bands.push(band)
      false
    end
    if missing_bands.empty?
      true
    else
      @error_msg = "#{nc_file} has missing bands - #{missing_bands.to_s}"
      false
    end
  end
rescue => e
  puts "exception:" + e.to_s
```

```
    puts e.backtrace
    false
  end
end

# ############################## get bands from opendap netcdf
#
# ##############################
def opendap_netcdf3_contains_band(nc_file, band)
  begin
    puts "nc file=#{nc_file}"
    if band[0] == '/'
      band = band[1..-1]
    end
    puts "band=#{band}"
    band1 = band.gsub("/","_")
    puts "band1=#{band1}"
    data = `#{NC3DUMP} #{nc_file} | grep '#{band1} '`
    if !data.nil? and !data.empty?
      data_dump_arr = data.split("=")
      subset_parameter = data_dump_arr[0]
      puts "subset parameter in nc file = #{subset_parameter}"
      if subset_parameter.include?(band1)
        true
      else
        puts " nc3 output does not contain parameter - #{band}"
        false
      end
    else
      puts " nc3 output does not contain parameter - #{band}"
      false
    end
  rescue => e
    puts "exception:" + e.to_s
    puts e.backtrace
    false
```

```
  end
end

#
# ################################################################
# data fields functions
#
#
# ################################################################
# TODO ##################################################################
def get_hdf_data_for_band(hdf_file, band1)
  begin
    # #remove group name
    band_arr = band1.split('/')
    if !band_arr.nil?
      band = band1[-1]
      data1 = `'#{H4DUMP}' -v '#{band}'  '#{hdf_file}' | grep -A20 'data:' | grep -A10 '#{band}'`
      if !data1.nil?
        # #data_dump = data1.split("data:") #if
        # !data_dump.nil?
        data_dump_arr = data1.split("\n")
        if !data_dump_arr.nil?
          # #data_dump_arr = data_dump_str
          # #data_dump_arr.split("\n") #skip the lines
          # till DATA { #skip_index =
          # data_dump_arr.index("DATA {") #skip_index
          # = data_dump_arr.find_index{|x|
          # x.match(/DATA /)}
          data = data_dump_arr[1..11]
          data10 = remove_space_comma(data)
          # #puts "band=#{subset_parameter},  input
          # file 10 fields
          # = #{data10}"
          data10
        else
          puts "h4dump returned#{subset_parameter} does not contain parameter - #{band}"
```

```
          nil
        end
      else
        puts "h4dump returned nil- data does not contain parameter - #{band}"
        nil
      end
    else
      puts "h4dump returned nil- data does not contain parameter - #{band1}"
    end
  rescue=> e
    puts "exception:" + e.to_s
    puts e.backtrace
    nil
  end
end


# ######################################################## h5dump
# -d /South_Polar_Projection/cell_tb_v_fore -y -w 1 -A 0
# filename process first 10 fields
# ########################################################
def get_h5_data_for_band(h5_file, band)
  begin
    data_dump = `'#{H5DUMP}' -d '#{band}' -y -w 1 -A 0  '#{h5_file}' `
    # #data_dump_arr = data_dump.split("\n ")
    data_dump_arr = data_dump.split("\n")
    # #puts data_dump_arr #data_arr[0] has the band
    # #subset_parameter = data_dump_arr[0] #filename is
    # data_dump_arr[0]
    subset_parameter = data_dump_arr[1]
    puts "subset parameter: #{subset_parameter}"
    if !subset_parameter.nil?
      if subset_parameter.include?(band)
        # #skip the lines till DATA { #skip_index =
        # data_arr[4..-5] has the data
        # data_dump_arr.index("DATA {")
```

```
        skip_index = data_dump_arr.find_index{|x| x.match(/DATA /)}
        # #end of arr = -5
        data_dump_real = data_dump_arr[skip_index+1..-5]
        if data_dump_real.nil? or data_dump_real.empty?
          puts  "The file has an empty dataset for the requested band parameter #{band}"
          data_dump_real
        else
          #data = data_dump_real[skip_index+1..skip_index+11]
          data = data_dump_real[0..11]
          data10 = remove_space_comma(data)
          puts "band=#{subset_parameter},  #{data10}"
          # file 10 fields
          # = #{data10}"
          data10
        end
      else
        puts "h5dump returned#{subset_parameter} does not contain parameter - #{band}"
        nil
      end
    else
      puts "h5dump returned nil- data does not contain parameter - #{band}"
      nil
    end
  rescue=> e
    puts "exception:" + e.to_s
    puts e.backtrace
    nil
  end
end

# #to get data within a lat or lon or time range #should return
# the indexes and values
def get_h5_data_range_array_for_band(h5_file, band, min,max)
  begin
    # #hash array with index as key
    data_range = Array.new
```

```
data_dump = `'#{H5DUMP}' -d '#{band}' -y -w 1 -A 0 -m '%.6f' '#{h5_file}' `
# #data_dump_arr = data_dump.split("\n ")
data_dump_arr = data_dump.split("\n")
# #puts data_dump_arr #data_arr[0] has the band
# #subset_parameter = data_dump_arr[0] #filename is
# data_dump_arr[0]
subset_parameter = data_dump_arr[1]
# #puts "subset parameter: #{subset_parameter}"
if !subset_parameter.nil?
  if subset_parameter.include?(band)
    # #skip the lines till DATA { #skip_index =
    # data_arr[4..-5] has the data
    # data_dump_arr.index("DATA {")
    skip_index = data_dump_arr.find_index{|x| x.match(/DATA /)}
    # #end of arr = -5
    data_dump_real = data_dump_arr[skip_index+1..-5]
    if data_dump_real.nil? or data_dump_real.empty?
      puts  "The file has an empty dataset for the requested band parameter #{band}"
      data_dump_real
    else
      #data = data_dump_real[skip_index+1..skip_index+11]
      data = data_dump_real[0..11]
      data10 = remove_space_comma(data)
      # #puts "band=#{subset_parameter},  input
      # file 10 fields
      # = #{data10}"
      data10
    end
  else
    puts "h5dump returned#{subset_parameter} does not contain parameter - #{band}"
    nil
  end
else
  puts "h5dump returned nil- data does not contain parameter - #{band}"
  nil
end
```

```
  rescue=> e
    puts "exception:" + e.to_s
    puts e.backtrace
    nil
  end
end

def get_h5_data_len_for_band(h5_file, band)
  begin
    # #hash array with index as key
    data_len = 0
    data_range = Array.new
    data_dump = `'#{H5DUMP}' -d '#{band}' -y -w 1 -A 0  '#{h5_file}' `
    # #data_dump_arr = data_dump.split("\n ")
    data_dump_arr = data_dump.split("\n")
    # #puts data_dump_arr #data_arr[0] has the band
    # #subset_parameter = data_dump_arr[0] #filename is
    # data_dump_arr[0]
    subset_parameter = data_dump_arr[1]
    # #puts "subset parameter: #{subset_parameter}"
    if !subset_parameter.nil?
      if subset_parameter.include?(band)
        # #skip the lines till DATA { #skip_index =
        # data_arr[4..-5] has the data
        # data_dump_arr.index("DATA {")
        skip_index = data_dump_arr.find_index{|x| x.match(/DATA /)}
        # #end of arr = -5
        data_dump_real = data_dump_arr[skip_index+1..-5]
        if data_dump_real.nil? or data_dump_real.empty?
          puts  "The file has an empty dataset for the requested band parameter #{band}"
          data_len
        else
          # #data =
          # data_dump_real[skip_index+1..skip_index+11]
          data = remove_space_comma_linefeed(data_dump_real)
          if !data.nil?
```

```
            data_len = data.length
            data_len
          else
            puts "null data"
            data_len
          end
        end
      else
        puts "h5dump returned#{subset_parameter} does not contain parameter - #{band}"
        data_len
      end
    else
      puts "h5dump returned nil- data does not contain parameter - #{band}"
      data_len
    end
  rescue=> e
    puts "exception:" + e.to_s
    puts e.backtrace
    data_len
  end
end

def validate_h5_data_range_for_band(h5_file, band, min,max)
  begin
    # #puts "min=#{min}, max=#{max}"

    @output_arr = Array.new
    data_dump = `'#{H5DUMP}' -d '#{band}' -y -w 1 -A 0 -m '%.6f' '#{h5_file}' `
    data_dump_arr = data_dump.split("\n")
    subset_parameter = data_dump_arr[1]

    if !subset_parameter.nil?
      if subset_parameter.include?(band)
        # #skip the lines till DATA { #skip_index =
        # data_arr[4..-5] has the data
        # data_dump_arr.index("DATA {")
```

```
    skip_index = data_dump_arr.find_index{|x| x.match(/DATA /)}
    # #end of arr = -5
    data_dump_real = data_dump_arr[skip_index+1..-5]
    if data_dump_real.nil? or data_dump_real.empty?
      @error_msg =   "The file has an empty dataset for the requested band parameter #{band}"
      puts @error_msg
      true
    else
      # #data =
      # data_dump_real[skip_index+1..skip_index+11]
      data = remove_space_comma_linefeed(data_dump_real)
      if !data.nil?
        data1 = data.collect {|x | x.to_f}
        data2 = data1.select { |x| x > max or x < min}
        if data2.nil? or data2.empty?
          @output_arr = data1
          true
        else
          @output_arr = data2[0..10]
          @error_msg =  "band:#{band}, data: #{data2[0..10]} - values outside the range"
          puts @error_msg
          false
        end
      else
        @error_msg =  "band:#{band} - null data"
        puts @error_msg
        true
      end
    end
  else
    @error_msg =  "h5dump returned #{subset_parameter} does not contain parameter - #{band}"
    puts @error_msg
    puts ("data: #{data_dump_arr[0..5]}")
    true
  end
else
```

```
      @error_msg =  "h5dump returned #{subset_parameter} does not contain parameter - #{band}"
      puts @error_msg
      puts ("data: #{data_dump_arr[0..5]}")
      true
    end
  rescue=> e
    @error_msg =  "exception:" + e.to_s
    puts @error_msg
    puts e.backtrace
    false
  end
end


def get_icesat_data_range_for_band(h5_file, band, min,max)
  # #puts "min=#{min}, max=#{max}"
  file_min = nil
  file_max = nil
  status = false
  begin
    @output_arr = Array.new
    data_dump = `'#{H5DUMP}' -d '#{band}' -y -w 1 -A 0 -m '%.6f' '#{h5_file}' `
    data_dump_arr = data_dump.split("\n")
    subset_parameter = data_dump_arr[1]

    if !subset_parameter.nil?
      if subset_parameter.include?(band)
        # #skip the lines till DATA { #skip_index =
        # data_arr[4..-5] has the data
        # data_dump_arr.index("DATA {")
        skip_index = data_dump_arr.find_index{|x| x.match(/DATA /)}
        # #end of arr = -5
        data_dump_real = data_dump_arr[skip_index+1..-5]
        if data_dump_real.nil? or data_dump_real.empty?
          @error_msg =   "The file has an empty dataset for the requested band parameter #{band}"
          puts @error_msg
```

```
  status = true
else
  # #data =
  # data_dump_real[skip_index+1..skip_index+11]
  data = remove_space_comma_linefeed(data_dump_real)
  if !data.nil?
    data1 = data.collect {|x | x.to_f}
    file_min = data1[0].to_f
    file_max = data1[-1].to_f
    if file_min >= min && file_max <= max
      status = true
    else
      status = false
    end
    puts "file_min:#{file_min}, file_max:#{file_max}"
    data2 = data1.select { |x| x > max or x < min}
    if data2.nil? or data2.empty?
      @output_arr = data1
      status = true
    else
      @output_arr = data2[0..10]
      @error_msg =  "band:#{band}, data: #{data2[0..10]} - values outside the range"
      puts @error_msg
      # #check if the last element is
      # still within range
      if data1[-1].to_f <= max
        # #this is considered ok for
        # icesat since some data
        # elements maybe outside range
        puts "min max data is valid"
        status = true
      end
    end
  else
    @error_msg =  "band:#{band} - null data"
    puts @error_msg
```

```
            status =  true
          end
        end
      else
        @error_msg =  "h5dump returned #{subset_parameter} does not contain parameter - #{band}"
        puts @error_msg
        puts ("data: #{data_dump_arr[0..5]}")
        status = true
      end
    else
      @error_msg =  "h5dump returned #{subset_parameter} does not contain parameter - #{band}"
      puts @error_msg
      puts ("data: #{data_dump_arr[0..5]}")
      status = true
    end

  rescue=> e
    @error_msg =  "exception:" + e.to_s
    puts @error_msg
    puts e.backtrace
    status = false
  end
  return status, file_min, file_max
end

# ##############################
#
# process first 10 fields ##############################
def get_opendap_ascii_data(ascii_file, band)
  begin
    puts band
    puts "ascii file=#{ascii_file}"
    band1 = band.gsub("/","_")[1..-1]
    puts "band=#{band1}"
    lines = Array.new
    lines = IO.readlines(ascii_file)
```

```
# #exclude leading underscore
select_line = lines.select {|line| line.include?(band1)}
if !select_line.nil? && !select_line.empty?
  # #puts "#{select_line}"
  if select_line.count == 0
    # #puts "line 0 from ascii file = #{filename}"
    data_dump_arr = select_line[0].split(",")
    subset_parameter = data_dump_arr[0]
    data = data_dump_arr[1..11]
    data10 = remove_space_comma(data)
    #puts "subset parameter in ascii file = #{subset_parameter}"
  else #several similar bands or multi dimensional band
    data10 = nil
    data110 = nil
    data210 = nil
    select_line.each do | line|
      data_dump_arr = line.split(',')
      if data_dump_arr[0].eql?(band1)
        subset_parameter = data_dump_arr[0]
        data = data_dump_arr[1..11]
        data10 = remove_space_comma(data)
      elsif data_dump_arr[0].eql?("#{band1}[0]")
        subset_parameter = data_dump_arr[0]
        data = data_dump_arr[1..-1]
        data10 = remove_space_comma_linefeed(data)
      elsif data_dump_arr[0].eql?("#{band1}[1]")
        data = data + data_dump_arr[1..-1]
        data10 = remove_space_comma_linefeed(data)
      elsif data_dump_arr[0].eql?("#{band1}[2]")
        data = data + data_dump_arr[1..-1]
        data10 = remove_space_comma_linefeed(data)
      elsif data_dump_arr[0].eql?("#{band1}[3]")
        data = data + data_dump_arr[1..-1]
        data10 = remove_space_comma_linefeed(data)
      elsif data_dump_arr[0].eql?("#{band1}_Bands_01[0]")
        subset_parameter = data_dump_arr[0]
```

```
        data = data_dump_arr[1..-1]
        data10 = remove_space_comma_linefeed(data)
      elsif data_dump_arr[0].eql?("#{band1}_Bands_02[0]")
        data1 = data_dump_arr[1..-1]
        data110 = remove_space_comma_linefeed(data)
      elsif data_dump_arr[0].eql?("#{band1}_Bands_03[0]")
        data2 = data_dump_arr[1..-1]
        data210 = remove_space_comma_linefeed(data)
      end
    end
    if !data10.nil? and data10.count > 10
      data10 = data10[0..10]
      # #break
    end
    if !data110.nil? and data110.count > 10
      data110 = data110[0..10]
      # #break
    end
    if !data210.nil? and data210.count > 10
      data210 = data210[0..10]
      # #break
    end
    if(!data10.nil? and !data110.nil? and !data210.nil?)
      datafinal = Array.new
      datafinal << data10[0] << data110[0] <<data210[0]
      datafinal << data10[1] << data110[1] <<data210[1]
      datafinal << data10[2] << data110[2] <<data210[2]
      datafinal << data10[3] << data110[3] <<data210[3]
      data10 = datafinal[0..10]
    end
  end
  data10
else
  puts " ascii output does not contain parameter: #{band1}"
  nil
end
```

```
  rescue => e
    puts "exception:" + e.to_s
    puts e.backtrace
    nil
  end
end

# ############################## process first 10 fields of nc
# output ##############################
def get_opendap_netcdf3_data(nc_file, band)
  begin
    puts "nc file=#{nc_file}"
    if band[0] == '/'
      band = band[1..-1]
    end
    puts "band=#{band}"
    band1 = band.gsub("/","_")
    puts "band1=#{band1}"

    data = `#{NC3DUMP} #{nc_file} | grep -A12 '#{band1} ='`
    if !data.nil? and !data.empty?
      data_dump_arr = data.split("=")
      subset_parameter = data_dump_arr[0]
      puts "subset parameter in nc file = #{subset_parameter}"
      if subset_parameter.include?(band1)
        data = data_dump_arr[1..11]
        data10 = remove_space_comma(data)
      elsif data_dump_arr[0].eql?("#{band1}_Bands_01[0]")
        subset_parameter = data_dump_arr[0]
        data = data_dump_arr[1..-1]
        data10 = remove_space_comma_linefeed(data)
      elsif data_dump_arr[0].eql?("#{band1}_Bands_02[0]")
        data1 = data_dump_arr[1..-1]
        data110 = remove_space_comma_linefeed(data)
      elsif data_dump_arr[0].eql?("#{band1}_Bands_03[0]")
```

```
        data2 = data_dump_arr[1..-1]
        data210 = remove_space_comma_linefeed(data)
      else
        puts " nc3 output does not contain parameter - #{band}"
        nil
      end
    else
      puts " nc3 output does not contain parameter - #{band}"
      nil
    end
  rescue => e
    puts "exception:" + e.to_s
    puts e.backtrace
    nil
  end
end

def remove_space_comma(data)
  if !data.nil?
    data1 = data.collect {|x | x.strip}
    data2 = data1.collect { |x| x.gsub(",","")}
  end
end

def remove_space_comma_linefeed(data)
  if !data.nil?
    data1 = data.collect {|x | x.strip}
    data2 = data1.collect { |x| x.gsub(",","")}
    data3 = data2.collect { |x| x.gsub("\n","")}
  end
end
def remove_space_comma_paranthesis(data)
  if !data.nil?
    data1 = data.collect {|x | x.strip}
    data2 = data1.collect { |x| x.gsub(",","")}
    data3 = data2.collect { |x| x.gsub("}","")}
```

```
  end
end

def compare_data_elements(input, output)
  if input.length == output.length
    result = true
    input.each_index do | i |
      in_val = input[i]
      out_val = output[i]
      if (in_val == out_val)
        result = true
      elsif (in_val.is_a?(Numeric)  and !approx?(in_val.to_f, out_val.to_f, 0.01))
        # #puts ("expected = #{in_val.to_f}, result=
        # #{out_val.to_f}")
        result = false
        break
      else
        result = true
      end
    end
    result
  else
    false
  end
end

def compare_attribute_elements(input, output)
  #if input.length < output1.length
  #  output = output1[(output1.length-input.length)..-1]
  #else
  #output = output1
  #end
  result = false
  if input.length <= output.length
    # the output may have additional attributes that was
    # added as part of the processing
```

```
          j=0
          input.each_index do | i|
            puts "#{input[i]}"
            puts "#{output[j]}"
            in_val = input[i].strip
            # #puts ("in_val=#{in_val}") #j =
            # output.index(in_val) #puts ("j=#{j}") #puts
            # ("output=#{output[i]}")
            out_val = output[j].strip
            if ((in_val.is_a?(String) and (in_val != out_val)) and ((out_val.include?("Processing") ||
out_val.include?("Subsetting"))))
              #skip to the next attribute - SMAP and ICESAT add additional attribute
              j=j+1
              out_val = output[j].strip
              result = compare_attribute(in_val, out_val)
            else
              result = compare_attribute(in_val, out_val)
            end
            if(result == false)
              break
            end
            j=j+1

          end
          # #puts "result=#{result}"
          result
        else
          # #input_arr = input.flatten #output_arr =
          # output.flatten
          puts "num attributes dont match. input=#{input.length},output=#{output.length}. missing attributes =
#{input - output}"
          @error_msg = "num attributes dont match.  input=#{input.length},ouput=#{output.length}. missing
attributes = #{input - output}"
          result = false
          # #puts "result=#{result}"
          result
```

```
  end

end

def compare_attribute(in_val, out_val)
  if (in_val == out_val)
    result = true
    puts "result=#{result}"
  elsif (in_val.is_a?(String) and (in_val != out_val))
    puts("string...expected=#{in_val}, result=#{out_val} " )
    @error_msg = "attributes dont match. expected=#{in_val}, result=#{out_val} "
    result = false
    puts "result=#{result}"
    result

  elsif (in_val.is_a?(Numeric)  and !approx?(in_val.to_f, out_val.to_f, 0.01))
    puts ("numeric...attributes dont match. expected = #{in_val.to_f}, result= {out_val.to_f}")
    @error_msg= "attributes dont match. expected = #{in_val.to_f}, result= {out_val.to_f}"
    result = false
    puts "result=#{result}"
    result
  end
end


def validate_ascii_output_data(input_file, output_file, band)
  extension = File.extname(input_file)
  if extension.casecmp(".h5") .zero?
    validate_h5_ascii_output_data(input_file, output_file, band)
  elsif extension.eql? (".hdf")
    validate_hdf_ascii_output_data(input_file, output_file, band)
  end
end

def validate_hdf_ascii_output_data(test_hdf, output_file, band)
  @input_arr = get_hdf_data_for_band(test_df, band)
```

```
  @output_arr = get_opendap_ascii_data(output_file, band)
  # #puts ("band =#{band}, expected=#{input_arr},
  # result=#{output_arr}")
  if @input_arr.nil? or @output_arr.nil?
    return false
  elsif @input_arr.eql?(@output_arr)
    return true
  elsif compare_data_elements(@input_arr, @output_arr) == false
    return false
  else
    return true
  end
end

def validate_h5_ascii_output_data(test_h5, output_file, band)
  @input_arr = get_h5_data_for_band(test_h5, band)
  @output_arr = get_opendap_ascii_data(output_file, band)
  # #puts ("band =#{band}, expected=#{input_arr},
  # result=#{output_arr}")
  if @input_arr.nil? or @output_arr.nil?
    return false
  elsif @input_arr.eql?(@output_arr)
    return true
  elsif compare_data_elements(@input_arr, @output_arr) == false
    return false
  else
    return true
  end
end

def validate_h5_netcdf3_output_data(test_h5, output_file, band)
  @input_arr = get_h5_data_for_band(test_h5, band)
  @output_arr = get_opendap_netcdf3_data(output_file, band)
  if @input_arr.nil? or @output_arr.nil?
    return false
  elsif @input_arr.eql?(@output_arr)
```

```
    return true
  elsif compare_data_elements(input_arr, output_arr) == true
    return true
  else
    return false
  end
end

def validate_hdf_netcdf3_output_data(test_hdf, output_file, band)
  @input_arr = get_hdf_data_for_band(test_hdf, band)
  @output_arr = get_opendap_netcdf3_data(output_file, band)
  if @input_arr.nil? or @output_arr.nil?
    return false
  elsif @input_arr.eql?(@output_arr)
    return true
  elsif compare_data_elements(input_arr, output_arr) == true
    return true
  else
    return false
  end
end

def validate_output_data (input_file, output_file, band)
  extension = File.extname(output_file)
  # #extension = File.extname(input_file)
  if extension.casecmp(".h5").zero? or extension.eql?(".he5")
    validate_h5_output_data(input_file, output_file, band)
  elsif extension.eql?(".hdf")
    validate_hdf_output_data(input_file, output_file, band)
  elsif extension.eql?(".nc")
    if netcdf_filetype(output_file).eql?("classic") && File.extname(input_file).eql?("hdf")
      validate_hdf_opendap_netcdf3_data(input_file, output_file, band)
    elsif netcdf_filetype(output_file).eql?("classic") && File.extname(input_file).eql?("h5")
      validate_h5_opendap_netcdf3_data(nc_file, band)
    else
      validate_h5_output_data(input_file, output_file, band)
```

```
    end
  elsif extension.eql?(".ascii")
    if File.extname(input_file).eql?("hdf")
      get_hdf_opendap_ascii_data(output_file, band)
    elsif netcdf_filetype(output_file).eql?("classic") && File.extname(input_file).eql?("h5")
      get_h5_opendap_ascii_data(nc_file, band)
    else
      validate_h5_output_data(input_file, output_file, band)
    end
  else
    validate_hdf_output_data(input_file, output_file, band)
  end

end

def validate_h5_output_data(test_h5, output_file, band)
  extension = File.extname(output_file)
  if extension.casecmp(".h5").zero?
    @input_arr = get_h5_data_for_band(test_h5, band)
    @output_arr = get_h5_data_for_band(output_file, band)
  elsif extension.eql?(".hdf")
    @input_arr = get_hdf_data_for_band(test_h5, band)
    @output_arr = get_hdf_data_for_band(output_file, band)
  end
  if @input_arr.nil? or @output_arr.nil?
    return false
  elsif @input_arr.eql?(@output_arr)
    puts " input arr data: #{@input_arr.length} equal to output arr data : #{@output_arr.length} "
    return true
  elsif compare_data_elements(input_arr, output_arr) == true
    return true
  else
    puts "band=#{band}"
    puts "input=#{@input_arr}"
    puts "output=#{@output_arr}"
    return false
```

```
    end
  end

  def validate_hdf_output_data(test_hdf, output_file, band)
    @input_arr = get_hdf_data_for_band(test_hdf, band)
    @output_arr = get_hdf_data_for_band(output_file, band)
    if @input_arr.nil? or @output_arr.nil?
      return false
    elsif @input_arr.eql?(@output_arr)
      puts " input arr data: #{@input_arr.length} equal to output arr data : #{@output_arr.length} "
      return true
    elsif compare_data_elements(input_arr, output_arr) == true
      return true
    else
      puts "band=#{band}"
      puts "input=#{@input_arr}"
      puts "output=#{@output_arr}"
      return false
    end
  end

  def validate_h5_spatial_subset_data(test_h5, output_file, band, spatial_subset_hash)
    # #check the lat/lon bands first
    if band.include?("lat")
      status = validate_h5_data_range_for_band(output_file, band, spatial_subset_hash[:s],
spatial_subset_hash[:n])
    elsif band.include?("lon")
      status = validate_h5_data_range_for_band(output_file, band, spatial_subset_hash[:w],
spatial_subset_hash[:e])
    end
    return status

  end

  def validate_h5_temporal_subset_data(test_h5, output_file, band, start_delta_time, end_delta_time)
    # #adjust time for IceSat EPOCH to UTC #puts
```

```
  # "epoch=#{epoch}" #puts "start=
  # #{temporal_subset_hash[:START]}" #puts "end=
  # #{temporal_subset_hash[:END]}"

  # #start_delta_time = get_epoch_delta(epoch,
  # temporal_subset_hash[:START]) #end_delta_time =
  # get_epoch_delta(epoch, temporal_subset_hash[:END]) #if
  # band.include?("time")
  status = validate_h5_data_range_for_band(output_file, band, start_delta_time, end_delta_time)
  # #end
  return status

end

def validate_icesat_temporal_subset_data(test_h5, output_file, band, start_delta_time, end_delta_time)
  # #adjust time for IceSat EPOCH to UTC #puts
  # "epoch=#{epoch}" #puts "start=
  # #{temporal_subset_hash[:START]}" #puts "end=
  # #{temporal_subset_hash[:END]}"

  # #start_delta_time = get_epoch_delta(epoch,
  # temporal_subset_hash[:START]) #end_delta_time =
  # get_epoch_delta(epoch, temporal_subset_hash[:END]) #if
  # band.include?("time") #get the first and last subsetted
  # data #results1 = get_icesat_data_range_for_band(test_h5,
  # band, start_delta_time, end_delta_time)

  results2 = get_icesat_data_range_for_band(output_file, band, start_delta_time, end_delta_time)
  results2[0]
  #                    if results2[0] == true
  #                        #compare the min and max in both cases
  #                        if results1[1] == results2[1] && results1[2] == results2[2]
  #                           true
  #                        else
  #                            false
  #                        end
```

```
  #                    else
  #                       false
  #                    end

  # #end
end

# ################################ Attributes
# ##################################
# ##########################################
#
# dump of all attributes without the hierarchy information
#
def get_h5_attributes(input_file)
  begin
    puts input_file
    if !input_file.nil?
      extension = File.extname(input_file)
      if extension.casecmp(".h5").zero?
        attr_list = `'#{H5DUMP}' -A  '#{input_file}'  | grep ATTRIBUTE`
        # #attr_list = `'#{H5DUMP}'   '#{h5_file}'  |
        # grep ATTRIBUTE`
        attr_list_arr = attr_list.split("\n ")
      end
    else
      puts "nil input file"
      attr_list = Array.new
    end
  rescue=> e
    puts "exception:" + e.to_s
    puts e.backtrace
    attr_list = Array.new
  end
end

# ######################################## global attributes
```

```
# ################################ group attributes
# ################################

# #######################################
#
# attributes for each dataset
# #######################################
def get_h5_attributes_for_band(h5_file, band)
  begin
    extension = File.extname(h5_file)
    if extension.casecmp(".h5").zero?
      attr_dump = `'#{H5DUMP}' -d '#{band}'  '#{h5_file}'  | grep ATTRIBUTE`
      attr_dump_arr = attr_dump.split("\n ")
      # #puts data_dump_arr
      first_row = attr_dump_arr[0]
      if !first_row.nil?
        if !first_row.include?('h5dump error')
          attribute_arr = attr_dump_arr
        else
          puts "h5dump attributes returned #{first_row} does not contain parameter - #{band}"
          nil
        end
      else
        # #puts "h5dump attributes returned nil- does
        # not contain parameter - #{band}"
        nil
      end
    end
  rescue=> e
    puts "exception:" + e.to_s
    puts e.backtrace
    nil
  end
end

#
```

```
#
# ##############################
#
# process first 10 fields ##############################
def get_opendap_ascii_attributes(ascii_file)
  begin
    # #puts "ascii file=#{ascii_file}" #band1 =
    # band.gsub("/","_") #puts "band=#{band1}"
    lines = IO.readlines(ascii_file)
  rescue => e
    puts "exception:" + e.to_s
    puts e.backtrace
    nil
  end
end

def get_opendap_ascii_attributes_for_band(ascii_file, band)
  begin
    # #puts "ascii file=#{ascii_file}" #band1 =
    # band.gsub("/","_") #puts "band=#{band1}"
    lines = IO.readlines(ascii_file)
  rescue => e
    puts "exception:" + e.to_s
    puts e.backtrace
    nil
  end
end

# ############################## process first 10 fields of nc
# output ##############################
def get_opendap_netcdf_attributes(nc_file)
  #               begin
  #                    # #puts "nc file=#{nc_file}"
  #                    band1 = band.gsub("/","_")
  #                    # #puts "band=#{band1}"
  #                    lines = IO.readlines(ascii_file)
```

```
  #                     # #first line is filename
  #                     filename = lines[0]
  #                     # #puts "line 0 from ascii file = #{filename}"
  #                     data_dump_arr = lines[1].split(",")
  #                     subset_parameter = data_dump_arr[0]
  #                     # #puts "subset parameter in ascii file =
  #                     # #{subset_parameter}"
  #                     if subset_parameter.include?(band1)
  #                         data = data_dump_arr[1..11]
  #                         data10 = remove_space_comma(data)
  #                         puts "band=#{subset_parameter}, first 10 fields in ascii file = #{data10}"
  #                         data10
  #                     else
  #                         puts " ascii output does not contain parameter"
  #                         nil
  #                     end
  #                 rescue => e
  #                     puts "exception:" + e.to_s
  #                     puts e.backtrace
  #                     nil
  #                 end
end

def get_opendap_netcdf_attributes_for_band(nc_file, band)
  #                 begin
  #                     # #puts "nc file=#{nc_file}"
  #                     band1 = band.gsub("/","_")
  #                     # #puts "band=#{band1}"
  #                     lines = IO.readlines(ascii_file)
  #                     # #first line is filename
  #                     filename = lines[0]
  #                     # #puts "line 0 from ascii file = #{filename}"
  #                     data_dump_arr = lines[1].split(",")
  #                     subset_parameter = data_dump_arr[0]
  #                     # #puts "subset parameter in ascii file =
  #                     # #{subset_parameter}"
```

```
  #                     if subset_parameter.include?(band1)
  #                         data = data_dump_arr[1..11]
  #                         data10 = remove_space_comma(data)
  #                         puts "band=#{subset_parameter}, first 10 fields in ascii file = #{data10}"
  #                         data10
  #                     else
  #                         puts " ascii output does not contain parameter"
  #                         nil
  #                     end
  #                 rescue => e
  #                     puts "exception:" + e.to_s
  #                     puts e.backtrace
  #                     nil
  #                 end
end

def validate_ascii_output_attributes(test_h5, output_file)
  input_arr = get_h5_attributes(test_h5)

  output_arr = get_opendap_ascii_attributes(output_file)
  if input_arr.nil? or output_arr.nil?
    return nil
  else
    if input_arr.eql?(output_arr)
      return true
    else compare_attribute_elements(input_arr, output_arr)
    end
  end
end

def validate_netcdf3_output_attributes(test_h5, output_file)
  input_arr = get_h5_attributes(test_h5)
  output_arr = get_opendap_netcdf_attributes(output_file)
  if input_arr.nil? or output_arr.nil?
    return nil
  else
```

```
    if input_arr.eql?(output_arr)
      return true
    else compare_data_elements(input_arr, output_arr)
    end
  end
end

def validate_netcdf3_output_attributes_for_band(test_h5, output_file, band)
  input_arr = get_h5_attributes_for_band(test_h5)
  output_arr = get_opendap_netcdf_attributes_for_band(output_file, band)
  if input_arr.nil? or output_arr.nil?
    return nil
  else
    if input_arr.eql?(output_arr)
      return true
    else compare_data_elements(input_arr, output_arr)
    end
  end
end

def validate_ascii_output_attributes_for_band(test_h5, output_file, band)
  input_arr = get_h5_attributes_for_band(test_h5,band)

  output_arr = get_opendap_ascii_attributes_for_band(output_file, band)
  if input_arr.nil? or output_arr.nil?
    return nil
  else
    if input_arr.eql?(output_arr)
      return true
    else compare_attribute_elements(input_arr, output_arr)
    end
  end
end

def validate_h5_output_attributes(input_file, output_file)
  # #puts "input file #{input_file}" #puts "output file
```

```
  # #{output_file}"
  @input_arr = get_h5_attributes(input_file)
  @output_arr = get_h5_attributes(output_file)
  if @input_arr.nil? and @output_arr.nil?
    puts "no attributes for input #{input_file}"
    @error_msg = "no attributes for input #{output_file}"
    return true
  elsif !@input_arr.nil? and @output_arr.nil?
    puts "no attributes for output #{output_file}=#{@input_arr}"
    @error_msg = "no attributes for output #{output_file}=#{@input_arr}"
    return false
  else
    # #puts "input attributes: #{@input_arr[0]}" #puts
    # "output attributes: #{@output_arr[0]}"
    if @input_arr.eql?(@output_arr)
      # #puts "attributes input arr: #{@input_arr.length}
      # equal to output arr : #{@output_arr.length} "
      return true
    else
      result = compare_attribute_elements(@input_arr, @output_arr)
      puts "result from compare for output #{output_file}=#{result}"
      result
    end
  end
end

def validate_h5_output_attributes_for_band(test_h5, output_file,band)
  extension = File.extname(test_h5)
  if extension.casecmp(".h5").zero?
    @input_arr = get_h5_attributes_for_band(test_h5,band)

    @output_arr = get_h5_attributes_for_band(output_file,band)
    if @input_arr.nil? and @output_arr.nil?
      puts " no attributes- #{band}"
      return true
    elsif !@input_arr.nil? and @output_arr.nil?
```

```
      puts "missing attributes for band#{band}=#{@input_arr}"
      @error_msg = "missing attributesfor band#{band}=#{@input_arr}"
      return false
    else
      if input_arr.eql?(output_arr)
        # puts "attributes for band:#{band} - input arr:
        # #{@input_arr.length} equal to output arr :
        # #{@output_arr.length} " #@error_msg = "attributes
        # for band:#{band} - input arr: #{@input_arr.length}
        # equal to output arr : #{@output_arr.length} "
        return true
      else
        result = compare_attribute_elements(input_arr, output_arr)
        puts "result from compare attributes for band:#{band} =#{result}"
        result
      end
    end
  end
end


# ##################################################################
#
# File copy methods
#
# ###################################################################

def copy_with_path(src, dest_dir)
  # #FileUtils.mkdir_p(File.dirname(dest))
  FileUtils.mkdir_p(dest_dir)
  FileUtils.cp(src, dest_dir)
end

def filesizes_comparable?(input_file, output_file, tolerance)
  @error_msg = ""
  insize = File.size(input_file).to_f
```

```
  outsize = File.size(output_file).to_f
  if insize > 0 and outsize > 0
    # #output size <= input size upto the percent of tolerance
    # #outsize/insize.should be greater than  0.9
    ratio = (outsize/insize).round(1)
    puts "tolerance=#{tolerance}, ratio=#{ratio}"
    # #if (ratio > tolerance) && (ratio <= 1.0)
    if (ratio < tolerance) or (ratio = tolerance)
      @error_msg = "#{input_file}=#{insize}, #{output_file} = #{outsize}, ratio=#{ratio}"
      true
    else
      @error_msg = "#{input_file}=#{insize}, #{output_file} = #{outsize}, ratio=#{ratio}"
      false
    end
  else
    @error_msg = "#{input_file}=#{insize}, #{output_file} = #{outsize}"
    false
  end
end



#
#            def validate_time(data, epoch, min,max)
#                        data1 = data.collect {|x | convert_to_utc(epoch,x.to_f)}
#                                data2 = data1.select { |x| x > max or x < min}
#                                if data2.nil? or data2.empty?
#                                    @output_arr = data1
#                                    true
#                                else
#                                    @output_arr = data2[0..10]
#                                    @error_msg =  "#{data2[0..10]} - values outside the range"
#                                    puts @error_msg
#                                    false
#                                end
#            end
```

```
      #
      #                #format string  "%Y-%m-%dT%H:%M:%S.%N"
      #             def convert_to_utc(epoch, delta_time)
      #                    datetime_format =  "%Y-%m-%dT%H:%M:%S.%N"
      #                     epoch_time = DateTime.strptime(epoch,datetime_format)
      #                     utc_time = epoch_time + Rational(delta_time,86400)
      #            end

      def get_epoch_delta(epoch, date_time_str)

        datetime_format =  "%Y-%m-%dT%H:%M:%S.%N"
        date_time_epoch = DateTime.strptime(date_time_str, datetime_format )
        epoch_time = DateTime.strptime(epoch,datetime_format)
        delta = ((date_time_epoch - epoch_time).to_f * 24 * 3600)
        delta1 = delta.round(3)
        puts "delta=#{delta1}"
        delta1
      end



    end # class Utils
  end #module da
end #module Sdps
Current Test Script: heg.rb

module Sdps
   module Da
      require 'nokogiri'
      require 'sdps/da/daxml'
      require 'sdps/da/dadb'
      require 'sdps/da/dautil'
      require 'sdps/da/service'

      class Sdps::Da::Heg< Sdps::Da::Service
        attr_reader :heg_service_request_url, :heg_service_response
```

```
        def initialize(mode)
          super(mode, "HegService")
           @host  ="f5dpl01:"
           if mode == "OPS" or mode == "ops"
               @service_url = "http://#{@host}#{@port}/#{@service}"
               @request_url = "http://#{@host}#{@port}/#{@service}"
            else
              @service_url = "http://#{@host}#{@port}/#{@service}_#{@mode}"
              @request_url = "http://#{@host}#{@port}/#{@service}_#{@mode}"
           end
        end

             ##########################################################################
            #
            #
            ##########################################################################
        def submit_sync_request(esdt, granuleid, subagent_id, format:"", projection:"", projection_param:"",
band_parameter:"" , bbox:"" , interpolation:"" , include_meta:"N")
                begin
                    file_ids = "#{granuleid}"
                    @heg_request = Sdps::Da::Heg::Request.new(@request_url, file_ids, subagent_id, format,
projection, projection_param, band_parameter , bbox , interpolation , include_meta)

                   #add IDENTIFIER and DEBUG options
                    identifier = Time.now.strftime("autotest_%Y_%m_%d_%H_%M_%9N")
                   @heg_service_request_url = "#{@heg_request.heg_request_url}&IDENTIFIER=#{identifier}"
                    #debug option


                   @heg_service_response = `curl -s -XGET '#{@heg_service_request_url}'`
                rescue=>e
                    puts e.backtrace
                    e.to_s
                end
        end
```

```
    def parse_response(xml_buff)
       begin
         #puts xml_buff
         @heg_xml = Sdps::Da::DaXml.new(xml_buff)
         if @heg_xml.nil?
           response = Sdps::Da::Heg::Response.new(RESPONSE_FILE_NOT_FOUND, "", xml_buff)
         elsif @heg_xml.contains?("/Exception")
           status = @heg_xml.get_value('/Exception/Code')
           message = @heg_xml.get_value('/Exception')[0..198]
           response = Sdps::Da::Heg::Response.new(status, "FAILURE",@heg_xml)
         elsif @heg_xml.contains?("/agentResponse")
           urls = @heg_xml.get_value('//downloadUrls')
           urls = urls.strip
           if !urls.nil? and  !urls.empty?
              response = Sdps::Da::Heg::Response.new(status, "SUCCESS", urls)
           else
              response = Sdps::Da::Heg::Response.new(status, "FAILURE", "no download urls")
           end
         else
            response = Sdps::Da::Heg::Response.new(RESPONSE_FILE_PARSE_ERROR, "FAILURE", @heg_xml)
         end
       rescue=>e
         puts e.backtrace
         response = Sdps::Da::Heg::Response.new(RESPONSE_FILE_PARSE_EXCEPTION, "FAILURE", e.to_s)
       end
       response
  end # parse_response

  class Sdps::Da::Heg::Response < Sdps::Da::Service::Response

  end

class Sdps::Da::Heg::Request < Sdps::Da::Service::Request
     attr_reader :heg_request_url
      def initialize(req_url, file_ids, subagent_id, format, projection, projection_param, band_parameter ,
```

```
bbox , interpolation , include_meta)
               super(subagent_id, format, projection, projection_param, band_parameter , bbox , interpolation ,
include_meta)
                    #puts "#{@command_url}"
                   request_str = "#{req_url}"
                   # request_str = "#{request_str}?FILE_IDS=#{file_ids}"
                     request_str = "#{request_str}?FILE_URLS=#{@test_hdf}"
                   @heg_request_url ="#{request_str}#{@request_url}"
                   #puts "#{@heg_request_url}"
          end
     end

     end # class Heg

   end # module Da
end # module Sdps
Current Test Script: dadb.rb

module Sdps
  module Da
    class Sdps::Da::DaDb

      def initialize(dbConn, current_mode)
        @conn = dbConn
        @mode = current_mode
      end

      def do_query(sql, *params)
        @conn.exec_params(sql, params)
      end


##################################################################################################
################
      # Returns maximum granuleid for each collection in the mode as a hash(esdt,granuleid)
      # Returns nil if there is a failure
```

```
##################################################################################################
################
      GET_MAX_GRANULEID_COLLECTION_SQL = "select esdt(shortname,versionid) as esdt,MAX(granuleid) as granuleid
from amgranule group by shortname, versionid"
      def get_max_granuleid_collection()
         rs = @conn.exec(GET_MAX_GRANULEID_COLLECTION_SQL)
         if rs != nil and rs.count > 0
             granuleid_hash = Hash.new
             rs.each do |row|
               granuleid = row['granuleid']
               esdt = row['esdt']
               granuleid_hash[esdt] = granuleid
             end
             granuleid_hash
         else
           #puts "no rows"
           nil
         end
      end


##################################################################################################
################
      # Returns maximum granuleid hdf for each collection in the mode as a hash(esdt,granuleid)
      # Returns nil if there is a failure

##################################################################################################
################
      GET_MAX_GRANULEID_HDF_COLLECTION_SQL = "select esdt(shortname,versionid) as esdt, MAX(granuleid) as
granuleid, getdatafile(MAX(granuleid)) as filename from amgranule where shortname not in ('QA', 'PH', 'DAP')
group by shortname, versionid "
      def get_max_granuleid_hdfs_collection()
         rs = @conn.exec(GET_MAX_GRANULEID_HDF_COLLECTION_SQL)
         if rs != nil and rs.count > 0
             granuleid_hash = Hash.new
```

```
            rs.each do |row|
              granuleid = row['granuleid']
              esdt = row['esdt']
              geoid = "#{esdt}:#{granuleid}"
              granuleid_hash[geoid] = row['filename']
              #puts geoid
              #puts row['filename']
            end
            granuleid_hash
         else
           #puts "no rows"
            nil
         end
      end



##################################################################################################
#############
      #
      #

##################################################################################################
#############
      GET_DA_FORMATS_SQL = "select name, formatid from amdaformat"
      def get_da_formats()
         rs = @conn.exec(GET_DA_FORMATS_SQL)
         if rs != nil and rs.count > 0
             formats = Hash.new
             rs.each do |row|
               format = row['name']
               formatid = row['formatid']
               formats[format] = formatid
             end
             formats
         else
```

```
          #puts "no rows"
          nil
       end
    end


##########################################################################################
#############
    #
    #

##########################################################################################
#############
    GET_DA_SERVICES_SQL = "select name, serviceid from amdaservice"
    def get_da_services()
       rs = @conn.exec(GET_DA_SERVICES_SQL)
       if rs != nil and rs.count > 0
           services = Hash.new
           rs.each do |row|
             service =  row['name']
             serviceid =  row['serviceid']
             services[service] = serviceid
           end
           services
       else
          #puts "no rows"
          nil
       end
    end


##########################################################################################
#############
    #
    #
```

```
##################################################################################################
#############
      GET_DA_SERVICEID_SQL = "select serviceid from amdaservice where name = $1"
      def get_serviceid(service_name)
          serviceid = nil
          rs = do_query(GET_DA_SERVICEID_SQL, service_name)
          if rs != nil and rs.count > 0
              rs.each do |row|
                  serviceid = row['serviceid']
              end
          end
          serviceid
      end


##################################################################################################
#############
      #
      #

##################################################################################################
#############
      GET_DA_SERVICE_FORMATS_SQL = "select name, formatid from amdaformat where formatid in (select formatid
from amdaformatsrvxref  where serviceid = $1)"
      def get_da_service_formats(serviceid)
         rs = do_query(GET_DA_SERVICE_FORMATS_SQL, serviceid)
         if rs != nil and rs.count > 0
             service_formats = Hash.new
             rs.each do |row|
               formatid =  row['formatid']
               format = row['name']
               service_formats[format] = formatid
             end
             service_formats
         else
           #puts "no rows"
```

```
          nil
        end
      end

##################################################################################################
#############
      #
      #

##################################################################################################
#############
      GET_HEG_SERVICE_FORMATS_SQL = "select f.name, f.description from amdaformat f join amdaformatsrvxref x on
f.formatid = x.formatid join amdaservice s on x.serviceid = s.serviceid where s.name = 'HEG'";

      def get_heg_service_formats()
        rs = do_query(GET_HEG_SERVICE_FORMATS_SQL)
        if rs != nil and rs.count > 0
            heg_formats = Hash.new
            rs.each do |row|
              description =  row['description']
              format = row['name']
              heg_formats[format] = description
            end
            heg_formats
        else
          #puts "no rows"
          nil
        end
      end


##################################################################################################
#############
      #
      #
```

```
##################################################################################################
#############
      GET_GDAL_SERVICE_FORMATS_SQL = "select f.name, f.description from amdaformat f join amdaformatsrvxref x on
f.formatid = x.formatid join amdaservice s on x.serviceid = s.serviceid where s.name = 'GDAL'";

      def get_gdal_service_formats()
         rs = do_query(GET_GDAL_SERVICE_FORMATS_SQL)
         if rs != nil and rs.count > 0
             gdal_formats = Hash.new
             rs.each do |row|
               description =  row['description']
               format = row['name']
               gdal_formats[format] = description
             end
             gdal_formats
         else
           #puts "no rows"
           nil
         end
      end


##################################################################################################
#############
      #
      #

##################################################################################################
#############
      GET_DA_FORMATID_SQL = "select formatid from amdaformat where name = $1"
      def get_formatid(format_name)
          formatid = nil
          rs = do_query(GET_DA_FORMATID_SQL, format_name)
          if rs != nil and rs.count > 0
              rs.each do |row|
                  formatid = row['formatid']
```

```
            end
        end
        formatid
    end

##########################################################################################################
#############
    #
    #

##########################################################################################################
#############
    GET_GRANULE_FORMAT_CONFIG = "select granuleid, formatid from amdagranuleformatconfig where success = 't"
    def granule_format_config?()
         rs = do_query(GET_GRANULE_FORMAT_CONFIG)
         if rs != nil and rs.count > 0
            rs.each do |row|
                formatid = row['formatid']
            end
         else
            false
         end
    end



##########################################################################################################
#############
    #
    #

##########################################################################################################
#############
    UPDATE_AUTOTEST_GRANULE_FORMAT_CONFIG = "update autotest.amdagranuleformatconfig set serviceid = $2,
formatid = $3, status = $4, message = $5, starttime = $6, duration = $7  where granuleid = $1 and serviceid = $2
and formatid = $3"
```

```
      def update_autotest_granule_format_config(granuleid, serviceid, formatid, status, message, starttime,
duration)
         #if exists, replace old value
          if autotest_granule_format_config?(granuleid, serviceid, formatid)
              do_query(UPDATE_AUTOTEST_GRANULE_FORMAT_CONFIG, granuleid, serviceid, formatid, status, message,
starttime, duration)
          else
              #insert
              insert_autotest_granule_format_config(granuleid, serviceid, formatid, status, message, starttime,
duration)
          end
      end


##############################################################################################################
#############
      #
      #

##############################################################################################################
#############
      GET_AUTOTEST_GRANULE_FORMAT_CONFIG = "select granuleid, serviceid, formatid, status, message from
autotest.amdagranuleformatconfig where granuleid = $1 and serviceid = $2 and formatid = $3"
      def autotest_granule_format_config?(granuleid, serviceid, formatid)
           rs = do_query(GET_AUTOTEST_GRANULE_FORMAT_CONFIG, granuleid, serviceid, formatid)
           if rs != nil and rs.count > 0
              true
           else
              false
           end
      end


##############################################################################################################
#############
      #
```

```
      #

##############################################################################################
#############
      INSERT_AUTOTEST_GRANULE_FORMAT_CONFIG = "insert into autotest.amdagranuleformatconfig values ($1, $2,
$3,$4, $5, $6, $7)"
      def insert_autotest_granule_format_config(granuleid, serviceid, formatid, status, message, starttime,
duration)
         do_query(INSERT_AUTOTEST_GRANULE_FORMAT_CONFIG, granuleid, serviceid, formatid, status, message,
starttime, duration)
      end


##############################################################################################
#############
      #
      #

##############################################################################################
#############
      def update_temp_config_format_table(granuleid, service_name, format, status, message, starttime, duration)
             serviceid = get_serviceid(service_name)
             formatid = get_formatid(format)
             update_autotest_granule_format_config(granuleid, serviceid, formatid, status, message, starttime,
duration)
      end #update_temp_config_format_table


##############################################################################################
#############
      #
      #

##############################################################################################
#############
      UPDATE_AUTOTEST_COLLECTION_FORMAT_CONFIG = "update autotest.amdacollectionformatconfig set filename = $2,
```

```
status = $5, message = $6, starttime = $7, duration = $8  where esdt = $1 and service = $3 and format = $4"
      def update_autotest_collection_format_config(esdt, sample_hdf, service, format, status, message,
starttime, duration)
         #if exists, replace old value
          if autotest_collection_format_config?(esdt, service, format)
              do_query(UPDATE_AUTOTEST_COLLECTION_FORMAT_CONFIG, esdt, sample_hdf, service, format, status,
message, starttime, duration)
          else
              #insert
              insert_autotest_collection_format_config(esdt, sample_hdf, service, format, status, message,
starttime, duration)
          end
      end



##################################################################################################
#############
      #
      #

##################################################################################################
#############
      GET_AUTOTEST_COLLECTION_FORMAT_CONFIG = "select esdt, service, format from
autotest.amdacollectionformatconfig where esdt = $1 and service = $2 and format = $3"
      def autotest_collection_format_config?(esdt, service, format)
           rs = do_query(GET_AUTOTEST_COLLECTION_FORMAT_CONFIG, esdt, service, format)
           if rs != nil and rs.count > 0
              true
           else
              false
           end
      end


##################################################################################################
```

```
#############
      #
      #

##########################################################################################################
#############
      INSERT_AUTOTEST_COLLECTION_FORMAT_CONFIG = "insert into autotest.amdacollectionformatconfig values ($1,
$2, $3,$4, $5, $6, $7, $8)"
      def insert_autotest_collection_format_config(esdt, sample_hdf, service, format, status, message,
starttime, duration)
         do_query(INSERT_AUTOTEST_COLLECTION_FORMAT_CONFIG, esdt, sample_hdf, service, format, status, message,
starttime, duration)
      end


##########################################################################################################
################
      # Returns minimum granuleid in the mode
      # Returns nil if there is a failure

##########################################################################################################
################
      GET_MIN_GRANULEID_SQL = "select MIN(granuleid) as granuleid from amgranule"
      def get_min_granuleid_collection()
         granuleid = nil
         rs = @conn.exec(GET_MIN_GRANULEID_SQL)
         if rs != nil and rs.count > 0
             rs.each do |row|
               granuleid = row[0]
             end
             return granuleid
         else
           #puts "no rows"
           nil
         end
      end
```

```
##################################################################################################
################
      # Returns objects and fields for requested collection and service
      # Returns nil if there is a failure

##################################################################################################
################
      GET_HDF_OBJECTS_FOR_COLLECTION_AND_SERVICE_SQL = "select o.objectname, f.fieldname from
DlDaColHdfObjectSrvXref x join AmDaService s on x.serviceId = s.serviceId
       join DlHdfObjects o on o.objectId = x.objectid join DlHdfFields f on f.fieldId = x.fieldId join
AmCollection c on c.CollectionId = x.CollectionId
       where shortname = $1 and versionid = $2 and name = $3"

      def get_hdf_objects_for_collection_and_service(shortname, versionid, service_name)
         hdf_fields = Hash.new
         rs = do_query(GET_HDF_OBJECTS_FOR_COLLECTION_AND_SERVICE_SQL, shortname, versionid, service_name)
         if rs != nil and rs.count > 0
             rs.each do |row|
               hdf_object = row['objectname']
               hdf_field = row['fieldname']
               hdf_fields[hdf_field] = hdf_object
             end
             hdf_fields
         else
           #puts "no rows"
           nil
         end
      end


##################################################################################################
################
      # Returns objects and fields for requested collection and service configured for exclusions. using only
```

```
objects and fields
      # Returns nil if there is a failure

##################################################################################################
################

      GET_OBJECTS_FIELDS_EXCLUDED_SQL1 = "SELECT  concat_ws('/', o.objectname, f.fieldname) as band_parameter
FROM AmDaParameterConfig bcfg  LEFT JOIN AmCollection c ON bcfg.collectionid = c.collectionid LEFT JOIN
AmDaService srv ON bcfg.serviceid = srv.serviceid LEFT JOIN AmDaProjection proj ON bcfg.projectionid =
proj.projectionid LEFT JOIN DlHdfObjects o ON bcfg.objectid = o.objectid  LEFT JOIN DlHdfFields f ON
bcfg.fieldid = f.fieldid  WHERE c.shortname = $1 and c.versionid = $2 and srv.name = $3 and proj.name = $4 and
requireflag = 'N' "

      def get_objects_excluded_for_collection_service_projection(shortname, versionid, service_name,
projection)
        # puts "excluded bands query"
         hdf_bands = Array.new
         #rs = do_query(GET_OBJECTS_FIELDS_EXCLUDED_SQL, shortname, versionid, service_name, format,
band_parameter)
        #puts "#{GET_OBJECTS_FIELDS_EXCLUDED_SQL1}, #{shortname} , #{versionid}, #{service_name}, #{projection}"
          rs = do_query(GET_OBJECTS_FIELDS_EXCLUDED_SQL1, shortname, versionid, service_name, projection)
          if rs != nil and rs.count > 0
             rs.each do |row|
               #p row
               band = row['band_parameter']
               if !band.nil? && !band.strip.empty?
                  hdf_bands.push(band)
               end
             end
             hdf_bands.map! { |e| e.start_with?("/") ? e : "/" + e }
             #puts hdf_bands
             hdf_bands
         else
           #puts "no rows"
           hdf_bands
         end
```

```
      end


##################################################################################################
################
      # Returns objects and fields for requested collection and service configured for exclusions. using only
objects and fields
      # Returns nil if there is a failure

##################################################################################################
################

      GET_OBJECTS_FIELDS_EXCLUDED_SQL2 = "SELECT  concat_ws('/', o.objectname, f.fieldname) as band_parameter
FROM AmDaParameterConfig bcfg  LEFT JOIN AmCollection c ON bcfg.collectionid = c.collectionid LEFT JOIN
AmDaService srv ON bcfg.serviceid = srv.serviceid LEFT JOIN AmDaFormat fmt ON bcfg.formatid = fmt.formatid  LEFT
JOIN DlHdfObjects o ON bcfg.objectid = o.objectid  LEFT JOIN DlHdfFields f ON bcfg.fieldid = f.fieldid  WHERE
c.shortname = $1 and c.versionid = $2 and srv.name = $3 and fmt.name = $4  and requireflag = 'N' "

      def get_objects_excluded_for_collection_service_format(shortname, versionid, service_name, format)
         #puts "excluded bands"
         hdf_bands = Array.new
         #rs = do_query(GET_OBJECTS_FIELDS_EXCLUDED_SQL, shortname, versionid, service_name, format,
band_parameter)
         #puts "#{GET_OBJECTS_FIELDS_EXCLUDED_SQL}, #{shortname} , #{versionid}, #{service_name}, #{format}"
          rs = do_query(GET_OBJECTS_FIELDS_EXCLUDED_SQL2, shortname, versionid, service_name, format)
          if rs != nil and rs.count > 0
             rs.each do |row|
               band = row['band_parameter']
               if !band.nil? and !band.empty?
                 hdf_bands.push(band)
               end
             end
             hdf_bands.map! { |e| e.start_with?("/") ? e : "/" + e }
             puts hdf_bands
             hdf_bands
         else
```

```
          #puts "no rows"
          hdf_bands
        end
      end

      GET_OBJECTID_SQL = "select objectid from dlhdfobects where objectname = $1"

##########################################################################################
################
      #  get objectid for object name

##########################################################################################
##################
     def get_hdf_objectids(objectname)
        hdf_object_ids =  Array.new
        rs = do_query(GET_OBJECTID_SQL, objectname)
        if rs != nil and rs.count > 0
            rs.each do |row|
              hdf_object_ids.push (row['objectname'])
            end
            hdf_object_ids
        else
          #puts "no rows"
          nil
        end
      end


     GET_FIELDID_SQL = "select fieldid from dlhdffields where fieldname = $1"

##########################################################################################
################
      #  get fieldid for field name

##########################################################################################
##################
```

```
     def get_hdf_fieldids(objectname)
         hdf_field_ids =  Array.new
         rs = do_query(GET_FIELDID_SQL, objectname)
         if rs != nil and rs.count > 0
             rs.each do |row|
               hdf_field_ids.push (row['fieldname'])
             end
             hdf_field_ids
         else
           #puts "no rows"
           nil
         end
      end

    end

  end
end
Current Test Script: glas.rb
#glas

module Sdps
   module Da
      require 'nokogiri'
      require 'sdps/da/daxml'
      require 'sdps/da/dadb'

      class Sdps::Da::Glas
       RESPONSE_FILE_NOT_FOUND = "RESPONSE_FILE_NOT_FOUND"
       RESPONSE_FILE_PARSE_ERROR = "RESPONSE_FILE_PARSE_ERROR"
       RESPONSE_FILE_PARSE_EXCEPTION = "RESPONSE_FILE_PARSE_EXCEPTION"
       ESI_RESPONSE_SCHEMA = "http://eosdis.nasa.gov/esi/rsp/i
http://newsroom.gsfc.nasa.gov/esi/8.1/schemas/ESIAgentResponseInternal.xsd"

       SERVICE_NAME = "GLAS"
```

```
        def initialize(mode)
           @host  ="f5dpl01v:"
           @port  ="22500"
           @service  ="GLAS"
           @mode      = mode
           @service_url = "http://#{@host}#{@port}/#{@rqs}_#{@mode}/GLAS"
           #need to remove the release ...or make it come from the yaml config
           @output_response_dir  = "/workingdata/TestResults/RelB/DataAccess/GlasService/#{@mode}/response/"
           @output_results_dir  = "/workingdata/TestResults/Rel82/DataAccess/GlasService#{@mode}/results/"
        end


        def submit_request(esdt, granuleid, request_url, parameters)
          output_filename = nil
          begin
            fieldvalue1 = "?file_urls=#{request_url}"
            fieldvalue2  = "&FORMAT=#{format}"
            fieldvalue3  = "&OUTPUT_DIR=#{@output_results_dir}"
            timestamp = Time.now.strftime('%Y_%m_%d_%H_%M_%S')

            format1 = format.gsub(" ", "_")
            format1 = format1.gsub("-", "_")
            format1 = format1.gsub(":", "_")
            output_filename = "#{@output_response_dir}#{@service}_#{esdt}_#{granuleid}_#{format1}.xml"
            command = ("curl -# -XGET '#{@service_url}#{fieldvalue1}#{fieldvalue2}#{fieldvalue3}' >
#{output_filename}")
            puts "command= #{command}"
            system (command)
            output_filename
          rescue=>e
            puts e.backtrace
            e.to_s
          end
        end #submitRequest

        def parse_response(xml_filename)
```

```
        end # parse_response


     end # class Glas

   end # module Da
end # module Sdps
Current Test Script: service.rb
# ##########################################################################################
# Service
#
#
# ##########################################################################################
module Sdps
    module Da
        require 'nokogiri'
        require 'sdps/da/daxml'
        require 'sdps/da/dadb'
        require 'sdps/da/dautil'

        class Sdps::Da::Service
            attr_reader :service_request_url, :service_response

            RESPONSE_FILE_NOT_FOUND = "RESPONSE_FILE_NOT_FOUND"
            RESPONSE_FILE_PARSE_ERROR = "RESPONSE_FILE_PARSE_ERROR"
            RESPONSE_FILE_PARSE_EXCEPTION = "RESPONSE_FILE_PARSE_EXCEPTION"


            ##########################################################################
            #
            #
            ##########################################################################
            def initialize(mode, service_type)

                # need to change to
                #let @port  ="22500"
```

```
            case mode
            when "OPS"
                @port = 45000
            when "DEV07"
                @port = 45017
            when "DEV06"
                @port = 45016
            when "DEV08"
                @port = 45018
            when "DEV09"
                @port = 45019
            else
                @port = 22500
            end

            @service  = service_type
            @mode      = mode
        end


        class Sdps::Da::Service::Response
            attr_reader :status, :message, :urls
            def initialize (_status, _message, _urls)
                @status = _status
                @message = _message
                @urls = _urls
            end
        end

        class Sdps::Da::Service::Request
            attr_reader :request_url
            attr_writer :request_url

            def initialize(subagent_id, format, projection, projection_param, band_parameter , bbox ,
temporal, interpolation , include_meta)
                command_str ="&SUBAGENT_ID=#{subagent_id}"
```

```
    if !format.nil? and !format.empty?
        format_encoded = URI::encode_www_form_component(format)
        command_str = "#{command_str}&FORMAT=#{format}"
    end
    if !projection.nil? and !projection.empty?
        projection_encoded = URI::encode_www_form_component(projection)
        command_str ="#{command_str}&PROJECTION=#{projection_encoded}"
    end
    if !projection_param.nil? and !projection_param.empty?
        projection_param_encoded = URI::encode_www_form_component(projection_param)
        command_str ="#{command_str}&PROJECTION_PARAMETERS=#{projection_param_encoded}"
    end
    if !band_parameter.nil? and !band_parameter.empty?
        band_encoded = URI::encode_www_form_component(band_parameter)
        command_str = "#{command_str}&SUBSET_DATA_LAYERS=#{band_encoded}"
    end
    if !bbox.nil? and !bbox.empty?
        #bbox_encoded = URI::encode_www_form_component(bbox)
        command_str = "#{command_str}&BBOX=#{bbox}"
    end
     if !temporal.nil? and !temporal.empty?
        temporal_arr = temporal.split(",")
        temporal_str = "START=#{temporal_arr[0]}&END=#{temporal_arr[1]}"
        #temporal_encoded = URI::encode_www_form_component(temporal_str)
        command_str = "#{command_str}&#{temporal_str}"
    end
     if !interpolation.nil? and !interpolation.empty?
        command_str = "#{command_str}&INTERPOLATION=#{interpolation}"
    end
     if !include_meta.nil? and !include_meta.empty?
        command_str = "#{command_str}&INCLUDE_META=#{include_meta}"
    end
    #@request_url = "#{command_str}&REQUEST_MODE=sync"
    @request_url = "#{command_str}"

end
```

```
          end
        end # class Service
    end # module Da
end # module Sdps
Current Test Script: esi.rb
module Sdps
  module Da
    require 'nokogiri'
    require 'sdps/da/daxml'
    require 'sdps/da/dadb'
    require 'sdps/da/dautil'
    require 'sdps/da/service'

    class Sdps::Da::Esi < Sdps::Da::Service
      attr_reader :esi_service_request_url, :esi_service_response

      def initialize(mode)
        super(mode, "esi")
      #  @host  ="f5dpl01:"
        if @mode == "DEV07" or @mode == "OPS"
            @host  ="f7dpl01.edn.ecs.nasa.gov"
        else
            @host  ="f5dpl01.edn.ecs.nasa.gov"
        end
        #if mode == "OPS" or mode == "ops"
        #  @service_url = "http://#{@host}#{@port}/#{@service}"
        #  @request_url = "http://#{@host}#{@port}/#{@service}/processing"
          @service_url = "https://#{@host}/#{mode.downcase}/#{@service}"
          @request_url = "https://#{@host}/#{mode.downcase}/#{@service}/processing"
        #else
        puts "mode=#{mode}"
        puts "Service URL=#{@service_url}"
        puts "Request URL=#{@request_url}"
        #  @service_url = "http://#{@host}#{@port}/#{@service}_#{mode}"
        #  @request_url = "http://#{@host}#{@port}/#{@service}_#{mode}/processing"
        #  @service_url = "https://#{@host}/#{mode}/#{@service}"
```

```
        #  @request_url = "https://#{@host}/#{mode}/#{@service}/processing"
        #end
      end

      ##########################################################################
      #
      #
      ##########################################################################
      def submit_request(esdt, granuleid, subagent_id, format:"", projection:"", projection_param:"",
band_parameter:"" , bbox:"" ,temporal:"", interpolation:"" , include_meta:"N")
        begin
          file_ids = "#{granuleid}"
          @esi_request = Sdps::Da::Esi::Request.new(@request_url, file_ids, subagent_id, format, projection,
projection_param, band_parameter , bbox , temporal,interpolation , include_meta)
          #@esi_service_request_url = @esi_request.esi_request_url
          #add IDENTIFIER and DEBUG options
          identifier = Time.now.strftime("autotest_%Y_%m_%d_%H_%M_%9N")
          @esi_service_request_url = "#{@esi_request.esi_request_url}&IDENTIFIER=#{identifier}"
          #debug option
          #@esi_service_request_url = "#{command_str}&IDENTIFIER=#{identifier}&DEBUG=Y"
          start_time = Time.now
          puts "#{@esi_service_request_url}"
          @esi_service_response = `curl -s -XGET '#{@esi_service_request_url}'`
          end_time = Time.now
          #puts "Time elapsed - #{end_time - start_time} secs"
          @esi_service_response
        rescue=>e
          puts e.backtrace
          e.to_s
        end
      end

      def parse_response(xml_buff)
        begin
          #puts xml_buff
          @esi_xml = Sdps::Da::DaXml.new(xml_buff)
```

```
          if @esi_xml.nil?
            response = Sdps::Da::Esi::Response.new(RESPONSE_FILE_NOT_FOUND, "", xml_buff)
          elsif @esi_xml.contains?("/Exception")
            status = @esi_xml.get_value('/Exception/Code')
            message = @esi_xml.get_value('/Exception')[0..198]
            response = Sdps::Da::Esi::Response.new(status, "FAILURE",@esi_xml)
          elsif @esi_xml.contains?("/agentResponse")
            urls = @esi_xml.get_value('//downloadUrls')
            urls = urls.strip
            if !urls.nil? and  !urls.empty?
              response = Sdps::Da::Esi::Response.new(status, "SUCCESS", urls)
            else
              response = Sdps::Da::Esi::Response.new(status, "FAILURE", "no download urls")
            end
          else
            response = Sdps::Da::Esi::Response.new(RESPONSE_FILE_PARSE_ERROR, "FAILURE", @esi_xml)
          end
        rescue=>e
          puts e.backtrace
          response = Sdps::Da::Esi::Response.new(RESPONSE_FILE_PARSE_EXCEPTION, "FAILURE", e.to_s)
        end
        response
      end # parse_response

      class Sdps::Da::Esi::Response < Sdps::Da::Service::Response

      end

      class Sdps::Da::Esi::Request < Sdps::Da::Service::Request
        attr_reader :esi_request_url
        def initialize(req_url, file_ids, subagent_id, format, projection, projection_param, band_parameter ,
bbox , temporal, interpolation , include_meta)
          super(subagent_id, format, projection, projection_param, band_parameter , bbox , temporal,
interpolation , include_meta)
          #puts "#{@command_url}"
          request_str = "#{req_url}"
```

```
          request_str = "#{request_str}?FILE_IDS=#{file_ids}"
          @esi_request_url ="#{request_str}#{@request_url}"

        end
      end

    end # class Esi

  end # module Da
end # module Sdps
Current Test Script: gdal.rb
#gdal and gdal webservice methods

module Sdps
   module Da
      require 'nokogiri'
      require 'sdps/da/daxml'
      require 'sdps/da/dadb'
      require 'sdps/da/dautil'

      class Sdps::Da::Gdal
       RESPONSE_FILE_NOT_FOUND = "RESPONSE_FILE_NOT_FOUND"
       RESPONSE_FILE_PARSE_ERROR = "RESPONSE_FILE_PARSE_ERROR"
       RESPONSE_FILE_PARSE_EXCEPTION = "RESPONSE_FILE_PARSE_EXCEPTION"
       ESI_RESPONSE_SCHEMA = "http://eosdis.nasa.gov/esi/rsp/i
http://newsroom.gsfc.nasa.gov/esi/8.1/schemas/ESIAgentResponseInternal.xsd"
       GDAL_TRANSLATE_FORMATS_CMD = "/tools/gdal/bin/gdal_translate --formats"
       GDAL_TRANSLATE_CMD = "/tools/gdal/bin/gdal_translate"
       GDAL_INFO_CMD = "/tools/gdal/bin/gdalinfo"

       PARSE_WEST_DEG = " | grep 'Upper Left' | sed  's/[()]//g' | sed 's/[,]/, /g' | sed 's/[d]/d /g' | awk
'{print $5}'"
       PARSE_WEST_MIN = " | grep 'Upper Left' | sed  's/[()]//g' | sed 's/[,]/, /g' | sed 's/[d]/d /g' | awk
'{print $6}'"
       PARSE_WEST_DIR =  " | grep 'Upper Left' | sed  's/[()]//g' | sed 's/[,]/, /g' | sed 's/[d]/d /g' | awk
'{print $7}'"
```

```
       PARSE_NORTH_DEG = " | grep 'Upper Left' | sed  's/[()]//g' | sed 's/[,]/, /g' | sed 's/[d]/d /g'  | sed
\"s/['']/' /g\" | awk '{print $8}'"
       PARSE_NORTH_MIN = " | grep 'Upper Left' | sed  's/[()]//g' | sed 's/[,]/, /g' | sed 's/[d]/d /g'  | sed
\"s/['']/' /g\" | awk '{print $9}'"
       PARSE_NORTH_DIR = " | grep 'Upper Left' | sed  's/[()]//g' | sed 's/[,]/, /g' | sed 's/[d]/d /g'   | sed
\"s/['']/' /g\" | awk '{print $10}'"

       PARSE_EAST_DEG = " | grep 'Lower Right' | sed  's/[()]//g' | sed 's/[,]/, /g' | sed 's/[d]/d /g'  | sed
\"s/['']/' /g\" | awk '{print $5}'"
       PARSE_EAST_MIN = " | grep 'Lower Right' | sed  's/[()]//g' | sed 's/[,]/, /g' | sed 's/[d]/d /g'  | sed
\"s/['']/' /g\" | awk '{print $6}'"
       PARSE_EAST_DIR = " | grep 'Lower Right' | sed  's/[()]//g' | sed 's/[,]/, /g' | sed 's/[d]/d /g'  | sed
\"s/['']/' /g\" | awk '{print $7}'"

       PARSE_SOUTH_DEG =" | grep 'Lower Right' | sed  's/[()]//g' | sed 's/[,]/, /g' | sed 's/[d]/d /g'  | sed
\"s/['']/' /g\" | awk '{print $8}'"
       PARSE_SOUTH_MIN =" | grep 'Lower Right' | sed  's/[()]//g' | sed 's/[,]/, /g' | sed 's/[d]/d /g'  | sed
\"s/['']/' /g\" | awk '{print $9}'"
       PARSE_SOUTH_DIR =" | grep 'Lower Right' | sed  's/[()]//g' | sed 's/[,]/, /g' | sed 's/[d]/d /g'  | sed
\"s/['']/' /g\" | awk '{print $10}'"

        SERVICE_NAME = "GDAL"

        def initialize(mode)
           @host  ="f5dpl01v:"
           @port  ="22500"
           @service  ="GdalService"
           @mode      = mode
           if mode == "OPS" or mode == "ops"
               @service_url = "http://#{@host}#{@port}/#{@service}"
           else
              @service_url = "http://#{@host}#{@port}/#{@service}_#{@mode}"
           end
        end
```

```
        def submit_request(esdt, granuleid, request_url, format)

          begin
            fieldvalue1 = "?file_urls=#{request_url}"
            fieldvalue2  = "&FORMAT=#{format}"
            #fieldvalue3  = "&OUTPUT_DIR=#{@output_results_dir}"
            #timestamp = Time.now.strftime('%Y_%m_%d_%H_%M_%S')

            format1 = format.gsub(" ", "_")
            format1 = format1.gsub("-", "_")
            format1 = format1.gsub(":", "_")

            #output_filename = "#{@output_response_dir}#{@service}_#{esdt}_#{granuleid}_#{format1}.xml"
            #command = ("curl -# -XGET '#{@service_url}#{fieldvalue1}#{fieldvalue2}#{fieldvalue3}' >
#{output_filename}")
            #command = ("curl -# -XGET '#{@service_url}#{fieldvalue1}#{fieldvalue2}' ")

            output_buff = `curl -# -XGET '#{@service_url}#{fieldvalue1}#{fieldvalue2}'`
            output_buff
          rescue=>e
            puts e.to_s
            puts e.backtrace
          end
        end #submitRequest

        def parse_response(xml_buff)
           begin
             @gdal_xml = Sdps::Da::DaXml.new(xml_buff)

             if @gdal_xml.nil?
               response = Response.new(RESPONSE_FILE_NOT_FOUND, xml_buff)
             elsif @gdal_xml.contains?("/Exception")
               status = @gdal_xml.get_value('/Exception/Code')
               message = @gdal_xml.get_value('/Exception')[0..198]
               #info = @gdal_xml.get_xml('/Exception/addlInfo')[0..198]
```

```
      #message = message1[0..198]
      #if info.nil? = false
      #   message = (message1 << info)[0..198]
      #end
      response = Sdps::Da::Gdal::Response.new(status, message)
    elsif @gdal_xml.contains?("/agentResponse")
      status = @gdal_xml.get_value('/agentResponse/processInfo/message')
      #message = @gdal_xml.get_value('/agentResponse')
      urls = @gdal_xml.get_value('/agentResponse/downloadUrls')
             urls = @gdal_xml.get_value('/agentResponse/downloadUrls')
      response = Sdps::Da::Gdal::Response.new(status, "", urls)
    else
       response = Sdps::Da::Gdal::Response.new(RESPONSE_FILE_PARSE_ERROR, @gdal_xml)
    end
  rescue=>e
    puts e.backtrace
    response = Sdps::Da::Gdal::Response.new(RESPONSE_FILE_PARSE_EXCEPTION, e.to_s)
  end
  response
end # parse_response


def get_gdal_formats
    formats = `#{GDAL_TRANSLATE_FORMATS_CMD}`
    if formats.nil?
       puts "no formats"
       nil
    else
       arr = formats.split("\n  ")
       #p arr
       if arr.nil? or arr.size == 0
          puts "no valid formats"
          nil
       else
          # delete first row
          arr.delete_at(0)
```

```
            p arr
            gdal_formats = Hash[ arr.map { |v| [ v.split(" (")[0], v.split(" (")[1].split("): ")[1] ] } ]

            p gdal_formats
            gdal_formats
         end
      end
end


def run_gdal_translate (inputfile, format)
      #/tools/gdal/bin/gdal_translate -of PNG

      output_extn = Sdps::Da::FORMAT_EXTENSIONS[format]
      input_extn = File.extname(inputfile)
      outputfile = inputfile.gsub(input_extn, output_extn)
      cmd_string = "#{GDAL_TRANSLATE_CMD} -q -of #{format} #{inputfile} #{outputfile}"
     #puts cmd_string
      system(cmd_string)
end

 def get_projection_info(input_file)
    projection =  `#{GDAL_TRANSLATE_CMD} | grep "PROJECTION" #{input_file}`
    if projection.include?("CYLINDRICAL_EQUAL_AREA")
        projection_parameter = "cea"
    elsif projection.include?("LAMBERT_AZIMUTHAL_EQUAL_AREA")
        projection_parameter = "laea"
    else
        nil
    end
 end

 def get_proj4_params(input_file)
      params = `#{GDAL_INFO_CMD} -proj4 #{input_file} | grep proj`
      params1 = params.chomp
      proj4_params = params1.gsub("'","")
```

```
        end

        def get_spatial_extent(inputfile)
            west_deg = `#{GDAL_INFO_CMD} #{inputfile} #{PARSE_WEST_DEG}`
            west_min = `#{GDAL_INFO_CMD} #{inputfile} #{PARSE_WEST_MIN}`
            west_dir = `#{GDAL_INFO_CMD} #{inputfile} #{PARSE_WEST_DIR}`

           south_deg = `#{GDAL_INFO_CMD} #{inputfile} #{PARSE_SOUTH_DEG}`
           south_min = `#{GDAL_INFO_CMD} #{inputfile} #{PARSE_SOUTH_MIN}`
           south_dir = `#{GDAL_INFO_CMD} #{inputfile} #{PARSE_SOUTH_DIR}`

            east_deg = `#{GDAL_INFO_CMD} #{inputfile} #{PARSE_EAST_DEG}`
            east_min = `#{GDAL_INFO_CMD} #{inputfile} #{PARSE_EAST_MIN}`
            east_dir = `#{GDAL_INFO_CMD} #{inputfile} #{PARSE_EAST_DIR}`

            north_deg = `#{GDAL_INFO_CMD} #{inputfile} #{PARSE_NORTH_DEG}`
            north_min = `#{GDAL_INFO_CMD} #{inputfile} #{PARSE_NORTH_MIN}`
            north_dir = `#{GDAL_INFO_CMD} #{inputfile} #{PARSE_NORTH_DIR}`
            #puts south_deg
            #puts south_min
            #puts "southdir = #{south_dir}"

            #puts "southdir = #{south_dir.strip.chomp(",")[-1]}"

            @utils = Sdps::Da::DaUtil.new()
            west =@utils.dms_to_degrees(west_deg.strip!.to_f , west_min.strip!.to_f,
west_dir.strip.chomp(",")[-1])
            south =@utils.dms_to_degrees(south_deg.strip!.to_f , south_min.strip!.to_f,
south_dir.strip.chomp(",")[-1])
            east =@utils.dms_to_degrees(east_deg.strip!.to_f , east_min.strip!.to_f,
east_dir.strip.chomp(",")[-1])
            north =@utils.dms_to_degrees(north_deg.strip!.to_f ,north_min.strip!.to_f,
north_dir.strip.chomp(",")[-1])

            spatialextent = [west, south, east, north]
```

```
        spatialextent
   end

  class Sdps::Da::Gdal::Info
     attr_reader  :filename, :projection,:spatialextent
     def initialize (buffer)
           arr = buffer.split("\n")
           #puts arr
           begin

               deg = west[0].strip
               min =  west[1].split("'")[0].strip
               sign =  west[1][-1]


               @spatialextent = "west,south,east,north"
           rescue=>e
              puts e.backtrace
           end
     end
  end


  class Sdps::Da::Gdal::Response
     attr_reader :status, :message, :downloadurls
     def initialize (code, description, urls)
        @status = code
        @message = description
        @downloadurls = urls
     end
  end

class Sdps::Da::Gdal::Request
     attr_reader :url, :format, :options, :operand
end
```

```
     end # class Gdal

   end # module Da
end # module Sdps
Current Test Script: smap.rb
#gdal and gdal webservice methods

module Sdps
   module Da
      require 'nokogiri'
      require 'sdps/da/daxml'
      require 'sdps/da/dadb'

      class Sdps::Da::Smap
       RESPONSE_FILE_NOT_FOUND = "RESPONSE_FILE_NOT_FOUND"
       RESPONSE_FILE_PARSE_ERROR = "RESPONSE_FILE_PARSE_ERROR"
       RESPONSE_FILE_PARSE_EXCEPTION = "RESPONSE_FILE_PARSE_EXCEPTION"
       ESI_RESPONSE_SCHEMA = "http://eosdis.nasa.gov/esi/rsp/i
http://newsroom.gsfc.nasa.gov/esi/8.1/schemas/ESIAgentResponseInternal.xsd"

       SERVICE_NAME = "SMAPL1L2"

        def initialize(mode)
           @host  ="f5dpl01v:"
           @port  ="22500"
           @service  ="SMAPL1L2"
           @mode      = mode
           @service_url = "http://#{@host}#{@port}/#{@rqs}_#{@mode}/GLAS"
           #need to remove the release ...or make it come from the yaml config
           @output_response_dir  = "/workingdata/TestResults/RelB/DataAccess/GlasService/#{@mode}/response/"
           @output_results_dir  = "/workingdata/TestResults/Rel82/DataAccess/GlasService#{@mode}/results/"
        end


        def submit_request(esdt, granuleid, request_url, parameters)
```

```
          output_filename = nil
          begin
            fieldvalue1 = "?file_urls=#{request_url}"
            fieldvalue2  = "&FORMAT=#{format}"
            fieldvalue3  = "&OUTPUT_DIR=#{@output_results_dir}"
            timestamp = Time.now.strftime('%Y_%m_%d_%H_%M_%S')

            format1 = format.gsub(" ", "_")
            format1 = format1.gsub("-", "_")
            format1 = format1.gsub(":", "_")
            output_filename = "#{@output_response_dir}#{@service}_#{esdt}_#{granuleid}_#{format1}.xml"
            command = ("curl -# -XGET '#{@service_url}#{fieldvalue1}#{fieldvalue2}#{fieldvalue3}' >
#{output_filename}")
            puts "command= #{command}"
            system (command)
            output_filename
          rescue=>e
            puts e.backtrace
            e.to_s
          end
        end #submitRequest

        def parse_response(xml_filename)

        end # parse_response

     end # class
   end # module Da
end # module Sdps
Current Test Script: qauudb.rb
module Sdps
  class QAUU
     class QAUUDb
         def initialize(dbconn)
             @conn = dbconn
         end
```

```
        # can be updated to get a whole row
        GET_SHORTNAME_SITE_SQL = "select shortname, site from dsqamutesdtsite where shortname = $1 and site =
$2"
        ADD_SHORTNAME_SITE_SQL = "insert into dsqamutesdtsite values ($1, $2)"

        def get_shortname_for_site (shortname, sitename)
                params = Array.new()
                params << shortname
                params << sitename
                rs = @conn.exec_params(GET_SHORTNAME_SITE_SQL, params)
                puts "get shortname="
                puts rs
                puts rs.count
                if !rs.nil? and rs.count > 0
                   puts rs[0]
                end
                rs
        end

        def add_collection_to_site(shortname, sitename)
              if get_shortname_for_site(shortname,sitename).nil?
                   params = Array.new()
                   params << shortname
                   params << sitename
                   @conn.exec_params(ADD_SHORTNAME_SITE_SQL, params)
              end
              true
        end

     end
  end
end
Current Test Script: qarequest.rb
module Sdps
     class QAUU
```

```
        class QaRequest
            attr_reader :mode, :sitename, :shortname, :versionid, :qa_request_filename

            def initialize(mode, site, collection_shortname, collection_versionid)
                 @mode = mode
                @sitename = site
                @shortname = collection_shortname
                @versionid = collection_versionid
            end

            def initialize_qa_request_file(file_description)
                current_time = Time.new.getlocal.strftime("%Y%m%d%H%M%S")
                @qa_request_filename = "#{@mode}_#{@sitename}_QAUPDATE#{file_description}.#{current_time}"
                puts @qa_request_filename
                @qa_request_filename
            end

            def create_science_qa_request_with_granule_ur(granuleid, measured_parameter_name, qa_flag_value,
qa_flag_explanation)
                   update_buffer = ""
                    update_buffer << "From #{@sitename}\n"
                    update_buffer << "begin QAMetadataUpdate Science GranuleUR\n"
                    granule_ur = "#{granuleid}"
                    update_buffer <<
"#{shortname}\t#{versionid}\t#{granule_ur}\t#{measured_parameter_name}\t#{qa_flag_value}\t#{qa_flag_explanation}
\n"
                    update_buffer << "end QAMetadataUpdate\n"
                    puts update_buffer
                    update_buffer.split("\n")
            end

        end
     end
end
Current Test Script: spec_helper.rb
# Conventionally, all specs live under a `spec` directory, which RSpec
```

```
# adds to the `$LOAD_PATH`.
# Require this file using `require "spec_helper"` to ensure that it is only
# loaded once.
#
# See http://rubydoc.info/gems/rspec-core/RSpec/Core/Configuration
RSpec.configure do |config|
  config.treat_symbols_as_metadata_keys_with_true_values = true
  config.run_all_when_everything_filtered = true
  config.filter_run :focus
  config.mock_framework = :rspec

  # Use color in STDOUT
  config.color = true

#  # Use color not only in STDOUT but also in pagers and files
#  config.tty = true

  # Use the specified formatter
  # config.formatter = :progress
  # config.formatter = :html
  # config.formatter = :textmate
  config.formatter = :documentation

  # Run specs in random order to surface order dependencies. If you find an
  # order dependency and want to debug it, you can fix the order by providing
  # the seed, which is printed after each run.
  #     --seed 1234
  config.order = 'random'
end

old_VERBOSE, $VERBOSE = $VERBOSE, nil
require 'sdps'
$VERBOSE = old_VERBOSE

Current Test Script: tcpproxylogger_spec.rb
require 'spec_helper'
```

```
describe Sdps::TcpProxyLogger do
  describe '.parse_file' do
    it "parses a file of TCP log entries into a list of LogEntrys" do
      tcp_proxy_log_pathname = File.expand_path('../tcp.log.2013-10-31T161246.443092', __FILE__)
      num_lines = open(tcp_proxy_log_pathname){|f|f.lines.to_a.length}
      timestamp_re = /^\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}\.\d{6}$/

      log_entries = Sdps::TcpProxyLogger.parse_file(tcp_proxy_log_pathname)

      expect(log_entries.length).to eq(num_lines)
      expect(log_entries.all?{|e| e.kind_of? Sdps::TcpProxyLogger::LogEntry}).to be_true
      expect(log_entries.all?{|e| timestamp_re =~ e.timestamp}).to be_true
      expect(log_entries.all?{|e| e.request? or e.response? or e.msg}).to be_true
    end
  end
end
```

Current Test Script: aimdb_spec.rb

```
require 'spec_helper.rb'

def str_to_datetime_no_usec(str)
  Time.strptime(str, '%Y-%m-%d %H:%M:%S')
end

class Array
  def has_all_ints?
    self.all? {|n| Fixnum === n or Bignum === n}
  end
end

# Create an AimDb instance with a mock PG instance that expects to have
# its #exec_params method called with specific arguments and returns a
# specified result.  The result should be taken from an actual call to
# #exec_params.
#
def aim_with_mock_pg(expected_sql, expected_arglist, pg_result)
```

```
  mockpg = double("pg")
  mockpg.should_receive(:exec_params).with(expected_sql, expected_arglist)
  mockpg.stub(:exec_params) { pg_result }
  Sdps::AimDb.new(mockpg)
end

describe Sdps::AimDb do

  describe '#find_collection_by_shortname_versionid' do
    it "returns a Collection" do
      pg_result = [{"collectionid"=>"71171", "shortname"=>"NISE", "longname"=>"Near-Real-Time SSMIS EASE-Grid
Daily Global Ice Concentration and Snow Extent", "versionid"=>"4", "collectiondescription"=>"Data for the
Near-Real-Time SSMIS EASE-Grid Daily Global Ice Concentration and Snow Extent product are provided in two 25-km
grids, in an azimuthal, equal area projection (the NSIDC SL and NL EASE-Grids).", "revisiondate"=>"2009-08-14
00:00:00", "type"=>"SC", "subtype"=>"NISE.004", "groupid"=>"OTHR        ", "filesystemlabel"=>"FS2",
"spatialsearchtype"=>"Rectangle", "inserttime"=>"2013-10-02 10:56:33.950006", "lastupdate"=>"2013-10-02
11:51:18.559145", "lastcollectionmovetime"=>"2013-10-02 10:56:33.950006", "publicretentiontime"=>"3650",
"hiddenretentiontime"=>"87600", "bulkexportedflag"=>"Y", "convertenabledflag"=>"N", "cloudcoversourceid"=>nil,
"daynightflag"=>"N", "globalflag"=>"Y", "exclusionscflag"=>"N", "publishbydefaultflag"=>"Y",
"allowpublishflag"=>"Y", "moveflag"=>"N", "orderonlyfslabel"=>nil, "orderonlysndirname"=>"NISE.004eCtvEHkm",
"replacementonflag"=>"N", "ecsflag"=>"Y", "orderviewphflag"=>"N", "orderviewqaflag"=>"N",
"orderbrowseflag"=>"Y", "twentyfourhourflag"=>"Y", "nominalcoveragerule"=>nil, "publicpriority"=>"255",
"gridlevel"=>"6", "qualitysummary"=>nil, "themepsaname"=>nil, "processingcenter"=>"NSIDC",
"archivecenter"=>"NSIDC", "processinglevelid"=>"3     ", "sciencemimetypeid"=>"1", "browsemimetypeid"=>"1",
"browseonlinemimetypeid"=>"2", "metadatamimetypeid"=>"4", "ingestcreatemapflag"=>"N",
"orderviewhdf4mapflag"=>"Y", "filedataformatid"=>"8", "duplicategranruleno"=>"2", "datamodeltype"=>"ECS"}]
      row = pg_result[0]
      expected_sql = Sdps::AimDb::COLLECTION_BY_SHORTNAME_VERSIONID_SQL
      expected_arglist = [row['shortname'], row['versionid'].to_i]

      aim = aim_with_mock_pg(expected_sql, expected_arglist, pg_result)

      actual = aim.find_collection_by_shortname_versionid('NISE', 4)

      expect(actual.collectionid).to eq(row['collectionid'].to_i)
      expect(actual.class).to eq(Sdps::AimDb::AmCollection)
```

```
      expect(actual.shortname).to eq(row['shortname'])
      expect(actual.versionid).to eq(row['versionid'].to_i)
      expect(actual.longname).to eq(row['longname'])
      expect(actual.revisiondate).to eq(str_to_datetime_no_usec(row['revisiondate']))
    end
  end

  describe '#find_collection_by_collectionid' do
    it "returns a Collection" do
      pg_result = [{"collectionid"=>"47260", "shortname"=>"MOD13Q1", "longname"=>"MODIS/Terra Vegetation Indices
16-Day L3 Global 250m SIN Grid", "versionid"=>"4", "collectiondescription"=>"MODIS/Terra Vegetation Indices
16-Day L3 Global 250m SIN Grid", "revisiondate"=>"2009-06-18 00:00:00", "type"=>"SC", "subtype"=>"MOD13Q1.004",
"groupid"=>"MOLT        ", "filesystemlabel"=>"DEFAULT", "spatialsearchtype"=>"GPolygon",
"inserttime"=>"2008-03-26 16:37:19.62", "lastupdate"=>"2009-06-18 09:25:06.943",
"lastcollectionmovetime"=>"2005-02-08 11:14:01.71", "publicretentiontime"=>"5000",
"hiddenretentiontime"=>"120000", "bulkexportedflag"=>"Y", "convertenabledflag"=>"Y", "cloudcoversourceid"=>"2",
"daynightflag"=>"Y", "globalflag"=>"N", "exclusionscflag"=>"N", "publishbydefaultflag"=>"N",
"allowpublishflag"=>"Y", "moveflag"=>"N", "orderonlyfslabel"=>nil, "orderonlysndirname"=>"MOD13Q1.004WLwUQVqM",
"replacementonflag"=>"N", "ecsflag"=>"Y", "orderviewphflag"=>"N", "orderviewqaflag"=>"N",
"orderbrowseflag"=>"N", "twentyfourhourflag"=>"Y", "nominalcoveragerule"=>"MODIS_8_16_LGID",
"publicpriority"=>"255", "gridlevel"=>"6", "qualitysummary"=>nil, "themepsaname"=>nil,
"processingcenter"=>"MODAPS", "archivecenter"=>"LPDAAC", "processinglevelid"=>"3     ",
"sciencemimetypeid"=>"1", "browsemimetypeid"=>"1", "browseonlinemimetypeid"=>"2", "metadatamimetypeid"=>"4",
"ingestcreatemapflag"=>"N", "orderviewhdf4mapflag"=>"N", "filedataformatid"=>"8", "duplicategranruleno"=>"3",
"datamodeltype"=>"ECS"}]
      row = pg_result[0]
      collectionid = row['collectionid'].to_i
      expected_sql = Sdps::AimDb::COLLECTION_BY_COLLECTIONID_SQL
      expected_arglist = [collectionid]
      aim = aim_with_mock_pg(expected_sql, expected_arglist, pg_result)

      actual = aim.find_collection_by_collectionid(collectionid)

      expect(actual.collectionid).to eq(collectionid)
      expect(actual.class).to eq(Sdps::AimDb::AmCollection)
      expect(actual.shortname).to eq(row['shortname'])
```

```
    expect(actual.versionid).to eq(row['versionid'].to_i)
    expect(actual.longname).to eq(row['longname'])
    expect(actual.revisiondate).to eq(str_to_datetime_no_usec(row['revisiondate']))
    ## FIXME Fields of type 'character(12)' pad to 12 characters with spaces.
    #expect(actual.groupid).to eq(row['groupid'].strip)
  end
end

describe "#find_active_granuleids_by_collectionid" do
  it "returns a list of GranuleIDs" do
    pg_values = [["53968"], ["53970"], ["53971"], ["53972"],
                 ["53973"], ["53974"], ["53975"], ["53979"],
                 ["53980"], ["53981"], ["70700"], ["70699"],
                 ["70701"], ["70702"], ["70639"], ["70638"],
                 ["70640"], ["70642"], ["70673"], ["70674"],
                 ["70675"], ["70676"], ["53969"], ["53964"],
                 ["53977"], ["53963"], ["53962"], ["53967"],
                 ["53965"], ["53966"], ["53978"], ["53976"]]
    pg_result = double("pg_result")
    pg_result.stub(:values) { pg_values }
    collectionid = 52047
    expected_arglist = [collectionid]
    expected_sql = Sdps::AimDb::ACTIVE_GRANULEIDS_SQL
    aim = aim_with_mock_pg(expected_sql, expected_arglist, pg_result)

    actual = aim.find_active_granuleids_by_collectionid(collectionid)

    expect(actual.length).to eq(pg_values.length)
    expect(actual).to eq(pg_values.flatten.map(&:to_i))
    expect(actual).to_not include nil
    expect(actual).to have_all_ints, "Expected all ints, got #{actual}"
  end
end

describe "#find_granule_by_granuleid" do
  it "returns a Granule" do
```

```
      pg_result = [{"granuleid"=>"68008", "collectionid"=>"42920", "shortname"=>"MOD10A2", "versionid"=>"5",
"ecsflag"=>"Y", "granulestate"=>"AR", "registrationtime"=>"2009-09-30 17:54:55.906", "archivetime"=>"2009-09-30
17:54:56.316", "publishtime"=>nil, "lastupdate"=>"2013-04-10 14:02:59.469751", "deletefromarchive"=>"N",
"deleteeffectivedate"=>nil, "externalid"=>nil, "isorderonly"=>"H", "versionnumber"=>"3",
"archiveneededflag"=>"Y", "pgeversion"=>"5.0.7", "productiondatetime"=>"2009-06-15 17:59:44",
"rangebeginningdate"=>"2007-09-30", "rangebeginningtime"=>"00:00:00.0000000", "rangeendingdate"=>"2007-10-07",
"rangeendingtime"=>"23:59:59.0000000", "calendardate"=>nil, "timeofday"=>nil, "beginningdatetime"=>"2007-09-30
00:00:00", "endingdatetime"=>"2007-10-07 23:59:59", "daynightflag"=>"Day  ", "sizembecsdatagranule"=>"0.142593",
"localgranuleid"=>"MOD10A2.A2007273.h27v06.005.2009166175943.hdf", "localversionid"=>"SCF V5.0.0",
"radiometricdbversion"=>nil, "reprocessingplanned"=>"further update is anticipated",
"reprocessingactual"=>"reprocessed", "geometricdbversion"=>nil, "dapid"=>nil, "dapinsertdate"=>nil}]
      row = pg_result[0]
      granuleid = row['granuleid'].to_i
      expected_arglist = [granuleid]
      expected_sql = Sdps::AimDb::GRANULE_BY_GRANULEID_SQL
      aim = aim_with_mock_pg(expected_sql, expected_arglist, pg_result)

      productiondatetime = str_to_datetime_no_usec(row['productiondatetime'])
      rangeendingdate = Date.strptime(row['rangeendingdate'])

      actual = aim.find_granule_by_granuleid(granuleid)

      expect(actual.granuleid).to eq(row['granuleid'].to_i)
      expect(actual.shortname).to eq(row['shortname'])
      expect(actual.versionid).to eq(row['versionid'].to_i)
      expect(actual.productiondatetime).to eq(productiondatetime)
      expect(actual.rangeendingdate).to eq(rangeendingdate)
      expect(actual.rangeendingtime).to eq(row['rangeendingtime'])
    end
  end

  describe '#find_granules_by_shortname_versionid' do
    it "finds all granules in one collection" do
      pg_result = [{"granuleid"=>"68011", "collectionid"=>"42920", "shortname"=>"MOD10A2", "versionid"=>"5",
"ecsflag"=>"Y", "granulestate"=>"AR", "registrationtime"=>"2009-10-01 11:06:31.296", "archivetime"=>"2009-10-01
11:06:32.623", "publishtime"=>"2009-10-01 11:06:44.823", "lastupdate"=>"2009-10-01 11:06:45.326",
```

"deletefromarchive"=>"N", "deleteeffectivedate"=>nil, "externalid"=>nil, "isorderonly"=>nil, "versionnumber"=>"4", "archiveneededflag"=>"Y", "pgeversion"=>"5.0.7", "productiondatetime"=>"2009-06-15 17:59:44", "rangebeginningdate"=>"2007-09-30", "rangebeginningtime"=>"00:00:00.0000000", "rangeendingdate"=>"2007-10-07", "rangeendingtime"=>"23:59:59.0000000", "calendardate"=>nil, "timeofday"=>nil, "beginningdatetime"=>"2007-09-30 00:00:00", "endingdatetime"=>"2007-10-07 23:59:59", "daynightflag"=>"Day ", "sizembecsdatagranule"=>"0.142593", "localgranuleid"=>"MOD10A2.A2007273.h27v06.005.2009166175943.hdf", "localversionid"=>"SCF V5.0.0", "radiometricdbversion"=>nil, "reprocessingplanned"=>"further update is anticipated", "reprocessingactual"=>"reprocessed", "geometricdbversion"=>nil, "dapid"=>nil, "dapinsertdate"=>nil},
{"granuleid"=>"68054", "collectionid"=>"42920", "shortname"=>"MOD10A2", "versionid"=>"5", "ecsflag"=>"Y", "granulestate"=>"AR", "registrationtime"=>"2010-08-19 14:06:28.243", "archivetime"=>"2010-08-19 14:06:29.41", "publishtime"=>"2010-08-19 14:06:44.186", "lastupdate"=>"2010-08-19 14:06:52.593", "deletefromarchive"=>"N", "deleteeffectivedate"=>nil, "externalid"=>nil, "isorderonly"=>nil, "versionnumber"=>nil, "archiveneededflag"=>"Y", "pgeversion"=>"5.0.8", "productiondatetime"=>"2010-07-14 08:11:08", "rangebeginningdate"=>"2010-07-04", "rangebeginningtime"=>"00:00:00.0000000", "rangeendingdate"=>"2010-07-11", "rangeendingtime"=>"23:59:59.0000000", "calendardate"=>nil, "timeofday"=>nil, "beginningdatetime"=>"2010-07-04 00:00:00", "endingdatetime"=>"2010-07-11 23:59:59", "daynightflag"=>"Day ", "sizembecsdatagranule"=>"0.0762033", "localgranuleid"=>"MOD10A2.A2010185.h00v08.005.2010195081107.hdf", "localversionid"=>"SCF V5.0.0", "radiometricdbversion"=>nil, "reprocessingplanned"=>"further update is anticipated", "reprocessingactual"=>"reprocessed", "geometricdbversion"=>nil, "dapid"=>nil, "dapinsertdate"=>nil},
{"granuleid"=>"68005", "collectionid"=>"42920", "shortname"=>"MOD10A2", "versionid"=>"5", "ecsflag"=>"Y", "granulestate"=>"AR", "registrationtime"=>"2009-09-30 16:46:48.94", "archivetime"=>"2009-09-30 16:46:49.51", "publishtime"=>nil, "lastupdate"=>"2013-04-10 14:02:51.609949", "deletefromarchive"=>"N", "deleteeffectivedate"=>nil, "externalid"=>nil, "isorderonly"=>"H", "versionnumber"=>"2", "archiveneededflag"=>"Y", "pgeversion"=>"5.0.7", "productiondatetime"=>"2009-06-15 17:59:44", "rangebeginningdate"=>"2007-09-30", "rangebeginningtime"=>"00:00:00.0000000", "rangeendingdate"=>"2007-10-07", "rangeendingtime"=>"23:59:59.0000000", "calendardate"=>nil, "timeofday"=>nil, "beginningdatetime"=>"2007-09-30 00:00:00", "endingdatetime"=>"2007-10-07 23:59:59", "daynightflag"=>"Day ", "sizembecsdatagranule"=>"0.142593", "localgranuleid"=>"MOD10A2.A2007273.h27v06.005.2009166175943.hdf", "localversionid"=>"SCF V5.0.0", "radiometricdbversion"=>nil, "reprocessingplanned"=>"further update is anticipated", "reprocessingactual"=>"reprocessed", "geometricdbversion"=>nil, "dapid"=>nil, "dapinsertdate"=>nil},
{"granuleid"=>"68008", "collectionid"=>"42920", "shortname"=>"MOD10A2", "versionid"=>"5", "ecsflag"=>"Y", "granulestate"=>"AR", "registrationtime"=>"2009-09-30 17:54:55.906", "archivetime"=>"2009-09-30 17:54:56.316", "publishtime"=>nil, "lastupdate"=>"2013-04-10 14:02:59.469751", "deletefromarchive"=>"N", "deleteeffectivedate"=>nil, "externalid"=>nil, "isorderonly"=>"H", "versionnumber"=>"3",

"archiveneededflag"=>"Y", "pgeversion"=>"5.0.7", "productiondatetime"=>"2009-06-15 17:59:44",
"rangebeginningdate"=>"2007-09-30", "rangebeginningtime"=>"00:00:00.0000000", "rangeendingdate"=>"2007-10-07",
"rangeendingtime"=>"23:59:59.0000000", "calendardate"=>nil, "timeofday"=>nil, "beginningdatetime"=>"2007-09-30
00:00:00", "endingdatetime"=>"2007-10-07 23:59:59", "daynightflag"=>"Day ", "sizembecsdatagranule"=>"0.142593",
"localgranuleid"=>"MOD10A2.A2007273.h27v06.005.2009166175943.hdf", "localversionid"=>"SCF V5.0.0",
"radiometricdbversion"=>nil, "reprocessingplanned"=>"further update is anticipated",
"reprocessingactual"=>"reprocessed", "geometricdbversion"=>nil, "dapid"=>nil, "dapinsertdate"=>nil},
{"granuleid"=>"68003", "collectionid"=>"42920", "shortname"=>"MOD10A2", "versionid"=>"5", "ecsflag"=>"Y",
"granulestate"=>"AR", "registrationtime"=>"2009-09-30 14:22:47.153", "archivetime"=>"2009-09-30 14:22:48.23",
"publishtime"=>nil, "lastupdate"=>"2013-04-10 14:03:31.849926", "deletefromarchive"=>"N",
"deleteeffectivedate"=>nil, "externalid"=>nil, "isorderonly"=>"H", "versionnumber"=>"1",
"archiveneededflag"=>"Y", "pgeversion"=>"5.0.7", "productiondatetime"=>"2009-06-15 17:59:44",
"rangebeginningdate"=>"2007-09-30", "rangebeginningtime"=>"00:00:00.0000000", "rangeendingdate"=>"2007-10-07",
"rangeendingtime"=>"23:59:59.0000000", "calendardate"=>nil, "timeofday"=>nil, "beginningdatetime"=>"2007-09-30
00:00:00", "endingdatetime"=>"2007-10-07 23:59:59", "daynightflag"=>"Day ", "sizembecsdatagranule"=>"0.142593",
"localgranuleid"=>"MOD10A2.A2007273.h27v06.005.2009166175943.hdf", "localversionid"=>"SCF V5.0.0",
"radiometricdbversion"=>nil, "reprocessingplanned"=>"further update is anticipated",
"reprocessingactual"=>"reprocessed", "geometricdbversion"=>nil, "dapid"=>nil, "dapinsertdate"=>nil}]

```
      shortname = 'MOD10A2'
      versionid = 5
      expected_arglist = [shortname, versionid]
      expected_sql = Sdps::AimDb::GRANULES_BY_SHORTNAME_VERSIONID_SQL
      aim = aim_with_mock_pg(expected_sql, expected_arglist, pg_result)

      actual = aim.find_granules_by_shortname_versionid(shortname, versionid)

      collectionid = actual[0].collectionid
      expect(actual.all?{|g| collectionid == g.collectionid}).to be_true
      expect(actual.all?{|g| shortname == g.shortname}).to be_true
      expect(actual.all?{|g| versionid == g.versionid}).to be_true
    end
  end

  describe '#find_granules_by_localgranuleid' do
    it "finds all granules with the same localgranuleid" do
      pg_result = [{"granuleid"=>"71114", "collectionid"=>"47625", "shortname"=>"AE_Land", "versionid"=>"2",
```

"ecsflag"=>"Y", "granulestate"=>"AR", "registrationtime"=>"2013-03-27 10:29:31.179882",
"archivetime"=>"2013-03-27 10:29:31.423311", "publishtime"=>nil, "lastupdate"=>"2013-10-25 13:20:07.114145",
"deletefromarchive"=>"N", "deleteeffectivedate"=>nil, "externalid"=>nil, "isorderonly"=>"H",
"versionnumber"=>"14", "archiveneededflag"=>"Y", "pgeversion"=>"08", "productiondatetime"=>"2008-09-01
22:39:56", "rangebeginningdate"=>"2008-09-01", "rangebeginningtime"=>"09:22:56.260000",
"rangeendingdate"=>"2008-09-01", "rangeendingtime"=>"10:12:51.540000", "calendardate"=>nil, "timeofday"=>nil,
"beginningdatetime"=>"2008-09-01 09:22:56.26", "endingdatetime"=>"2008-09-01 10:12:51.54", "daynightflag"=>nil,
"sizembecsdatagranule"=>"0.655132293701172", "localgranuleid"=>"AMSR_E_L2_Land_T08_200809010922_D.hdf",
"localversionid"=>nil, "radiometricdbversion"=>nil, "reprocessingplanned"=>nil,
"reprocessingactual"=>"reprocessed once", "geometricdbversion"=>nil, "dapid"=>nil, "dapinsertdate"=>nil},
{"granuleid"=>"71123", "collectionid"=>"47625", "shortname"=>"AE_Land", "versionid"=>"2", "ecsflag"=>"Y",
"granulestate"=>"AR", "registrationtime"=>"2013-03-28 15:12:24.703345", "archivetime"=>"2013-03-28
15:12:24.896577", "publishtime"=>nil, "lastupdate"=>"2013-10-25 13:21:09.896344", "deletefromarchive"=>"N",
"deleteeffectivedate"=>nil, "externalid"=>nil, "isorderonly"=>"H", "versionnumber"=>"15",
"archiveneededflag"=>"Y", "pgeversion"=>"08", "productiondatetime"=>"2008-09-01 22:39:56",
"rangebeginningdate"=>"2008-09-01", "rangebeginningtime"=>"09:22:56.260000", "rangeendingdate"=>"2008-09-01",
"rangeendingtime"=>"10:12:51.540000", "calendardate"=>nil, "timeofday"=>nil, "beginningdatetime"=>"2008-09-01
09:22:56.26", "endingdatetime"=>"2008-09-01 10:12:51.54", "daynightflag"=>nil,
"sizembecsdatagranule"=>"0.655132293701172", "localgranuleid"=>"AMSR_E_L2_Land_T08_200809010922_D.hdf",
"localversionid"=>nil, "radiometricdbversion"=>nil, "reprocessingplanned"=>nil,
"reprocessingactual"=>"reprocessed once", "geometricdbversion"=>nil, "dapid"=>nil, "dapinsertdate"=>nil},
{"granuleid"=>"71127", "collectionid"=>"47625", "shortname"=>"AE_Land", "versionid"=>"2", "ecsflag"=>"Y",
"granulestate"=>"AR", "registrationtime"=>"2013-03-28 15:38:25.330162", "archivetime"=>"2013-03-28
15:42:09.064558", "publishtime"=>"2013-10-25 13:21:51.436097", "lastupdate"=>"2013-10-25 13:21:51.436097",
"deletefromarchive"=>"N", "deleteeffectivedate"=>nil, "externalid"=>nil, "isorderonly"=>nil,
"versionnumber"=>"16", "archiveneededflag"=>"Y", "pgeversion"=>"08", "productiondatetime"=>"2008-09-01
22:39:56", "rangebeginningdate"=>"2008-09-01", "rangebeginningtime"=>"09:22:56.260000",
"rangeendingdate"=>"2008-09-01", "rangeendingtime"=>"10:12:51.540000", "calendardate"=>nil, "timeofday"=>nil,
"beginningdatetime"=>"2008-09-01 09:22:56.26", "endingdatetime"=>"2008-09-01 10:12:51.54", "daynightflag"=>nil,
"sizembecsdatagranule"=>"0.655132293701172", "localgranuleid"=>"AMSR_E_L2_Land_T08_200809010922_D.hdf",
"localversionid"=>nil, "radiometricdbversion"=>nil, "reprocessingplanned"=>nil,
"reprocessingactual"=>"reprocessed once", "geometricdbversion"=>nil, "dapid"=>nil, "dapinsertdate"=>nil}]

```
      lgid = 'AMSR_E_L2_Land_T08_200809010922_D.hdf'
      expected_arglist = [lgid]
      expected_sql = Sdps::AimDb::GRANULES_BY_LOCALGRANULEID_SQL
      aim = aim_with_mock_pg(expected_sql, expected_arglist, pg_result)
```

```
      actual = aim.find_granules_by_localgranuleid(lgid)

      expect(actual).to_not be_empty
      expect(actual.all?{|g| g.is_a? Sdps::AimDb::AmGranule}).to be_true
      expect(actual.all?{|g| lgid == g.localgranuleid}).to be_true
    end
  end

  describe '#find_granules_by_grfilename' do
    it "finds all granules with the same userdatafile" do
      pg_result = [{"granuleid"=>"71114", "collectionid"=>"47625", "shortname"=>"AE_Land", "versionid"=>"2",
"ecsflag"=>"Y", "granulestate"=>"AR", "registrationtime"=>"2013-03-27 10:29:31.179882",
"archivetime"=>"2013-03-27 10:29:31.423311", "publishtime"=>nil, "lastupdate"=>"2013-10-25 13:20:07.114145",
"deletefromarchive"=>"N", "deleteeffectivedate"=>nil, "externalid"=>nil, "isorderonly"=>"H",
"versionnumber"=>"14", "archiveneededflag"=>"Y", "pgeversion"=>"08", "productiondatetime"=>"2008-09-01
22:39:56", "rangebeginningdate"=>"2008-09-01", "rangebeginningtime"=>"09:22:56.260000",
"rangeendingdate"=>"2008-09-01", "rangeendingtime"=>"10:12:51.540000", "calendardate"=>nil, "timeofday"=>nil,
"beginningdatetime"=>"2008-09-01 09:22:56.26", "endingdatetime"=>"2008-09-01 10:12:51.54", "daynightflag"=>nil,
"sizembecsdatagranule"=>"0.655132293701172", "localgranuleid"=>"AMSR_E_L2_Land_T08_200809010922_D.hdf",
"localversionid"=>nil, "radiometricdbversion"=>nil, "reprocessingplanned"=>nil,
"reprocessingactual"=>"reprocessed once", "geometricdbversion"=>nil, "dapid"=>nil, "dapinsertdate"=>nil,
"internalfilename"=>":SC:AE_Land.002:71114:1.HDF-EOS", "userdatafile"=>"AMSR_E_L2_Land_T08_200809010922_D.hdf",
"onlinefilename"=>"AMSR_E_L2_Land_T08_200809010922_D_14.hdf",
"directorypath"=>"/AE_Land.002ZRxNmubb/2008.09.01/", "filesize"=>"686956", "checksum"=>"995056962",
"checksumtypeid"=>"2", "checksumoriginid"=>"4", "archivechecksumlastverified"=>"2013-03-27 10:29:31.423311",
"archivechecksumstatus"=>"N", "onlinechecksumlastverified"=>"2013-03-27 10:29:31.122611",
"onlinechecksumstatus"=>"N", "originalproviderchecksumtypeid"=>nil, "md5"=>"94d9e9724f800fd0095934ab22b93b3d"},
{"granuleid"=>"71123", "collectionid"=>"47625", "shortname"=>"AE_Land", "versionid"=>"2", "ecsflag"=>"Y",
"granulestate"=>"AR", "registrationtime"=>"2013-03-28 15:12:24.703345", "archivetime"=>"2013-03-28
15:12:24.896577", "publishtime"=>nil, "lastupdate"=>"2013-10-25 13:21:09.896344", "deletefromarchive"=>"N",
"deleteeffectivedate"=>nil, "externalid"=>nil, "isorderonly"=>"H", "versionnumber"=>"15",
"archiveneededflag"=>"Y", "pgeversion"=>"08", "productiondatetime"=>"2008-09-01 22:39:56",
"rangebeginningdate"=>"2008-09-01", "rangebeginningtime"=>"09:22:56.260000", "rangeendingdate"=>"2008-09-01",
"rangeendingtime"=>"10:12:51.540000", "calendardate"=>nil, "timeofday"=>nil, "beginningdatetime"=>"2008-09-01
09:22:56.26", "endingdatetime"=>"2008-09-01 10:12:51.54", "daynightflag"=>nil,
```

"sizembecsdatagranule"=>"0.655132293701172", "localgranuleid"=>"AMSR_E_L2_Land_T08_200809010922_D.hdf",
"localversionid"=>nil, "radiometricdbversion"=>nil, "reprocessingplanned"=>nil,
"reprocessingactual"=>"reprocessed once", "geometricdbversion"=>nil, "dapid"=>nil, "dapinsertdate"=>nil,
"internalfilename"=>":SC:AE_Land.002:71123:1.HDF-EOS", "userdatafile"=>"AMSR_E_L2_Land_T08_200809010922_D.hdf",
"onlinefilename"=>"AMSR_E_L2_Land_T08_200809010922_D_15.hdf",
"directorypath"=>"/AE_Land.002ZRxNmubb/2008.09.01/", "filesize"=>"686956", "checksum"=>"995056962",
"checksumtypeid"=>"2", "checksumoriginid"=>"4", "archivechecksumlastverified"=>"2013-03-28 15:12:24.896577",
"archivechecksumstatus"=>"N", "onlinechecksumlastverified"=>"2013-03-28 15:12:24.44831",
"onlinechecksumstatus"=>"P", "originalproviderchecksumtypeid"=>nil, "md5"=>"94d9e9724f800fd0095934ab22b93b3d"},
{"granuleid"=>"71127", "collectionid"=>"47625", "shortname"=>"AE_Land", "versionid"=>"2", "ecsflag"=>"Y",
"granulestate"=>"AR", "registrationtime"=>"2013-03-28 15:38:25.330162", "archivetime"=>"2013-03-28
15:42:09.064558", "publishtime"=>"2013-10-25 13:21:51.436097", "lastupdate"=>"2013-10-25 13:21:51.436097",
"deletefromarchive"=>"N", "deleteeffectivedate"=>nil, "externalid"=>nil, "isorderonly"=>nil,
"versionnumber"=>"16", "archiveneededflag"=>"Y", "pgeversion"=>"08", "productiondatetime"=>"2008-09-01
22:39:56", "rangebeginningdate"=>"2008-09-01", "rangebeginningtime"=>"09:22:56.260000",
"rangeendingdate"=>"2008-09-01", "rangeendingtime"=>"10:12:51.540000", "calendardate"=>nil, "timeofday"=>nil,
"beginningdatetime"=>"2008-09-01 09:22:56.26", "endingdatetime"=>"2008-09-01 10:12:51.54", "daynightflag"=>nil,
"sizembecsdatagranule"=>"0.655132293701172", "localgranuleid"=>"AMSR_E_L2_Land_T08_200809010922_D.hdf",
"localversionid"=>nil, "radiometricdbversion"=>nil, "reprocessingplanned"=>nil,
"reprocessingactual"=>"reprocessed once", "geometricdbversion"=>nil, "dapid"=>nil, "dapinsertdate"=>nil,
"internalfilename"=>":SC:AE_Land.002:71127:1.HDF-EOS", "userdatafile"=>"AMSR_E_L2_Land_T08_200809010922_D.hdf",
"onlinefilename"=>"AMSR_E_L2_Land_T08_200809010922_D.hdf", "directorypath"=>"/AE_Land.002/2008.09.01/",
"filesize"=>"686956", "checksum"=>"995056962", "checksumtypeid"=>"2", "checksumoriginid"=>"4",
"archivechecksumlastverified"=>"2013-03-28 15:42:09.064558", "archivechecksumstatus"=>"N",
"onlinechecksumlastverified"=>"2013-03-28 15:38:25.236931", "onlinechecksumstatus"=>"P",
"originalproviderchecksumtypeid"=>nil, "md5"=>"94d9e9724f800fd0095934ab22b93b3d"}]

```
      userdatafile = 'AMSR_E_L2_Land_T08_200809010922_D.hdf'
      expected_arglist = [userdatafile]
      expected_sql = Sdps::AimDb::GRANULES_BY_GRFILENAME_SQL
      aim = aim_with_mock_pg(expected_sql, expected_arglist, pg_result)

      actual = aim.find_granules_by_grfilename(userdatafile)

      expect(actual).to_not be_empty
      collectionid = actual[0].collectionid
      shortname = actual[0].shortname
```

```
      versionid = actual[0].versionid
      expect(actual.all?{|g| collectionid == g.collectionid}).to be_true
      expect(actual.all?{|g| shortname == g.shortname}).to be_true
      expect(actual.all?{|g| versionid == g.versionid}).to be_true
    end
  end

  describe '#find_associated_browse' do
    it "finds browse granules associated with the granuleid" do
      pg_result = [{"browseid"=>"70557", "externalid"=>nil, "misbrid"=>nil, "granulestate"=>"AR",
"registrationtime"=>"2011-12-01 15:28:26.32", "archivetime"=>"2011-12-01 15:28:26.383", "browsesize"=>"24350",
"lastupdate"=>"2011-12-01 15:28:26.3", "deleteeffectivedate"=>nil, "archiveneededflag"=>"Y",
"publishtime"=>"2013-04-10 14:03:33.951252", "isorderonly"=>nil, "versionnumber"=>nil, "browsedescription"=>nil,
"browseproductiondatetime"=>"2011-12-01 20:28:26.18"}]
      scienceid = 70556
      expected_arglist = [scienceid]
      expected_sql = Sdps::AimDb::BROWSE_BY_SCIENCEID_SQL
      aim = aim_with_mock_pg(expected_sql, expected_arglist, pg_result)

      actual = aim.find_associated_browse(scienceid)

      expect(actual.length).to eq(1)
      expect(actual[0]).to be_instance_of(Sdps::AimDb::AmBrowse)
    end
  end

  describe '#find_associated_ph' do
    it "finds PH granules associated with the granuleid" do
      pg_result = [{"granuleid"=>"71129", "collectionid"=>"1068", "shortname"=>"PH", "versionid"=>"1",
"ecsflag"=>"Y", "granulestate"=>"AR", "registrationtime"=>"2013-03-28 16:53:36.34566",
"archivetime"=>"2013-03-28 16:53:36.653766", "publishtime"=>"2013-10-25 13:20:06.852779",
"lastupdate"=>"2013-10-25 13:20:06.852779", "deletefromarchive"=>"N", "deleteeffectivedate"=>nil,
"externalid"=>nil, "isorderonly"=>nil, "versionnumber"=>"1", "archiveneededflag"=>"Y", "pgeversion"=>nil,
"productiondatetime"=>nil, "rangebeginningdate"=>nil, "rangebeginningtime"=>nil, "rangeendingdate"=>nil,
"rangeendingtime"=>nil, "calendardate"=>nil, "timeofday"=>nil, "beginningdatetime"=>nil, "endingdatetime"=>nil,
"daynightflag"=>nil, "sizembecsdatagranule"=>"0.000166893005371094", "localgranuleid"=>nil,
```

```
"localversionid"=>nil, "radiometricdbversion"=>nil, "reprocessingplanned"=>nil, "reprocessingactual"=>nil,
"geometricdbversion"=>nil, "dapid"=>nil, "dapinsertdate"=>nil}]
      scienceid = 70421
      expected_arglist=[scienceid]
      expected_sql = Sdps::AimDb::ASSOCIATED_PH_SQL
      aim = aim_with_mock_pg(expected_sql, expected_arglist, pg_result)

      actual = aim.find_associated_ph(scienceid)

      expect(actual.length).to eq(1)
      ph = actual[0]
      expect(ph).to be_instance_of(Sdps::AimDb::AmGranule)
      expect(ph.shortname).to eq('PH')
      expect(ph.versionid).to eq(1)
      expect(ph.granuleid).to eq(71129)
    end
  end

  describe '#find_associated_qa' do
    it "finds QA granules associated with the granuleid" do
      pg_result = [{"granuleid"=>"71130", "collectionid"=>"1067", "shortname"=>"QA", "versionid"=>"1",
"ecsflag"=>"Y", "granulestate"=>"AR", "registrationtime"=>"2013-03-28 15:42:30.137858",
"archivetime"=>"2013-03-28 16:53:36.206391", "publishtime"=>"2013-10-25 13:21:50.969142",
"lastupdate"=>"2013-10-25 13:21:50.969142", "deletefromarchive"=>"N", "deleteeffectivedate"=>nil,
"externalid"=>nil, "isorderonly"=>nil, "versionnumber"=>"1", "archiveneededflag"=>"Y", "pgeversion"=>nil,
"productiondatetime"=>nil, "rangebeginningdate"=>nil, "rangebeginningtime"=>nil, "rangeendingdate"=>nil,
"rangeendingtime"=>nil, "calendardate"=>nil, "timeofday"=>nil, "beginningdatetime"=>nil, "endingdatetime"=>nil,
"daynightflag"=>nil, "sizembecsdatagranule"=>"0.00172042846679688", "localgranuleid"=>nil,
"localversionid"=>nil, "radiometricdbversion"=>nil, "reprocessingplanned"=>nil, "reprocessingactual"=>nil,
"geometricdbversion"=>nil, "dapid"=>nil, "dapinsertdate"=>nil}]
      scienceid = 70421
      expected_arglist=[scienceid]
      expected_sql = Sdps::AimDb::ASSOCIATED_QA_SQL
      aim = aim_with_mock_pg(expected_sql, expected_arglist, pg_result)

      actual = aim.find_associated_qa(scienceid)
```

```
    expect(actual.length).to eq(1)
    qa = actual[0]
    expect(qa).to be_instance_of(Sdps::AimDb::AmGranule)
    expect(qa.shortname).to eq('QA')
    expect(qa.versionid).to eq(1)
    expect(qa.granuleid).to eq(71130)
  end

end

describe '#find_associated_hdfmap' do
  it "finds HDF_MAP granules associated with the granuleid" do
    pg_result = [{
      "granuleid" => "198540",
      "collectionid" => "198324",
      "shortname" => "HDF_MAP",
      "versionid" => "1",
      "ecsflag" => "Y",
      "granulestate" => "AR",
      "registrationtime" => "2013-08-15 18:23:54.289403",
      "archivetime" => "2013-08-15 18:23:54.32918",
      "publishtime" => "2013-08-15 18:24:18.619769",
      "lastupdate" => "2013-08-15 18:24:18.619769",
      "deletefromarchive" => "N",
      "deleteeffectivedate" => nil,
      "externalid" => nil,
      "isorderonly" => nil,
      "versionnumber" => "1",
      "archiveneededflag" => "Y",
      "pgeversion" => nil,
      "productiondatetime" => nil,
      "rangebeginningdate" => nil,
      "rangebeginningtime" => nil,
      "rangeendingdate" => nil,
      "rangeendingtime" => nil,
```

```
        "calendardate" => nil,
        "timeofday" => nil,
        "beginningdatetime" => nil,
        "endingdatetime" => nil,
        "daynightflag" => nil,
        "sizembecsdatagranule" => "0.00956153869628906",
        "localgranuleid" => nil,
        "localversionid" => nil,
        "radiometricdbversion" => nil,
        "reprocessingplanned" => nil,
        "reprocessingactual" => nil,
        "geometricdbversion" => nil,
        "dapid" => nil,
        "dapinsertdate" => nil
      }]
      row = pg_result[0]
      scienceid = 7770777
      expected_arglist=[scienceid]
      expected_sql = Sdps::AimDb::ASSOCIATED_HDFMAP_SQL
      aim = aim_with_mock_pg(expected_sql, expected_arglist, pg_result)

      actual = aim.find_associated_hdfmap(scienceid)

      expect(actual.length).to eq(1)
      hdfmap = actual[0]
      expect(hdfmap).to be_instance_of(Sdps::AimDb::AmGranule)
      expect(hdfmap.shortname).to eq(row["shortname"])
      expect(hdfmap.versionid).to eq(row["versionid"].to_i)
      expect(hdfmap.granuleid).to eq(row["granuleid"].to_i)
    end
  end

end
Current Test Script: logentry_spec.rb
require 'spec_helper'
```

```
describe Sdps::TcpProxyLogger::LogEntry do

  describe ".parse" do
    it "parses a plain TCP proxy log entry" do
      timestamp = '2013-10-31T15:59:27.130070'
      severity = 'INFO'
      uuid = '2097faeb829b73684be1792dd6efd555'
      msg = 'Receiving from 127.0.0.1:60782'
      line = "{:timestamp=>\"#{timestamp}\", :severity=>\"#{severity}\", :uuid=>\"#{uuid}\", :msg=>\"#{msg}\"}"
      entry = Sdps::TcpProxyLogger::LogEntry.parse(line)
      expect(entry.timestamp).to eq(timestamp)
      expect(entry.severity).to eq(severity)
      expect(entry.uuid).to eq(uuid)
      expect(entry.msg).to eq(msg)
      expect(entry.request?).to be_false
      expect(entry.request).to be_nil
      expect(entry.response?).to be_false
      expect(entry.response).to be_nil
    end

    it "parses a TCP request log entry" do
      timestamp = '2013-10-31T15:59:27.130541'
      severity = 'INFO'
      uuid = '2097faeb829b73684be1792dd6efd555'
      request = "PUT
/catalog-rest/providers/EDF_DEV06/datasets/MODIS%2FAqua%20Thermal%20Anomalies%2FFire%205-Min%20L2%20Swath%201km%
20V005 HTTP/1.1\r\nEcho-Token: A7B6FC42-D826-9579-87EB-F42AA24052BF\r\nContent-Length: 6838\r\nContent-Type:
text/plain; charset=ISO-8859-1\r\nHost: localhost:3600\r\nUser-Agent: BMGT Ingest Client\r\n\r\n"
      line = "{:timestamp=>\"#{timestamp}\", :severity=>\"#{severity}\", :uuid=>\"#{uuid}\",
:request=>#{request.inspect}}"
      entry = Sdps::TcpProxyLogger::LogEntry.parse(line)
      expect(entry.timestamp).to eq(timestamp)
      expect(entry.severity).to eq(severity)
      expect(entry.uuid).to eq(uuid)
      expect(entry.msg).to be_nil
      expect(entry.request?).to be_true
```

```
      expect(entry.request).to eq(request)
      expect(entry.response?).to be_false
      expect(entry.response).to be_nil
    end

    it "parses a TCP response log entry" do
      timestamp = '2013-10-31T15:59:27.759703'
      severity = 'INFO'
      uuid = '2097faeb829b73684be1792dd6efd555'
      response = "HTTP/1.1 200 OK\r\nServer: Apache-Coyote/1.1\r\nCatalog-Location:
https://testbed.echo.nasa.gov/catalog-rest/echo_catalog/datasets/C1000000024-EDF_DEV06\r\nLocation:
https://testbed.echo.nasa.gov/catalog-rest/providers/EDF_DEV06/datasets/MODIS%2FAqua%20Thermal%20Anomalies%2FFir
e%205-Min%20L2%20Swath%201km%20V005\r\nX-UA-Compatible: IE=Edge,chrome=1\r\nETag:
\"7215ee9c7d9dc229d2921a40e899ec5f\"\r\nCache-Control: max-age=0, private, must-revalidate\r\nX-Request-Id:
c867a2d090a438bf5fa0a19a55a5abd2\r\nX-Runtime: 0.589000\r\nDate: Thu, 31 Oct 2013 19:59:27 GMT\r\nX-Rack-Cache:
invalidate, pass\r\nContent-Type: application/xml;charset=utf-8\r\nContent-Length: 1\r\n\r\n "
      line = "{:timestamp=>\"#{timestamp}\", :severity=>\"#{severity}\", :uuid=>\"#{uuid}\",
:response=>#{response.inspect}}"
      entry = Sdps::TcpProxyLogger::LogEntry.parse(line)
      expect(entry.timestamp).to eq(timestamp)
      expect(entry.severity).to eq(severity)
      expect(entry.uuid).to eq(uuid)
      expect(entry.msg).to be_nil
      expect(entry.request?).to be_false
      expect(entry.request).to be_nil
      expect(entry.response?).to be_true
      expect(entry.response).to eq(response)
    end

  end

  describe "#time" do
    it "returns a Time object matching the timestamp" do
      timestamp = '2013-10-31T15:59:27.130070'
      time = Time.strptime(timestamp, "%Y-%m-%dT%H:%M:%S.%N")
      line = "{:timestamp=>\"#{timestamp}\", :severity=>\"INFO\", :uuid=>\"2097faeb829b73684be1792dd6efd555\",
```

```
:msg=>\"Receiving from 127.0.0.1:60782\"}"
      entry = Sdps::TcpProxyLogger::LogEntry.parse(line)
      expect(entry.timestamp).to eq(timestamp)
      expect(entry.time).to eq(time)
    end
  end

  describe "#request?" do

    it "returns true when the log entry is for a TCP request" do
      line = '{:timestamp=>"2013-10-31T15:50:22.674946", :severity=>"INFO",
:uuid=>"a0ab882d08af74e60f9f77b8b3a50614", :request=>"DELETE
/echo-rest/tokens/746F5064-E5C5-7CB1-BF21-EEE9EF910DA2 HTTP/1.1\r\nEcho-Token:
746F5064-E5C5-7CB1-BF21-EEE9EF910DA2\r\nHost: localhost:3600\r\nUser-Agent: BMGT Token Client\r\n\r\n"}'
      entry = Sdps::TcpProxyLogger::LogEntry.parse(line)
      expect(entry.request?).to be_true
    end

    it "returns false when the log entry is for a TCP response" do
      line = '{:timestamp=>"2013-10-31T15:50:23.083947", :severity=>"INFO",
:uuid=>"5a60a340e80d695d63a68568fb358c39", :response=>"HTTP/1.1 204 No Content\r\nServer:
Apache-Coyote/1.1\r\nCache-Control: no-cache\r\nX-UA-Compatible: IE=Edge,chrome=1\r\nSet-Cookie:
_EchoRestRails_session=BAh7BiIPc2Vzc2lvbl9pZCIlMmYxMjhlMmVhNWJiYjRmM2IwYzIwNmNkOGQ2NTMyYTA%3D--469997dd6f7e54395
1a27c666ebd0b961e63ae5c; path=/; HttpOnly\r\nX-Runtime: 0.248000\r\nDate: Thu, 31 Oct 2013 19:50:23
GMT\r\n\r\n"}'
      entry = Sdps::TcpProxyLogger::LogEntry.parse(line)
      expect(entry.request?).to be_false
    end

    it "returns false for a plain log entry" do
      line = '{:timestamp=>"2013-10-31T15:50:23.084777", :severity=>"INFO",
:uuid=>"5a60a340e80d695d63a68568fb358c39", :msg=>"Local end closed connection."}'
      entry = Sdps::TcpProxyLogger::LogEntry.parse(line)
      expect(entry.request?).to be_false
    end
```

```
  end

  describe "#response?" do

    it "returns true when the log entry is for a TCP response" do
      line = '{:timestamp=>"2013-10-31T15:50:23.083947", :severity=>"INFO",
:uuid=>"5a60a340e80d695d63a68568fb358c39", :response=>"HTTP/1.1 204 No Content\r\nServer:
Apache-Coyote/1.1\r\nCache-Control: no-cache\r\nX-UA-Compatible: IE=Edge,chrome=1\r\nSet-Cookie:
_EchoRestRails_session=BAh7BiIPc2Vzc2lvbl9pZCIlMmYxMjhlMmVhNWJiYjRmM2IwYzIwNmNkOGQ2NTMyYTA%3D--469997dd6f7e54395
1a27c666ebd0b961e63ae5c; path=/; HttpOnly\r\nX-Runtime: 0.248000\r\nDate: Thu, 31 Oct 2013 19:50:23
GMT\r\n\r\n"}'
      entry = Sdps::TcpProxyLogger::LogEntry.parse(line)
      expect(entry.response?).to be_true
    end

    it "returns false when the log entry is for a TCP request" do
      line = '{:timestamp=>"2013-10-31T15:50:22.674946", :severity=>"INFO",
:uuid=>"a0ab882d08af74e60f9f77b8b3a50614", :request=>"DELETE
/echo-rest/tokens/746F5064-E5C5-7CB1-BF21-EEE9EF910DA2 HTTP/1.1\r\nEcho-Token:
746F5064-E5C5-7CB1-BF21-EEE9EF910DA2\r\nHost: localhost:3600\r\nUser-Agent: BMGT Token Client\r\n\r\n"}'
      entry = Sdps::TcpProxyLogger::LogEntry.parse(line)
      expect(entry.response?).to be_false
    end

    it "returns false for a plain log entry" do
      line = '{:timestamp=>"2013-10-31T15:50:23.084777", :severity=>"INFO",
:uuid=>"5a60a340e80d695d63a68568fb358c39", :msg=>"Local end closed connection."}'
      entry = Sdps::TcpProxyLogger::LogEntry.parse(line)
      expect(entry.response?).to be_false
    end

  end

end
Current Test Script: dbrecord_spec.rb
require 'spec_helper'
```

```
require 'time'

class MyDbRecord < Sdps::DbRecord

  db_attr :id, 'bigint'
  db_attr :foreignid, 'bigint'
  db_attr :versionid, 'smallint'
  db_attr :versionnumber, 'integer'
  db_attr :name, 'character varying(8)'
  db_attr :flag, 'character(1)'
  db_attr :inserttime, 'timestamp without time zone'
  db_attr :lastupdate, 'timestamp without time zone'
  db_attr :begindate, 'date'
  db_attr :enddate, 'date'
  db_attr :size, 'double precision'
  db_attr :weight, 'double precision'

end

describe Sdps::DbRecord do
  describe "db_attr" do

    context "given a tuple with no nils" do
      before(:all) do
        @tuple = {
          id: 123456,
          foreignid: 654321,
          versionid: 255,
          versionnumber: 13,
          name: "Example",
          flag: "Y",
          inserttime: "2013-10-17 08:38:12",
          lastupdate: "2013-10-17 08:38:12.345678",
          begindate: "2013-10-01",
          enddate: "2013-10-07",
          size: "0.000123",
```

```
    weight: "7.654321"
  }
  @my_rec = MyDbRecord.new(@tuple)
end
context "with type 'bigint'" do
  it "defines an integer field" do
    expect(@my_rec.id).to eq(@tuple[:id].to_i)
    expect(@my_rec.foreignid).to eq(@tuple[:foreignid].to_i)
  end
end
context "with type 'smallint'" do
  it "defines an integer field" do
    expect(@my_rec.versionid).to eq(@tuple[:versionid].to_i)
  end
end
context "with type 'integer'" do
  it "defines an integer field" do
    expect(@my_rec.versionnumber).to eq(@tuple[:versionnumber].to_i)
  end
end
context "with type 'character varying'" do
  it "defines a string field" do
    expect(@my_rec.name).to eq(@tuple[:name])
  end
end
context "with type 'character'" do
  it "defines a string field" do
    expect(@my_rec.flag).to eq(@tuple[:flag])
  end
end
context "with type 'timestamp without time zone'" do
  it "defines a datetime field" do
    inserttime = Time.strptime(@tuple[:inserttime], '%Y-%m-%d %H:%M:%S')
    lastupdate = Time.strptime(@tuple[:lastupdate], '%Y-%m-%d %H:%M:%S.%N')
    expect(@my_rec.inserttime).to eq(inserttime)
    expect(@my_rec.lastupdate).to eq(lastupdate)
```

```
    end
  end
  context "with type 'date'" do
    it "defines a date field" do
      begindate = Date.strptime(@tuple[:begindate])
      enddate = Date.strptime(@tuple[:enddate])
      expect(@my_rec.begindate).to eq(begindate)
      expect(@my_rec.enddate).to eq(enddate)
    end
  end
  context "with type 'double precision'" do
    it "defines a float field" do
      expect(@my_rec.size).to eq(@tuple[:size].to_f)
      expect(@my_rec.weight).to eq(@tuple[:weight].to_f)
    end
  end
end

context "given a tuple with some nils" do
  before(:all) do
    @tuple = {
      id: 123456,
      foreignid: nil,
      versionid: 255,
      versionnumber: nil,
      name: "Example",
      flag: nil,
      inserttime: "2013-10-17 08:38:12",
      lastupdate: nil,
      begindate: "2013-10-01",
      enddate: nil,
      size: "0.000123",
      weight: nil
    }
    @my_rec = MyDbRecord.new(@tuple)
  end
```

```
context "with type 'bigint'" do
  it "defines an integer field" do
    expect(@my_rec.id).to eq(@tuple[:id].to_i)
    expect(@my_rec.foreignid).to be_nil
  end
end
context "with type 'smallint'" do
  it "defines an integer field" do
    expect(@my_rec.versionid).to eq(@tuple[:versionid].to_i)
  end
end
context "with type 'integer'" do
  it "defines an integer field" do
    expect(@my_rec.versionnumber).to be_nil
  end
end
context "with type 'character varying'" do
  it "defines a string field" do
    expect(@my_rec.name).to eq(@tuple[:name])
  end
end
context "with type 'character'" do
  it "defines a string field" do
    expect(@my_rec.flag).to be_nil
  end
end
context "with type 'timestamp without time zone'" do
  it "defines a datetime field" do
    inserttime = Time.strptime(@tuple[:inserttime], '%Y-%m-%d %H:%M:%S')
    expect(@my_rec.inserttime).to eq(inserttime)
    expect(@my_rec.lastupdate).to be_nil
  end
end
context "with type 'date'" do
  it "defines a date field" do
    begindate = Date.strptime(@tuple[:begindate])
```

```
      expect(@my_rec.begindate).to eq(begindate)
      expect(@my_rec.enddate).to be_nil
    end
  end
  context "with type 'double precision'" do
    it "defines a float field" do
      expect(@my_rec.size).to eq(@tuple[:size].to_f)
      expect(@my_rec.weight).to be_nil
    end
  end

end

context "given a tuple with other nils" do
  before(:all) do
    @tuple = {
      id: nil,
      foreignid: 654321,
      versionid: nil,
      versionnumber: 13,
      name: nil,
      flag: "Y",
      inserttime: nil,
      lastupdate: "2013-10-17 08:38:12.345678",
      begindate: nil,
      enddate: "2013-10-07",
      size: nil,
      weight: "7.654321"
    }
    @my_rec = MyDbRecord.new(@tuple)
  end
  context "with type 'bigint'" do
    it "defines an integer field" do
      expect(@my_rec.id).to be_nil
      expect(@my_rec.foreignid).to eq(@tuple[:foreignid].to_i)
    end
```

```
end
context "with type 'smallint'" do
  it "defines an integer field" do
    expect(@my_rec.versionid).to be_nil
  end
end
context "with type 'integer'" do
  it "defines an integer field" do
    expect(@my_rec.versionnumber).to eq(@tuple[:versionnumber].to_i)
  end
end
context "with type 'character varying'" do
  it "defines a string field" do
    expect(@my_rec.name).to be_nil
  end
end
context "with type 'character'" do
  it "defines a string field" do
    expect(@my_rec.flag).to eq(@tuple[:flag])
  end
end
context "with type 'timestamp without time zone'" do
  it "defines a datetime field" do
    lastupdate = Time.strptime(@tuple[:lastupdate], '%Y-%m-%d %H:%M:%S.%N')
    expect(@my_rec.inserttime).to be_nil
    expect(@my_rec.lastupdate).to eq(lastupdate)
  end
end
context "with type 'date'" do
  it "defines a date field" do
    enddate = Date.strptime(@tuple[:enddate])
    expect(@my_rec.begindate).to be_nil
    expect(@my_rec.enddate).to eq(enddate)
  end
end
context "with type 'double precision'" do
```

```
        it "defines a float field" do
          expect(@my_rec.size).to be_nil
          expect(@my_rec.weight).to eq(@tuple[:weight].to_f)
        end
      end
    end

  end
end
Current Test Script: run_nighly_regression_tests.rb
#!/usr/bin/env ruby
supported_modes = ['OPS', 'DEV08','DEV07','DEV04','DEV06']
mode = ENV['MODE']
if mode.nil? or mode.empty? or mode.strip.size.zero?
  puts "\n You should set MODE environment variable to run the tests\n"
  exit
elsif supported_modes.include?(mode) == false
  puts "\n #{mode} is not supported by autotest.\n"
  puts "\n Currently supported modes are #{supported_modes.to_s}.\n"
  exit
elsif ENV['USER'] != 'cmshared'
  puts "\n\nYou must login as cmshared to run the tests\n\n"
  exit
end

time = Time.now.strftime('%Y%m%d-%H%M%S')

log_dir = "/workingdata/cucumber/#{mode}/logs/"
my_log = "#{log_dir}nightly_regression_test-#{time}.log"
system("touch #{my_log}")
system("echo 'Started Nightly Autotest at #{time}' >> #{my_log}")
system("echo '====================================' >> #{my_log}")

smap_egi_features = [ 'egi_smapl1_format.feature',
                      'egi_smapl2_smp_format.feature',
                      'egi_smapl2_smp_parameter_subset.feature',
```

```
                       'egi_smapl3_smap_parameter_subset_reprojection.feature',
                       'egi_smapl4_smau_parameter_subset_reprojection.feature',
                       'egi_smapl4_smlm_parameter_subset_reprojection.feature',
                       'egi_smapl4_smlm_reprojection.feature']

viirs_egi_features = [ 'egi_viirs_l2_snow_format.feature',
                        'egi_viirs_l2_snow_parameter_subset.feature',
                        'egi_viirs_l2_snow_spatial_subset.feature',
                        'egi_viirs_l2_snow_reprojection.feature',
                        'egi_viirs_l3_vegetation_format.feature',
                        'egi_viirs_l3_vegetation_parameter_subset.feature']

glas_egi_features = ['egi_glas_spatial_subset.feature',
                      'egi_glas_format.feature']

modis_egi_features = ['egi_modis_l2_aqua_seaice_parameter_subset.feature']

atlas_egi_features = ['egi_atlas_parameter_subset.feature']



smap_egi_features.each do |smap_feature|
   system("cucumber --profile email_log features/DataAccess/egi/smap/#{smap_feature}")
   system("echo 'test: #{smap_feature}\n' >> #{my_log}")
   system("ls #{log_dir}#{smap_feature}_*log | tail -1 >> #{my_log}")
   system("ls #{log_dir}#{smap_feature}_*log | tail -1 | xargs tail -3 >> #{my_log}")
   system("echo '-------------------------------------' >> #{my_log}")
end

viirs_egi_features.each do |viirs_feature|
   system("cucumber --profile email_log features/DataAccess/egi/viirs/#{viirs_feature}")
   system("echo 'test: #{viirs_feature}\n' >> #{my_log}")
   system("ls #{log_dir}#{viirs_feature}_*log | tail -1 >> #{my_log}")
   system("ls #{log_dir}#{viirs_feature}_*log | tail -1 | xargs tail -3 >> #{my_log}")
   system("echo '-------------------------------------' >> #{my_log}")
end
```

```
glas_egi_features.each do |glas_feature|
   system("cucumber --profile email_log features/DataAccess/egi/glas/#{glas_feature}")
   system("echo 'test: #{glas_feature}\n' >> #{my_log}")
   system("ls #{log_dir}#{glas_feature}_*log | tail -1 >> #{my_log}")
   system("ls #{log_dir}#{glas_feature}_*log | tail -1 | xargs tail -3 >> #{my_log}")
   system("echo '-------------------------------------' >> #{my_log}")
end

modis_egi_features.each do |modis_feature|
   system("cucumber --profile email_log features/DataAccess/egi/modis/#{modis_feature}")
   system("echo 'test: #{modis_feature}\n' >> #{my_log}")
   system("ls #{log_dir}#{modis_feature}_*log | tail -1 >> #{my_log}")
   system("ls #{log_dir}#{modis_feature}_*log | tail -1 | xargs tail -3 >> #{my_log}")
   system("echo '-------------------------------------' >> #{my_log}")
end

atlas_egi_features.each do |atlas_feature|
   system("cucumber --profile email_log features/DataAccess/egi/atlas/#{atlas_feature}")
   system("echo 'test: #{atlas_feature}\n' >> #{my_log}")
   system("ls #{log_dir}#{atlas_feature}_*log | tail -1 >> #{my_log}")
   system("ls #{log_dir}#{atlas_feature}_*log | tail -1 | xargs tail -3 >> #{my_log}")
   system("echo '-------------------------------------' >> #{my_log}")
end


system("cat  #{my_log}  | mail -s \"Nightly AUTOTEST Results for #{mode} `date +%D`\" jonathon.soohoo@nasa.gov
elizabeth.o.ajayi@nasa.gov Sudha.Murthy@nasa.gov hung.q.dinh@nasa.gov")
Current Test Script: mode_admin.rb
#!/usr/bin/env ruby

require_relative '../../lib/sdps/mode_admin_lib'

# Script: ModeAdmin.rb

# Author: Jonathon Soohoo
```

```
# Created: 10.15.13
# Description: ModeAdmin allows users to start and stop modes.
#
# ModeAdmin uses the following parameters
# mode_admin.rb  <start_mode>  <MODE> [<MODE 2> ... <MODE N>]        or
#              <start_view>  <MODE> <view>                          or
#              <stop_mode>   <MODE> [<MODE 2> ... <MODE N>]         or
#              <bounce_mode> <MODE> [<MODE 2> ... <MODE N>]         or
#              <mode_status> <MODE>                                 or
#              <start_all_modes>                                    or
#              <stop_all_modes>                                     or
#              <bounce_all_modes>                                   or
#              <start_subsys>  <BMGT|INS|OMS|SSS> <MODE>            or
#              <stop_subsys>   <BMGT|INS|OMS|SSS> <MODE>            or
#              <subsys_status> <BMGT|INS|OMS|SSS> <MODE>

commands = %w(start_mode stop_mode bounce_mode start_view
              start_all_modes stop_all_modes bounce_all_modes
              start_subsys stop_subsys subsys_status mode_status)
@modes = []
user_command = ARGV[0]
if !ARGV[0].nil?
  user_command = ARGV[0].downcase
end

if ARGV.length < 1 || !commands.include?(user_command)
  puts "\n\nTo run ModeAdmin please use the following parameters"
  puts "\nmode_admin.rb  <start_mode>  <MODE> [<MODE 2> ... <MODE N>]        or"
  puts "               <stop_mode>   <MODE> [<MODE 2> ... <MODE N>]        or"
  puts "               <bounce_mode> <MODE> [<MODE 2> ... <MODE N>]        or"
  puts "               <start_view>  <MODE> <view>                         or"
  puts "               <mode_status>  <MODE>                               or"
  puts "               <start_all_modes>                                   or"
  puts "               <stop_all_modes>                                    or"
  puts "               <bounce_all_modes>                                  or"
  puts "               <start_subsys>  <BMGT|INS|OMS|SSS> <MODE>           or"
```

```
  puts "               <stop_subsys>   <BMGT|INS|OMS|SSS> <MODE>           or"
  puts "               <subsys_status> <BMGT|INS|OMS|SSS> <MODE>\n\n "
  exit
elsif user_command=='start_view' and (ARGV[1].nil? or ARGV[2].nil?)
  puts "\n\nMODE and View arguments are required for start_view\n"
  puts "Usage: mode_admin.rb start_view <MODE> <view>\n\n"
  exit
elsif ENV['USER'] != 'cmshared'
  puts "\n\nYou must login as cmshared to run mode_admin.rb\n\n"
  exit
end

mode_dir = "/home/cmshared/LAB/LinuxStart"
linux_modes_dir = "/home/cmshared/LAB/LinuxStart/LINUXMODES"


arg_count = 1
@verbose = true

case user_command
when 'start_mode'
  puts "\n\n Start Mode: \n\n"
  while my_mode = ARGV[arg_count]
    start_mode(my_mode)
    arg_count += 1
  end
when 'stop_mode'
  puts "\n\n Stop Mode: \n\n"
  while my_mode = ARGV[arg_count]
    stop_mode(my_mode)
    arg_count += 1
  end
when 'bounce_mode'
  puts "\n\n Bounce Mode: \n\n"
  while my_mode = ARGV[arg_count]
    stop_mode(my_mode)
```

```
    sleep 30
    start_mode(my_mode)
    arg_count += 1
  end
when 'start_view'
  puts "\n\n Start View: \n\n"
  start_mode(ARGV[1], ARGV[2])
when 'mode_status'
  puts "\n\n Mode Status: \n\n"
  mode_status(ARGV[1])
when 'start_subsys'
  puts "\n\n Start Subsystem: \n\n"
  start_subsys(ARGV[1],ARGV[2])
when 'stop_subsys'
  puts "\n\n Stop Subsystem: \n\n"
  stop_subsys(ARGV[1],ARGV[2])
when 'subsys_status'
  puts "\n\n Subsystem Status: \n\n"
  subsys_status(ARGV[1],ARGV[2])
when 'start_all_modes'
  puts "\n\n Start All Modes: \n\n"
  get_configured_modes
  @modes.each do |mode|
    start_mode(mode)
  end
when 'stop_all_modes'
  puts "\n\n Stop All Modes: \n\n"
  get_configured_modes
  @modes.each do |mode|
    stop_mode(mode)
  end
when 'bounce_all_modes'
  "\n\n Bounce All Mode: \n\n"
  get_configured_modes
  @modes.each do |mode|
    stop_mode(mode)
```

```
    start_mode(mode)
  end
end
```

Current Test Script: check_db_versions.rb

```
#!/usr/bin/env ruby

#Author: Jonathon Soohoo
#Date: 11.21.13
#Description: For a mode print the schema patch level in the database
#             and print the number of latest patch in clearcase

require 'pg'

class PostgresConnect
  # Create the connection instance.
  def connect
    @conn = PG.connect(
        :host => 'f4dbl03',
        :port => 5432,
        :dbname => 'ecs',
        :user => 'autotest')
  end

  # Get the latest patch level in the database
  def getSchemaPatchLevel(schema)
    @conn.exec( "select EcDbSchemaVersionId from #{schema}.EcDbDatabaseVersions where
EcDbCurrentVersionFlag='Y'" ) do |result|
      result.each do |row|
        yield row if block_given?
      end
    end
  end

  # Disconnect the back-end connection.
  def disconnect
    @conn.close
```

```
  end
end

def get_baseline(mode)
  view = "no_view_defined"
  File.open("/home/cmshared/LAB/LinuxStart/LINUXMODES", 'r') do |f1|
    while line = f1.gets
      mode1, baseline, host =line.split(",")
      if mode1 == mode
        if baseline.strip == 'RELB'
          view = "current_test"
        elsif
          view = baseline+"_linux"
        end
      end
    end
  end
  view
end


def main
  if ENV['USER'] != 'cmshared'
    puts "\n\nYou must login as cmshared to run this script\n\n"
    exit
  end
  if ARGV[0].nil?
    puts "\n\nMissing MODE parameter\n"
    puts "\nUsage: Check_Db_Versions.rb <MODE>\n\n"
    exit
  end
  p = PostgresConnect.new()
  p.connect
  mode = ARGV[0].downcase
  baseline = get_baseline(mode.upcase)
  baseline_levels = Hash['aim' => `/usr/atria/bin/cleartool setview -exec "ls -rt
```

```
/ecs/formal/DSS/AIM/database/patches | grep Patch | grep rb |  grep -iv Template | tail -1  "
#{baseline}`.gsub(/[^0-9]/, ''),
                         'ingest' => `/usr/atria/bin/cleartool setview -exec "ls -rt
/ecs/formal/INGEST/common/src/db/dplIngest/patches | grep Patch | grep rb |  grep -iv Template | tail -1   "
#{baseline}`.gsub(/[^0-9]/, ''),
                         'oms' => `/usr/atria/bin/cleartool setview -exec "ls -rt
/ecs/formal/OMS/Database/patches | grep Patch | grep rb |  grep -iv Template | tail -1   "
#{baseline}`.gsub(/[^0-9]/, ''),
                         'sss' => `/usr/atria/bin/cleartool setview -exec "ls -rt
/ecs/formal/SSS/Database/patches | grep Patch |  grep -iv Template | tail -1   " #{baseline}`.gsub(/[^0-9]/,
'')]

  subsystems = ['aim','ingest','oms','sss']

  puts "\nCheck Database Patch Levels  "
  puts "--------------------------------------------------"
  printf "  Mode\t\t : %-20s\n" , mode.upcase
  printf "  Baseline\t : %-20s\n" , baseline
  puts "==================================================="
  puts "  Schema \t | Mode Version\t| Baseline Version"
  puts "==================================================="

  begin
    subsystems.each do |subsys|
      schema = subsys.dup
      if mode!='ops'
        schema << "_#{mode}"
      end
      p.getSchemaPatchLevel(schema) {|row| @schema_id = row['ecdbschemaversionid'] }
      printf "  %-15s| %-13s| %s\n",  schema, @schema_id, baseline_levels[subsys].to_s
    end
  rescue Exception => e
    puts e.message
    puts e.backtrace.inspect
  ensure
    p.disconnect
```

```
  end
  puts "--------------------------------------------------"
end

main
Current Test Script: echo10_validate_metadata.rb
#!/usr/bin/env ruby

XMLLINT = '/tools/libxml2-2.9.1/bin/xmllint'
SCHEMA_PATH='/tools/common/test/BE_82_01/echo10/schema'
ITEMTYPES = { c: 'Collection', g: 'Granule', b: 'Browse' }

def usage
  $stderr.puts "\
Usage: #{$0} ITEM_TYPE ITEM_XML
ITEM_TYPE is one of
  c   Collection
  g   Granule
  b   Browse
ITEM_XML is a file of ECHO 10 metadata, matching the item type."
end

def parsecommandline(argv)
  exit_status = 0
  config = {}
  itemtype_char = argv.shift
  if itemtype_char.nil?
    usage
    exit_status = 1
  end

  if exit_status == 0
    itemtype = ITEMTYPES[itemtype_char.downcase.to_sym]
    if itemtype.nil?
      $stderr.puts "#{itemtype_char}: Unknown item type."
      usage
```

```
      exit_status = 2
    else
      config[:schema] = "#{itemtype}.xsd"
    end
  end

  if exit_status == 0
    config[:xml_file] = argv.shift
    if config[:xml_file].nil?
      usage
      exit_status = 3
    end
  end

  [exit_status, config]
end

exit_status, config = parsecommandline ARGV
if exit_status == 0
  `#{XMLLINT} --noout --path #{SCHEMA_PATH} --schema #{config[:schema]} #{config[:xml_file]}`
  exit $?.exitstatus
else
  exit exit_status
end
```

Current Test Script: update_collection_format.rb

```
#!/usr/bin/env ruby
$LOAD_PATH << 'lib'
require 'rspec'
require 'sdps/db'
require 'sdps/aimdb'
require 'sdps/da'

module GDalConfigureFormats

   class << self
     def main
```

```
        #check mode
        @mode = ENV['MODE']
        puts "MODE:"
        puts @mode

        @db_subsystem = Sdps::Da::DB_SUBSYSTEM
        puts "SUBSYSTEM:"
        puts @db_subsystem

        @db_conn = Sdps::Db.connect(@db_subsystem, @mode)
        @dadb = Sdps::Da::DaDb.new(@db_conn, @mode)
        @aimdb = Sdps::AimDb.new(@db_conn)

        @dadb.get_granule_formats();
        @dadb.update_collection_formats();

     end #main

   end #class GdalConfigureFormats

end

 GDalConfigureFormats.main
Current Test Script: run_oml_regression_tests.rb
#!/usr/bin/env ruby
supported_modes = ['OPS', 'DEV04', 'DEV08', 'DEV07']
mode = ENV['MODE']
if mode.nil? or mode.empty? or mode.strip.size.zero?
  puts "\n You should set MODE environment variable to run the tests\n"
  exit
elsif supported_modes.include?(mode) == false
  puts "\n #{mode} is not supported by autotest.\n"
  puts "\n Currently supported modes are #{supported_modes.to_s}.\n"
  exit
elsif ENV['USER'] != 'cmshared'
  puts "\n\nYou must login as cmshared to run the tests\n\n"
```

```
  exit
end

#Dir.chdir "/tools/common/test/autotest"

#bmgt tests
#all commented out tests are not working
#system ("cucumber features/bmgt/nominal_granule_manual_export_40_S1.feature")  - undefined feature
system ("cucumber --profile feature_log features/bmgt/AutoExport_Collection_Enable.feature")
system ("cucumber --profile feature_log features/bmgt/AutoExport_Collection_Insert.feature")
system ("cucumber --profile feature_log features/bmgt/AutoExport_Collection_Update.feature")
system ("cucumber --profile feature_log features/bmgt/Ecs_Collection_Additional_Metadata_S1.feature")
system ("cucumber --profile feature_log features/bmgt/Ecs_Collection_Additional_Metadata_S2.feature")
system ("cucumber --profile feature_log features/bmgt/Ecs_Collection_Additional_Metadata_S3.feature")
system ("cucumber --profile feature_log features/bmgt/Ecs_Collection_Additional_Metadata_S4.feature")
system ("cucumber --profile feature_log features/bmgt/Ecs_Collection_Additional_Metadata_S5.feature")
system ("cucumber --profile feature_log features/bmgt/Ecs_Collection_Additional_Metadata_S6.feature")
#the following test needs the collections installed in the mode
#system ("cucumber features/bmgt/Ecs_Collection_Additional_Metadata_S7.feature --out
/workingdata/cucumber/#{mode}/Ecs_Collection_Additional_Metadata_S7.feature.#{Time.now.strftime("%Y_%m_%d_%H_%M_
%S")}.out")
system ("cucumber --profile feature_log features/bmgt/Ecs_Collection_Additional_Metadata_S8.feature")
system ("cucumber --profile feature_log features/bmgt/Ecs_Collection_Additional_Metadata_S12.feature")
system ("cucumber --profile feature_log features/bmgt/Ecs_Collection_Additional_Metadata_S13.feature")
system ("cucumber --profile feature_log features/bmgt/Longform_Collection_Export.feature")
system ("cucumber --profile feature_log features/bmgt/Longform_Granule_Export.feature")
#system ("cucumber features/bmgt/Longform_Granule_Updates_Export.feature") #unfinished; ask Sri
#system ("cucumber features/bmgt/Longform_InsertTime_Export.feature --out
/workingdata/cucumber/#{mode}/Longform_InsertTime_Export.feature.#{Time.now.strftime("%Y_%m_%d_%H_%M_%S")}.out")
#system ("cucumber features/bmgt/Longform_LastUpdate_Export.feature --out
/workingdata/cucumber/#{mode}/Longform_LastUpdate_Export.feature.#{Time.now.strftime("%Y_%m_%d_%H_%M_%S")}.out")
system ("cucumber --profile feature_log features/bmgt/ManualExport_Granule_Delete.feature")
system ("cucumber --profile feature_log features/bmgt/ManualExport_Granule_Insert.feature")
#system ("cucumber features/bmgt/ManualExport_Insert_Time.feature --out
/workingdata/cucumber/#{mode}/ManualExport_Insert_Time.feature.#{Time.now.strftime("%Y_%m_%d_%H_%M_%S")}.out")
system ("cucumber --profile feature_log features/bmgt/ManualExport_LastUpdate.feature")
```

```
system ("cucumber --profile feature_log features/bmgt/ManualExport_Opendap_URL.feature")
#system ("cucumber features/bmgt/Manual_Collection_Deletes.feature --out
/workingdata/cucumber/#{mode}/Manual_Collection_Deletes.feature.#{Time.now.strftime("%Y_%m_%d_%H_%M_%S")}.out")
system ("cucumber --profile feature_log features/bmgt/Manual_Collection_Force_Delete.feature")
system ("cucumber --profile feature_log features/bmgt/Manual_Collection_Full_Update.feature")
system ("cucumber --profile feature_log features/bmgt/Manual_Collection_Inserts.feature")
system ("cucumber --profile feature_log features/bmgt/nominal_collection_export.feature")
system ("cucumber --profile feature_log features/bmgt/nominal_manual_granule_export_40_S1.feature")
Current Test Script: tcpproxylogger.rb
#!/usr/bin/env ruby

require 'time'
require 'logger'

module Timestamp
  def Timestamp.included klass
    klass.extend Timestamp
  end
  def timestamp(datetime = Time.now)
    datetime.strftime("%Y-%m-%dT%H:%M:%S.%6N")
  end
end

class LogEntry < Hash
  def request?
    self.has_key? :request
  end
  def response?
    self.has_key? :response
  end
  def timestamp
    Time.strptime(self[:timestamp], "%Y-%m-%dT%H:%M:%S.%N")
  end
end

class TcpProxyLogger < Logger
```

```
include Timestamp

class << self
  def parse(log_line)
    LogEntry[eval(log_line)]
  end
  def parse_file(log_filename)
    open(log_filename) do |log|
      log.map { |line| parse(line) }
    end
  end
end

def initialize(*args)
  super
  self.level = INFO
  self.formatter = proc { |severity, datetime, progname, data|
    hash = {timestamp:timestamp(datetime), severity: severity}
    if data.kind_of? Exception
      hash[:msg] = data.to_s
      hash[:exception] = data.class
      hash[:backtrace] = data.backtrace
    elsif data.kind_of? Hash
      data.each do |k,v|
        if v.kind_of? Exception
          hash[:msg] = v.to_s
          hash[:exception] = v.class
          hash[:backtrace] = v.backtrace
        else
          hash[k] = v
        end
      end
    else
      hash[:msg] = data.to_s
    end
```

```
      hash.to_s + "\n"
    }
  end

  def request(uuid, data, msg=nil)
    tcp(:request, uuid, data, msg)
  end

  def response(uuid, data, msg=nil)
    tcp(:response, uuid, data, msg)
  end

  def tcp(direction, uuid, data, msg)
    hash = {uuid:uuid, direction => data}
    hash[:msg] = msg if msg
    self.info(hash)
  end

#  def info(string_or_hash)
#    self.info(to_hash(string_or_hash))
#  end
#
#  def debug(string_or_hash)
#    self.info(to_hash(string_or_hash))
#  end

#  def warn(uuid, msg, exception=nil)
#    hash = {uuid:uuid, msg:msg}
#    hash[:exception] = exception.inspect if exception
#    self.warn(hash)
#  end

#  def error(uuid, msg, exception=nil)
#    hash = {uuid:uuid, msg:msg}
#    hash[:exception] = exception.inspect if exception
#    self.error(hash)
```

```
#  end

  private

  def to_hash(string_or_hash)
    begin
      Hash[string_or_hash]
    rescue ArgumentError
      {msg: string_or_hash.to_s}
    end
  end

end

class Logger
  class LogDevice
    private
    def add_log_header(file)
      # Prevent writing initial comment to a new log file.
    end
  end
end
```

Current Test Script: simplemockecho.rb

```
#!/usr/bin/env ruby
#!/usr/bin/env ruby

# Simplistic mock ECHO.
# Returns (mostly) static strings for all requests.

require 'socket'
require 'time'
require 'securerandom'
require_relative './tcpproxylogger.rb'

SELECT_TIMEOUT = 0.4
BLOCKSIZE = 10_000
```

```
ECHO_HOST = "testbed.echo.nasa.gov"
MAIN_RETRY_MAX = 10
RECV_RETRY_MAX = 10
PROGNAME = File.basename(__FILE__)
#ABSPATH = File.expand_path(File.dirname(__FILE__))
#LOG_PATH = "#{ABSPATH}/logs"
#LOG_NAME = "#{File.basename(PROGNAME, '.rb')}.log"
#DATA_PATH = "#{ABSPATH}/data"
#LOG_PATHNAME = "#{LOG_PATH}/#{LOG_NAME}"
STDOUT.sync = true

# The maximum length of the queue for pending connections.
backlog = 10

def uuidgen
  SecureRandom.hex
end

# POST /echo-rest/tokens/ HTTP/1.1
def token_response(providerid)
  $log.debug("token_response(#{providerid.inspect})")
  token = "00000000-0000-0000-0000-000000000000"
  body = [
    "<?xml version=\"1.0\" encoding=\"UTF-8\"?>",
    "<token>",
    "  <id>#{token}</id>",
    "  <username>bmgt_tester</username>",
    "  <client_id>clientName</client_id>",
    "  <user_ip_address>127.0.0.0</user_ip_address>",
    "  <provider>#{providerid}</provider>",
    "</token>",
  ].join("\n")
  [
    "HTTP/1.1 201 Created",
    "Server: Apache-Coyote/1.1",
    "Location: https://testbed.echo.nasa.gov/echo-rest/tokens/#{token}?clientId=unknown",
```

```
    "Cache-Control: no-cache",
    "X-UA-Compatible: IE=Edge,chrome=1",
    "Set-Cookie:
_EchoRestRails_session=BAh7BiIPc2Vzc2lvbl9pZCIlN2RmNDQ4YjY5ZjQxOGJjY2NjMzg0Y2UzMWE1ZmI5ZDg%3D--278ca1ce24a5b9261
8917fd6bb5198fc1d6d75b3; path=/; HttpOnly",
    "X-Runtime: 0.000000",
    "Content-Type: application/xml;charset=utf-8",
    "Content-Length: #{body.length}",
    "Date: #{Time.new.httpdate}",
    "",
    body
  ].join("\r\n")
end

# TODO Remove if no longer needed
#
# GET
/catalog-rest/providers/DEV07/datasets/AMSR-E%2FAqua%20Monthly%20L3%205x5%20deg%20Rainfall%20Accumulations%20V00
2/verify HTTP/1.1
def get_verify_success(providerid, path)
  catalogid = path.include?("/datasets/") ? "granules/G" : "datasets/C"
  body = [
    "<?xml version=\"1.0\" encoding=\"UTF-8\"?>",
    "<GranulesShortForm>",
    " <Provider>#{providerid}</Provider>",
    " <DatasetId>AMSR-E/Aqua Monthly L3 5x5 deg Rainfall Accumulations V002</DatasetId>",
    " <GranuleReferences>",
    "  <GranuleRef>",
    "   <GranuleUR>SC:AE_RnGd.002:69201</GranuleUR>",
    "   <LastUpdate>2013-03-01T11:56:31Z</LastUpdate>",
    "  </GranuleRef>",
    " </GranuleReferences>",
    "</GranulesShortForm>",
  ].join("\n")
  [
    "HTTP/1.1 200 OK",
```

```
    "Server: Apache-Coyote/1.1",
    "X-UA-Compatible: IE=Edge,chrome=1",
    "ETag: \"00000000000000000000000000000000\"",
    "Cache-Control: max-age=0, private, must-revalidate",
    "X-Request-Id: 00000000000000000000000000000000",
    "X-Runtime: 0.000000",
    "Date: #{Time.new.httpdate}",
    "X-Runtime: 0.000000",
    "X-Rack-Cache: miss",
    "Content-Type: application/xml;charset=utf-8",
    "Content-Length: #{body.length}",
    "Date: #{Time.new.httpdate}",
    "",
    body
  ].join("\r\n")
end


# TODO Remove if no longer needed
#
# PUT /catalog-rest/providers/DEV08/granules/SC%3AMCD43A2.005%3A263826?semantic_diff=true HTTP/1.1
# PUT
/catalog-rest/providers/DEV08/datasets/MODIS%2FTerra%2BAqua%20BRDF%2FAlbedo%20Quality%2016-Day%20L3%20Global%205
00m%20SIN%20Grid%20V005?semantic_diff=true HTTP/1.1
def put_semantic_diff_success(providerid, path)
  catalogid = path.include?("/granules/") ? "granules/G" : "datasets/C"
  body = [
    "<?xml version=\"1.0\" encoding=\"UTF-8\"?>",
    #"<reconciliation_report>",
    #"</reconciliation_report>",
    #"<?xml version="1.0" encoding="UTF-8"?>",
    "<reconciliation_report>",
    "<item_error_group>",
    "<item_errors type=\"array\">",
    "<item_error>",
    "<error_code>METADATA_MISMATCH</error_code>",
```

"<error_path>Granule/InsertTime</error_path>",
"<message>Expected: [2008-10-03T08:52:56.890Z] Actual: [2008-10-03T08:52:55.279Z]</message>",
"</item_error>",
"<item_error>",
"<error_code>METADATA_MISMATCH</error_code>",
"<error_path>Granule/Temporal/RangeDateTime/BeginningDateTime</error_path>",
"<message>Expected: [2001-05-04T10:35:00.000000Z] Actual: [2001-05-04T10:35:00.000Z]</message>",
"</item_error>",
"<item_error>",
"<error_code>METADATA_MISMATCH</error_code>",
"<error_path>Granule/Temporal/RangeDateTime/EndingDateTime</error_path>",
"<message>Expected: [2001-05-04T10:40:00.000000Z] Actual: [2001-05-04T10:40:00.000Z]</message>",
"</item_error>",
"<item_error>",
"<error_code>METADATA_MISMATCH</error_code>",

"<error_path>Granule/OrbitCalculatedSpatialDomains/OrbitCalculatedSpatialDomain/EquatorCrossingDateTime</error_path>",
"<message>Expected: [2001-05-04T10:50:47.344527Z] Actual: [2001-05-04T10:50:47.344Z]</message>",
"</item_error>",
"<item_error>",
"<error_code>METADATA_MISMATCH</error_code>",
"<error_path>Granule/AdditionalAttributes/AdditionalAttribute/Values/Value</error_path>",
"<message>Expected: [02] Actual: [2]</message>",
"</item_error>",
"<item_error>",
"<error_code>METADATA_MISMATCH</error_code>",
"<error_path>Granule/AdditionalAttributes/AdditionalAttribute/Values/Value</error_path>",
"<message>Expected: [00] Actual: [0]</message>",
"</item_error>",
"<item_error>",
"<error_code>METADATA_MISMATCH</error_code>",
"<error_path>Granule/OnlineAccessURLs/OnlineAccessURL/URL</error_path>",
"<message>Expected: [ftp://n5eil01u.ecs.nsidc.org/DP0/MOST/MOD10_L2.005/2001.05.04/MOD10_L2.A2001124.1035.005.2008277144607.hdf] Actual:

[ftp://n4ftl01u.ecs.nasa.gov/SAN/MOST/MOD10_L2.005/2001.05.04/MOD10_L2.A2001124.1035.005.2008277144607.hdf]</mes
sage>",
    "</item_error>",
    "<item_error>",
    "<error_code>METADATA_MISMATCH</error_code>",
    "<error_path>Granule/OnlineResources/OnlineResource/URL</error_path>",
    "<message>Expected:
[ftp://n5eil01u.ecs.nsidc.org/DP0/BRWS/Browse.001/2008.10.03/BROWSE.MOD10_L2.A2001124.1035.005.2008277144607.1.j
pg] Actual:
[ftp://n4ftl01u.ecs.nasa.gov/SAN/BRWS/Browse.001/2008.10.03/BROWSE.MOD10_L2.A2001124.1035.005.2008277144607.1.jp
g]</message>",
    "</item_error>",
    "<item_error>",
    "<error_code>METADATA_MISMATCH</error_code>",
    "<error_path>Granule/OnlineResources/OnlineResource/URL</error_path>",
    "<message>Expected:
[ftp://n5eil01u.ecs.nsidc.org/DP0/MOST/MOD10_L2.005/2001.05.04/MOD10_L2.A2001124.1035.005.2008277144607.hdf.xml]
Actual:
[ftp://n4ftl01u.ecs.nasa.gov/SAN/MOST/MOD10_L2.005/2001.05.04/MOD10_L2.A2001124.1035.005.2008277144607.hdf.xml]<
/message>",
    "</item_error>",
    "<item_error>",
    "<error_code>METADATA_ADDITION</error_code>",
    "<error_path>Granule/AssociatedBrowseImageUrls</error_path>",
    "<message>Found extra item
[{\"ProviderBrowseUrl\"=>{\"URL\"=>\"http://browse.echo.nasa.gov/NSIDC_ECS/2008/10/04/:BR:Browse.001:28103
747:1.BINARY\", \"FileSize\"=>\"52855\", \"MimeType\"=>\"application/x-hdfeos\"}}].</message>",
    "</item_error>",
  "</item_errors>",
  "<item_id>SC:MOD10_L2.005:28103722</item_id>",
  "<item_type>GRANULE</item_type>",
  "</item_error_group>",
  "</reconciliation_report> ",
  ].join("\n")
  [
    "HTTP/1.1 200 OK",

```
    "Server: Apache-Coyote/1.1",
    "Catalog-Location: https://#{ECHO_HOST}/catalog-rest/echo_catalog/#{catalogid}0000000000-#{providerid}",
    "Location: https://#{ECHO_HOST}#{path}",
    "X-UA-Compatible: IE=Edge,chrome=1",
    "ETag: \"00000000000000000000000000000000\"",
    "Cache-Control: max-age=0, private, must-revalidate",
    "X-Request-Id: 00000000000000000000000000000000",
    "X-Runtime: 0.000000",
    "X-Rack-Cache: invalidate, pass",
    "Content-Type: application/xml;charset=utf-8",
    "Content-Length: 1",
    "Content-Length: #{body.length}",
    "Date: #{Time.new.httpdate}",
    "",
    body
  ].join("\r\n")
end

# PUT /catalog-rest/providers/DEV08/granules/SC%3AMCD43A2.005%3A263826
# PUT
/catalog-rest/providers/DEV08/datasets/MODIS%2FTerra%2BAqua%20BRDF%2FAlbedo%20Quality%2016-Day%20L3%20Global%205
00m%20SIN%20Grid%20V005 HTTP/1.1
def put_success(providerid, path)
  catalogid = path.include?("/granules/") ? "granules/G" : "datasets/C"
  [
    "HTTP/1.1 200 OK",
    "Server: Apache-Coyote/1.1",
    "Catalog-Location: https://#{ECHO_HOST}/catalog-rest/echo_catalog/#{catalogid}0000000000-#{providerid}",
    "Location: https://#{ECHO_HOST}#{path}",
    "X-UA-Compatible: IE=Edge,chrome=1",
    "ETag: \"00000000000000000000000000000000\"",
    "Cache-Control: max-age=0, private, must-revalidate",
    "X-Request-Id: 00000000000000000000000000000000",
    "X-Runtime: 0.000000",
    "Date: #{Time.new.httpdate}",
    "X-Rack-Cache: invalidate, pass",
```

```
    "Content-Type: application/xml;charset=utf-8",
    "Content-Length: 1",
    "",
    " "
  ].join("\r\n")
end

# DELETE /echo-rest/tokens/E50BAE2B-2A22-D377-6559-17F76653E192 HTTP/1.1^M
def delete_token_success
  [
    "HTTP/1.1 204 No Content",
    "Server: Apache-Coyote/1.1",
    "Cache-Control: no-cache",
    "X-UA-Compatible: IE=Edge,chrome=1",
    "Set-Cookie:
_EchoRestRails_session=BAh7BiIPc2Vzc2lvbl9pZCIlNjlkYTM5OGUxYmIyOGY0YzkzMTY4NTIwYTg1ZmNmNzM%3D--958e8d1509326c190
3bb633cd16e74f38aa5140a; path=/; HttpOnly",
    "X-Runtime: 0.000000",
    "Date: #{Time.new.httpdate}",
    ""
  ].join("\r\n")
end

# DELETE /catalog-rest/providers/DEV08/granules/SC%3AMCD43A2.005%3A263826 HTTP/1.1^M
def delete_item_success
  [
    "HTTP/1.1 204 No Content",
    "Server: Apache-Coyote/1.1",
    "X-UA-Compatible: IE=Edge,chrome=1",
    "Cache-Control: no-cache",
    "X-Request-Id: d65bb94f88f112f0130aba32e37ea472",
    "X-Runtime: 0.000000",
    "Date: #{Time.new.httpdate}",
    "X-Rack-Cache: invalidate, pass",
    ""
  ].join("\r\n")
```

```
end

# Default response.
def http_success
  $log.debug("http_success()")
  [
    "HTTP/1.1 200 OK",
    "Date: #{Time.new.httpdate}",
    "Content-Type: application/xml;charset=utf-8",
    "Content-Length: 1",
    "",
    "\n"
  ].join("\r\n")
end

def timestamp(datetime = Time.now)
  datetime.strftime("%Y-%m-%dT%H:%M:%S.") + ("%06d" % datetime.usec)
end

# Obscure the password in the log file.
def scrubbed(data)
  data.sub(/<password>[^<]*<\/password>/, "<password>********<password>")
end

# NOTE This is getting a bit complex.  Consider creating a proper router.
# Dispatch to the appropriate response for the given request.
def response(request)
  if request.start_with?("POST /echo-rest/tokens")
    token_response($providerid)
  elsif request.start_with?("PUT /")
    path = request.split(' ', 3)[1]
    if path.include?("semantic")
      put_semantic_diff_success($providerid, path)
    else
      put_success($providerid, path)
    end
```

```
  elsif request.start_with?('DELETE /echo-rest/tokens/')
    delete_token_success
  elsif request.start_with?('DELETE /catalog-rest/providers/')
    delete_item_success
  elsif request.start_with?("GET /")
    path = request.split(' ', 3)[1]
    if(path.include?("verify"))
      get_verify_success($providerid, path)
    else
      http_success
    end
  else # unknown
    http_success
  end
end

#######################################################################

if ARGV.length < 2
  $stderr.printf "Usage: %s LOCALPORT PROVIDERID [LOG_PATHNAME]\n", $0
  exit 1
end

localport = ARGV.shift.to_i
$providerid = ARGV.shift
$log_pathname = ARGV.shift

### $log = Logger.new($log_pathname || STDOUT)
$log = TcpProxyLogger.new($log_pathname || STDOUT)
###$log.progname = PROGNAME
###$log.level = Logger::INFO
###$log.formatter = proc do |severity,datetime,progname,msg|
###          "#{datetime}: #{msg}\n"
###end

server = TCPServer.open(localport)
```

```
server.listen(backlog)

# Abort on exceptions, otherwise threads will be silently killed in case
# of unhandled exceptions.
Thread.abort_on_exception = true

def connThread(client)
  uuid = uuidgen
  $log.debug(">>> connThread(#{client.inspect})")
  port, name = client.peeraddr[1..2]
  $log.debug("    receiving from #{name}:#{port}")

  request = ""
  while true do
    $log.debug("    while true do ...")
    # readable, writable, error = select([client], [client], nil, 0)
    ready = select([client], nil, nil, SELECT_TIMEOUT)
    $log.debug("    ready: #{ready.inspect}")
    if ready.nil?
      if request.empty?
        break
      else
        $log.request(uuid, scrubbed(request))
        resp = response(request)
        $log.response(uuid, resp)
        client.write(resp)
        request = ""
      end
    else
      if ready[0].include?(client)
        recv_retry_count = 0
        begin
          $log.debug("    client.recv(#{BLOCKSIZE})...")
          data = client.recv(BLOCKSIZE)
          $log.debug("    ...recv'd #{data.length} bytes")
          request << data
```

```
        rescue Exception => e
          if recv_retry_count < RECV_RETRY_MAX
            recv_retry_count += 1
            $log.warn(uuid: uuid, msg: "Retrying read #{recv_retry_count} of #{RECV_RETRY_MAX} times.",
exception: e)
            retry
          else
            $log.error(uuid: uuid, msg: "Exceeded read retry limit.")
          end
        end
        if data.empty?
          $log.debug("    client closed connection.")
          break
        else
          $log.debug("    data: #{data.inspect}")
        end
        #$log.info("Request\n" + scrubbed(request))
        #resp = response(request)
        #$log.info("Response\n" + resp)
        #client.write(resp)
      end
    end
  end
  client.close
  $log.debug("<<< connThread(#{client.inspect})")
end

main_retry_count = 0
while true do
  # Whenever server.accept returns a new connection, start
  # a handler thread for that connection.
  begin
    Thread.start(server.accept) { |client| connThread(client) }
    # Reset the count after success
    main_retry_count = 0
  rescue Interrupt => e
```

```
    $log.info("Received #{e.inspect}. Stopping.")
    break
  rescue SignalException => e
    $log.info("Received #{e.inspect}. Stopping.")
    break
  rescue Exception => e
    $log.error("Main loop caught #{e.class}")
    $log.error(e)
    if main_retry_count < MAIN_RETRY_MAX
      main_retry_count += 1
      sleep 1
      $log.warn("Retrying main loop #{main_retry_count} of #{MAIN_RETRY_MAX} times.")
      retry
    else
      $log.error("Exceeded main loop retry limit. Stopping.")
    end
  end
end
```

Current Test Script: tcpproxy.rb

```
#!/usr/bin/env ruby

# 2013-03-06 Script originally from
#   A TCP Proxy in Ruby
#   http://hpyblg.wordpress.com/2010/07/18/a-tcp-proxy-in-ruby/

require 'socket'
require 'time'
require 'securerandom'
require_relative './tcpproxylogger.rb'

RETRY_MAX = 10

# read buffer
BLOCKSIZE = 1024 * 1024
```

```
ACCEPTED_BEHAVIOR = %w(200 403 409 422 500 pause)

def usage
  $stderr.printf "Usage: %s LOCALPORT REMOTEHOST:REMOTEPORT LOGFILE [RESPONSE]\n", $0
  $stderr.printf "RESPONSE may be one of\n"
  $stderr.printf "  #{ACCEPTED_BEHAVIOR.join("\n  ")}\n"
end

def uuidgen
  SecureRandom.hex
end

def scrubbed(data)
  data.sub(/<password>[^<]*<\/password>/, "<password>********<password>")
end

def http_response(http_status, content, x_request_id, date)
  "HTTP/1.1 #{http_status}
Server: Apache-Coyote/1.1
X-UA-Compatible: IE=Edge,chrome=1
Cache-Control: no-cache
X-Request-Id: #{x_request_id}
X-Runtime: 0.123000
Date: #{date}
X-Rack-Cache: invalidate, pass
Content-Type: application/xml;charset=utf-8
Content-Length: #{content.length}

#{content}"
end

def http403(x_request_id=SecureRandom.hex, date=Time.now.httpdate)
  http_response('403 Forbidden Access error from ECHO',
                "<errors><error code=\"FORBIDDEN_ACCESS\">Forbidden Access error from ECHO</error></errors>",
               x_request_id, date)
end
```

```
def http409(x_request_id=SecureRandom.hex, date=Time.now.httpdate)
  http_response('409 Conflict',
                "<errors><error code=\"CONFLICT\">Conflict.</error></errors>",
               x_request_id, date)
end

def http422(x_request_id=SecureRandom.hex, date=Time.now.httpdate)
  http_response('422 Record Invalid',
                "<errors><error code=\"RECORD_INVALID\">Record is Invalid.</error></errors>",
               x_request_id, date)
end

def http500(x_request_id=SecureRandom.hex, date=Time.now.httpdate)
  http_response('500 Internal Server Error',
                "<errors><error code=\"INTERNAL_SYSTEM_ERROR\">Internal system error.</error></errors>",
               x_request_id, date)
end

def http503(x_request_id=SecureRandom.hex, date=Time.now.httpdate)
  http_response('503 Service Not Available',
                "<errors><error code=\"SERVICE_NOT_AVAILABLE\">Service Not Available</error></errors>",
               x_request_id, date)
end

def http200(x_request_id=SecureRandom.hex, date=Time.now.httpdate)
  http_response('200 OK', "\n", x_request_id, date)
end

def connThread(local)
  $log.debug(">>> connThread()")
  uuid = uuidgen
  port, name = local.peeraddr[1..2]
  $log.info(uuid: uuid, msg: "Receiving from #{name}:#{port}")

  remote = $remoteHost.nil? ? nil : TCPSocket.new($remoteHost, $remotePort)
```

```
# start reading from both ends
while true do
  ready = select([local, remote], nil, nil)
  if ready[0].include? local
    # local -> remote
    count = 0
    begin
      data = local.recv(BLOCKSIZE)
    rescue Exception => e
      count += 1
      $log.warn(uuid: uuid, msg: "Exception #{count} on local.recv(#{BLOCKSIZE})")
      $log.warn(e)
      retry unless count > RETRY_MAX
      $log.error(uuid: uuid, msg: "Bailing after retrying local.recv #{count - 1} times.")
    end
    if data.empty?
      $log.info(uuid: uuid, msg: "Local end closed connection.")
      break
    end
    if $behavior == '200'
      $log.request(uuid, scrubbed(data), "Ignoring request")
      data = http200
      local.write(data)
      $log.response(uuid, scrubbed(data), "Returned 200 response")
    elsif $behavior == '403'
      $log.request(uuid, scrubbed(data), "Ignoring request")
      data = http403
      local.write(data)
      $log.response(uuid, scrubbed(data), "Returned 403 response")
    elsif $behavior == '409'
      $log.request(uuid, scrubbed(data), "Ignoring request")
      data = http409
      local.write(data)
      $log.response(uuid, scrubbed(data), "Returned 409 response")
    elsif $behavior == '422'
      $log.request(uuid, scrubbed(data), "Ignoring request")
```

```
    data = http422
    local.write(data)
    $log.response(uuid, scrubbed(data), "Returned 422 response")
  elsif $behavior == '500'
    $log.request(uuid, scrubbed(data), "Ignoring request")
    data = http500
    local.write(data)
    $log.response(uuid, scrubbed(data), "Returned 500 response")
  elsif $behavior == '503'
    $log.request(uuid, scrubbed(data), "Ignoring request")
    data = http503
    local.write(data)
    $log.response(uuid, scrubbed(data), "Returned 503 response")
  elsif $behavior == 'pause'
    $log.request(uuid, scrubbed(data), "Ignoring request")
    $log.response(uuid, nil, "Paused. No response.")
  else
    count = 0
    begin
      remote.write(data)
      $log.request(uuid, scrubbed(data))
    rescue Exception => e
      count += 1
      $log.warn(uuid: uuid, msg: "Exception #{count} on remote.write(data)", exception: e.inspect)
      retry unless count > RETRY_MAX
      $log.error(uuid: uuid, msg: "Bailing after retrying remote.write #{count - 1} times.")
    end
  end
end
if ready[0].include? remote
  # remote -> local
  count = 0
  begin
    data = remote.recv(BLOCKSIZE)
  rescue Exception => e
    count += 1
```

```
        $log.warn(uuid: uuid, msg: "Exception #{count} on remote.recv(#{BLOCKSIZE})", exception: e.inspect)
        retry unless count > RETRY_MAX
        $log.error(uuid: uuid, msg: "Bailing after retrying remote.recv #{count - 1} times.")
      end
      if data.empty?
        $log.info(uuid: uuid, msg: "Remote end closed connection.")
        break
      end
      if $behavior != 'pause'
        count = 0
        begin
          local.write(data)
          $log.response(uuid, scrubbed(data))
        rescue Exception => e
          count += 1
          $log.warn(uuid: uuid, msg: "Exception #{count} on local.write(data)", exception: e.inspect)
          retry unless count > RETRY_MAX
          $log.error(uuid: uuid, msg: "Bailing after retrying local.write #{count - 1} times.")
        end
      end
    end
  end

  local.close
  remote.close
  $log.info(uuid: uuid, msg: "Closed #{name}:#{port}")
  $log.debug("<<< connThread()")
end

#######################################################################

if ARGV.length < 3
  usage
  exit 1
end
```

```
$remoteHost = nil
$remotePort = nil

local_port = ARGV.shift
$remoteHost, $remotePort = ARGV.shift.split(':')

# LOGDEVICE = File.expand_path('../../logs/tcp.log', __FILE__)
LOGDEVICE = ARGV.shift
$log = TcpProxyLogger.new(LOGDEVICE)
# $log.level = Logger::DEBUG

$behavior = ARGV.shift
if $behavior and not %w(200 403 409 422 500 503 pause).include?($behavior)
  usage
  exit 2
end

$log.info("Target address: #{$remoteHost}:#{$remotePort}")
$log.info("Behavior: #{$behavior ? $behavior : 'proxy'}")

server = TCPServer.open(local_port)
server.listen(500)

port = server.addr[1]
addrs = server.addr[2..-1].uniq

$log.info("Listening on #{addrs.collect{|a|"#{a}:#{port}"}.join(' ')}")

# Abort on exceptions, otherwise threads will be silently killed in case
# of unhandled exceptions.
Thread.abort_on_exception = true

# Have a thread just to process Ctrl-C events on Windows
# (although Ctrl-Break always works).
Thread.new { loop { sleep 1 } }
```

```
running = true
while running do
  # Whenever server.accept returns a new connection, start
  # a handler thread for that connection.
  begin
    Thread.start(server.accept) { |local| connThread(local) }
  rescue Interrupt => e
    $log.info("Caught interrupt.  Exiting.")
    running = false
  rescue SignalException => e
    $log.info("Caught signal #{e.to_s}.  Exiting.")
    running = false
  end
end

$log.info("TCPProxy ended.")
Current Test Script: install.rb
#!/usr/bin/env ruby

CONFIG_PATHNAME = File.expand_path('../../config/tcpproxy.yml', __FILE__)
SOURCE_DIR = File.expand_path('../source', __FILE__)

begin
  $VERBOSE, verbose_bak = nil, $VERBOSE
  require 'pry' ### DEBUG ###
  require 'optparse'
  require 'erb'
  require 'yaml'
  require 'ostruct'
  require 'fileutils'
  require 'pp'
ensure
  $VERBOSE = verbose_bak
end

class CommandLine
```

```
def self.parse(args)
  options = {}
  opt_parser = OptionParser.new do |opts|
    opts.banner = "Usage: #{$0} MODE [-c CONFIG] [-d DESTDIR]"
    opts.on("-c", "--config CONFIG", "Use an alternate config file") do |config|
      if File.exists?(config)
        options[:config_pathname] = config
      else
        puts "#{config}: File not found."
        exit 1
      end
    end
    opts.on("-d", "--directory DESTDIR",
            "Install under DESTDIR instead of the directory specified in the config file") do |dir|
      options[:dest_dir] = File.expand_path(dir)
    end
    opts.on_tail("-h", "--help", "Show this message") do
      puts opts
      exit
    end
  end
  begin
    opt_parser.parse!(args)
    unless args.empty? # or args[0].start_with?("-")
      options[:mode] = args.shift
    else
      puts "\nMODE is required\n\n"
      puts opt_parser
      exit
    end
    options[:config_pathname] ||= CONFIG_PATHNAME
  rescue OptionParser::ParseError => e
    puts "\n#{e.to_s.capitalize}\n\n"
    puts opt_parser
    exit 1
  end
```

```
    options
  end
end

def read_config(config_pathname)
  begin
    YAML.load_file(config_pathname)
  rescue Exception => e
    puts "Error reading YAML file #{config_pathname.inspect}: #{e}"
    exit 1
  end
end

def config_hash(options)
  config = read_config(options[:config_pathname])
  mode = options[:mode]
  if config.has_key?(mode)
    result = config[mode].merge(options)
    result[:mode_dir] = File.join(result[:dest_dir], mode)
    result[:logs_dir] = File.join(result[:mode_dir], 'logs')
    result
  else
    puts "#{mode}: mode not defined in #{options[:config_pathname]}."
    exit 2
  end
end

def source_to_dest(mode_dir, source)
  if source.start_with?(SOURCE_DIR)
    mode_dir + source[SOURCE_DIR.length..-1]
  else
    source
  end
end

def walk_dir_tree(root, &block)
```

```
  block.yield root
  Dir[root + '/*'].each do |child|
    if File.directory?(child)
      walk_dir_tree(child, &block)
    else
      block.yield child
    end
  end
end

def transform(config, erb_pathname, dest_pathname)
  open(dest_pathname, 'w') do |out|
    erb = ERB.new(open(erb_pathname){|f| f.read})
    content = erb.result(binding)
    out.write(content)
  end
end

config = OpenStruct.new(config_hash(CommandLine.parse(ARGV)))

# Copy directory tree and expand templates.
walk_dir_tree(SOURCE_DIR) do |x|
  dest = source_to_dest(config.mode_dir, x)
  if File.directory?(x)
    FileUtils.mkdir_p(dest, mode:0755)
  elsif x.end_with?('.erb')
    transform(config, x, dest[0..-5])
  else
    FileUtils.cp(x, dest)
  end
end

# Move files out of bin/ and make them executable.
bin_dir = File.join(config.mode_dir, 'bin')
bin_files = Dir[File.join(bin_dir, '*')]
FileUtils.chmod(0744, bin_files)
```

```
FileUtils.mv(bin_files, File.join(bin_dir, '..'))
FileUtils.remove_dir(bin_dir)
Current Test Script: data_visualize.rb
#data_visualize.rb
require 'erb'
require 'find'

#should be logged in as cmshared
if ENV['USER'] != 'cmshared'
  puts "\n\nYou must login as cmshared to run the tests\n\n"
  exit
end
#the paths can be command line or default in the file
@base_paths = ["/workingdata/cucumber/OPS", "/workingdata/cucumber/DEV08", "/workingdata/cucumber/DEV07",
"/workingdata/cucumber/DEV06"]
@base_path_options = @base_paths.collect { |x|  x.gsub("/","_")  }
@imagefilepattern = ".png"
#puts @base_paths

#search for all png files in the paths
@root_path = @base_paths[0]
@root_path_option = @base_path_options[0]
#puts root_path
@image_file_paths = []
Find.find(@root_path) do |path|
    @image_file_paths << path if path =~ /.*\.png$/
end
#p image_file_paths
@sub_paths = Hash.new

@image_file_paths.each do |file_path|
     #remove base path and filename
     #puts "file_path: #{file_path}"
     base_name = File.basename(file_path)
     #puts "base_name={base_name}"
     image_sub_path = file_path.gsub(@root_path,"").gsub(base_name, "")
```

```
     filename_without_extn = base_name.sub(".png", "")

     if @sub_paths.has_key?(image_sub_path)
         puts "has key #{image_sub_path}"
         # add the filename and base name hash to it
         filepaths_hash = @sub_paths[image_sub_path]
         filepaths_hash[filename_without_extn] = file_path
         @sub_paths[image_sub_path] = filepaths_hash
         p filepaths_hash
     else
         puts "has no key #{image_sub_path}"
         # create a new hash
         filepaths_hash = Hash.new()
         filepaths_hash[filename_without_extn] = file_path
         @sub_paths[image_sub_path] = filepaths_hash
         p filepaths_hash
     end
     #p @sub_paths
end
#p @sub_paths

old_stdout = STDOUT.dup

$stdout.reopen '/workingdata/cucumber/data_visualize.html', 'w'
erb = ERB.new(File.open("data_visualize.html.erb").read, 0, '>')
puts erb.result  binding

$stdout.reopen old_stdout
#puts @base_paths

Current Test Script: run_dpl_regression_tests.rb
#!/usr/bin/env ruby
supported_modes = ['OPS', 'DEV08','DEV07','DEV04','DEV06']
mode = ENV['MODE']
if mode.nil? or mode.empty? or mode.strip.size.zero?
  puts "\n You should set MODE environment variable to run the tests\n"
```

```
  exit
elsif supported_modes.include?(mode) == false
  puts "\n #{mode} is not supported by autotest.\n"
  puts "\n Currently supported modes are #{supported_modes.to_s}.\n"
  exit
elsif ENV['USER'] != 'cmshared'
  puts "\n\nYou must login as cmshared to run the tests\n\n"
  exit
end

#Dir.chdir "/tools/common/test/autotest"

#dpl tests

#data access tests
#system ("cucumber features/da/gdal_service_requests.feature --out
/workingdata/cucumber/#{mode}/egi_smap_nosubset_format.feature.#{Time.now.strftime("%Y_%m_%d_%H_%M_%S")}.out")
#system ("cucumber features/da/simple_inventory_interface.feature")
#system("cucumber --profile feature_log features/DataAccess/egi/smap/egi_smapl1_parameter_subset.feature")
#system("cucumber --profile feature_log features/DataAccess/egi/smap/egi_smapl2_parameter_subset.feature")

#system("cucumber --profile feature_log features/DataAccess/egi/smap/egi_smapl1_parameter_subset.feature")
#system("cucumber --profile feature_log features/DataAccess/egi/smap/egi_smapl2_smp_parameter_subset.feature")
#system("cucumber --profile feature_log features/DataAccess/egi/smap/egi_smapl2_smap_parameter_subset.feature")
#system("cucumber --profile feature_log features/DataAccess/egi/smap/egi_smapl2_sma_parameter_subset.feature")
#
#system("cucumber --profile feature_log features/DataAccess/egi/smap/egi_smapl1_format.feature")
#system("cucumber --profile feature_log features/DataAccess/egi/smap/egi_smapl2_smp_format.feature")
#system("cucumber --profile feature_log features/DataAccess/egi/smap/egi_smapl2_smap_format.feature")
#system("cucumber --profile feature_log features/DataAccess/egi/smap/egi_smapl2_sma_format.feature")

#system("cucumber --profile feature_log features/DataAccess/egi/smap/egi_smapl3_smp_parameter_subset.feature")
#system("cucumber --profile feature_log features/DataAccess/egi/smap/egi_smapl3_smap_parameter_subset.feature")
#system("cucumber --profile feature_log features/DataAccess/egi/smap/egi_smapl3_sma_parameter_subset.feature")
#system("cucumber --profile feature_log features/DataAccess/egi/smap/egi_smapl3_fta_parameter_subset.feature")
#system("cucumber --profile feature_log features/DataAccess/egi/smap/egi_smapl4_smau_parameter_subset.feature")
```

```
#system("cucumber --profile feature_log features/DataAccess/egi/smap/egi_smapl4_smlm_parameter_subset.feature")
#system("cucumber --profile feature_log features/DataAccess/egi/smap/egi_smapl4_smgp_parameter_subset.feature")
#system("cucumber --profile feature_log features/DataAccess/egi/smap/egi_smapl4_cmdl_parameter_subset.feature")
#
#system("cucumber --profile feature_log features/DataAccess/egi/smap/egi_smapl3_smp_format.feature")
#system("cucumber --profile feature_log features/DataAccess/egi/smap/egi_smapl3_smap_format.feature")
#system("cucumber --profile feature_log features/DataAccess/egi/smap/egi_smapl3_sma_format.feature")
#system("cucumber --profile feature_log features/DataAccess/egi/smap/egi_smapl3_fta_format.feature")
#system("cucumber --profile feature_log features/DataAccess/egi/smap/egi_smapl4_smau_format.feature")
#system("cucumber --profile feature_log features/DataAccess/egi/smap/egi_smapl4_smlm_format.feature")
#system("cucumber --profile feature_log features/DataAccess/egi/smap/egi_smapl4_smgp_format.feature")
#system("cucumber --profile feature_log features/DataAccess/egi/smap/egi_smapl4_cmdl_format.feature")


#system("cucumber --profile feature_log
features/DataAccess/egi/egi_smapl1_parameter_subset_netcdf4.feature")Current Test Script:
Manual_Collection_Force_Delete.rb
Given(/^criteria_165 mode is checked out$/) do
  @mode = ENV['MODE']
  @mode.should_not be_nil
  #host should be checked to be f5oml01v
  @mode_checker = Sdps::Mode.new(@mode)
  @mode_checker.mode_checked_out?().should be_truthy
  @db_subsystem = Sdps::Bmgt::DB_SUBSYSTEM
  @db_conn = Sdps::Db.connect(@db_subsystem, @mode)
  @db_conn.should_not be_nil

  @xmlUtil = Sdps::Bmgt::Xml.new()
  @bmgtdb = Sdps::Bmgt::BmgtDb.new(@db_conn)

  #Initialize critinfo and SoTestData
  @critinfo = "BE_82_01:165"
  @sotestdata = Sdps::SoTestData.new(@db_conn)

  @proxy_log_location = "/tools/common/test/BE_82_01/proxy/#{@mode}/logs/tcp.log"
end
```

```
Given(/^criteria_165 testdata is checked out$/) do
  criteria_165_testdata_checked_out?().should be_truthy
end

Given(/^criteria_165 bmgt subsystem is checked out$/) do
  @mode_checker.bmgt_subsystem_checked_out?().should be_truthy, "bmgt system check failed"
end

Given(/^criteria_165 collection is installed$/) do
  @aimdb.is_collection_installed?(@coll.shortname, @coll.versionid).should be_truthy, "Collection for 165 not
installed."
end

Given(/^criteria_165  collection is enabled for collection and granule export\.$/) do
  if (!@bmgtdb.collection_export_enabled?(@coll.collectionid))
    @bmgtdb.enable_collection_granule_export(@coll.collectionid).should be_truthy
  end
end

When(/^criteria_165 Request force delete for collection C1$/) do
  @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
  manual_coll_forcedelete_export?(@coll.collectionid).should_not be_falsey
end


Then(/^criteria_165 Verify that the request for C1 is queued and completed$/) do
  #poll and wait
  status = Sdps::Util.poll(180, 15){
    result = @bmgtdb.get_coll_export_request_status(@request_time, @coll.collectionid, 'MAN')
    if (result == 'PENDING' || result == 'STARTED')
      nil
    else
      result
    end
  }
```

```
  status.should eq 'SUCCESS'
end

Then(/^criteria_165 Verify that the TCP proxy log reports a single HTTP DELETE request for collection C1$/) do
  dataset =  @coll.datasetid()
  @xmlUtil.http_delete_for_collection?(@proxy_log_location, @request_time, dataset).should be_truthy
end

Then(/^criteria_165 Verify that the collection C1 is automatically disabled for both collection and granule
metadata export$/) do
  check_exportflags_disabled?(@coll.collectionid).should be_truthy
end

def criteria_165_testdata_checked_out?

  @data = @sotestdata[@critinfo]
  @data.should_not be_nil

  rv = true
  if @data.length != 1
    puts "There should be one collections returned for #{@critinfo}, check AutoTestSoTestDataInfo table"
    rv = false
  else
    @shortname = @data[0].shortname
    @versionid = @data[0].versionid

    @aimdb = Sdps::AimDb.new(@db_conn)
    @coll = @aimdb.find_collection_by_shortname_versionid(@shortname, @versionid)

    if (@coll == nil || !@aimdb.is_collection_installed?(@shortname, @versionid))
      puts "#{@shortname}, #{@versionid} is not installed"
      rv = false
    else
      #collection should have datamodeltype = 'ECS'
      if @coll.datamodeltype != 'ECS'
        puts "#{@shortname}, #{@versionid}'s datamodeltype is not ECS"
```

```
        rv = false
      end
    end
  end
  rv.should be_truthy
end

def manual_coll_forcedelete_export?(collectionid)
  begin
    request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
    #options = Array.new
    #options.push('-fd')
    #Sdps::Bmgt::ExportRequest.new(@mode).manual_export_for_collection?(collectionid, options).should be_truthy,
"Falied running Manual Force delete Export for collection"
    cdir = "/usr/ecs/#{@mode}/CUSTOM/utilities"
    Dir.chdir("#{cdir}") do
      status = system("./EcBmBMGTManualStart #{@mode} -fd -c #{collectionid}")
    end
    #check if request was queued

    if @bmgtdb.get_coll_export_request_status(request_time, collectionid, 'MAN') == nil
      false
    else
      true
    end
  rescue =>e
    puts e.backtrace
    false
  end
end

def check_exportflags_disabled?(collectionid)
  begin
    collectionexportflag = @bmgtdb.get_collection_granule_export(collectionid)['collectionexportflag']
    granuleexportflag = @bmgtdb.get_collection_granule_export(collectionid)['granuleexportflag']
    if collectionexportflag == 'Y' ||  granuleexportflag == 'Y'
```

```
      false
    else
      true
    end
  end
end
```

Current Test Script: BE_82_01_40_V6.rb

```
Then(/^Verify that the metadata generated by the action in S-2 subclause e contains a Restriction Flag value of
255$/) do
    @test_granules_xml.keys.each do |granuleid|
      granule_xml = @test_granules_xml[granuleid]
      @xml_utils.granule_xml_contains?(granule_xml,"//Granule/RestrictionFlag",
"<RestrictionFlag>255</RestrictionFlag>").should be_truthy, "granule - #{granuleid} xml does not contain
RestrictionFlag = 255"
    end
end
```

Current Test Script: nominal_granule_export.rb

```
When(/^Logically delete a granule from the ECS inventory$/) do
  pending # express the regexp above with the code you wish you had
end

Then(/^Verify that the operations in each result in the export of a single HTTP DELETE, with the ID of the
granule in the URL$/) do
  pending # express the regexp above with the code you wish you had
end

Then(/^but containing no granule metadata in the request body$/) do
  pending # express the regexp above with the code you wish you had
end

When(/^Physically delete a granule from the ECS inventory$/) do
  pending # express the regexp above with the code you wish you had
end

When(/^DFA a granule$/) do
```

```
  pending # express the regexp above with the code you wish you had
end




When(/^Perform XML replacement on a granule$/) do
  pending # express the regexp above with the code you wish you had
end

Then(/^Verify that the metadata generated by the action in S2 subclause o contains XML reflecting the
replacement which was performed$/) do
  pending # express the regexp above with the code you wish you had
end

Current Test Script: BE_82_01_40_S2e.rb
When(/^Hide a granule$/) do
   @aimdb = Sdps::AimDb.new(@db_conn);
   @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
   @export_queue = 'EVENT'
   @poll_wait_time = 300 #need to get this value from the configuration property table
   status = @aimdb.hide_granule(@test_granule.granuleid);
   status.should be_truthy
end
Current Test Script: Ecs_Collection_Additional_Metadata_S5.rb
require 'sdps/sotestdata'
require 'sdps/aimdb'
require 'sdps/mode'
require 'sdps/db'
require 'sdps/util'
require 'sdps/bmgt'

Given(/^Criteria_20_5 Initial Setup$/) do
  #Check mode.
  @mode = ENV['MODE']
  @mode.should_not be_nil
```

```
  @mode_checker = Sdps::Mode.new(@mode)
  @mode_checker.mode_checked_out?().should be_truthy
  @proxy_log_location = @mode_checker.get_proxy_log_location()
  @xmlUtil = Sdps::Bmgt::Xml.new()

  #set up db connection
  @db_subsystem = Sdps::Bmgt::DB_SUBSYSTEM
  @conn = Sdps::Db.connect(@db_subsystem, @mode)
  @conn.should_not be_nil

  #Make sure BMGT is up and running, proxy needs to be up too
  @mode_checker.bmgt_subsystem_checked_out?().should be_truthy, "bmgt system check failed"

  #Initialize critinfo and SoTestData
  @critinfo = "BE_82_01:20:5"
  @sotestdata = Sdps::SoTestData.new(@conn)
end

Given(/^Criteria_20_5 All test collections are installed in the mode$/) do

  aimdb = Sdps::AimDb.new(@conn)
  @collections = Array.new

  data = @sotestdata[@critinfo]
  data.should_not be_nil

  rv = true
  if data.length != 6
    puts "There should be six collections returned for #{@critinfo}, check AutoTestSoTestDataInfo table"
    rv = false
  else
    #loop through @data
    data.each do |dataitem|
      shortname = dataitem.shortname
      versionid = dataitem.versionid
      collection = aimdb.find_collection_by_shortname_versionid(shortname,versionid)
```

```
      if collection == nil
        puts "test data #{shortname}, #{versionid} not installed"
        rv = false
      else
        @collections.push(collection)
        #collection should have datamodeltype = 'ECS'
        if collection.datamodeltype != 'ECS'
          puts "collection's datamodeltype is not ECS"
          rv = false
        end
      end
    end
  end
  rv.should be_truthy
end

Given(/^Criteria_20_5 All test collections are configured for collection and granule metadata export$/) do
  bmgtdb = Sdps::Bmgt::BmgtDb.new(@conn)
  @collections.each do |collection|
    collectionid = collection.collectionid
    if bmgtdb.collection_export_enabled?(collectionid) == false
      bmgtdb.enable_collection_granule_export(collectionid).should be_truthy
    end
  end
end

Given(/^Criteria_20_5 All test collections have valid CoordinateSystem and GranuleSpatialRepresentation$/) do
  rv = true
  @collections.each do |collection|
    #set CoordinateSystem to CARTESIAN and GranuleSpatialRepresentation to GEODETIC
    collectionid = collection.collectionid
    begin
        rs = @conn.exec ("update bg_collection_configuration set GranuleSpatialRepresentation='GEODETIC',
CollectionCoordinateSystem='CARTESIAN' where collectionid = '#{collectionid}' returning 1")
        if rs == nil or rs.count != 1
```

```
          rv = false
        end
    rescue => e
        puts e.backtrace
        rv = false
    end
  end
  rv.should be_truthy
end

Given(/^Criteria_20_5 A local copy of the collection schema is available to validate against$/) do
  #verify Sdps::Bmgt::ECHO10_COLLECTION_SCHEMA exists
  schemaFile = Sdps::Bmgt::ECHO10_COLLECTION_SCHEMA
  File.exists?(schemaFile).should be_truthy
end

Given(/^Criteria_20_5 A TCP proxy is capturing BMGT traffic$/) do
 #This is checked at the initial setup
 puts "If it gets here, the TCP proxy is up and running"
end

When(/^Criteria_20_5 S-5 For each of the valid backtrack types, configure a collection for that type$/) do
  #There are six valid types, each associated with a unique metadata value

  valueAMSRA = "<OrbitParameters>\n  <SwathWidth>1600.0</SwathWidth>\n  <Period>101.0</Period>\n
<InclinationAngle>98.62</InclinationAngle>\n  <NumberOfOrbits>0.5</NumberOfOrbits>\n
<StartCircularLatitude>-90.0</StartCircularLatitude>\n</OrbitParameters>"

  valueAMSRE = "<OrbitParameters>\n  <SwathWidth>1450.0</SwathWidth>\n  <Period>98.88</Period>\n
<InclinationAngle>98.15</InclinationAngle>\n  <NumberOfOrbits>0.5</NumberOfOrbits>\n
<StartCircularLatitude>-90.0</StartCircularLatitude>\n</OrbitParameters>"

  valueGLASQuarter = "<OrbitParameters>\n  <SwathWidth>2.0</SwathWidth>\n  <Period>96.7</Period>\n
<InclinationAngle>94.0</InclinationAngle>\n  <NumberOfOrbits>0.25</NumberOfOrbits>\n
<StartCircularLatitude>-50.0</StartCircularLatitude>\n</OrbitParameters>"
```

```
  #in the doc, this one is for GLAS Single Orbit, but that's considered invalid for orbitgroup,it works for GLAS
Two Orbit
  valueGLASTwo = "<OrbitParameters>\n  <SwathWidth>2.0</SwathWidth>\n  <Period>96.7</Period>\n
<InclinationAngle>94.0</InclinationAngle>\n  <NumberOfOrbits>2.0</NumberOfOrbits>\n
<StartCircularLatitude>50.0</StartCircularLatitude>\n</OrbitParameters>"

  valueGLAS14 = "<OrbitParameters>\n  <SwathWidth>2.0</SwathWidth>\n  <Period>96.7</Period>\n
<InclinationAngle>94.0</InclinationAngle>\n  <NumberOfOrbits>14.0</NumberOfOrbits>\n
<StartCircularLatitude>50.0</StartCircularLatitude>\n</OrbitParameters>"

  valueMISR = "<OrbitParameters>\n  <SwathWidth>400.0</SwathWidth>\n  <Period>98.88</Period>\n
<InclinationAngle>98.3</InclinationAngle>\n  <NumberOfOrbits>1.0</NumberOfOrbits>\n
<StartCircularLatitude>0.0</StartCircularLatitude>\n</OrbitParameters>"


  @BackTrackTypesHash = Hash.new()
  @BackTrackTypesHash["AMSR-A"] = "#{valueAMSRA}"
  @BackTrackTypesHash["AMSR-E"] = "#{valueAMSRE}"
  @BackTrackTypesHash["GLAS Quarter Orbit"] = "#{valueGLASQuarter}"
  @BackTrackTypesHash["GLAS Two Orbit"] = "#{valueGLASTwo}"
  @BackTrackTypesHash["GLAS 14 Orbit"] = "#{valueGLAS14}"
  @BackTrackTypesHash["MISR"] = "#{valueMISR}"

  @BackTrackTypes = Array.new
  @BackTrackTypes.push('AMSR-A')
  @BackTrackTypes.push('AMSR-E')
  @BackTrackTypes.push('GLAS Quarter Orbit')
  @BackTrackTypes.push('GLAS Two Orbit')
  @BackTrackTypes.push('GLAS 14 Orbit')
  @BackTrackTypes.push('MISR')


  rv = true
  count = 0
  @collections.each do |collection|
    BackTrackType = @BackTrackTypes[count]
```

```
    puts "BackTrackType is #{BackTrackType}, ShortName is #{collection.shortname}"
    collectionid = collection.collectionid
    begin
      rs = @conn.exec ("update bg_collection_configuration set orbitGroup = '#{BackTrackType}' where
collectionid = '#{collectionid}' returning 1")
      if rs == nil or rs.count != 1
        rv = false
      end
    rescue => e
      puts e.backtrace
      rv = false
    end
    count = count + 1
  end
  rv.should be_truthy
end

When(/^Criteria_20_5 S-5 Request the export of collection metadata for a collection which is configured for that
type$/) do
  #record the request time
  @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")

  @collections.each do |collection|
    collectionid = collection.collectionid
    Sdps::Bmgt::ExportRequest.new(@mode).manual_export_for_collection?(collectionid, nil).should be_truthy,
"Falied running Manual Export for collection"
    ##make sure the request is queued
    Sdps::Bmgt::BmgtDb.new(@conn).get_coll_export_request_status(@request_time, collectionid, "MAN").should_not
be_nil
  end
end

Then(/^Criteria_20_5 Manual Export succeeds$/) do
  @collections.each do |collection|
    collectionid = collection.collectionid
    #poll and wait
```

```
    export_req = Sdps::Bmgt::BmgtDb.new(@conn)
    status = Sdps::Util.poll(180, 15){
      result = export_req.get_coll_export_request_status(@request_time, collectionid, "MAN")
      if (result == 'PENDING' || result == 'STARTED')
        nil
      else
        result
      end
    }
    status.should eq 'SUCCESS'
  end
end

Then(/^Criteria_20_5 V-12 Verify that all Collection metadata in the previous steps validates against
https:\/\/api\.echo\.nasa\.gov\/ingest\/schema\/Collection\.xsd$/) do
  #Verify the S-12 collection's exported XML validates against the schema:
  #The following method makes use of the local copy of the collection.xsd file:
Sdps::Bmgt::ECHO10_COLLECTION_SCHEMA
  count = 0
  @collections.each do |collection|
    collection_xml = @xmlUtil.get_collection_xml(@proxy_log_location, @request_time, collection)
    #puts "xml is #{collection_xml}"
    @xmlUtil.collection_xml_valid?(collection_xml).should be_truthy
    puts "Verification okay"

    #Verify the collection xml file contains the Spatial/OrbitParameters element, easier to combine the next
step here
    BackTrackType = @BackTrackTypes[count]
    puts "BackTrackType is #{BackTrackType}, ShortName is #{collection.shortname}"
    @xmlUtil.collection_xml_contains?(collection_xml, "//Collection/Spatial/OrbitParameters",
"#{@BackTrackTypesHash[BackTrackType]}").should be_truthy, "#{collection_xml}"
    puts "Contains Spatial/OrbitParameters okay"

    #Verify the collection xml file contains configured CoordinateSystem and GranuleSpatialRepresentation
values, again, easier to combine it here
    @xmlUtil.collection_xml_contains?(collection_xml,
```

```
"//Collection/Spatial/HorizontalSpatialDomain/Geometry/CoordinateSystem",
"<CoordinateSystem>CARTESIAN</CoordinateSystem>").should be_truthy
    puts "Contains CoordinateSystem okay"

    @xmlUtil.collection_xml_contains?(collection_xml, "//Collection/Spatial/GranuleSpatialRepresentation",
"<GranuleSpatialRepresentation>GEODETIC</GranuleSpatialRepresentation>").should be_truthy
    puts "contains GranuleSpatialRepresentation okay"
    count = count + 1
  end
end

Then(/^Criteria_20_5 V-5 Verify that the collection metadata generated in S-5 contains a
Spatial\/OrbitParameters element which is populated according to the rules for the backtrack metadata type
associated with the collection$/) do
  puts "If it gets here, the step is verified"

end

Then(/^Criteria_20_5 V-15 For the exports in S-5 to S-9, verify that the configured CoordinateSystem and
GranuleSpatialRepresentation values are included in the metadata$/) do
  puts "if it gets here, the step is verified"
end

Current Test Script: AutoExport_Collection_Update.rb
Given(/^criteria_10_S2_c mode is checked out$/) do

  @mode = ENV['MODE']
  @mode.should_not be_nil
  #host should be checked to be f5oml01v
  @mode_checker = Sdps::Mode.new(@mode)
  @mode_checker.mode_checked_out?().should be_truthy
  @db_subsystem = Sdps::Bmgt::DB_SUBSYSTEM
  @db_conn = Sdps::Db.connect(@db_subsystem, @mode)
  @db_conn.should_not be_nil

  @xmlUtil = Sdps::Bmgt::Xml.new()
```

```
  @bmgtdb = Sdps::Bmgt::BmgtDb.new(@db_conn)

  #Initialize critinfo and SoTestData
  @critinfo = "BE_82_01:10:2_C"
  @sotestdata = Sdps::SoTestData.new(@db_conn)

  @proxy_log_location = "/tools/common/test/BE_82_01/proxy/#{@mode}/logs/tcp.log"
end

Given(/^criteria_10_S2_c testdata is checked out$/) do
  criteria_10_s2_c_testdata_checked_out?().should be_truthy
end

Given(/^criteria_10_S2_c bmgt subsystem is checked out$/) do
  @mode_checker.bmgt_subsystem_checked_out?().should be_truthy, "bmgt system check failed"
end

Given(/^criteria_10_S2_c collection is installed$/) do
  @aimdb.is_collection_installed?(@coll.shortname, @coll.versionid).should be_truthy, "Collection for 10_S2_c
not installed."
end

Given(/^criteria_10_S2_c  collection is enabled for collection and granule export\.$/) do
  if (!@bmgtdb.collection_export_enabled?(@coll.collectionid))
    @bmgtdb.enable_collection_export(@coll.collectionid).should be_truthy
  end
end

When(/^criteria_10_S2_c Update the collection\.$/) do
  update_collection?().should be_truthy
end

Then(/^criteria_10_S2_c Verify the bg_export_request table shows a successful export for collection C1$/) do

  #poll and wait
  status = Sdps::Util.poll(300, 30){
```

```
    result = @bmgtdb.get_coll_export_request_status(@request_time, @coll.collectionid, "EVENT")
    if (result == 'PENDING' || result == 'STARTED')
      nil
    else
      result
    end
  }
  status.should eq 'SUCCESS'
end

Then(/^criteria_10_S2_c Verify that the TCP proxy log reports a single HTTP PUT request for collection C1$/) do
  dataset =  @coll.datasetid()
  @xmlUtil.http_put_for_collection?(@proxy_log_location, @request_time, dataset).should be_truthy
end

Then(/^criteria_10_S2_c Verify that the HTTP PUT request contains collection C1's full collection metadata\.$/)
do
  @collection_xml = @xmlUtil.get_collection_xml(@proxy_log_location, @request_time, @coll)
  @xmlUtil.collection_xml_valid?(@collection_xml).should be_truthy
end

Then(/^criteria_10_S2_c Verify that the metadata exported validates against the ECHO collection metadata schema
\(https:\/\/api\.echo\.nasa\.gov\/ingest\/schema\/Collection\.xsd\)$/) do
  @xmlUtil.collection_xml_valid?(@collection_xml).should be_truthy
end

Then(/^criteria_10_S2_c Verify that the metadata exported has the element Visible = true$/) do
  @xmlUtil.collection_xml_contains?(@collection_xml,"//Collection/Visible", "<Visible>true</Visible>").should
be_truthy
end

Then(/^criteria_10_S2_c Verify that the metadata exported has the element Orderable = false$/) do
  @xmlUtil.collection_xml_contains?(@collection_xml,"//Collection/Orderable",
"<Orderable>false</Orderable>").should be_truthy
end
```

```
Then(/^criteria_10_S2_c Verify that the metadata exported has the element InsertTime = The insert time of the
collection recorded in the AIM database\.$/) do

  insertTime = @coll.inserttime
  insertTime.should_not be_nil
  insertTime_in_db = DateTime.parse(insertTime.to_s).strftime('%Y-%m-%dT%H:%M:%S.%3NZ')
  @xmlUtil.collection_xml_contains?(@collection_xml, "//Collection/InsertTime",
"<InsertTime>#{insertTime_in_db}</InsertTime>").should be_truthy

end

Then(/^criteria_10_S2_c Verify that the metadata exported has the element LastUpdate = The last update time of
the collection recorded in the AIM database$/) do

  lastUpdate = @coll.lastupdate
  lastUpdate.should_not be_nil
  lastUpdate_in_db = DateTime.parse(lastUpdate.to_s).strftime('%Y-%m-%dT%H:%M:%S.%3NZ')

  @xmlUtil.collection_xml_contains?(@collection_xml, "//Collection/LastUpdate",
"<LastUpdate>#{lastUpdate_in_db}</LastUpdate>").should be_truthy

end

Then(/^criteria_10_S2_c Verify that the metadata exported contains version numbers with leading zeroes$/) do
  vId = "%03d" %  @coll.versionid
  @xmlUtil.collection_xml_contains?(@collection_xml, "//Collection/VersionId",
"<VersionId>#{vId}</VersionId>").should be_truthy
end

def criteria_10_s2_c_testdata_checked_out?

  @data = @sotestdata[@critinfo]
  @data.should_not be_nil

  rv = true
  if @data.length != 1
```

```
    puts "There should be one collections returned for #{@critinfo}, check AutoTestSoTestDataInfo table"
    rv = false
  else
    @shortname = @data[0].shortname
    @versionid = @data[0].versionid

    @aimdb = Sdps::AimDb.new(@db_conn)
    @coll = @aimdb.find_collection_by_shortname_versionid(@shortname, @versionid)

    if (@coll == nil || !@aimdb.is_collection_installed?(@shortname, @versionid))
      puts "#{@shortname}, #{@versionid} is not installed"
      rv = false
    else
      #collection should have datamodeltype = 'ECS'
      if @coll.datamodeltype != 'ECS'
        puts "#{@shortname}, #{@versionid}'s datamodeltype is not ECS #{@coll.datamodeltype} **"
        rv = false
      end
    end
  end
  rv.should be_truthy
end

def update_collection?

  begin
    @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
    @db_conn.exec("update DsGeESDTConfiguredType set esdtstate = 'updating'
            where configuredname = '#{@coll.shortname}' and versionid = #{@coll.versionid}")
    @db_conn.exec("update DsGeESDTConfiguredType set esdtstate = 'installed'
            where configuredname = '#{@coll.shortname}' and versionid = #{@coll.versionid}")

   rs =  @db_conn.exec("select eventkey, dbid from dsmdgreventhistory
            where dbid = #{@coll.collectionid} and eventtype = 'CLUPDATE'and eventTime > '#{@request_time}'")
    if rs != nil and rs.count > 0
      true
```

```
    else
      false
    end
  rescue =>e
    puts e.backtrace
    false
  end

end
```

Current Test Script: BE_82_01_40_V5b.rb

Current Test Script: ManualExport_Insert_Time.rb

```
Given(/^criteria_200 mode is checked out$/) do
  @mode = ENV['MODE']
  @mode.should_not be_nil
  #host should be checked to be f5oml01v
  @mode_checker = Sdps::Mode.new(@mode)
  @mode_checker.mode_checked_out?().should be_truthy
  @db_subsystem = Sdps::Bmgt::DB_SUBSYSTEM
  @db_conn = Sdps::Db.connect(@db_subsystem, @mode)
  @db_conn.should_not be_nil

  @xmlUtil = Sdps::Bmgt::Xml.new()
  @bmgtdb = Sdps::Bmgt::BmgtDb.new(@db_conn)

  #Initialize critinfo and SoTestData
  @critinfo = "BE_82_01:200"
  @sotestdata = Sdps::SoTestData.new(@db_conn)

  @proxy_log_location = "/tools/common/test/BE_82_01/proxy/#{@mode}/logs/tcp.log"
  @manual_log = "/usr/ecs/#{@mode}/CUSTOM/logs/EcBmBMGTManualDriver.log"
end

Given(/^criteria_200 bmgt subsystem is checked out$/) do
```

```
  @mode_checker.bmgt_subsystem_checked_out?().should be_truthy, "bmgt system check failed"
end

Given(/^criteria_200 testdata is checked out$/) do
  criteria_200_testdata_checked_out?().should be_truthy
end

Given(/^criteria_200 collections are installed$/) do
  @colls.each do |coll |
    @aimdb.is_collection_installed?(coll.shortname, coll.versionid).should be_truthy,
"#{coll.shortname}.#{coll.versionid} for Criteria 200 not installed."
  end
end

Given(/^criteria_200  collections are enabled for collection and granule export$/) do
  @colls.each do |coll |
    @bmgtdb.enable_collection_granule_export(coll.collectionid).should be_truthy
  end
end

Given(/^criteria_200 collections have few granules ingested$/) do
  check_for_granules?().should be_truthy
end

Given(/^criteria_200 Identify two granules within a collection and their insert times$/) do
  identify_criteria200_granules();
end

When(/^criteria_200 Request export of granule metadata for collection based on insert times$/) do
  options  = Array.new
  options.push('-c')
  options.push(@colls[0].collectionid)

  time_based_manual_export?('-metg', options).should be_truthy
  gran_requests_processed_successfully?().should be_truthy
end
```

```
Then(/^criteria_200 Verify the granule at starttime is exported but not the one with the end time$/) do
  @bmgtdb.get_gran_export_request_status(@request_time, @granules[0], 'MAN').should_not be_nil
  @bmgtdb.get_gran_export_request_status(@request_time, @granules[3], 'MAN').should be_nil
end

Then(/^criteria_200 Verify that the TCP proxy shows an HTTP PUT\/DELETE request each for granules g1, g2, g3$/)
do
  i = 0;
  until i > 2 do

    if(@aimdb.granule_deleted?(@granules[i]))
      @xmlUtil.http_delete_for_granule?(@proxy_log_location, @request_time, @granules[i]).should be_truthy
    else
      @xmlUtil.http_put_for_granule?(@proxy_log_location, @request_time, @granules[i]).should be_truthy
    end

    i += 1
  end
end

Then(/^criteria_200 Verify that the TCP proxy does not show any request for granule g4$/) do
  @xmlUtil.http_put_for_granule?(@proxy_log_location, @request_time, @granules[3]).should be_falsey
  @xmlUtil.http_delete_for_granule?(@proxy_log_location, @request_time, @granules[3]).should be_falsey
end

Then(/^criteria_200 Verify that the TCP proxy shows full granule metadata for each of the non deleted granules
g1, g2, g3$/) do
  i = 0;
  until i > 2 do
    if(!@aimdb.granule_deleted?(@granules[i]))
      granule_xml = @xmlUtil.get_granule_xml(@proxy_log_location, @request_time, @granules[i])
      @xmlUtil.granule_xml_valid?(granule_xml).should be_truthy
    end
    i += 1
  end
```

```
end

Then(/^criteria_200 Identify two collections and their inserttimes$/) do
  @starttime = @colls[0].inserttime.strftime("%Y-%m-%d %H:%M:%S")
  @endtime = @colls[3].inserttime.strftime("%Y-%m-%d %H:%M:%S")
end

Then(/^criteria_200 Request export of collection metadata based on insert times$/) do
  time_based_manual_export?('-metc',nil).should be_truthy
  coll_requests_processed_successfully?().should be_truthy
end

Then(/^criteria_200 Verify the collection at starttime is exported but not the one with the end time$/) do
  @bmgtdb.get_coll_export_request_status(@request_time, @colls[0].collectionid, 'MAN').should_not be_nil
  @bmgtdb.get_coll_export_request_status(@request_time, @colls[3].collectionid, 'MAN').should be_nil
end

Then(/^criteria_200 Verify that the TCP proxy shows an HTTP PUT request each for collection C1, C2, C3$/) do
  i = 0;
  until i > 2 do
    dataset =  @colls[i].datasetid()
    @xmlUtil.http_put_for_collection?(@proxy_log_location, @request_time, dataset).should be_truthy
    i += 1
  end
end

Then(/^criteria_200 Verify that the TCP proxy does not show any request for collection C4$/) do
  dataset =  @colls[3].datasetid()
  @xmlUtil.http_put_for_collection?(@proxy_log_location, @request_time, dataset).should be_falsey
  @xmlUtil.http_delete_for_collection?(@proxy_log_location, @request_time, dataset).should be_falsey
end

Then(/^criteria_200 Verify that the TCP proxy shows full collection metadata for each of the collection C1, C2,
C3$/) do
  i = 0;
  until i > 2 do
```

```
    collection_xml = @xmlUtil.get_collection_xml(@proxy_log_location, @request_time, @colls[i])
    @xmlUtil.collection_xml_valid?(collection_xml).should be_truthy
    i += 1
  end
end

def criteria_200_testdata_checked_out?

  @colls = Array.new
  @data = @sotestdata[@critinfo]
  @data.should_not
  cids = Array.new
  i = 0
  rv = true
  if @data.length != 4
    puts "There should be 4 collections returned for #{@critinfo}, check AutoTestSoTestDataInfo table"
    rv = false
  else
    @data.each do |esdt|
      shortname = @data[i].shortname
      versionid = @data[i].versionid
      i += 1

      @aimdb = Sdps::AimDb.new(@db_conn)
      coll = @aimdb.find_collection_by_shortname_versionid(shortname, versionid)

      if (coll == nil || !@aimdb.is_collection_installed?(shortname, versionid))
        puts "#{shortname}, #{versionid} is not installed"
        rv = false
      else
        #collection should have datamodeltype = 'ECS'
        if coll.datamodeltype != 'ECS'
          puts "collection's datamodeltype is not ECS"
          rv = false
        end
      end
```

```
      rv.should be_truthy
      @colls.push(coll)
    end
    @colls.sort_by!(&:inserttime)

  end
  rv.should be_truthy
end

def identify_criteria200_granules()

  @granules = @bmgtdb.get_granules_by_time(@colls[0].collectionid, 'archivetime', 4 )
  @granules.should_not be_nil
  last  = @granules.size - 1
  @starttime = @aimdb.find_granule_by_granuleid(@granules[0]).archivetime.strftime("%Y-%m-%d %H:%M:%S")
  @endtime = @aimdb.find_granule_by_granuleid(@granules[3]).archivetime.strftime("%Y-%m-%d %H:%M:%S")
end

def check_for_granules?()
  rv = false
  granules = @aimdb.find_notdeleted_granuleids_by_collectionid(@colls[0].collectionid)

  if(granules.length > 0)
    rv = true;
  else
    rv = false;
  end
  return rv
end

def time_based_manual_export?(type, options)
  @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
  rv = Sdps::Bmgt::ExportRequest.new(@mode).manual_time_export?(type, @starttime, @endtime, options)
  rv.should be_truthy
  return rv
end
```

```
def coll_requests_processed_successfully?()
  begin
    rv = true;
    i = 0
    until i > 2 do
      status = Sdps::Util.poll(180, 15){
        result = @bmgtdb.get_coll_export_request_status(@request_time,@colls[i].collectionid, 'MAN')
        if (result == 'PENDING' || result == 'STARTED')
          nil
        else
          result
        end
      }
      rv = false if (status != 'SUCCESS')
      i += 1
    end
    return rv;
  end
end
def gran_requests_processed_successfully?()
  begin
    rv = true;
    i = 0
    until i > 2 do
      status = Sdps::Util.poll(180, 15){
        result = @bmgtdb.get_gran_export_request_status(@request_time,@granules[i], 'MAN')
        if (result == 'PENDING' || result == 'STARTED')
          nil
        else
          result
        end
      }
      rv = false if (status != 'SUCCESS')
      i += 1
```

```
    end
    return rv;
  end
end
Current Test Script: Longform_InsertTime_Export.rb
Given(/^criteria_430 mode is checked out$/) do
  @mode = ENV['MODE']
  @mode.should_not be_nil
  #host should be checked to be f5oml01v
  @mode_checker = Sdps::Mode.new(@mode)
  @mode_checker.mode_checked_out?().should be_truthy
  @db_subsystem = Sdps::Bmgt::DB_SUBSYSTEM
  @db_conn = Sdps::Db.connect(@db_subsystem, @mode)
  @db_conn.should_not be_nil

  @xmlUtil = Sdps::Bmgt::Xml.new()
  @bmgtdb = Sdps::Bmgt::BmgtDb.new(@db_conn)

  #Initialize critinfo and SoTestData
  @critinfo = "BE_82_01:200"
  @sotestdata = Sdps::SoTestData.new(@db_conn)

  @proxy_log_location = "/tools/common/test/BE_82_01/proxy/#{@mode}/logs/tcp.log"

end

Given(/^criteria_430 bmgt subsystem is checked out$/) do
  @mode_checker.bmgt_subsystem_checked_out?().should be_truthy, "bmgt system check failed"
end

Given(/^criteria_430 testdata is checked out$/) do
  criteria_430_testdata_checked_out?().should be_truthy
end

Given(/^criteria_430 collections are installed$/) do
  @colls.each do |coll |
```

```
    @aimdb.is_collection_installed?(coll.shortname, coll.versionid).should be_truthy,
"#{coll.shortname}.#{coll.versionid} for Criteria 430 not installed."
  end
end

Given(/^criteria_430 collections are enabled for collection and granule export$/) do
  @colls.each do |coll |
    @bmgtdb.enable_collection_granule_export(coll.collectionid).should be_truthy
  end
end

Given(/^criteria_430 collections have few granules ingested$/) do
  check_for_granules?().should be_truthy
end

Given(/^criteria_430 Identify two granules within a collection and their insert times$/) do
  identify_criteria_granules();
end

When(/^criteria_430 Request longform verification export of granule metadata for collection based on insert
times$/) do
  options  = Array.new
  options.push('-vl')
  options.push('-c')
  options.push(@colls[0].collectionid)

  time_based_verification_export?('-metg', options).should be_truthy
  ver_gran_requests_processed_successfully?().should be_truthy
end

Then(/^criteria_430 Verify the granule at starttime is exported but not the one with the end time$/) do
  @bmgtdb.get_gran_export_request_status(@request_time, @granules[0], 'VER').should_not be_nil
  @bmgtdb.get_gran_export_request_status(@request_time, @granules[3], 'VER').should be_nil
end

Then(/^criteria_430 Verify that the TCP proxy shows an HTTP PUT request each for granules g1, g2, g3$/) do
```

```
  i = 0;
  until i > 2 do
    @xmlUtil.http_put_for_granule?(@proxy_log_location, @request_time, @granules[i]).should be_truthy
    i += 1
  end
end

Then(/^criteria_430 Verify that the TCP proxy does not show any request for granule g4$/) do
  @xmlUtil.http_put_for_granule?(@proxy_log_location, @request_time, @granules[3]).should be_falsey
  @xmlUtil.http_delete_for_granule?(@proxy_log_location, @request_time, @granules[3]).should be_falsey
end

Then(/^criteria_430 Verify that the TCP proxy shows full granule metadata for each of the granules g1, g2, g3$/)
do
  i = 0;
  until i > 2 do
    granule_xml = @xmlUtil.get_granule_xml(@proxy_log_location, @request_time, @granules[i])
    @xmlUtil.granule_xml_valid?(granule_xml).should be_truthy
    i += 1
  end
end

Then(/^criteria_430 Verify that the TCP proxy shows that the export request contains a HTTP query parameter
xml_diff=true for each of the granules g1, g2, g3 indicating verification request$/) do
  i = 0;
  until i > 2 do
    @xmlUtil.http_verification_put_for_granule?(@proxy_log_location, @request_time, @granules[i]).should
be_truthy
    i += 1
  end
end

Then(/^criteria_430 Identify two collections and their inserttimes$/) do
  @starttime = @colls[0].inserttime.strftime("%Y-%m-%d %H:%M:%S")
  @endtime = @colls[3].inserttime.strftime("%Y-%m-%d %H:%M:%S")
end
```

```
Then(/^criteria_430 Request longform verification export of collection metadata based on insert times$/) do
  options  = Array.new
  options.push('-vl')
  time_based_verification_export?('-metc', options).should be_truthy
  ver_coll_requests_processed_successfully?().should be_truthy
end

Then(/^criteria_430 Verify the collection at starttime is exported but not the one with the end time$/) do
  @bmgtdb.get_coll_export_request_status(@request_time, @colls[0].collectionid, 'VER').should_not be_nil
  @bmgtdb.get_coll_export_request_status(@request_time, @colls[3].collectionid, 'VER').should be_nil
end

Then(/^criteria_430 Verify that the TCP proxy shows an HTTP PUT request each for collection C1, C2, C3$/) do
  i = 0;
  until i > 2 do
    dataset =  @colls[i].datasetid()
    @xmlUtil.http_put_for_collection?(@proxy_log_location, @request_time, dataset).should be_truthy
    i += 1
  end
end

Then(/^criteria_430 Verify that the TCP proxy does not show any request for collection C4$/) do
  dataset =  @colls[3].datasetid()
  @xmlUtil.http_put_for_collection?(@proxy_log_location, @request_time, dataset).should be_falsey
  @xmlUtil.http_delete_for_collection?(@proxy_log_location, @request_time, dataset).should be_falsey
end

Then(/^criteria_430 Verify that the TCP proxy shows full collection metadata for each of the collection C1, C2,
C3$/) do
  i = 0;
  until i > 2 do
    collection_xml = @xmlUtil.get_collection_xml(@proxy_log_location, @request_time, @colls[i])
    @xmlUtil.collection_xml_valid?(collection_xml).should be_truthy
    i += 1
  end
```

```
end
Then(/^criteria_430 Verify that the TCP proxy shows that the export request contains a HTTP query parameter
xml_diff=true for each of the collections C1, C2, C3 indicating verification request$/) do
  i = 0;
  until i > 2 do
        dataset = @colls[i].datasetid()
        @xmlUtil.http_verification_put_for_collection?(@proxy_log_location, @request_time, dataset).should
be_truthy
    i += 1
  end
end

def criteria_430_testdata_checked_out?

  @colls = Array.new
  @data = @sotestdata[@critinfo]
  @data.should_not
  cids = Array.new
  i = 0
  rv = true
  if @data.length != 4
    puts "There should be 4 collections returned for #{@critinfo}, check AutoTestSoTestDataInfo table"
    rv = false
  else
    @data.each do |esdt|
      shortname = @data[i].shortname
      versionid = @data[i].versionid
      i += 1

      @aimdb = Sdps::AimDb.new(@db_conn)
      coll = @aimdb.find_collection_by_shortname_versionid(shortname, versionid)

      if (coll == nil || !@aimdb.is_collection_installed?(shortname, versionid))
        puts "#{shortname}, #{versionid} is not installed"
        rv = false
      else
```

```
        #collection should have datamodeltype = 'ECS'
        if coll.datamodeltype != 'ECS'
          puts "collection's datamodeltype is not ECS"
          rv = false
        end
      end
      rv.should be_truthy
      @colls.push(coll)
    end
    @colls.sort_by!(&:inserttime)

  end
  rv.should be_truthy
end

def identify_criteria_granules()

  @granules = @bmgtdb.get_granules_by_time(@colls[0].collectionid, 'archivetime', 4 )
  @granules.should_not be_nil
  last  = @granules.size - 1
  @starttime = @aimdb.find_granule_by_granuleid(@granules[0]).archivetime.strftime("%Y-%m-%d %H:%M:%S")
  @endtime = @aimdb.find_granule_by_granuleid(@granules[3]).archivetime.strftime("%Y-%m-%d %H:%M:%S")
end

def check_for_granules?()
  rv = false
  granules = @aimdb.find_notdeleted_granuleids_by_collectionid(@colls[0].collectionid)

  if(granules.length > 0)
    rv = true;
  else
    rv = false;
  end
  return rv
end
```

```
def time_based_verification_export?(type, options)
  @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
  rv = Sdps::Bmgt::ExportRequest.new(@mode).manual_time_export?(type, @starttime, @endtime, options)
  rv.should be_truthy
  return rv
end


def ver_coll_requests_processed_successfully?()
  begin
    rv = true;
    i = 0
    until i > 2 do
      status = Sdps::Util.poll(180, 15){
        result = @bmgtdb.get_coll_export_request_status(@request_time,@colls[i].collectionid, 'VER')
        if (result == 'PENDING' || result == 'STARTED')
          nil
        else
          result
        end
      }
      rv = false if (status != 'SUCCESS' && status != 'WARNING')
      i += 1
    end
    return rv;
  end
end
def ver_gran_requests_processed_successfully?()
  begin
    rv = true;
    i = 0
    until i > 2 do
      status = Sdps::Util.poll(180, 15){
        result = @bmgtdb.get_gran_export_request_status(@request_time,@granules[i], 'VER')
        if (result == 'PENDING' || result == 'STARTED')
          nil
```

```
        else
          result
        end
      }
      rv = false if (status != 'SUCCESS' && status != 'WARNING')
      i += 1
    end
    return rv;
  end
end

Current Test Script: BE_82_01_40_common.rb


Given(/^nocustomdatacheck$/) do
  true
end

Then(/^nocustomverify$/) do
  true
end
Current Test Script: BE_82_01_40_V4.rb

Current Test Script: AutoExport_Collection_Enable.rb
Given(/^criteria_10_S2_d mode is checked out$/) do
  @mode = ENV['MODE']
  @mode.should_not be_nil
  #host should be checked to be f5oml01v
  @mode_checker = Sdps::Mode.new(@mode)
  @mode_checker.mode_checked_out?().should be_truthy
  @db_subsystem = Sdps::Bmgt::DB_SUBSYSTEM
  @db_conn = Sdps::Db.connect(@db_subsystem, @mode)
  @db_conn.should_not be_nil

  @xmlUtil = Sdps::Bmgt::Xml.new()
  @bmgtdb = Sdps::Bmgt::BmgtDb.new(@db_conn)
```

```
  #Initialize critinfo and SoTestData
  @critinfo = "BE_82_01:10:2_D"
  @sotestdata = Sdps::SoTestData.new(@db_conn)

  @proxy_log_location = "/tools/common/test/BE_82_01/proxy/#{@mode}/logs/tcp.log"
end

Given(/^criteria_10_S2_d testdata is checked out$/) do
  criteria_10_S2_d_testdata_checked_out?().should be_truthy
end

Given(/^criteria_10_S2_d bmgt subsystem is checked out$/) do
  @mode_checker.bmgt_subsystem_checked_out?().should be_truthy, "bmgt system check failed"
end

Given(/^criteria_10_S2_d collection is installed$/) do

  @aimdb.is_collection_installed?(@shortname, @versionid).should be_truthy, "Collection for 10_S2_d not
installed."

end

Given(/^criteria_10_S2_d  collection is disabled for collection and granule export\.$/) do
      @bmgtdb.disable_collection_granule_export(@coll.collectionid).should be_truthy
end

When(/^criteria_10_S2_d Enable the collection for collection Export\.$/) do

  @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
  @bmgtdb.enable_collection_export(@coll.collectionid).should be_truthy

end

Then(/^criteria_10_S2_d Verify the bg_export_request table shows a successful export for collection C1$/) do
```

```
  #poll and wait

  status = Sdps::Util.poll(180, 15){
    result = @bmgtdb.get_coll_export_request_status(@request_time, @coll.collectionid, "NEW")
    if (result == 'PENDING' || result == 'STARTED')
      nil
    else
      result
    end
  }
  status.should eq 'SUCCESS'
end

Then(/^criteria_10_S2_d Verify that the TCP proxy log reports a single HTTP PUT request for collection C1$/) do

  dataset =  @coll.datasetid()
  @xmlUtil.http_put_for_collection?(@proxy_log_location, @request_time, dataset).should be_truthy

end

Then(/^criteria_10_S2_d Verify that the HTTP PUT request contains collection C1's full collection metadata\.$/)
do

  @collection_xml = @xmlUtil.get_collection_xml(@proxy_log_location, @request_time, @coll)
  @xmlUtil.collection_xml_valid?(@collection_xml).should be_truthy

end

Then(/^criteria_10_S2_d Verify that the metadata exported validates against the ECHO collection metadata schema
\(https:\/\/api\.echo\.nasa\.gov\/ingest\/schema\/Collection\.xsd\)$/) do

  @xmlUtil.collection_xml_valid?(@collection_xml).should be_truthy

end

Then(/^criteria_10_S2_d Verify that the metadata exported has the element Visible = true$/) do
```

```
  @xmlUtil.collection_xml_contains?(@collection_xml,"//Collection/Visible", "<Visible>true</Visible>").should
be_truthy

end

Then(/^criteria_10_S2_d Verify that the metadata exported has the element Orderable = false$/) do

  @xmlUtil.collection_xml_contains?(@collection_xml,"//Collection/Orderable",
"<Orderable>false</Orderable>").should be_truthy

end

Then(/^criteria_10_S2_d Verify that the metadata exported has the element InsertTime = The insert time of the
collection recorded in the AIM database\.$/) do

  insertTime = @coll.inserttime
  insertTime.should_not be_nil
  insertTime_in_db = DateTime.parse(insertTime.to_s).strftime('%Y-%m-%dT%H:%M:%S.%3NZ')
  @xmlUtil.collection_xml_contains?(@collection_xml, "//Collection/InsertTime",
"<InsertTime>#{insertTime_in_db}</InsertTime>").should be_truthy

end

Then(/^criteria_10_S2_d Verify that the metadata exported has the element LastUpdate = The last update time of
the collection recorded in the AIM database$/) do

  lastUpdate = @coll.lastupdate
  lastUpdate.should_not be_nil
  lastUpdate_in_db = DateTime.parse(lastUpdate.to_s).strftime('%Y-%m-%dT%H:%M:%S.%3NZ')

  @xmlUtil.collection_xml_contains?(@collection_xml, "//Collection/LastUpdate",
"<LastUpdate>#{lastUpdate_in_db}</LastUpdate>").should be_truthy

end
```

```
Then(/^criteria_10_S2_d Verify that the metadata exported contains version numbers with leading zeroes$/) do
  vId = "%03d" %  @coll.versionid #changed to reflect NCR 8052154's bmgt version id conversion
  @xmlUtil.collection_xml_contains?(@collection_xml, "//Collection/VersionId",
"<VersionId>#{vId}</VersionId>").should be_truthy
end

def criteria_10_S2_d_testdata_checked_out?

  @data = @sotestdata[@critinfo]
  @data.should_not be_nil

  rv = true
  if @data.length != 1
    puts "There should be one collections returned for #{@critinfo}, check AutoTestSoTestDataInfo table"
    rv = false
  else
    @shortname = @data[0].shortname
    @versionid = @data[0].versionid

    @aimdb = Sdps::AimDb.new(@db_conn)
    @coll = @aimdb.find_collection_by_shortname_versionid(@shortname, @versionid)

    if (@coll == nil || !@aimdb.is_collection_installed?(@shortname, @versionid))
      puts "#{@shortname}, #{@versionid} is not installed"
      rv = false
    else
      #collection should have datamodeltype = 'ECS'
      if @coll.datamodeltype != 'ECS'
        puts "#{@shortname}, #{@versionid} datamodeltype is not ECS"
        rv = false
      end
    end
  end
  rv.should be_truthy
end
```

Current Test Script: BE_82_01_40_test_data.rb

```
Given(/^Ensure test granule is not ingested$/) do
  # test 40:S2a
  @so_test_data_granules = get_so_test_data_granules(@so_test_data_items)
  @so_test_data_granules.each { |so_test_data_granule|
        granules = @aimdb.find_granules_by_localgranuleid(so_test_data_granule.localgranuleid)
        if granules != nil and granules.count > 0
          granules.each { |granule|
             #granule should not be ingested or should be deleted
             #TODO: Call granule delete to delete all the test granules
             if granule != nil
                 granule.deleteeffectivedate.should be_nil, "granule not deleted- #{granule.granuleid}"
            end
          }
        end
  }
end

Given(/^Ensure test granule is not hidden$/) do
  # test 40:S2e
  get_test_granules_ingested_notdeleted(@so_test_data_items)
  @test_granules1.should_not be_nil, "granules not ingested for collection1"
  @test_granules1.length.should_not be_zero, "granules not ingested for collection1"

  @test_granule = @test_granules1[0]
  @test_granule.should_not be_nil
  if @test_granule.deletefromarchive == 'H'
     #unhide granule
     @aimdb.unhide_granule(@test_granule.granuleid)
     @test_granule = @aimdb.find_granule_by_granuleid(@test_granule.granuleid)
  end
  @test_granule.deletefromarchive.should_not eq 'H'
end

Given(/^Ensure test granule is not restricted$/) do
```

```
  # test 40:S2f
  get_test_granules_ingested_notdeleted(@so_test_data_items)
  @test_granules1.should_not be_nil, "granules not ingested for collection1"
  @test_granules1.length.should_not be_zero, "granules not ingested for collection1"

  @test_granule = @test_granules1[0]
  if @aimdb.granule_restricted?(@test_granule.granuleid) == true
     request_time =  Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S");
     @aimdb.unrestrict_granule(@test_granule.granuleid)
     #poll status and wait till timeout
     status = Sdps::Util.poll(300, 5){
       result = Sdps::Bmgt::BmgtDb.new(@db_conn).get_gran_export_request_status(request_time,
@test_granule.granuleid, 'EVENT')
       if result != 'SUCCESS'
          nil
       else
          result
       end
     }
     puts status
     status.should eq 'SUCCESS'
  end
  @aimdb.granule_restricted?(@test_granule.granuleid).should be_falsey, "granule is already restricted"
end

Given(/^Ensure test granule is restricted$/) do
  # test 40:S2g
  get_test_granules_ingested_notdeleted(@so_test_data_items)
  @test_granules1.should_not be_nil, "granules not ingested for collection1"
  @test_granules1.length.should_not be_zero, "granules not ingested for collection1"

  @test_granule = @test_granules1[0]

  if @aimdb.granule_restricted?(@test_granule.granuleid) == false
     @restriction_flag = 7
     @description = "Testing BE_82_01 Crit 40 S2g"
```

```
     @aimdb.add_restriction(@restriction_flag, @description);
     request_time =  Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S");
     status = @aimdb.restrict_granule(@test_granule.granuleid,@restriction_flag);
     #poll status and wait till timeout
     status = Sdps::Util.poll(300, 5){
     result = Sdps::Bmgt::BmgtDb.new(@db_conn).get_gran_export_request_status(request_time,
@test_granule.granuleid, 'EVENT')
       if result != 'SUCCESS'
          nil
       else
          result
       end
     }
     status.should eq 'SUCCESS'
  end
  @aimdb.granule_restricted?(@test_granule.granuleid).should be_truthy, "granule is not restricted"
end


Given(/^Ensure test granule is not published$/) do
  # test 40:S2i
  get_test_granules_ingested_notdeleted(@so_test_data_items)
  @test_granules1.should_not be_nil, "granules not ingested for collection1"
  @test_granules1.length.should_not be_zero, "granules not ingested for collection1"

  @test_granule = @test_granules1[0]
  @aimdb = Sdps::AimDb.new(@db_conn)
  if @aimdb.granule_published?(@test_granule.granuleid) == true
     request_time =  Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S");
     Sdps::Publishing.new(@mode).unpublish_granule(@test_granule.granuleid)
     #poll status and wait till timeout
     status = Sdps::Util.poll(300, 5){
       result = Sdps::Bmgt::BmgtDb.new(@db_conn).get_gran_export_request_status(request_time,
@test_granule.granuleid, 'EVENT')
       if result != 'SUCCESS'
```

```
         nil
      else
         result
      end
    }
    puts status
    status.should eq 'SUCCESS'
  end
  @aimdb.granule_published?(@test_granule.granuleid).should be_falsey, "granule is already published"
end

Given(/^Ensure test granule is published$/) do
  # test 40:S2j
  get_test_granules_ingested_notdeleted(@so_test_data_items)
  @test_granules1.should_not be_nil, "granules not ingested for collection1"
  @test_granules1.length.should_not be_zero, "granules not ingested for collection1"

  @test_granule = @test_granules1[0]

  @aimdb = Sdps::AimDb.new(@db_conn)
  if @aimdb.granule_published?(@test_granule.granuleid) == false
    request_time =  Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S");
    status = Sdps::Publishing.new(@mode).publish_granule(@test_granule.granuleid);
    #poll status and wait till timeout
    status = Sdps::Util.poll(300, 5){
    result = Sdps::Bmgt::BmgtDb.new(@db_conn).get_gran_export_request_status(request_time,
@test_granule.granuleid, 'EVENT')
      if result != 'SUCCESS'
         nil
      else
         result
      end
    }
    status.should eq 'SUCCESS'
  end
  @aimdb.granule_published?(@test_granule.granuleid).should be_truthy, "granule is not published"
```

```
end

Given(/^Ensure test granule is not linked to browse$/) do
  # test 40:S2k
    get_test_granules_ingested_notdeleted(@so_test_data_items)
    @test_granules1.should_not be_nil, "granules not ingested for collection1"
    @test_granules1.length.should_not be_zero, "granules not ingested for collection1"

    @test_granules1.length.should_not be_zero
    @test_granule = @test_granules1[0]
                #browse granule should not be ingested
     if @test_granule != nil
       browse_granules = @aimdb.find_associated_browse_by_granuleid(@test_granule.granuleid)
          if browse_granules != nil
               #TODO: delete the test granules with the browse and check again
               browse_granules.count.should be_zero, "browse granule already installed"
          end
     end
end

Given(/^Ensure test granule is linked to browse$/) do
  # test 40:S2l
    get_test_granules_ingested_notdeleted(@so_test_data_items)
    @test_granules1.should_not be_nil, "granules not ingested for collection1"
    @test_granules1.length.should_not be_zero, "granules not ingested for collection1"

     # both science granule and browse granule should be ingested
     # they should be linked
     @test_granules1.length.should_not be_zero
     @test_granule = @test_granules1[0]
     #browse granule should not be ingested
     if @test_granule != nil
           browse_granules = @aimdb.find_associated_browse_by_granuleid(@test_granule.granuleid)
           browse_granules.should_not be_nil, "browse granule not ingested and linked"
           browse_granules.count.should_not be_zero, "browse granule not ingested and linked"
     end
```

```
end

Given(/^Ensure test granules are public$/) do
  #test 40:S2n - exports all granules..so not doing this step
  #need to get all undeleted granules for the collection being moved
       get_test_granules_ingested_notdeleted(@so_test_data_items)
       @test_granules1.should_not be_nil, "granules not ingested for collection1"
       @test_granules1.count.should_not be_zero, "granules not ingested for collection1"

       #save the non public granules to a separate array
       @test_granules2 = @test_granules1.select { |test_granule|
         @aimdb.granule_published?(test_granule.granuleid) == false
       }

       #delete non public granules from test granules
       @test_granules1.delete_if  { |test_granule|
               @aimdb.granule_published?(test_granule.granuleid) == false
        }
end

Given(/^Ensure test granules are ingested$/) do
      get_test_granules_ingested_notdeleted(@so_test_data_items)
      @test_granules1.should_not be_nil, "granules not ingested for collection1"
      @test_granules1.count.should_not be_zero, "granules not ingested for collection1"
end

Given(/^Ensure test granules are ingested and QAUU is configured with a test site$/) do
      get_test_granules_ingested_notdeleted(@so_test_data_items)
      @test_granules1.should_not be_nil, "granules not ingested for collection1"
      @test_granules1.count.should_not be_zero, "granules not ingested for collection1"

       @test_granule = @test_granules1[0]
       @test_granule.should_not be_nil

       #Ensure QAUU is configured with a test site
       @qauu = Sdps::QAUU.new(@mode, @db_conn)
```

```
       sitename = "EDF"
       email = "labuser@f5eil01v.edn.ecs.nasa.gov"
       status = @qauu.configure_site(@collection1.shortname, sitename, email)
       status.should be_truthy, "configuring QAUU test site failed"
end


def get_all_undeleted_granules_for_collection()

end
```

Current Test Script: bmgt_common.rb

```
Given(/^bmgt subsystem is checked out in the mode$/) do
  #Need to add mode scheck out script
  @mode = ENV['MODE']
  @mode.should_not be_nil
  @mode_checker = Sdps::Mode.new(@mode)
  #host should be checked to be f5oml01v
  @mode_checker.mode_checked_out?().should be_truthy

  @db_subsystem = Sdps::Bmgt::DB_SUBSYSTEM
  @db_conn = Sdps::Db.connect(@db_subsystem, @mode)
  @db_conn.should_not be_nil
  @mode_checker.bmgt_subsystem_checked_out?().should be_truthy, "bmgt system check failed"
  @proxy_log_location = @mode_checker.proxy_log_location
  @proxy_log_location.should_not be_nil

end
```

Current Test Script: BE_82_01_40_S2l.rb

```
When(/^Unlink a granule from a browse granule$/) do
```

```
  #remove the link in the ambrowsegranulexref table
  @test_granule.should_not be_nil, "test granule not available"
  granuleid = @test_granule.granuleid
  granuleid.should_not be_nil, "granuleid cannot be nil"
  @poll_wait_time = 300
  @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
  @export_queue = 'EVENT'
  status = @aimdb.unlink_browse(granuleid)
  status.should be_truthy, "unlink browse failed"
end
Current Test Script: ManualExport_Granule_Insert.rb

Given(/^criteria_180 mode is checked out$/) do
  @mode = ENV['MODE']
  @mode.should_not be_nil
  #host should be checked to be f5oml01v
  @mode_checker = Sdps::Mode.new(@mode)
  @mode_checker.mode_checked_out?().should be_truthy
  @db_subsystem = Sdps::Bmgt::DB_SUBSYSTEM
  @db_conn = Sdps::Db.connect(@db_subsystem, @mode)
  @db_conn.should_not be_nil

  @xmlutil = Sdps::Bmgt::Xml.new()
  @bmgtdb = Sdps::Bmgt::BmgtDb.new(@db_conn)

  #Initialize critinfo and SoTestData
  @critinfo = "BE_82_01:180"
  @sotestdata = Sdps::SoTestData.new(@db_conn)

  @proxy_log_location = "/tools/common/test/BE_82_01/proxy/#{@mode}/logs/tcp.log"
  @manual_log = "/usr/ecs/#{@mode}/CUSTOM/logs/EcBmBMGTManualDriver.log"
end

Given(/^criteria_180 bmgt subsystem is checked out$/) do
  @mode_checker.bmgt_subsystem_checked_out?().should be_truthy, "bmgt system check failed"
end
```

```
Given(/^criteria_180 testdata is checked out$/) do
  criteria_180_testdata_checked_out?().should be_truthy
end

Given(/^criteria_180 collections are installed$/) do
  @colls.each do |coll |
    @aimdb.is_collection_installed?(coll.shortname, coll.versionid).should be_truthy,
"#{coll.shortname}.#{coll.versionid} for Criteria 200 not installed."
  end
end

Given(/^criteria_180  collections are enabled for collection and granule export$/) do
  @colls.each do |coll |
    @bmgtdb.enable_collection_granule_export(coll.collectionid).should be_truthy
  end
end

Given(/^criteria_180 Identify a granule from collection C1 that can be exported$/) do
  granules = @aimdb.find_notdeleted_granuleids_by_collectionid(@colls[0].collectionid)
  rv = true
  if(granules.length < 1)
    rv = false
  else
    @g1 = granules[0]
  end
  rv.should be_truthy
end

When(/^criteria_180 Request the manual export of metadata for a single granule, specifying the granule ID$/) do
  @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
  rv = Sdps::Bmgt::ExportRequest.new(@mode).manual_export_for_granules?(@g1, nil)
  rv.should be_truthy
end

Then(/^criteria_180 Verify the granule g1 is exported$/) do
```

```
  is_gran_request_successful?(@g1).should be_truthy
end

Then(/^criteria_180 Verify that the TCP proxy shows an HTTP PUT request granule g1$/) do
  @xmlutil.http_put_for_granule?(@proxy_log_location, @request_time, @g1).should be_truthy
end

Then(/^criteria_180 Verify that the TCP proxy shows full granule metadata for granule g1$/) do
  gran_xml = @xmlutil.get_granule_xml(@proxy_log_location, @request_time, @g1)
    @xmlutil.granule_xml_valid?(gran_xml).should be_truthy
end

Then(/^criteria_180 Identify 2 Logically delete granules\(g2, g3\) or logically delete 2 granules from
collection C2$/) do
  @delgrans = @aimdb.find_deleted_granuleids_by_collectionid(@colls[1].collectionid)
  rv = true
  if(@delgrans.length < 2)
    rv = false
  else
    @g2 = @delgrans[0]
    @g3 = @delgrans[1]
  end
  rv.should be_truthy

end

Then(/^criteria_180 Manually Export granules in Collection C2$/) do
  @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
  rv = Sdps::Bmgt::ExportRequest.new(@mode).manual_granule_export_for_collection?(@colls[1].collectionid, nil)
  rv.should be_truthy
end

Then(/^criteria_180 Verify that the TCP proxy log shows an HTTP PUT request for each granule in Collection C2
except for logically deleted granules$/) do

  @granules = @aimdb.find_notdeleted_granuleids_by_collectionid(@colls[1].collectionid)
```

```
  @granules.each do |gran|
    is_gran_request_successful?(gran).should be_truthy
    @xmlutil.http_put_for_granule?(@proxy_log_location, @request_time, gran).should be_truthy
  end
end

Then(/^criteria_180 Verify that the TCP proxy shows that the request contains the full granule metadata for each
granule in Collection C2 except for logically deleted granules$/) do
  @granules.each do |gran|
    gran_xml = @xmlutil.get_granule_xml(@proxy_log_location, @request_time, gran)
    @xmlutil.granule_xml_valid?(gran_xml).should be_truthy
  end
end

Then(/^criteria_180 Verify that the TCP proxy log shows an HTTP DELETE is exported for each of granules g2 and
g3$/) do
  is_gran_request_successful?(@g2).should be_truthy
  @xmlutil.http_delete_for_granule?(@proxy_log_location, @request_time, @g2).should be_truthy
  is_gran_request_successful?(@g3).should be_truthy
  @xmlutil.http_delete_for_granule?(@proxy_log_location, @request_time, @g3).should be_truthy
end

Then(/^criteria_180 Verify that the TCP proxy log shows an HTTP DELETE is also exported for any other logically
deleted granules in the collection$/) do
  @delgrans = @aimdb.find_deleted_granuleids_by_collectionid(@colls[1].collectionid)
  @delgrans.each do |gran|
    is_gran_request_successful?(gran).should be_truthy
    @xmlutil.http_delete_for_granule?(@proxy_log_location, @request_time, gran).should be_truthy
  end
end

Then(/^criteria_180 Identify granule g4 in Collection C3 as a normal science granule that can be exported\.$/)
do
  granules = @aimdb.find_notdeleted_granuleids_by_collectionid(@colls[2].collectionid)
  rv = true
  if(granules.length < 1)
```

```
    rv = false
  else
    @g4 = granules[0]
  end
  rv.should be_truthy
end

Then(/^criteria_180 Identify granule g5 that is logically deleted in Collection C3$/) do
  @delgrans = @aimdb.find_deleted_granuleids_by_collectionid(@colls[2].collectionid)
  rv = true
  if(@delgrans.length < 1)
    rv = false
  else
    @g5 = @delgrans[0]
  end
  rv.should be_truthy
end

Then(/^criteria_180 Identify granule g6 which is physically deleted in Collection C3 or make up a nonexistent
granuleid in Collection C3$/) do
  @g6id = '12345'
  @g6 = "SC:#{@colls[2].shortname}.#{@colls[2].versionid}:#{@g6id}"
end

Then(/^criteria_180 Request, in a single manual operation, the export of metadata for g4, g5, g6$/) do
  @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
  @granules = "'#{@g4}, #{@g5}, #{@g6}'"
  rv = Sdps::Bmgt::ExportRequest.new(@mode).manual_export_for_granules?(@granules, nil)
  rv.should be_truthy
end

Then(/^criteria_180 Verify that the TCP proxy log shows an HTTP PUT request  for granule g4$/) do
  is_gran_request_successful?(@g4).should be_truthy
  @xmlutil.http_put_for_granule?(@proxy_log_location, @request_time, @g4).should be_truthy
end
```

```
Then(/^criteria_180 Verify that the TCP proxylog shows that the request contains the full granule metadata for
granule g4$/) do
  gran_xml = @xmlutil.get_granule_xml(@proxy_log_location, @request_time, @g4)
  @xmlutil.granule_xml_valid?(gran_xml).should be_truthy
end

Then(/^criteria_180 Verify that the TCP proxy log shows an HTTP DELETE is exported for granule g5$/) do
  is_gran_request_successful?(@g5).should be_truthy
  @xmlutil.http_delete_for_granule?(@proxy_log_location, @request_time, @g5).should be_truthy
end

Then(/^criteria_180 Verify that the TCP proxy log shows an HTTP DELETE is exported for granule g6$/) do
  is_gran_request_successful?(@g6id).should be_truthy
  @xmlutil.http_delete_for_granule?(@proxy_log_location, @request_time, @g6id).should be_truthy
end

Then(/^criteria_180 Request, in a single manual operation, specifying the export of inserts\-only for
g4,g5,g6$/) do
  @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
  options = Array.new
  options.push('-ins')
  rv = Sdps::Bmgt::ExportRequest.new(@mode).manual_export_for_granules?(@granules, options)
  rv.should be_truthy
end

Then(/^criteria_180 Insert Verify that the TCP proxy log shows an HTTP PUT request  for granule g4$/) do
  is_gran_request_successful?(@g4).should be_truthy
  @xmlutil.http_put_for_granule?(@proxy_log_location, @request_time, @g4).should be_truthy
end

Then(/^criteria_180 Insert Verify that the TCP proxylog shows that the request contains the full granule
metadata for granule g4$/) do
  gran_xml = @xmlutil.get_granule_xml(@proxy_log_location, @request_time, @g4)
  @xmlutil.granule_xml_valid?(gran_xml).should be_truthy
end
```

```
Then(/^criteria_180 Insert Verify that the TCP proxy log shows No Export for granule g5$/) do
  @xmlutil.http_put_for_granule?(@proxy_log_location, @request_time, @g5).should be_falsey
  @xmlutil.http_delete_for_granule?(@proxy_log_location, @request_time, @g5).should be_falsey
end

Then(/^criteria_180 Insert Verify that the TCP proxy log shows No Export for granule g6$/) do
  @xmlutil.http_put_for_granule?(@proxy_log_location, @request_time, @g6id).should be_falsey
  @xmlutil.http_delete_for_granule?(@proxy_log_location, @request_time, @g6id).should be_falsey
end

Then(/^criteria_180 Request, in a single manual operation, specifying the export of delete\-only for g4,g5,g6$/)
do
  @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
  options = Array.new
  options.push('-del')
  rv = Sdps::Bmgt::ExportRequest.new(@mode).manual_export_for_granules?(@granules, options)
  rv.should be_truthy
end

Then(/^criteria_180 Delete Verify that the TCP proxy log shows No Export for granule g4$/) do
  @xmlutil.http_put_for_granule?(@proxy_log_location, @request_time, @g4).should be_falsey
  @xmlutil.http_delete_for_granule?(@proxy_log_location, @request_time, @g4).should be_falsey
end

Then(/^criteria_180 Delete Verify that the TCP proxy log shows an HTTP DELETE is exported for granule g5$/) do
  is_gran_request_successful?(@g5).should be_truthy
  @xmlutil.http_delete_for_granule?(@proxy_log_location, @request_time, @g5).should be_truthy
end

Then(/^criteria_180 Delete Verify that the TCP proxy log shows an HTTP DELETE is exported for granule g6$/) do
  is_gran_request_successful?(@g6id).should be_truthy
  @xmlutil.http_delete_for_granule?(@proxy_log_location, @request_time, @g6id).should be_truthy
end

def criteria_180_testdata_checked_out?
```

```
  @colls = Array.new
  @data = @sotestdata[@critinfo]
  @data.should_not
  cids = Array.new
  i = 0
  rv = true
  if @data.length != 3
    puts "There should be 3 collections returned for #{@critinfo}, check AutoTestSoTestDataInfo table"
    rv = false
  else
    @data.each do |esdt|
      shortname = @data[i].shortname
      versionid = @data[i].versionid
      i += 1

      @aimdb = Sdps::AimDb.new(@db_conn)
      coll = @aimdb.find_collection_by_shortname_versionid(shortname, versionid)

      if (coll == nil || !@aimdb.is_collection_installed?(shortname, versionid))
        puts "#{shortname}, #{versionid} is not installed"
        rv = false
      else
        #collection should have datamodeltype = 'ECS'
        if coll.datamodeltype != 'ECS'
          puts "collection's datamodeltype is not ECS"
          rv = false
        end
      end
      rv.should be_truthy
      @colls.push(coll)
    end
   end
  rv.should be_truthy
end

def is_gran_request_successful?(granid)
```

```
  puts "Request successful #{granid}"
  begin
    rv = true;
    status = Sdps::Util.poll(180, 15){
      result = @bmgtdb.get_gran_export_request_status(@request_time,granid, 'MAN')
      if (result == 'PENDING' || result == 'STARTED')
        nil
      else
        result
      end
    }
    rv = false if (status != 'SUCCESS')

    return rv;
  end
end
Current Test Script: Ecs_Collection_Additional_Metadata_S7.rb
require 'sdps/sotestdata'
require 'sdps/aimdb'
require 'sdps/mode'
require 'sdps/db'
require 'sdps/util'
require 'sdps/bmgt'

Given(/^Criteria_20_7 Initial Setup$/) do
  #Check mode.
  @mode = ENV['MODE']
  @mode.should_not be_nil
  @mode_checker = Sdps::Mode.new(@mode)
  @mode_checker.mode_checked_out?().should be_truthy
  @proxy_log_location = @mode_checker.get_proxy_log_location()
  @xmlUtil = Sdps::Bmgt::Xml.new()

  #set up db connection
  @db_subsystem = Sdps::Bmgt::DB_SUBSYSTEM
  @conn = Sdps::Db.connect(@db_subsystem, @mode)
```

```
  @conn.should_not be_nil

  #Make sure BMGT is up and running, proxy needs to be up too
  @mode_checker.bmgt_subsystem_checked_out?().should be_truthy, "bmgt system check failed"

  #Initialize critinfo and SoTestData
  @critinfo = "BE_82_01:20:7"
  @sotestdata = Sdps::SoTestData.new(@conn)
end

Given(/^Criteria_20_7 All test collections are installed in the mode$/) do

  aimdb = Sdps::AimDb.new(@conn)
  @collections = Array.new

  data = @sotestdata[@critinfo]
  data.should_not be_nil

  rv = true
  if data.length != 3
    puts "There should be three collection returned for #{@critinfo}, check AutoTestSoTestDataInfo table"
    rv = false
  else
    #loop through @data
    data.each do |dataitem|
      shortname = dataitem.shortname
      versionid = dataitem.versionid
      collection = aimdb.find_collection_by_shortname_versionid(shortname,versionid)

      if collection == nil
        puts "test data #{shortname}, #{versionid} not installed"
        rv = false
      else
        @collections.push(collection)
        #collection should have datamodeltype = 'ECS'
        if collection.datamodeltype != 'ECS'
```

```
          puts "collection's datamodeltype is not ECS"
          rv = false
        end
      end
    end
  end
  rv.should be_truthy
end

Given(/^Criteria_20_7 All test collections are configured for collection and granule metadata export$/) do
  bmgtdb = Sdps::Bmgt::BmgtDb.new(@conn)
  @collections.each do |collection|
    collectionid = collection.collectionid
    if bmgtdb.collection_export_enabled?(collectionid) == false
      bmgtdb.enable_collection_granule_export(collectionid).should be_truthy
    end
  end
end

Given(/^Criteria_20_7 All test collections have valid CoordinateSystem and GranuleSpatialRepresentation$/) do
  rv = true
  @collections.each do |collection|
    #set CoordinateSystem to CARTESIAN and GranuleSpatialRepresentation to GEODETIC
    collectionid = collection.collectionid
    begin
        rs = @conn.exec ("update bg_collection_configuration set GranuleSpatialRepresentation='GEODETIC',
CollectionCoordinateSystem='CARTESIAN' where collectionid = '#{collectionid}' returning 1")
        if rs == nil or rs.count != 1
          rv = false
        end
    rescue => e
        puts e.backtrace
        rv = false
    end
  end
  rv.should be_truthy
```

```
end

Given(/^Criteria_20_7 A local copy of the collection schema is available to validate against$/) do
  #verify Sdps::Bmgt::ECHO10_COLLECTION_SCHEMA exists
  schemaFile = Sdps::Bmgt::ECHO10_COLLECTION_SCHEMA
  File.exists?(schemaFile).should be_truthy
end

Given(/^Criteria_20_7 A TCP proxy is capturing BMGT traffic$/) do
 #This is checked at the initial setup
 puts "If it gets here, the TCP proxy is up and running"
end

When(/^Criteria_20_7 S-7 For each of the valid TwoDCoordinateSystem types, configure a collection for that
type$/) do
  #There are three valid types and three collections
  @TwoDTypes = Array.new
  @TwoDTypes.push('MISR')
  @TwoDTypes.push('MODIS Tile SIN')
  @TwoDTypes.push('MODIS Tile EASE')

  rv = true
  count = 0
  @collections.each do |collection|
    TwoDType = @TwoDTypes[count]
    puts "TwoDType is #{TwoDType}, shortname is: #{collection.shortname}"
    #set CoordinateSystem and GranuleSpatialRepresentation to null
    collectionid = collection.collectionid
    begin
      rs = @conn.exec ("update bg_collection_configuration set TwoDCoordinateSystem = '#{TwoDType}' where
collectionid = '#{collectionid}' returning 1")
      if rs == nil or rs.count != 1
        rv = false
      end
    rescue => e
      puts e.backtrace
```

```
      rv = false
    end
    count = count + 1
  end
  rv.should be_truthy
end

When(/^Criteria_20_7 S-7 Request the export of collection metadata for a collection which is configured for that
type$/) do
  #record the request time
  @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")

  @collections.each do |collection|
    collectionid = collection.collectionid
    Sdps::Bmgt::ExportRequest.new(@mode).manual_export_for_collection?(collectionid, nil).should be_truthy,
"Falied running Manual Export for collection"
    ##make sure the request is queued
    Sdps::Bmgt::BmgtDb.new(@conn).get_coll_export_request_status(@request_time, collectionid, "MAN").should_not
be_nil
  end
end

Then(/^Criteria_20_7 Manual Export succeeds$/) do
  @collections.each do |collection|
    collectionid = collection.collectionid
    #poll and wait
    export_req = Sdps::Bmgt::BmgtDb.new(@conn)
    status = Sdps::Util.poll(180, 15){
      result = export_req.get_coll_export_request_status(@request_time, collectionid, "MAN")
      if (result == 'PENDING' || result == 'STARTED')
        nil
      else
        result
      end
    }
    status.should eq 'SUCCESS'
```

```
  end
end

Then(/^Criteria_20_7 V-12 Verify that all Collection metadata in the previous steps validates against
https:\/\/api\.echo\.nasa\.gov\/ingest\/schema\/Collection\.xsd$/) do
  #Verify the S-12 collection's exported XML validates against the schema:
  #The following method makes use of the local copy of the collection.xsd file:
Sdps::Bmgt::ECHO10_COLLECTION_SCHEMA
  count = 0
  @collections.each do |collection|
    collection_xml = @xmlUtil.get_collection_xml(@proxy_log_location, @request_time, collection)
    #puts "xml is #{collection_xml}"
    @xmlUtil.collection_xml_valid?(collection_xml).should be_truthy
    puts "Verification okay"

    #Verify the collection xml file contains the TwoD element, easier to combine the next step here
    TwoDType = @TwoDTypes[count]
    @xmlUtil.collection_xml_contains?(collection_xml,
"//Collection/TwoDCoordinateSystems/TwoDCoordinateSystem/TwoDCoordinateSystemName",
"<TwoDCoordinateSystemName>#{TwoDType}</TwoDCoordinateSystemName>").should be_truthy
    puts "Contains TwoD element okay"

    #Verify the collection xml file contains configured CoordinateSystem and GranuleSpatialRepresentation
values, again, easier to combine it here
    @xmlUtil.collection_xml_contains?(collection_xml,
"//Collection/Spatial/HorizontalSpatialDomain/Geometry/CoordinateSystem",
"<CoordinateSystem>CARTESIAN</CoordinateSystem>").should be_truthy
    puts "Contains CoordinateSystem okay"

    @xmlUtil.collection_xml_contains?(collection_xml, "//Collection/Spatial/GranuleSpatialRepresentation",
"<GranuleSpatialRepresentation>GEODETIC</GranuleSpatialRepresentation>").should be_truthy
    puts "contains GranuleSpatialRepresentation okay"
    count = count + 1
  end
end
```

```
Then(/^Criteria_20_7 V-7 Verify that the collection metadata generated in S-7 contains a
TwoDCoordinateSystems\/TwoDCoordinateSystem element which is populated according to the rules for the
TwoDCoordinateSystem type associated with the collection$/) do
  puts "If it gets here, the step is verified"

end

Then(/^Criteria_20_7 V-15 For the exports in S-5 to S-9, verify that the configured CoordinateSystem and
GranuleSpatialRepresentation values are included in the metadata$/) do
  puts "if it gets here, the step is verified"
end

Current Test Script: Ecs_Collection_Additional_Metadata_S3.rb
require 'sdps/sotestdata'
require 'sdps/aimdb'
require 'sdps/mode'
require 'sdps/db'
require 'sdps/util'
require 'sdps/bmgt'

Given(/^Criteria_20_3 Initial Setup$/) do
  #Check mode.
  @mode = ENV['MODE']
  @mode.should_not be_nil
  @mode_checker = Sdps::Mode.new(@mode)
  @mode_checker.mode_checked_out?().should be_truthy
  @proxy_log_location = @mode_checker.get_proxy_log_location()
  @generator_log_location = "/usr/ecs/#{@mode}/CUSTOM/logs/EcBmBMGTGenerator.log"
  @xmlUtil = Sdps::Bmgt::Xml.new()

  #set up db connection
  @db_subsystem = Sdps::Bmgt::DB_SUBSYSTEM
  @conn = Sdps::Db.connect(@db_subsystem, @mode)
  @conn.should_not be_nil

  #Make sure BMGT is up and running, proxy needs to be up too
```

```
  @mode_checker.bmgt_subsystem_checked_out?().should be_truthy, "bmgt system check failed"

  #Initialize critinfo and SoTestData
  @critinfo = "BE_82_01:20:3"
  @sotestdata = Sdps::SoTestData.new(@conn)
end

Given(/^Criteria_20_3 All test collections are installed in the mode$/) do
  @data = @sotestdata[@critinfo]
  @data.should_not be_nil

  rv = true
  if @data.length != 1
    puts "There should be one collection returned for #{@critinfo}, check AutoTestSoTestDataInfo table"
    rv = false
  else
    @shortname = @data[0].shortname
    @versionid = @data[0].versionid

    @aimdb = Sdps::AimDb.new(@conn)
    @collection =@aimdb.find_collection_by_shortname_versionid(@shortname,@versionid)

    if @collection == nil
      puts "test data #{@shortname}, #{@versionid} not installed"
      rv = false
    else
      #collection should have datamodeltype = 'ISO-SMAP'
      if @collection.datamodeltype != 'ISO-SMAP'
        puts "collection's datamodeltype is not ISO-SMAP"
        rv = false
      end
    end
  end
  rv.should be_truthy
end
```

```
Given(/^Criteria_20_3 All test collections are configured for collection and granule metadata export$/) do
  @collectionid = @collection.collectionid
  @bmgtdb = Sdps::Bmgt::BmgtDb.new(@conn)

  if @bmgtdb.collection_export_enabled?(@collectionid) == false
    @bmgtdb.enable_collection_granule_export().should be_truthy
  end
end

Given(/^Criteria_20_3 A local copy of the collection schema is available to validate against$/) do
 puts "In the case of Criteria_20_3, no xml should be exported, therefore no need for the schema file"
end

Given(/^Criteria_20_3 A TCP proxy is capturing BMGT traffic$/) do
 #This is checked at the initial setup
 puts "If it gets here, the TCP proxy is up and running"

end

When(/^Criteria_20_3 Set the test collection's CoordinateSystem to any string$/) do
  #There's only one collection in this test

  begin
    rs = @conn.exec ("update bg_collection_configuration set CollectionCoordinateSystem='GEODETIC' where
ShortName='#{@shortname}' and versionid = '#{@versionid}' returning 1")

    if rs != nil and rs.count == 1
      rv = true
    else
      rv = false
    end
  rescue => e
    puts e.backtrace
    rv = false
  end
```

```
  rv.should be_truthy
end

Then(/^Criteria_20_3 Verify that it succeeds$/) do

  puts "If it gets here, the update successful is verified"

end

When(/^Criteria_20_3 Request a Manual Export of the S-3 collection$/) do
  #record the request time
  @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
  Sdps::Bmgt::ExportRequest.new(@mode).manual_export_for_collection?(@collection.collectionid, nil).should_not
be_falsey, "Falied running Manual Export for collection"

  ##make sure the request is queued
  Sdps::Bmgt::BmgtDb.new(@conn).get_coll_export_request_status(@request_time, @collection.collectionid,
"MAN").should_not be_nil
end

Then(/^Criteria_20_3 Verify the manual export fails$/) do
  #check the EcBmBMGTManualDriver.log? key word?
  #actually the manual run log just queues the request, the actual run error appears in the generator log
  #but first have to wait till the request is processed
  #poll and wait
  export_req = Sdps::Bmgt::BmgtDb.new(@conn)
  status = Sdps::Util.poll(180, 15){
    result = export_req.get_coll_export_request_status(@request_time, @collection.collectionid, "MAN")
    if (result == 'PENDING' || result == 'STARTED')
      nil
    else
      result
    end
  }
  status.should eq 'SKIPPED'
```

```
end

Then(/^Criteria_20_3 Verify the log file records the export failure$/) do
   #Check the generator log, after @request_time
  founderror = false
  open (@generator_log_location) do |log|
    # move the file pointer to the line after the timestamp
    log.find{|line| line =~ /^\d{4}-\d\d-\d\d \d\d:\d\d/ and line > @request_time}
    #find all the lines with ERROR for the collection esdt
    len =  log.grep(/ERROR.*#{@collection.esdt}.*ISO-SMAP metadata should not have a CollectionCoordinateSystem
configured/).map{|line| line.chomp}.length
    if len > 0
      founderror = true
    end
  end
  founderror.should be_truthy
end

Then(/^Criteria_20_3 Verify the log file indicates the invalid CoordinateSystem value caused the export
failure\.$/) do
  puts "If it gets here, the failure msg is verified"
end

Then(/^Criteria_20_3 Verify that all Collection metadata in the previous steps validates against
https:\/\/api\.echo\.nasa\.gov\/ingest\/schema\/Collection\.xsd$/) do
  puts "In the case of Criteria_20_3, there is no xml file exported, therefore nothing to validate against the
schema file"
end

Then(/^Criteria_20_3 Verify that no collection metadata should have been exported$/) do
  #there is no Put request in the proxy log?
  dataset =  @collection.datasetid()
  @xmlUtil.http_put_for_collection?(@proxy_log_location, @request_time, dataset).should be_falsey
end
```

Current Test Script: Ecs_Collection_Additional_Metadata_S8.rb

```
require 'sdps/sotestdata'
require 'sdps/aimdb'
require 'sdps/mode'
require 'sdps/db'
require 'sdps/util'
require 'sdps/bmgt'

Given(/^Criteria_20_8 Initial Setup$/) do
  #Check mode.
  @mode = ENV['MODE']
  @mode.should_not be_nil
  @mode_checker = Sdps::Mode.new(@mode)
  @mode_checker.mode_checked_out?().should be_truthy
  @proxy_log_location = @mode_checker.get_proxy_log_location()
  @xmlUtil = Sdps::Bmgt::Xml.new()

  #set up db connection
  @db_subsystem = Sdps::Bmgt::DB_SUBSYSTEM
  @conn = Sdps::Db.connect(@db_subsystem, @mode)
  @conn.should_not be_nil

  #Make sure BMGT is up and running, proxy needs to be up too
  @mode_checker.bmgt_subsystem_checked_out?().should be_truthy, "bmgt system check failed"

  #Initialize critinfo and SoTestData
  @critinfo = "BE_82_01:20:8"
  @sotestdata = Sdps::SoTestData.new(@conn)
end

Given(/^Criteria_20_8 All test collections are installed in the mode$/) do

  aimdb = Sdps::AimDb.new(@conn)
  @collections = Array.new

  data = @sotestdata[@critinfo]
  data.should_not be_nil
```

```
  rv = true
  if data.length != 1
    puts "There should be one collection returned for #{@critinfo}, check AutoTestSoTestDataInfo table"
    rv = false
  else
    #loop through @data
    data.each do |dataitem|
      shortname = dataitem.shortname
      versionid = dataitem.versionid
      collection = aimdb.find_collection_by_shortname_versionid(shortname,versionid)

      if collection == nil
        puts "test data #{shortname}, #{versionid} not installed"
        rv = false
      else
        @collections.push(collection)
        #collection should have datamodeltype = 'ECS'
        if collection.datamodeltype != 'ECS'
          puts "collection's datamodeltype is not ECS"
          rv = false
        end
      end
    end
  end
  rv.should be_truthy
end

Given(/^Criteria_20_8 All test collections are configured for collection and granule metadata export$/) do
  bmgtdb = Sdps::Bmgt::BmgtDb.new(@conn)
  @collections.each do |collection|
    collectionid = collection.collectionid
    if bmgtdb.collection_export_enabled?(collectionid) == false
      bmgtdb.enable_collection_granule_export(collectionid).should be_truthy
    end
  end
```

```
end

Given(/^Criteria_20_8 All test collections have valid CoordinateSystem and GranuleSpatialRepresentation$/) do
  rv = true
  @collections.each do |collection|
    #set CoordinateSystem to CARTESIAN and GranuleSpatialRepresentation to GEODETIC
    collectionid = collection.collectionid
    begin
        rs = @conn.exec ("update bg_collection_configuration set GranuleSpatialRepresentation='GEODETIC',
CollectionCoordinateSystem='CARTESIAN' where collectionid = '#{collectionid}' returning 1")
        if rs == nil or rs.count != 1
          rv = false
        end
    rescue => e
        puts e.backtrace
        rv = false
    end
  end
  rv.should be_truthy
end

Given(/^Criteria_20_8 A local copy of the collection schema is available to validate against$/) do
  #verify Sdps::Bmgt::ECHO10_COLLECTION_SCHEMA exists
  schemaFile = Sdps::Bmgt::ECHO10_COLLECTION_SCHEMA
  File.exists?(schemaFile).should be_truthy
end

Given(/^Criteria_20_8 A TCP proxy is capturing BMGT traffic$/) do
 #This is checked at the initial setup
 puts "If it gets here, the TCP proxy is up and running"
end

When(/^Criteria_20_8 S-8 Request the export of metadata for a collection which is not configured for a
TwoDCoordinateSystem$/) do
  #record the request time
  @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
```

```
  @collections.each do |collection|
    collectionid = collection.collectionid
    rv = true
    begin
      rs = @conn.exec ("update bg_collection_configuration set TwoDCoordinateSystem = null where collectionid =
'#{collectionid}' returning 1")
      if rs == nil or rs.count != 1
        rv = false
      end
    rescue => e
      puts e.backtrace
      rv = false
    end
    rv.should be_truthy

    Sdps::Bmgt::ExportRequest.new(@mode).manual_export_for_collection?(collectionid, nil).should be_truthy,
"Failed running Manual Export for collection"
    ##make sure the request is queued
    Sdps::Bmgt::BmgtDb.new(@conn).get_coll_export_request_status(@request_time, collectionid, "MAN").should_not
be_nil
  end
end

Then(/^Criteria_20_8 Manual Export succeeds$/) do
  @collections.each do |collection|
    collectionid = collection.collectionid
    #poll and wait
    export_req = Sdps::Bmgt::BmgtDb.new(@conn)
    status = Sdps::Util.poll(180, 15){
      result = export_req.get_coll_export_request_status(@request_time, collectionid, "MAN")
      if (result == 'PENDING' || result == 'STARTED')
        nil
      else
        result
      end
```

```
    }
    status.should eq 'SUCCESS'
  end
end

Then(/^Criteria_20_8 V-12 Verify that all Collection metadata in the previous steps validates against
https:\/\/api\.echo\.nasa\.gov\/ingest\/schema\/Collection\.xsd$/) do
  #Verify the S-12 collection's exported XML validates against the schema:
  #The following method makes use of the local copy of the collection.xsd file:
Sdps::Bmgt::ECHO10_COLLECTION_SCHEMA
  count = 0
  @collections.each do |collection|
    collection_xml = @xmlUtil.get_collection_xml(@proxy_log_location, @request_time, collection)
    #puts "xml is #{collection_xml}"
    @xmlUtil.collection_xml_valid?(collection_xml).should be_truthy
    puts "Verification okay"

    #Verify the collection xml file contains no TwoD element, easier to combine the next step here
    @xmlUtil.collection_xml_contains?(collection_xml, "//Collection/TwoDCoordinateSystems/TwoDCoordinateSystem",
"").should be_truthy
    puts "Not contains TwoD element okay"

    #Verify the collection xml file contains configured CoordinateSystem and GranuleSpatialRepresentation
values, again, easier to combine it here
    @xmlUtil.collection_xml_contains?(collection_xml,
"//Collection/Spatial/HorizontalSpatialDomain/Geometry/CoordinateSystem",
"<CoordinateSystem>CARTESIAN</CoordinateSystem>").should be_truthy
    puts "Contains CoordinateSystem okay"

    @xmlUtil.collection_xml_contains?(collection_xml, "//Collection/Spatial/GranuleSpatialRepresentation",
"<GranuleSpatialRepresentation>GEODETIC</GranuleSpatialRepresentation>").should be_truthy
    puts "contains GranuleSpatialRepresentation okay"
    count = count + 1
  end
end
```

```
Then(/^Criteria_20_8 V-8 Verify that the collection metadata generated in S-8 contains no
TwoDCoordinateSystems\/TwoDCoordinateSystem element$/) do
  puts "If it gets here, the step is verified"

end

Then(/^Criteria_20_8 V-15 For the exports in S-5 to S-9, verify that the configured CoordinateSystem and
GranuleSpatialRepresentation values are included in the metadata$/) do
  puts "if it gets here, the step is verified"
end
```

Current Test Script: BE_82_01_40_V2.rb

Current Test Script: BE_82_01_40_S2g.rb

```
When(/^Unrestrict a granule$/) do
   @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
   @export_queue = 'EVENT'
   @poll_wait_time = 300 #need to get this value from the configuration property table

   status = @aimdb.unrestrict_granule(@test_granule.granuleid);
   status.should be_truthy
end
```

Current Test Script: Manual_Collection_Deletes.rb

```
Given(/^criteria_160 mode is checked out$/) do
  @mode = ENV['MODE']
  @mode.should_not be_nil
  #host should be checked to be f5oml01v
  @mode_checker = Sdps::Mode.new(@mode)
  @mode_checker.mode_checked_out?().should be_truthy
  @db_subsystem = Sdps::Bmgt::DB_SUBSYSTEM
  @db_conn = Sdps::Db.connect(@db_subsystem, @mode)
  @db_conn.should_not be_nil
```

```
  @xmlUtil = Sdps::Bmgt::Xml.new()
  @bmgtdb = Sdps::Bmgt::BmgtDb.new(@db_conn)
  @aimdb = Sdps::AimDb.new(@db_conn)
  #Initialize critinfo and SoTestData
  @critinfo = "BE_82_01:160"
  @sotestdata = Sdps::SoTestData.new(@db_conn)

  @proxy_log_location = "/tools/common/test/BE_82_01/proxy/#{@mode}/logs/tcp.log"
  @manual_log = "/usr/ecs/#{@mode}/CUSTOM/logs/EcBmBMGTManualDriver.log"
end

Given(/^criteria_160 testdata is checked out$/) do
  criteria_160_testdata_checked_out?().should be_truthy
end

Given(/^criteria_160 bmgt subsystem is checked out$/) do
  @mode_checker.bmgt_subsystem_checked_out?().should be_truthy, "bmgt system check failed"
end

Given(/^criteria_160 collection which is enabled for collection metadata export which is present in AmCollection
but is not in the installed state in AIM DsGeESDTConfiguredType\.$/) do

  @aimdb.find_collection_by_shortname_versionid(@colls[0].shortname, @colls[0].versionid).should_not be_nil
  @db_conn.exec("update DsGeESDTConfiguredType set esdtstate = 'updating'
            where configuredname = '#{@colls[0].shortname}' and versionid = #{@colls[0].versionid}")

  @aimdb.is_collection_installed?(@colls[0].shortname, @colls[0].versionid).should be_falsey, "Collection1 for
160 is installed."

  id = @aimdb.find_collection_by_shortname_versionid(@colls[0].shortname, @colls[0].versionid).collectionid

  if (!@bmgtdb.collection_export_enabled?(id))
    @bmgtdb.enable_collection_export(id).should be_truthy
  end
end
```

```
Given(/^criteria_160  A collection which is enabled for collection metadata export which is not present in
AmCollection and is not in the installed state in AIM DsGeESDTConfiguredType\.$/) do

  insert_fake_esdt_in_collconfig?(@colls[1], 1, 'Y', 'Y').should be_truthy

  @aimdb.find_collection_by_shortname_versionid(@colls[1].shortname, @colls[1].versionid).should be_nil

  @bmgtdb.collection_export_enabled_by_shortname?(@colls[1].shortname, @colls[1].versionid).should be_truthy

end

Given(/^criteria_160 A collection which is present in AmCollection, and in the installed state in AIM
DsGeESDTConfiguredType, but which is not enabled for collection metadata export\.$/) do

  @aimdb.find_collection_by_shortname_versionid(@colls[2].shortname, @colls[2].versionid).should_not be_nil
  @aimdb.is_collection_installed?(@colls[2].shortname, @colls[2].versionid).should be_truthy, "Collection3 for
160 not installed."
  id = @aimdb.find_collection_by_shortname_versionid(@colls[2].shortname, @colls[2].versionid).collectionid

  if (@bmgtdb.collection_export_enabled?(id))
    @bmgtdb.disable_collection_granule_export(id).should be_truthy
  end
end

Given(/^criteria_160 A collection which does not exist in AIM AmCollection and DsGeESDTConfiguredType and is not
enabled for collection metadata export i\.e\. a completely made up short name and version ID\.$/) do

  insert_fake_esdt_in_collconfig?(@colls[3], 2, 'N', 'N').should be_truthy
  @aimdb.find_collection_by_shortname_versionid(@colls[3].shortname, @colls[3].versionid).should be_nil
  @bmgtdb.collection_export_enabled_by_shortname?(@colls[3].shortname, @colls[3].versionid).should be_falsey

end

Given(/^crieria_160  A collection which is present in AmCollection, in the installed state
DsGeESDTConfiguredType, and is enabled for collection metadata export\.$/) do
```

```
  @aimdb.find_collection_by_shortname_versionid(@colls[4].shortname, @colls[4].versionid).should_not be_nil
  @aimdb.is_collection_installed?(@colls[4].shortname, @colls[4].versionid).should be_truthy, "Collection5 for
160 not installed."
  id = @aimdb.find_collection_by_shortname_versionid(@colls[4].shortname, @colls[4].versionid).collectionid
  if (!@bmgtdb.collection_export_enabled?(id))
    @bmgtdb.enable_collection_export(id).should be_truthy
  end
end

Given(/^criteria_160 Ensure all 5 collections are listed in a text file e\.g\., collections\.txt$/) do

  @collfile = '/home/cmshared/collections.txt'
  File.open(@collfile, 'w') do |f1|
    @colls.each do |coll|
      f1.puts(coll.esdt())
    end
  end

end

When(/^criteria_160 Manually export all 5 collections$/) do

  manual_file_export_for_collection?(@collfile, nil).should be_truthy
  coll_deletes_processed_successfully?().should be_truthy

end

Then(/^criteria_160_nominal Verify the TCP proxy or mock ECHO log shows no export of collection C1$/) do
  dataset =  @colls[0].datasetid()
  @xmlUtil.http_put_for_collection?(@proxy_log_location, @request_time, dataset).should be_falsey
  @xmlUtil.http_delete_for_collection?(@proxy_log_location, @request_time, dataset).should be_falsey
end

Then(/^criteria_160_nominal Verify the TCP proxy or mock ECHO log shows an HTTP DELETE for collection C2$/) do
  dataset =  @colls[1].datasetid()
```

```
  @xmlUtil.http_delete_for_collection?(@proxy_log_location, @request_time, dataset).should be_truthy
end

Then(/^criteria_160_nominal Verify the TCP proxy or mock ECHO log shows no export of collection C3$/) do
  dataset =  @colls[2].datasetid()
  @xmlUtil.http_put_for_collection?(@proxy_log_location, @request_time, dataset).should be_falsey
  @xmlUtil.http_delete_for_collection?(@proxy_log_location, @request_time, dataset).should be_falsey
end

Then(/^criteria_160_nominal Verify the TCP proxy or mock ECHO log shows no export of collection C4$/) do
  dataset =  @colls[3].datasetid()
  @xmlUtil.http_put_for_collection?(@proxy_log_location, @request_time, dataset).should be_falsey
  @xmlUtil.http_delete_for_collection?(@proxy_log_location, @request_time, dataset).should be_falsey
end

Then(/^criteria_160_nominal Verify the TCP proxy or mock ECHO log shows an HTTP PUT for collection C5$/) do
  dataset =  @colls[4].datasetid()
  @xmlUtil.http_put_for_collection?(@proxy_log_location, @request_time, dataset).should be_truthy
end

Then(/^criteria_160_nominal Verify the TCP proxy or mock ECHO log shows the PUT for collection C5 includes full
collection metadata$/) do
  collection_xml = @xmlUtil.get_collection_xml(@proxy_log_location, @request_time, @colls[4])
  @xmlUtil.collection_xml_valid?(collection_xml).should be_truthy
end

Then(/^criteria_160_nominal Verify the BMGT Manual log includes a message indicating that collection C3 could
not be not exported because it was not enabled or not installed\.$/) do
  manuallog_no_coll_export?(@colls[2]).should be_truthy
end

Then(/^criteria_160_nominal Verify the BMGT Manual log includes a message indicating that collection C4 could
not be not exported because it was not enabled or not installed\.$/) do
  manuallog_no_coll_export?(@colls[3]).should be_truthy
end
```

```
Then(/^criteria_160 Manually export all 5 collections as insertonly$/) do
  manual_file_export_for_collection?(@collfile, '-ins').should be_truthy
  coll_deletes_processed_successfully?().should be_truthy
end

Then(/^criteria_160_insertonly Verify the TCP proxy or mock ECHO log shows no export of collection C1$/) do
  dataset =  @colls[0].datasetid()
  @xmlUtil.http_put_for_collection?(@proxy_log_location, @request_time, dataset).should be_falsey
  @xmlUtil.http_delete_for_collection?(@proxy_log_location, @request_time, dataset).should be_falsey
end

Then(/^criteria_160_insertonly Verify the TCP proxy or mock ECHO log shows no export of collection C2$/) do
  dataset =  @colls[1].datasetid()
  @xmlUtil.http_put_for_collection?(@proxy_log_location, @request_time, dataset).should be_falsey
  @xmlUtil.http_delete_for_collection?(@proxy_log_location, @request_time, dataset).should be_falsey
end

Then(/^criteria_160_insertonly Verify the TCP proxy or mock ECHO log shows no export of collection C3$/) do
  dataset =  @colls[2].datasetid()
  @xmlUtil.http_put_for_collection?(@proxy_log_location, @request_time, dataset).should be_falsey
  @xmlUtil.http_delete_for_collection?(@proxy_log_location, @request_time, dataset).should be_falsey
end

Then(/^criteria_160_insertonly Verify the TCP proxy or mock ECHO log shows no export of collection C4$/) do
  dataset =  @colls[3].datasetid()
  @xmlUtil.http_put_for_collection?(@proxy_log_location, @request_time, dataset).should be_falsey
  @xmlUtil.http_delete_for_collection?(@proxy_log_location, @request_time, dataset).should be_falsey
end

Then(/^criteria_160_insertonly Verify the TCP proxy or mock ECHO log shows an HTTP PUT for collection C5$/) do
  dataset =  @colls[4].datasetid()
  @xmlUtil.http_put_for_collection?(@proxy_log_location, @request_time, dataset).should be_truthy
end

Then(/^criteria_160_insertonly Verify the TCP proxy or mock ECHO log shows the PUT for collection C5 includes
```

```
full collection metadata$/) do
  collection_xml = @xmlUtil.get_collection_xml(@proxy_log_location, @request_time, @colls[4])
  @xmlUtil.collection_xml_valid?(collection_xml).should be_truthy
end

Then(/^criteria_160_insertonly Verify the BMGT Manual log includes a message indicating that collection C3 could
not be not exported because it was not enabled or not installed\.$/) do
  manuallog_no_coll_export?(@colls[2]).should be_truthy
end
Then(/^criteria_160_insertonly Verify the BMGT Manual log includes a message indicating that collection C4 could
not be not exported because it was not enabled or not installed\.$/) do
  manuallog_no_coll_export?(@colls[3]).should be_truthy
end

Then(/^criteria_160 Manually export all 5 collections as deleteonly$/) do
  if (!@bmgtdb.collection_export_enabled?(1))
          puts "UPDATING COLLECTIN CONFIG FOR ID 1"
      @bmgtdb.enable_collection_export(1).should be_truthy
  end
  manual_file_export_for_collection?(@collfile, '-deleteonly').should be_truthy
  coll_deletes_processed_successfully?().should be_truthy
end

Then(/^criteria_160_deleteonly Verify the TCP proxy or mock ECHO log shows no export of collection C1$/) do
  dataset =  @colls[0].datasetid()
  @xmlUtil.http_put_for_collection?(@proxy_log_location, @request_time, dataset).should be_falsey
  @xmlUtil.http_delete_for_collection?(@proxy_log_location, @request_time, dataset).should be_falsey
end

Then(/^criteria_160_deleteonly Verify the TCP proxy or mock ECHO log shows an HTTP DELETE for collection C2$/)
do
  dataset =  @colls[1].datasetid()
  @xmlUtil.http_delete_for_collection?(@proxy_log_location, @request_time, dataset).should be_truthy
end

Then(/^criteria_160_deleteonly Verify the TCP proxy or mock ECHO log shows no export of collection C3$/) do
```

```
  dataset =  @colls[2].datasetid()
  @xmlUtil.http_put_for_collection?(@proxy_log_location, @request_time, dataset).should be_falsey
  @xmlUtil.http_delete_for_collection?(@proxy_log_location, @request_time, dataset).should be_falsey
end

Then(/^criteria_160_deleteonly Verify the TCP proxy or mock ECHO log shows no export of collection C4$/) do
  dataset =  @colls[3].datasetid()
  @xmlUtil.http_put_for_collection?(@proxy_log_location, @request_time, dataset).should be_falsey
  @xmlUtil.http_delete_for_collection?(@proxy_log_location, @request_time, dataset).should be_falsey
end

Then(/^criteria_160_deleteonly Verify the TCP proxy or mock ECHO log shows no export of collection C5$/) do
  dataset =  @colls[4].datasetid()
  @xmlUtil.http_put_for_collection?(@proxy_log_location, @request_time, dataset).should be_falsey
  @xmlUtil.http_delete_for_collection?(@proxy_log_location, @request_time, dataset).should be_falsey
end

Then(/^criteria_160_deleteonly Verify the BMGT Manual log includes a message indicating that collection C1 could
not be not exported because it exists in AmCollection\.$/) do
  manuallog_no_delcoll_export?(@colls[0]).should be_truthy
end

Then(/^criteria_160_deleteonly Verify the BMGT Manual log includes a message indicating that collection C3 could
not be not exported because it was not enabled or not installed\.$/) do
  manuallog_no_coll_export?(@colls[2]).should be_truthy
end

Then(/^criteria_160_deleteonly Verify the BMGT Manual log includes a message indicating that collection C4 could
not be not exported because it was not enabled or not installed\.$/) do
  manuallog_no_coll_export?(@colls[3]).should be_truthy
  cleanup()
end

def criteria_160_testdata_checked_out?

  @colls = Array.new
```

```
  @data = @sotestdata[@critinfo]
  @data.should_not be_nil
  i = 0
  rv = true
  if @data.length != 5
    puts "There should be 5 collections returned for #{@critinfo}, check AutoTestSoTestDataInfo table"
    rv = false
  else
    @data.each do |esdt|
      shortname = @data[i].shortname
      versionid = @data[i].versionid
      longname = @data[i].longname
      pathname = @data[i].pathname

      dataitem = Sdps::SoTestData::SoTestDataItem.new(pathname, shortname,versionid,longname)
      @colls.push(dataitem)
      i = i + 1
    end
        puts @colls.inspect
  end
  rv.should be_truthy
end

def manual_file_export_for_collection?(file,  option)
  begin
    puts "Manual collection export from file"
    options = nil
    if option != nil
      options = Array.new
      options.push(option)
    end
    @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
    Sdps::Bmgt::ExportRequest.new(@mode).manual_file_export?(file, '-metc', options).should be_truthy, "Falied
running Manual Export for collection"

  rescue =>e
```

```
    puts e.backtrace
    false
  end
end

def manuallog_no_coll_export?(coll)
  foundmsg = false
  open (@manual_log) do |log|
    # move the file pointer to the line after the timestamp
    log.find{|line| line =~ /^\d{4}-\d\d-\d\d \d\d:\d\d/ and line > @request_time}
    #find all the lines with ERROR for the collection esdt
    msg =  log.grep(/Input collection #{coll.shortname}.#{coll.versionid} is either collection export disabled
or does not exist in AmCollection./).map{|line| line.chomp}
    len = msg.length
    if len > 0
      foundmsg = true
    end
  end
  return foundmsg

end
def manuallog_no_delcoll_export?(coll)
  foundmsg = false
  open (@manual_log) do |log|
    # move the file pointer to the line after the timestamp
    log.find{|line| line =~ /^\d{4}-\d\d-\d\d \d\d:\d\d/ and line > @request_time}

    puts "Coll esdt ** #{coll.esdt}"

    #find all the lines with ERROR for the collection esdt
    msg =  log.grep(/Collection #{coll.esdt} cannot be added as a Delete request since it exists in
AmCollection/).map{|line| line.chomp}
    len = msg.length
    if len > 0
      foundmsg = true
    end
```

```
  end
  return foundmsg

end

def insert_fake_esdt_in_collconfig?(coll, id, cflag, gflag)

  begin

    rv = true
    @db_conn.exec("delete from bg_collection_configuration where collectionid = #{id}")
    @db_conn.exec("insert into bg_collection_configuration(collectionid, shortname, versionid, longname,
              granuleexportflag, collectionexportflag, difid, maxgranulestoverify)
              select  #{id}, '#{coll.shortname}', #{coll.versionid}, '#{coll.longname}', '#{gflag}', '#{cflag}',
'#{coll.shortname}', 5000 ")

  rescue =>e
    puts e.backtrace
    rv = false
  end
  return rv
end

def coll_deletes_processed_successfully?()
  begin
    rv = true;
    @colls.each do|coll|

      id = @bmgtdb.get_collectionid_from_coll_config(coll.shortname, coll.versionid)

      if(id != nil)
        status = Sdps::Util.poll(180, 15){
        result = @bmgtdb.get_coll_export_request_status(@request_time,id, 'MAN')
        if(result == nil)
          result = 'NOREQ'
        end
```

```
        if (result == 'PENDING' || result == 'STARTED')
          nil
        else
          result
        end
      }

      end

      #rv = false if (status != 'SUCCESS')
    end
    return rv;
  end
end


def cleanup()
  begin
    rs = @db_conn.exec("delete from bg_collection_configuration where collectionid in (1, 2) ")
    if rs != nil and rs.count > 0
      rv = true
    end
    File.delete(@collfile)
    @db_conn.exec("update DsGeESDTConfiguredType set esdtstate = 'installed'
            where configuredname = '#{@colls[0].shortname}' and versionid = #{@colls[0].versionid}")
  rescue =>e
    puts e.backtrace
    rv = false
  end
end
Current Test Script: Longform_LastUpdate_Export.rb
Given(/^criteria_440 mode is checked out$/) do
  @mode = ENV['MODE']
  @mode.should_not be_nil
  #host should be checked to be f5oml01v
  @mode_checker = Sdps::Mode.new(@mode)
```

```
  @mode_checker.mode_checked_out?().should be_truthy
  @db_subsystem = Sdps::Bmgt::DB_SUBSYSTEM
  @db_conn = Sdps::Db.connect(@db_subsystem, @mode)
  @db_conn.should_not be_nil

  @xmlUtil = Sdps::Bmgt::Xml.new()
  @bmgtdb = Sdps::Bmgt::BmgtDb.new(@db_conn)

  #Initialize critinfo and SoTestData
  @critinfo = "BE_82_01:200"
  @sotestdata = Sdps::SoTestData.new(@db_conn)

  @proxy_log_location = "/tools/common/test/BE_82_01/proxy/#{@mode}/logs/tcp.log"

end

Given(/^criteria_440 bmgt subsystem is checked out$/) do
  @mode_checker.bmgt_subsystem_checked_out?().should be_truthy, "bmgt system check failed"
end

Given(/^criteria_440 testdata is checked out$/) do
  criteria_440_testdata_checked_out?().should be_truthy
end

Given(/^criteria_440 collections are installed$/) do
  @colls.each do |coll |
    @aimdb.is_collection_installed?(coll.shortname, coll.versionid).should be_truthy,
"#{coll.shortname}.#{coll.versionid} for Criteria 210 not installed."
  end
end

Given(/^criteria_440 collections are enabled for collection and granule export$/) do
  @colls.each do |coll |
    @bmgtdb.enable_collection_granule_export(coll.collectionid).should be_truthy
  end
end
```

```
Given(/^criteria_440 collections have few granules ingested$/) do
  check_for_granules?().should be_truthy
end

Given(/^criteria_440 Identify two granules within a collection and their lastupdate times$/) do
  identify_criteria440_granules()
end

When(/^criteria_440 Request longform verification export of granule metadata for collection based on lastupdate
times$/) do
  options  = Array.new
  options.push('-vl')
  options.push('-c')
  options.push(@colls[0].collectionid)
  options.push('-lastupdate')

  time_based_verification_export?('-metg', options).should be_truthy
  ver_gran_requests_processed_successfully?().should be_truthy
end

Then(/^criteria_440 Verify the granule at starttime is exported but not the one with the end time$/) do
  @bmgtdb.get_gran_export_request_status(@request_time, @granules[0], 'VER').should_not be_nil
  @bmgtdb.get_gran_export_request_status(@request_time, @granules[3], 'VER').should be_nil
end

Then(/^criteria_440 Verify that the TCP proxy shows an HTTP PUT request each for granules g1, g2, g3$/) do
  i = 0;
  until i > 2 do
    @xmlUtil.http_put_for_granule?(@proxy_log_location, @request_time, @granules[i]).should be_truthy
    i += 1
  end
end

Then(/^criteria_440 Verify that the TCP proxy does not show any request for granule g4$/) do
  @xmlUtil.http_put_for_granule?(@proxy_log_location, @request_time, @granules[3]).should be_falsey
```

```
  @xmlUtil.http_delete_for_granule?(@proxy_log_location, @request_time, @granules[3]).should be_falsey
end

Then(/^criteria_440 Verify that the TCP proxy shows full granule metadata for each of the granules g1, g2, g3$/)
do
  i = 0;
  until i > 2 do
    granule_xml = @xmlUtil.get_granule_xml(@proxy_log_location, @request_time, @granules[i])
    @xmlUtil.granule_xml_valid?(granule_xml).should be_truthy
    i += 1
  end
end

Then(/^criteria_440 Verify that the TCP proxy shows that the export request contains a HTTP query parameter
xml_diff=true for each of the granules g1, g2, g3 indicating verification request$/) do
  i = 0;
  until i > 2 do
    @xmlUtil.http_verification_put_for_granule?(@proxy_log_location, @request_time, @granules[i]).should
be_truthy
    i += 1
  end
end

Then(/^criteria_440 Identify two collections and their lastupdate times$/) do
  @starttime = @colls[0].lastupdate.strftime("%Y-%m-%d %H:%M:%S")
  @endtime = @colls[3].lastupdate.strftime("%Y-%m-%d %H:%M:%S")
end

Then(/^criteria_440 Request longform verification export of collection metadata based on lastupdate times$/) do
  options  = Array.new
  options.push('-vl')
  options.push('-lastupdate')
  time_based_verification_export?('-metc',options).should be_truthy
  ver_coll_requests_processed_successfully?().should be_truthy
end
```

```
Then(/^criteria_440 Verify the collection at starttime is exported but not the one with the end time$/) do
  @bmgtdb.get_coll_export_request_status(@request_time, @colls[0].collectionid, 'VER').should_not be_nil
  @bmgtdb.get_coll_export_request_status(@request_time, @colls[3].collectionid, 'VER').should be_nil
end

Then(/^criteria_440 Verify that the TCP proxy shows an HTTP PUT request each for collection C1, C2, C3$/) do
  i = 0;
  until i > 2 do
    dataset =  @colls[i].datasetid()
    @xmlUtil.http_put_for_collection?(@proxy_log_location, @request_time, dataset).should be_truthy
    i += 1
  end
end

Then(/^criteria_440 Verify that the TCP proxy does not show any request for collection C4$/) do
  dataset =  @colls[3].datasetid()
  @xmlUtil.http_put_for_collection?(@proxy_log_location, @request_time, dataset).should be_falsey
  @xmlUtil.http_delete_for_collection?(@proxy_log_location, @request_time, dataset).should be_falsey
end

Then(/^criteria_440 Verify that the TCP proxy shows full collection metadata for each of the collection C1, C2,
C3$/) do
  i = 0;
  until i > 2 do
    collection_xml = @xmlUtil.get_collection_xml(@proxy_log_location, @request_time, @colls[i])
    @xmlUtil.collection_xml_valid?(collection_xml).should be_truthy
    i += 1
  end
end
Then(/^criteria_440 Verify that the TCP proxy shows that the export request contains a HTTP query parameter
xml_diff=true for each of the collections C1, C2, C3 indicating verification request$/) do
  i = 0;
  until i > 2 do
        dataset = @colls[i].datasetid()
        @xmlUtil.http_verification_put_for_collection?(@proxy_log_location, @request_time, dataset).should
be_truthy
```

```
    i += 1
  end
end

def criteria_440_testdata_checked_out?

  @colls = Array.new
  @data = @sotestdata[@critinfo]
  @data.should_not be_nil
  i = 0
  rv = true
  if @data.length != 4
    puts "There should be 4 collections returned for #{@critinfo}, check AutoTestSoTestDataInfo table"
    rv = false
  else
    @data.each do |esdt|
      shortname = @data[i].shortname
      versionid = @data[i].versionid
      i = i + 1

      @aimdb = Sdps::AimDb.new(@db_conn)
      coll = @aimdb.find_collection_by_shortname_versionid(shortname, versionid)

      if (coll == nil || !@aimdb.is_collection_installed?(shortname, versionid))
        puts "#{shortname}, #{versionid} is not installed"
        rv = false
      else
        #collection should have datamodeltype = 'ECS'
        if coll.datamodeltype != 'ECS'
          puts "collection's datamodeltype is not ECS"
          rv = false
        end
      end
      rv.should be_truthy
      @colls.push(coll)
    end
```

```
    @colls.sort_by!(&:lastupdate)
  end
  rv.should be_truthy
end

def identify_criteria440_granules()
  @granules = @bmgtdb.get_granules_by_time(@colls[0].collectionid, 'lastupdate', 4 )
  last  = @granules.length - 1
  @starttime = @aimdb.find_granule_by_granuleid(@granules[0]).lastupdate.strftime("%Y-%m-%d %H:%M:%S")
  @endtime = @aimdb.find_granule_by_granuleid(@granules[last]).lastupdate.strftime("%Y-%m-%d %H:%M:%S")
end

def check_for_granules?()
  rv = false
  granules = @aimdb.find_notdeleted_granuleids_by_collectionid(@colls[0].collectionid)
  if(granules.length > 0)
    rv = true;
  else
    rv = false;
  end

  return rv

end

def time_based_verification_export?(type, options)
  @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
  rv = Sdps::Bmgt::ExportRequest.new(@mode).manual_time_export?(type, @starttime, @endtime, options)
  rv.should be_truthy
  return rv
end

def ver_coll_requests_processed_successfully?()
  begin
    rv = true;
    i = 0
```

```
    until i > 2 do
      status = Sdps::Util.poll(180, 15){
        result = @bmgtdb.get_coll_export_request_status(@request_time,@colls[i].collectionid, 'VER')
        if (result == 'PENDING' || result == 'STARTED')
          nil
        else
          result
        end
      }
      rv = false if (status != 'SUCCESS' && status != 'WARNING')
      i += 1
    end
    return rv;
  end
end
def ver_gran_requests_processed_successfully?()
  begin
    rv = true;
    i = 0
    until i > 2 do
      status = Sdps::Util.poll(180, 15){
        result = @bmgtdb.get_gran_export_request_status(@request_time,@granules[i], 'VER')
        if (result == 'PENDING' || result == 'STARTED')
          nil
        else
          result
        end
      }
      rv = false if (status != 'SUCCESS' && status != 'WARNING')
      i += 1
    end
    return rv;
  end
end
```

Current Test Script: Manual_Collection_Inserts.rb

```
Given(/^criteria_150 mode is checked out$/) do
  @mode = ENV['MODE']
  @mode.should_not be_nil
  #host should be checked to be f5oml01v
  @mode_checker = Sdps::Mode.new(@mode)
  @mode_checker.mode_checked_out?().should be_truthy
  @db_subsystem = Sdps::Bmgt::DB_SUBSYSTEM
  @db_conn = Sdps::Db.connect(@db_subsystem, @mode)
  @db_conn.should_not be_nil

  @xmlUtil = Sdps::Bmgt::Xml.new()
  @bmgtdb = Sdps::Bmgt::BmgtDb.new(@db_conn)

  #Initialize critinfo and SoTestData
  @critinfo = "BE_82_01:150"
  @sotestdata = Sdps::SoTestData.new(@db_conn)

  @proxy_log_location = "/tools/common/test/BE_82_01/proxy/#{@mode}/logs/tcp.log"
  @manual_log = "/usr/ecs/#{@mode}/CUSTOM/logs/EcBmBMGTManualDriver.log"
end

Given(/^criteria_150 testdata is checked out$/) do
  criteria_150_testdata_checked_out?().should be_truthy
end

Given(/^criteria_150 bmgt subsystem is checked out$/) do
  @mode_checker.bmgt_subsystem_checked_out?().should be_truthy, "bmgt system check failed"
end

Given(/^criteria_150 collections are installed$/) do

  @colls.each do |coll |
    @aimdb.is_collection_installed?(coll.shortname, coll.versionid).should be_truthy,
"#{coll.shortname}.#{coll.versionid} for Criteria 150 not installed."
  end
```

```
end

Given(/^criteria_150  collections are enabled for collection and granule export\.$/) do

  @colls.each do |coll |
    @bmgtdb.enable_collection_export(coll.collectionid).should be_truthy
  end

end

When(/^criteria_150 Request export of collection metadata for collection C1\.$/) do
  @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
  manual_option_export_for_collection?(@colls[0].collectionid, nil).should be_truthy
end

Then(/^criteria_150 Verify the bg_export_request table shows a successful export for collection C1$/) do
  status = Sdps::Util.poll(180, 15){
    result = @bmgtdb.get_coll_export_request_status(@request_time, @colls[0].collectionid, 'MAN')
    if (result == 'PENDING' || result == 'STARTED')
      nil
    else
      result
    end
  }
  status.should eq 'SUCCESS'
end

Then(/^criteria_150 Verify that the TCP proxy log reports a single HTTP PUT request for collection C1$/) do
  dataset =  @colls[0].datasetid()
  @xmlUtil.http_put_for_collection?(@proxy_log_location, @request_time, dataset).should be_truthy
end

Then(/^criteria_150 Verify that the HTTP PUT request contains collection C1's full collection metadata\.$/) do
  collection_xml = @xmlUtil.get_collection_xml(@proxy_log_location, @request_time, @colls[0])
  @xmlUtil.collection_xml_valid?(collection_xml).should be_truthy
end
```

```
Then(/^criteria_150 Request insert only export of collection metadata for collection C2$/) do
  @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
  manual_option_export_for_collection?(@colls[1].collectionid, '-ins').should be_truthy
end

Then(/^criteria_150 Verify the bg_export_request table shows a successful export for collection C2$/) do
  status = Sdps::Util.poll(180, 15){
    result = @bmgtdb.get_coll_export_request_status(@request_time, @colls[1].collectionid, 'MAN')
    if (result == 'PENDING' || result == 'STARTED')
      nil
    else
      result
    end
  }
  status.should eq 'SUCCESS'
end

Then(/^criteria_150 Verify that the TCP proxy log reports a single HTTP PUT request for collection C2$/) do
  dataset2 =  @colls[1].datasetid()
  @xmlUtil.http_put_for_collection?(@proxy_log_location, @request_time, dataset2).should be_truthy
end

Then(/^criteria_150 Verify that the HTTP PUT request contains collection C2's full collection metadata\.$/) do
  collection_xml2 = @xmlUtil.get_collection_xml(@proxy_log_location, @request_time, @colls[1])
  @xmlUtil.collection_xml_valid?(collection_xml2).should be_truthy
end

Then(/^criteria_150 Request delete only export of collection metadata for collection C3$/) do
  @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
  manual_option_export_for_collection?(@colls[2].collectionid, '-del').should be_falsey
end

Then(/^criteria_150 Verify that no export request was enqueued$/) do
  result = @bmgtdb.get_coll_export_request_status(@request_time, @colls[2].collectionid, 'MAN')
  result.should be_nil
```

```
end

Then(/^criteria_150 Verify that the Manual driver log indicates that no request was generated as the collection
was not deleted$/) do
  #Check the generator log, after @request_time
  foundmsg = false
  open (@manual_log) do |log|
    # move the file pointer to the line after the timestamp
    log.find{|line| line =~ /^\d{4}-\d\d-\d\d \d\d:\d\d/ and line > @request_time}

    puts "Coll2 esdt ** #{@colls[2].esdt}"

    #find all the lines with ERROR for the collection esdt
    msg =  log.grep(/Collection #{@colls[2].esdt} cannot be added as a Delete request since it exists in
AmCollection/).map{|line| line.chomp}
    len = msg.length
    if len > 0
      foundmsg = true
    end
  end
  foundmsg.should be_truthy
end

Then(/^criteria_150 Verify that the TCP proxy shows no export request sent$/) do
  dataset3 =  @colls[2].datasetid()
  @xmlUtil.http_put_for_collection?(@proxy_log_location, @request_time, dataset3).should be_falsey
  @xmlUtil.http_delete_for_collection?(@proxy_log_location, @request_time, dataset3).should be_falsey
end

def criteria_150_testdata_checked_out?

  @colls = Array.new
  @data = @sotestdata[@critinfo]
  @data.should_not be_nil
  i = 0
  rv = true
```

```
  if @data.length != 3
    puts "There should be 3 collections returned for #{@critinfo}, check AutoTestSoTestDataInfo table"
    rv = false
  else
    @data.each do |esdt|
      shortname = @data[i].shortname
      versionid = @data[i].versionid
      i = i + 1

      @aimdb = Sdps::AimDb.new(@db_conn)
      coll = @aimdb.find_collection_by_shortname_versionid(shortname, versionid)

      if (coll == nil || !@aimdb.is_collection_installed?(shortname, versionid))
        puts "#{shortname}, #{versionid} is not installed"
        rv = false
      else
        #collection should have datamodeltype = 'ECS'
        if coll.datamodeltype != 'ECS'
          puts "collection's datamodeltype is not ECS"
          rv = false
        end
      end
      rv.should be_truthy
      @colls.push(coll)
    end

  end
  rv.should be_truthy
end

def manual_option_export_for_collection?(collectionid,  option)
  begin

    options = nil
    if option != nil
      options = Array.new
```

```
      options.push(option)
    end
    Sdps::Bmgt::ExportRequest.new(@mode).manual_export_for_collection?(collectionid, options).should be_truthy,
"Falied running Manual Export for collection"

    #check if request was queued

    if @bmgtdb.get_coll_export_request_status(@request_time, collectionid, 'MAN') == nil
      false
    else
      true
    end
  rescue =>e
    puts e.backtrace
    false
  end
end
Current Test Script: BE_82_01_40_S1.rb
Given(/^for BE_82_01:40:1 test mode is checked out$/) do
  #Need to add mode scheck out script
  @mode = ENV['MODE']
  @mode.should_not be_nil
  @mode_checker = Sdps::Mode.new(@mode)
  #host should be checked to be f5oml01v
  @mode_checker.mode_checked_out?().should be_truthy

  @proxy_log_location = @mode_checker.proxy_log_location

  @db_subsystem = Sdps::Bmgt::DB_SUBSYSTEM
  @db_conn = Sdps::Db.connect(@db_subsystem, @mode)
  @db_conn.should_not be_nil
end

Given(/^for BE_82_01:40:1 bmgt subsystem is checked out$/) do
  @mode_checker.bmgt_subsystem_checked_out?().should be_truthy, "bmgt system check failed"
end
```

```
#BE_82_01:40:1
Given(/^BE_82_01:40:1 testdata is checked out$/) do
  @db_conn.should_not be_nil
  @so_test_data = Sdps::SoTestData.new(@db_conn)
  @so_test_data.should_not be_nil

  @so_test_data_items = @so_test_data['BE_82_01:40:1']
  @so_test_data_items.should_not be_nil
  @so_test_data_items.length.should_not be_zero

  # check the collections in the granules
  esdts = @so_test_data.get_unique_collection_esdts()
  collections = @so_test_data.get_unique_collection_data_items()

  @aimdb = Sdps::AimDb.new(@db_conn)
  @collection1 = @aimdb.find_collection_by_shortname_versionid(collections[0].shortname,
collections[0].versionid)
  @collection1.should_not be_nil

  if @so_test_data_items.length > 1
      @collection2 = @aimdb.find_collection_by_shortname_versionid(collections[1].shortname,
collections[1].versionid)
      @collection2.should_not be_nil, "test data not installed"
      #if @collection1 is not science, swap collections data

      if @collection1.type != 'SC' #swap them
          @collection1 = @collection2
          @collection2 = nil
      elsif @collection2.type != 'SC'
           @collection2 = nil
     end
  else
      @collection2 = nil
  end
end
```

```
Given(/^BE_82_01:40:1 two collections with ECS metadata and which share the same short name, but have different
version IDs$/) do
  #test 40:S1
  (@collection1.shortname == @collection2.shortname).should be_truthy, "the two test collections do not have the
same shortname"
  (@collection1.versionid == @collection2.versionid).should_not be_truthy, "the two test collections have the
same versionid"

  @test_granules1 = Array.new()
  @test_granules2 = Array.new()
  @so_test_data_granules = @so_test_data.get_so_test_data_granules(@so_test_data_items)

  @so_test_data_granules.each { |so_test_data_granule|
      granules = @aimdb.find_granules_by_localgranuleid(so_test_data_granule.localgranuleid)

      #if granules.count > 1
           #need to delete and reingest if granules count is > 1
           #check if they are deleted
      #else
           granules.each {  |granule|

           if granule.deleteeffectivedate == nil and granule.deletefromarchive != 'Y'
                if granule.collectionid ==  @collection1.collectionid
                    @test_granules1.push(granule)
               elsif @collection2 != nil and granule.collectionid == @collection2.collectionid
                    @test_granules2.push(granule)
               end
          end
           }
       #end
  }
end


Given(/^BE_82_01:40:1 test collections are enabled for collection and granule export$/) do
```

```
  # test 40
  @bmgtdb = Sdps::Bmgt::BmgtDb.new(@db_conn)
  if @collection1.type == "SC" #ignore associated granule esdts
    collectionexportflag =
@bmgtdb.get_collection_granule_export(@collection1.collectionid)['collectionexportflag']
    granuleexportflag = @bmgtdb.get_collection_granule_export(@collection1.collectionid)['granuleexportflag']

    if collectionexportflag != 'Y' or  granuleexportflag != 'Y'
      @bmgtdb.enable_collection_granule_export(@collection1.collectionid).should be_truthy, "enable collection
granule export for collection 1 failed"
    end
  end

  @collection2.should_not be_nil

  if @collection2.type == "SC" #ignore associated granule esdts
      collectionexportflag =
@bmgtdb.get_collection_granule_export(@collection2.collectionid)['collectionexportflag']
      granuleexportflag = @bmgtdb.get_collection_granule_export(@collection2.collectionid)['granuleexportflag']
      if collectionexportflag != 'Y' or granuleexportflag != 'Y'
         @bmgtdb.enable_collection_granule_export(@collection2.collectionid).should be_truthy, "enable
collection granule export for collection 2 failed"
      end
  end
end


When(/^BE_82_01:40:1 Request the manual export of granule metadata for all granules in one of these
collections$/) do
    @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
    @export_queue = 'MAN'
    @poll_wait_time = 300 # toget from config prop table
    Sdps::Bmgt::ExportRequest.new(@mode).manual_granule_export_for_collection?(@collection1.esdt,
nil).should_not be_falsey, "Manual Export of collection failed"
end
```

```
Then(/^for BE_82_01:40:1 Verify that the operations each result in the export of one or more HTTP PUTs
containing the full granule metadata of the exported granules in the collection$/) do
    status = nil
    @test_granules1.each  { |test_granule|

       #poll status and wait till timeout
         status = Sdps::Util.poll(@poll_wait_time, 5){
         result = Sdps::Bmgt::BmgtDb.new(@db_conn).get_gran_export_request_status(@request_time,
test_granule.granuleid, @export_queue)
         if result != 'SUCCESS'
           nil
         else
           result
         end
       }
    }
     status.should eq 'SUCCESS'
     # , "Export Request status not SUCCESS"
     if status == 'SUCCESS'
       @xml_utils = Sdps::Bmgt::Xml.new()
       #check tcp log
       @test_granules_xml = Hash.new()

       @test_granules1.each  do |test_granule|

         @xml_utils.http_put_for_granule?(@proxy_log_location, @request_time, test_granule.granuleid).should
be_truthy, "No HTTP PUT request for granule - #{test_granule.granuleid}"
         granule_xml = @xml_utils.get_granule_xml(@proxy_log_location, @request_time, test_granule.granuleid)
         granule_xml.should_not be_nil, "granule xml not found"

         #save xml for later validation
         @test_granules_xml[test_granule.granuleid] = granule_xml
       end
     end
     status.should eq 'SUCCESS'
     @test_granules_xml.should_not be_nil, "granule xml not found"
```

```
end

Then(/^for BE_82_01:40:1 Verify that no granules belonging to other collections sharing a short name but with a
different version ID$/) do
     #get granules for the collection 2
     # should check for granules in check test data
     #check tcp log
     @test_granules2.each  do |test_granule|
       @xml_utils.http_put_for_granule?(@proxy_log_location, @request_time, test_granule.granuleid).should
be_falsey, "There is a HTTP PUT request for granule - #{test_granule.granuleid} in collection 2"
     end
end

Then(/^for BE_82_01:40:1 Verify that the metadata exported validates against the ECHO granule metadata schema
\(https:\/\/api\.echo\.nasa\.gov\/ingest\/schema\/Granule\.xsd\)\.$/) do
   @test_granules_xml.should_not be_nil, "granule xml not found"
  @test_granules1.each { |test_granule|
      granule_xml = @test_granules_xml[test_granule.granuleid]
      granule_xml.should_not be_nil, "granule xml not found"
      @xml_utils.granule_xml_valid?(granule_xml).should be_truthy, "XML not valid for granule -
#{test_granule.granuleid}"
    }
end

Then(/^for BE_82_01:40:1 Verify that the metadata exported has the element Visible = true$/) do
    @test_granules_xml.keys.each do |granuleid|
      granule_xml = @test_granules_xml[granuleid]
      @xml_utils.granule_xml_contains?(granule_xml,"//Granule/Visible", "<Visible>true</Visible>").should
be_truthy, "granule - #{granuleid} xml does not contain Visible = true"
    end
end

Then(/^for BE_82_01:40:1 Verify that the metadata exported has the element Orderable = true$/) do
    @test_granules_xml.should_not be_nil, "granule xml not found"
    @test_granules_xml.keys.each do |granuleid|
      granule_xml = @test_granules_xml[granuleid]
```

```
      @xml_utils.granule_xml_contains?(granule_xml,"//Granule/Orderable", "<Orderable>true</Orderable>").should
be_truthy, "granule - #{granuleid} xml does not contain Orderable = true"
    end
end

Then(/^for BE_82_01:40:1 Verify that the metadata exported has the element InsertTime = The insert time of the
granule recorded in the AIM database\.$/) do
    @test_granules1.each do |test_granule|
      archivetime = test_granule.archivetime
      inserttime_in_metadata = DateTime.parse(archivetime.to_s).strftime('%Y-%m-%dT%H:%M:%S.%3NZ')
      granule_xml = @test_granules_xml[test_granule.granuleid]
      inserttime_in_xml = @xml_utils.granule_xml_getvalue(granule_xml, "//Granule/InsertTime")
      @xml_utils.granule_xml_contains?(granule_xml, "//Granule/InsertTime",
"<InsertTime>#{inserttime_in_metadata}</InsertTime>").should be_truthy, " The insert time in the metadata
exported is not equal to the archive time in the database for granule - #{test_granule.granuleid}"
    end
end

Then(/^for BE_82_01:40:1 Verify that the metadata exported has the element LastUpdate = The last update time of
the granule recorded in the AIM database$/) do
    @test_granules1.each do |test_granule|
      # in case the last update changed after the testdata checkoutout
      test_granule_new = @aimdb.find_granule_by_granuleid(test_granule.granuleid)
      lastupdate = test_granule_new.lastupdate
      lastupdate.should_not be_nil
      lastupdate_in_metadata = DateTime.parse(lastupdate.to_s).strftime('%Y-%m-%dT%H:%M:%S.%3NZ')
      granule_xml = @test_granules_xml[test_granule.granuleid]
      last_update_in_xml = @xml_utils.granule_xml_getvalue(granule_xml, "//Granule/LastUpdate")

      if @xml_utils.granule_xml_contains?(granule_xml, "//Granule/LastUpdate",
"<LastUpdate>#{lastupdate_in_metadata}</LastUpdate>") != true
         #check if there was another PUT in the tcp log
         granule_xml = @xml_utils.get_granule_xml(@proxy_log_location, lastupdate_in_metadata,
test_granule.granuleid)

         @xml_utils.granule_xml_contains?(granule_xml, "//Granule/LastUpdate",
```

```
"<LastUpdate>#{lastupdate_in_metadata}</LastUpdate>").should be_truthy, "The LastUpdate in the metadata exported
is not equal to the lastupdate in the database for granule - #{test_granule.granuleid}"
      end
    end
end
Current Test Script: BE_82_01_40_V8.rb
Then(/^Verify that the metadata generated by the action in S-2 subclause g \(and any other subclauses for which
the affected granule did not have a restriction flag set\) the restriction flag element is absent$/) do
    #get restriction flag set
    #@restriction_flag
    @test_granules_xml.keys.each do |granuleid|
      granule_xml = @test_granules_xml[granuleid]
      @xml_utils.granule_xml_getvalue(granule_xml,"//Granule/RestrictionFlag").should be_empty, "granule -
#{granuleid} xml contains RestrictionFlag"
    end
end
Current Test Script: BE_82_01_40_V15.rb
#Criteria definition
#Then(/^Verify that the action performed in S-2 subclause n results in the export of full granule metadata for
every public granule in the collection affected. Verify that no nonpublic granules are exported, and that the
exported metadata for public granules includes URLs which correspond to the target filesystem of the collection
move$/) do
#How it works now
Then(/^Verify that the action performed in S-2 subclause n results in the export of full granule metadata for
every granule in the collection affected. Verify that the exported metadata for public granules includes URLs
which correspond to the target filesystem of the collection move$/) do
  #get public granules in the collection - in  @test_granules1
  #verification of public granules metadata export done in V1
  #need to verify the urls contain the target filesystem
  @test_granules1.each  do |test_granule|
      puts test_granule.granuleid
      granule_xml = @test_granules_xml[test_granule.granuleid]
      granule_xml.should_not be_nil, " meta data not exported for granule #{test_granule.granuleid}"
      puts granule_xml
      if @aimdb.granule_published?(test_granule.granuleid) == true
         urls = @xml_utils.granule_xml_getvalue(granule_xml,"//OnlineResource/URL")
```

```
        urls.should_not be_nil, "no onlineresource urls exported"
        puts urls
        urls.include?(@current_filesystem_path).should be_falsey
        urls.include?(@target_filesystem_path).should be_truthy
        urls2 = @xml_utils.granule_xml_getvalue(granule_xml,"//OnlineAccessURL/URL")
        urls2.should_not be_nil, "no onlineaccess urls exported"
        puts urls2
        urls2.include?(@current_filesystem_path).should be_falsey
        urls2.include?(@target_filesystem_path).should be_truthy
      end
  end

  #get nonpublic granules in the collection - in @test_granules2
  #@test_granules2.each  do |test_granule|
  #        puts test_granule.granuleid
  #       @xml_utils.http_put_for_granule?(@proxy_log_location, @request_time,
test_granule.granuleid).should_not be_truthy, "HTTP PUT request for non public granule -
#{test_granule.granuleid}"
  #end

end
Current Test Script: BE_82_01_40_V5d.rb

Current Test Script: BE_82_01_40_S2h.rb
When(/^Perform a QAUpdate on a granule$/) do
     @test_granule.should_not be_nil, "test granule not available"
      granuleid = @test_granule.granuleid
      granuleid.should_not be_nil, "granuleid cannot be nil"
      puts granuleid
      @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
      @export_queue = 'EVENT'
      @poll_wait_time = 300 #need to get this value from the configuration property table
      @qauu_wait_time = 300 # from config

      #get granule metadata file
      granule_metadata_file = @aimdb.get_granule_metadata_file(granuleid)
```

```
      granule_metadata_file.should_not be_nil, "could not get granule metadata file from the database"
      puts "granule metadatafile=" + granule_metadata_file

      @qauu.site_configured.should be_truthy, "site not configured"

       #find measured parameter
       measured_parameter = @qauu.get_measured_parameter(granule_metadata_file)
       measured_parameter.should_not be_nil, "could not get measured parameter"
        puts "measured parameter=" + measured_parameter

       #get science quality flag value
       current_science_quality_flag = @qauu.get_measured_parameter_science_quality_flag(granule_metadata_file,
measured_parameter)
        puts "current science quality flag = " + current_science_quality_flag

       #get VALID_SCIENCE_QA_FLAGS
       valid_science_qa_flags = @qauu.get_valid_science_quality_flags()
       valid_science_qa_flags.should_not be_nil, "could not get valid science flags"
       valid_science_qa_flags.should_not be_empty, "valid science flags not configured"

       qa_flag_value = valid_science_qa_flags.select { |flag| flag != current_science_quality_flag}[0]
       qa_flag_value.should_not be_nil, "could not get another qa science flag"
       puts "qa flag update = " + qa_flag_value

       qa_flag_explanation  = "Testing QA UPDATE"

       #create a qa update request
       description = "BE_82_01_40_S2h"
       @qauu.configure_request(@collection1.versionid, description)

       request_file = @qauu.create_science_qa_request_with_granule_ur(@test_granule.granuleid,
measured_parameter, qa_flag_value, qa_flag_explanation)
       request_file.should_not be_nil, "could not create qa request"
       status = @qauu.qa_update(request_file)
       status.should be_truthy, "QA update failed"
end
```

```
Current Test Script: BE_82_01_40_S2i.rb
  When(/^Publish a granule in the datapool$/) do
    granuleid = @test_granule.granuleid
    @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
    @export_queue = 'EVENT'
    @poll_wait_time = 300 #need to get this value from the configuration property table
    @publish_wait_time = 300 # from config

    # get hostname for publish
    publisher = Sdps::Publishing.new(@mode)
    publisher.should_not be_nil
    publisher.publish_granule(granuleid).should be_truthy
    puts "done publishing"

    #check if granule is published
     if @aimdb.nil?
       @aimdb = Sdps::AimDb.new(@db_conn)
     end
    status = Sdps::Util.poll(@publish_wait_time, 5){
       if @aimdb.granule_published?(granuleid) == true
          puts "published in db"
          true
       else
          puts "not published in db"
          nil
       end
    }
    status.should be_truthy, "granule #{granuleid} not published"
  end
Current Test Script: BE_82_01_40_V11.rb
Then(/^Verify that the metadata generated by the action in S-2 subclause j contains no datapool URLs$/) do
    @test_granules_xml.keys.each do |granuleid|
      granule_xml = @test_granules_xml[granuleid]
      @xml_utils.granule_xml_getvalue(granule_xml,"//OnlineAccessURL/URL").should be_empty, "granule -
#{granuleid} xml contains OnlineAccessURL"
```

```
      @xml_utils.granule_xml_getvalue(granule_xml,"//OnlineResource/URL").should be_empty, "granule -
#{granuleid} xml contains OnlineResource"
    end
end
Current Test Script: Ecs_Collection_Additional_Metadata_S6.rb
require 'sdps/sotestdata'
require 'sdps/aimdb'
require 'sdps/mode'
require 'sdps/db'
require 'sdps/util'
require 'sdps/bmgt'

Given(/^Criteria_20_6 Initial Setup$/) do
  #Check mode.
  @mode = ENV['MODE']
  @mode.should_not be_nil
  @mode_checker = Sdps::Mode.new(@mode)
  @mode_checker.mode_checked_out?().should be_truthy
  @proxy_log_location = @mode_checker.get_proxy_log_location()
  @xmlUtil = Sdps::Bmgt::Xml.new()

  #set up db connection
  @db_subsystem = Sdps::Bmgt::DB_SUBSYSTEM
  @conn = Sdps::Db.connect(@db_subsystem, @mode)
  @conn.should_not be_nil

  #Make sure BMGT is up and running, proxy needs to be up too
  @mode_checker.bmgt_subsystem_checked_out?().should be_truthy, "bmgt system check failed"

  #Initialize critinfo and SoTestData
  @critinfo = "BE_82_01:20:6"
  @sotestdata = Sdps::SoTestData.new(@conn)
end

Given(/^Criteria_20_6 All test collections are installed in the mode$/) do
```

```
  aimdb = Sdps::AimDb.new(@conn)
  @collections = Array.new

  data = @sotestdata[@critinfo]
  data.should_not be_nil

  rv = true
  if data.length != 1
    puts "There should be one collection returned for #{@critinfo}, check AutoTestSoTestDataInfo table"
    rv = false
  else
    #loop through @data
    data.each do |dataitem|
      shortname = dataitem.shortname
      versionid = dataitem.versionid
      collection = aimdb.find_collection_by_shortname_versionid(shortname,versionid)

      if collection == nil
        puts "test data #{shortname}, #{versionid} not installed"
        rv = false
      else
        @collections.push(collection)
        #collection should have datamodeltype = 'ECS'
        if collection.datamodeltype != 'ECS'
          puts "collection's datamodeltype is not ECS"
          rv = false
        end
      end
    end
  end
  rv.should be_truthy
end

Given(/^Criteria_20_6 All test collections are configured for collection and granule metadata export$/) do
  bmgtdb = Sdps::Bmgt::BmgtDb.new(@conn)
  @collections.each do |collection|
```

```
    collectionid = collection.collectionid
    if bmgtdb.collection_export_enabled?(collectionid) == false
      bmgtdb.enable_collection_granule_export(collectionid).should be_truthy
    end
  end
end

Given(/^Criteria_20_6 All test collections have valid CoordinateSystem and GranuleSpatialRepresentation$/) do
  rv = true
  @collections.each do |collection|
    #set CoordinateSystem to CARTESIAN and GranuleSpatialRepresentation to GEODETIC
    collectionid = collection.collectionid
    begin
        rs = @conn.exec ("update bg_collection_configuration set GranuleSpatialRepresentation='GEODETIC',
CollectionCoordinateSystem='CARTESIAN' where collectionid = '#{collectionid}' returning 1")
        if rs == nil or rs.count != 1
          rv = false
        end
    rescue => e
        puts e.backtrace
        rv = false
    end
  end
  rv.should be_truthy
end

Given(/^Criteria_20_6 A local copy of the collection schema is available to validate against$/) do
  #verify Sdps::Bmgt::ECHO10_COLLECTION_SCHEMA exists
  schemaFile = Sdps::Bmgt::ECHO10_COLLECTION_SCHEMA
  File.exists?(schemaFile).should be_truthy
end

Given(/^Criteria_20_6 A TCP proxy is capturing BMGT traffic$/) do
 #This is checked at the initial setup
 puts "If it gets here, the TCP proxy is up and running"
end
```

```
When(/^Criteria_20_6 S-6 Request the export of metadata for a collection which is not configured for
backtrack$/) do
  #record the request time
  @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")

  @collections.each do |collection|
    collectionid = collection.collectionid
    rv = true
    begin
      rs = @conn.exec ("update bg_collection_configuration set orbitGroup = null where collectionid =
'#{collectionid}' returning 1")
      if rs == nil or rs.count != 1
        rv = false
      end
    rescue => e
      puts e.backtrace
      rv = false
    end
    rv.should be_truthy

    Sdps::Bmgt::ExportRequest.new(@mode).manual_export_for_collection?(collectionid, nil).should be_truthy,
"Failed running Manual Export for collection"
    ##make sure the request is queued
    Sdps::Bmgt::BmgtDb.new(@conn).get_coll_export_request_status(@request_time, collectionid, "MAN").should_not
be_nil
  end
end

Then(/^Criteria_20_6 Manual Export succeeds$/) do
  @collections.each do |collection|
    collectionid = collection.collectionid
    #poll and wait
    export_req = Sdps::Bmgt::BmgtDb.new(@conn)
    status = Sdps::Util.poll(180, 15){
      result = export_req.get_coll_export_request_status(@request_time, collectionid, "MAN")
```

```
      if (result == 'PENDING' || result == 'STARTED')
        nil
      else
        result
      end
    }
    status.should eq 'SUCCESS'
  end
end

Then(/^Criteria_20_6 V-12 Verify that all Collection metadata in the previous steps validates against
https:\/\/api\.echo\.nasa\.gov\/ingest\/schema\/Collection\.xsd$/) do
  #Verify the S-12 collection's exported XML validates against the schema:
  #The following method makes use of the local copy of the collection.xsd file:
Sdps::Bmgt::ECHO10_COLLECTION_SCHEMA
  count = 0
  @collections.each do |collection|
    collection_xml = @xmlUtil.get_collection_xml(@proxy_log_location, @request_time, collection)
    #puts "xml is #{collection_xml}"
    @xmlUtil.collection_xml_valid?(collection_xml).should be_truthy
    puts "Verification okay"

    #Verify the collection xml file contains no Spatial/OrbitParameters, easier to combine the next step here
    @xmlUtil.collection_xml_contains?(collection_xml, "//Collection/Spatial/OrbitParameters", "").should
be_truthy
    puts "Contains no Spatial/OrbitParameters element okay"

    #Verify the collection xml file contains configured CoordinateSystem and GranuleSpatialRepresentation
values, again, easier to combine it here
    @xmlUtil.collection_xml_contains?(collection_xml,
"//Collection/Spatial/HorizontalSpatialDomain/Geometry/CoordinateSystem",
"<CoordinateSystem>CARTESIAN</CoordinateSystem>").should be_truthy
    puts "Contains CoordinateSystem okay"

    @xmlUtil.collection_xml_contains?(collection_xml, "//Collection/Spatial/GranuleSpatialRepresentation",
"<GranuleSpatialRepresentation>GEODETIC</GranuleSpatialRepresentation>").should be_truthy
```

```
    puts "contains GranuleSpatialRepresentation okay"
    count = count + 1
  end
end

Then(/^Criteria_20_6 V-6 Verify that the collection metadata generated in S-6 contains no
Spatial\/OrbitParameters element$/) do
  puts "If it gets here, the step is verified"

end

Then(/^Criteria_20_6 V-15 For the exports in S-5 to S-9, verify that the configured CoordinateSystem and
GranuleSpatialRepresentation values are included in the metadata$/) do
  puts "if it gets here, the step is verified"
end
```

Current Test Script: BE_82_01_40_V5c.rb

Current Test Script: BE_82_01_40_V5a.rb

Current Test Script: BE_82_01_40_V7.rb

```
Then(/^Verify that the metadata generated by the action in S-2 subclause f contains the restriction flag value
that was set in the database for that granule$/) do
    #get restriction flag set
    #@restriction_flag
    @test_granules_xml.keys.each do |granuleid|
      granule_xml = @test_granules_xml[granuleid]
      @xml_utils.granule_xml_contains?(granule_xml,"//Granule/RestrictionFlag",
"<RestrictionFlag>#{@restriction_flag}</RestrictionFlag>").should be_truthy, "granule - #{granuleid} xml does
not contain RestrictionFlag = #{@restriction_flag}"
    end
end
```

Current Test Script: Ecs_Collection_Additional_Metadata_S4.rb

```
require 'sdps/sotestdata'
require 'sdps/aimdb'
require 'sdps/mode'
require 'sdps/db'
require 'sdps/util'
require 'sdps/bmgt'

Given(/^Criteria_20_4 Initial Setup$/) do
  #Check mode.
  @mode = ENV['MODE']
  @mode.should_not be_nil
  @mode_checker = Sdps::Mode.new(@mode)
  @mode_checker.mode_checked_out?().should be_truthy
  @proxy_log_location = @mode_checker.get_proxy_log_location()
  @generator_log_location = "/usr/ecs/#{@mode}/CUSTOM/logs/EcBmBMGTGenerator.log"
  @xmlUtil = Sdps::Bmgt::Xml.new()

  #set up db connection
  @db_subsystem = Sdps::Bmgt::DB_SUBSYSTEM
  @conn = Sdps::Db.connect(@db_subsystem, @mode)
  @conn.should_not be_nil

  #Make sure BMGT is up and running, proxy needs to be up too
  @mode_checker.bmgt_subsystem_checked_out?().should be_truthy, "bmgt system check failed"

  #Initialize critinfo and SoTestData
  @critinfo = "BE_82_01:20:4"
  @sotestdata = Sdps::SoTestData.new(@conn)
end

Given(/^Criteria_20_4 All test collections are installed in the mode$/) do
  @data = @sotestdata[@critinfo]
  @data.should_not be_nil

  rv = true
  if @data.length != 1
```

```
    puts "There should be one collection returned for #{@critinfo}, check AutoTestSoTestDataInfo table"
    rv = false
  else
    @shortname = @data[0].shortname
    @versionid = @data[0].versionid

    @aimdb = Sdps::AimDb.new(@conn)
    @collection =@aimdb.find_collection_by_shortname_versionid(@shortname,@versionid)

    if @collection == nil
      puts "test data #{@shortname}, #{@versionid} not installed"
      rv = false
    else
      #collection should have datamodeltype = 'ISO-SMAP'
      if @collection.datamodeltype != 'ISO-SMAP'
        puts "collection's datamodeltype is not ISO-SMAP"
        rv = false
      end
    end
  end
  rv.should be_truthy
end

Given(/^Criteria_20_4 All test collections are configured for collection and granule metadata export$/) do
  @collectionid = @collection.collectionid
  @bmgtdb = Sdps::Bmgt::BmgtDb.new(@conn)

  if @bmgtdb.collection_export_enabled?(@collectionid) == false
    @bmgtdb.enable_collection_granule_export().should be_truthy
  end
end

Given(/^Criteria_20_4 A local copy of the collection schema is available to validate against$/) do
 puts "In the case of Criteria_20_4, no xml should be exported, therefore no need for the schema file"
end
```

```
Given(/^Criteria_20_4 A TCP proxy is capturing BMGT traffic$/) do
 #This is checked at the initial setup
 puts "If it gets here, the TCP proxy is up and running"

end

When(/^Criteria_20_4 update the value of the S-4 collection's GranuleSpatialRepresentation to any string$/) do
  #There's only one collection in this test

  begin
    rs = @conn.exec ("update bg_collection_configuration set GranuleSpatialRepresentation='NO_SPATIAL',
CollectionCoordinateSystem = null where ShortName='#{@shortname}' and versionid = '#{@versionid}' returning 1")

    if rs != nil and rs.count == 1
      rv = true
    else
      rv = false
    end
  rescue => e
    puts e.backtrace
    rv = false
  end

  rv.should be_truthy
end

Then(/^Criteria_20_4 Verify that it succeeds$/) do

  puts "If it gets here, the update successful is verified"

end

When(/^Criteria_20_4 Request a Manual Export of the S-4 collection$/) do
  #record the request time
  @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
  Sdps::Bmgt::ExportRequest.new(@mode).manual_export_for_collection?(@collection.collectionid, nil).should_not
```

```
be_falsey, "Falied running Manual Export for collection"

  ##make sure the request is queued
  Sdps::Bmgt::BmgtDb.new(@conn).get_coll_export_request_status(@request_time, @collection.collectionid,
"MAN").should_not be_nil
end

Then(/^Criteria_20_4 Verify the manual export fails$/) do
  #check the EcBmBMGTManualDriver.log? key word?
  #actually the manual run log just queues the request, the actual run error appears in the generator log
  #but first have to wait till the request is processed
  #poll and wait
  export_req = Sdps::Bmgt::BmgtDb.new(@conn)
  status = Sdps::Util.poll(180, 15){
    result = export_req.get_coll_export_request_status(@request_time, @collection.collectionid, "MAN")
    if (result == 'PENDING' || result == 'STARTED')
      nil
    else
      result
    end
  }
  status.should eq 'SKIPPED'

end

Then(/^Criteria_20_4 Verify the log file records the export failure and the reason for failure$/) do
  #Check the generator log, after @request_time
  founderror = false
  open (@generator_log_location) do |log|
    # move the file pointer to the line after the timestamp
    log.find{|line| line =~ /^\d{4}-\d\d-\d\d \d\d:\d\d/ and line > @request_time}
    #find all the lines with ERROR for the collection esdt
    len =  log.grep(/ERROR.*#{@collection.esdt}.*ISO-SMAP metadata should not have a
GranuleSpatialRepresentation configured/).map{|line| line.chomp}.length
    if len > 0
      founderror = true
```

```
    end
  end
  founderror.should be_truthy
end

Then(/^Criteria_20_4 Verify the log file indicates the invalid GranuleSpatialRepresentation value caused the
export failure\.$/) do
  puts "If it gets here, the failure msg is verified"
end

Then(/^Criteria_20_4 Verify that all Collection metadata in the previous steps validates against
https:\/\/api\.echo\.nasa\.gov\/ingest\/schema\/Collection\.xsd$/) do
  puts "In the case of Criteria_20_4, there is no xml file exported, therefore nothing to validate against the
schema file"
end

Then(/^Criteria_20_4 Verify that no collection metadata should have been exported$/) do
  #there is no Put request in the proxy log?
  dataset =  @collection.datasetid()
  @xmlUtil.http_put_for_collection?(@proxy_log_location, @request_time, dataset).should be_falsey
end

Current Test Script: BE_82_01_40_V1.rb

Current Test Script: nominal_manual_granule_export.rb
Given(/^"(.*?)" and "(.*?)" testdata is checked out$/) do |esdt1, esdt2|
  @db_conn.should_not be_nil

  # check the collections in the granules
  esdts = [esdt1, esdt2]
  collection1_arr = esdt1.split('.')
  collection2_arr = esdt2.split('.')
  #collections = @so_test_data.get_unique_collection_data_items()

  @aimdb = Sdps::AimDb.new(@db_conn)
  @collection1 = @aimdb.find_collection_by_shortname_versionid(collection1_arr[0], collection1_arr[1].to_i)
```

```
  @collection2 = @aimdb.find_collection_by_shortname_versionid(collection2_arr[0], collection2_arr[1].to_i)

  @collection1.should_not be_nil, "test data #{esdt1} not installed"
  @collection2.should_not be_nil, "test data #{esdt2 }not installed"

  end

Given(/^two collections with ECS metadata and which share the same short name, but have different version IDs$/)
do
  #test 40:S1
  (@collection1.shortname == @collection2.shortname).should be_truthy, "the two test collections do not have the
same shortname"
  (@collection1.versionid == @collection2.versionid).should_not be_truthy, "the two test collections have the
same versionid"

  @test_granules1 = Array.new()
  @test_granules2 = Array.new()
  granules = @aimdb.find_granules_by_shortname_versionid(@collection1.shortname, @collection1.versionid)
  granules.each {  |granule|
           if granule.deleteeffectivedate == nil and granule.deletefromarchive != 'Y'
                    @test_granules1.push(granule)
           end
  }
  granules = @aimdb.find_granules_by_shortname_versionid(@collection2.shortname, @collection2.versionid)
  granules.each {  |granule|
           if granule.deleteeffectivedate == nil and granule.deletefromarchive != 'Y'
                    @test_granules2.push(granule)
           end
   }

end


Given(/^test collections are enabled for collection and granule export$/) do
  # test 40
```

```
  @bmgtdb = Sdps::Bmgt::BmgtDb.new(@db_conn)
  if @collection1.type == "SC" #ignore associated granule esdts
    collectionexportflag =
@bmgtdb.get_collection_granule_export(@collection1.collectionid)['collectionexportflag']
    granuleexportflag = @bmgtdb.get_collection_granule_export(@collection1.collectionid)['granuleexportflag']

    if collectionexportflag != 'Y' or  granuleexportflag != 'Y'
      @bmgtdb.enable_collection_granule_export(@collection1.collectionid).should be_truthy, "enable collection
granule export for collection 1 failed"
    end
  end

  @collection2.should_not be_nil

  if @collection2.type == "SC" #ignore associated granule esdts
      collectionexportflag =
@bmgtdb.get_collection_granule_export(@collection2.collectionid)['collectionexportflag']
      granuleexportflag = @bmgtdb.get_collection_granule_export(@collection2.collectionid)['granuleexportflag']
      if collectionexportflag != 'Y' or granuleexportflag != 'Y'
         @bmgtdb.enable_collection_granule_export(@collection2.collectionid).should be_truthy, "enable
collection granule export for collection 2 failed"
      end
  end
end


When(/^Request the manual export of granule metadata for all granules in one of these collections$/) do
    puts  "proxy - location : #{@proxy_log_location}"
    @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
    @export_queue = 'MAN'
    @poll_wait_time = 300 # toget from config prop table
    Sdps::Bmgt::ExportRequest.new(@mode).manual_granule_export_for_collection?(@collection1.esdt,
nil).should_not be_falsey, "Manual Export of collection failed"
end

Then(/^Verify that the operations each result in the export of one or more HTTP PUTs containing the full granule
```

```
metadata of the exported granules in the collection$/) do
    status = nil
    @test_granules1.each  { |test_granule|

       #poll status and wait till timeout
         status = Sdps::Util.poll(@poll_wait_time, 5){
         result = Sdps::Bmgt::BmgtDb.new(@db_conn).get_gran_export_request_status(@request_time,
test_granule.granuleid, @export_queue)
         if result != 'SUCCESS'
           nil
         else
           result
         end
       }
    }
     status.should eq 'SUCCESS'
     # , "Export Request status not SUCCESS"
     if status == 'SUCCESS'
       @xml_utils = Sdps::Bmgt::Xml.new()
       #check tcp log
       @test_granules_xml = Hash.new()

       @test_granules1.each  do |test_granule|
         puts @proxy_log_location
         @xml_utils.http_put_for_granule?(@proxy_log_location, @request_time, test_granule.granuleid).should
be_truthy, "No HTTP PUT request for granule - #{test_granule.granuleid}"
         granule_xml = @xml_utils.get_granule_xml(@proxy_log_location, @request_time, test_granule.granuleid)
         granule_xml.should_not be_nil, "granule xml not found"

         #save xml for later validation
         @test_granules_xml[test_granule.granuleid] = granule_xml
       end
     end
     status.should eq 'SUCCESS'
     @test_granules_xml.should_not be_nil, "granule xml not found"
end
```

```
Then(/^Verify that no granules belonging to other collections sharing a short name but with a different version
ID$/) do
     #get granules for the collection 2
     # should check for granules in check test data
     #check tcp log
     @test_granules2.each  do |test_granule|
       @xml_utils.http_put_for_granule?(@proxy_log_location, @request_time, test_granule.granuleid).should
be_falsey, "There is a HTTP PUT request for granule - #{test_granule.granuleid} in collection 2"
     end
end

Then(/^Verify that the metadata exported validates against the ECHO granule metadata schema
\(https:\/\/api\.echo\.nasa\.gov\/ingest\/schema\/Granule\.xsd\)\.$/) do
   @test_granules_xml.should_not be_nil, "granule xml not found"
  @test_granules1.each { |test_granule|
      granule_xml = @test_granules_xml[test_granule.granuleid]
      granule_xml.should_not be_nil, "granule xml not found"
      @xml_utils.granule_xml_valid?(granule_xml).should be_truthy, "XML not valid for granule -
#{test_granule.granuleid}"
    }
end

Then(/^Verify that the metadata exported has the element Visible = true$/) do
    @test_granules_xml.keys.each do |granuleid|
      granule_xml = @test_granules_xml[granuleid]
      @xml_utils.granule_xml_contains?(granule_xml,"//Granule/Visible", "<Visible>true</Visible>").should
be_truthy, "granule - #{granuleid} xml does not contain Visible = true"
    end
end

Then(/^Verify that the metadata exported has the element Orderable = true$/) do
    @test_granules_xml.should_not be_nil, "granule xml not found"
    @test_granules_xml.keys.each do |granuleid|
      granule_xml = @test_granules_xml[granuleid]
      @xml_utils.granule_xml_contains?(granule_xml,"//Granule/Orderable", "<Orderable>true</Orderable>").should
```

```
be_truthy, "granule - #{granuleid} xml does not contain Orderable = true"
    end
end

Then(/^Verify that the metadata exported has the element InsertTime = The insert time of the granule recorded in
the AIM database\.$/) do
    @test_granules1.each do |test_granule|
      archivetime = test_granule.archivetime
      inserttime_in_metadata = DateTime.parse(archivetime.to_s).strftime('%Y-%m-%dT%H:%M:%S.%3NZ')
      granule_xml = @test_granules_xml[test_granule.granuleid]
      inserttime_in_xml = @xml_utils.granule_xml_getvalue(granule_xml, "//Granule/InsertTime")
      @xml_utils.granule_xml_contains?(granule_xml, "//Granule/InsertTime",
"<InsertTime>#{inserttime_in_metadata}</InsertTime>").should be_truthy, " The insert time in the metadata
exported is not equal to the archive time in the database for granule - #{test_granule.granuleid}"
    end
end

Then(/^Verify that the metadata exported has the element LastUpdate = The last update time of the granule
recorded in the AIM database$/) do
    @test_granules1.each do |test_granule|
      # in case the last update changed after the testdata checkoutout
      test_granule_new = @aimdb.find_granule_by_granuleid(test_granule.granuleid)
      lastupdate = test_granule_new.lastupdate
      lastupdate.should_not be_nil
      lastupdate_in_metadata = DateTime.parse(lastupdate.to_s).strftime('%Y-%m-%dT%H:%M:%S.%3NZ')
      granule_xml = @test_granules_xml[test_granule.granuleid]
      last_update_in_xml = @xml_utils.granule_xml_getvalue(granule_xml, "//Granule/LastUpdate")

      if @xml_utils.granule_xml_contains?(granule_xml, "//Granule/LastUpdate",
"<LastUpdate>#{lastupdate_in_metadata}</LastUpdate>") != true
         #check if there was another PUT in the tcp log
         granule_xml = @xml_utils.get_granule_xml(@proxy_log_location, lastupdate_in_metadata,
test_granule.granuleid)

         @xml_utils.granule_xml_contains?(granule_xml, "//Granule/LastUpdate",
"<LastUpdate>#{lastupdate_in_metadata}</LastUpdate>").should be_truthy, "The LastUpdate in the metadata exported
```

```
is not equal to the lastupdate in the database for granule - #{test_granule.granuleid}"
      end
    end
end
Current Test Script: ManualExport_Granule_Delete.rb
Given(/^criteria_190 mode is checked out$/) do
  @mode = ENV['MODE']
  @mode.should_not be_nil
  #host should be checked to be f5oml01v
  @mode_checker = Sdps::Mode.new(@mode)
  @mode_checker.mode_checked_out?().should be_truthy
  @db_subsystem = Sdps::Bmgt::DB_SUBSYSTEM
  @db_conn = Sdps::Db.connect(@db_subsystem, @mode)
  @db_conn.should_not be_nil

  @xmlUtil = Sdps::Bmgt::Xml.new()
  @bmgtdb = Sdps::Bmgt::BmgtDb.new(@db_conn)

  #Initialize critinfo and SoTestData
  @critinfo = "BE_82_01:180"
  @sotestdata = Sdps::SoTestData.new(@db_conn)

  @proxy_log_location = "/tools/common/test/BE_82_01/proxy/#{@mode}/logs/tcp.log"
  @manual_log = "/usr/ecs/#{@mode}/CUSTOM/logs/EcBmBMGTManualDriver.log"
end

Given(/^criteria_190 bmgt subsystem is checked out$/) do
  @mode_checker.bmgt_subsystem_checked_out?().should be_truthy, "bmgt system check failed"
end

Given(/^criteria_190 testdata is checked out$/) do
  criteria_190_testdata_checked_out?().should be_truthy
end

Given(/^criteria_190 collections are installed$/) do
  @colls.each do |coll |
```

```
    @aimdb.is_collection_installed?(coll.shortname, coll.versionid).should be_truthy,
"#{coll.shortname}.#{coll.versionid} for Criteria 200 not installed."
  end
end

Given(/^criteria_190  collections are enabled for collection and granule export$/) do
  @colls.each do |coll |
    @bmgtdb.enable_collection_granule_export(coll.collectionid).should be_truthy
  end
end

Given(/^criteria_190 Identify a logically deleted granule from collection C1 that can be exported$/) do
  rv = true
  @delgrans = @aimdb.find_deleted_granuleids_by_collectionid(@colls[0].collectionid)
  if(@delgrans.length < 1)
    rv = false
  else
    @g1 = @delgrans[0]
  end
  rv.should be_truthy
end

When(/^criteria_190 Request the manual export of metadata for a single granule, specifying the granule ID$/) do
  @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
  rv = Sdps::Bmgt::ExportRequest.new(@mode).manual_export_for_granules?(@g1, nil)
  rv.should be_truthy
end

Then(/^criteria_190 Verify the granule g1 is exported$/) do
  is_gran_request_successful?(@g1).should be_truthy
end

Then(/^criteria_190 Verify that the TCP proxy shows an HTTP DELETE request granule g1$/) do
  @xmlUtil.http_delete_for_granule?(@proxy_log_location, @request_time, @g1).should be_truthy
end
```

```
Then(/^criteria_190 Identify 2 Logically delete granules\(g2, g3\) or logically delete 2 granules from
collection C2$/) do
  @delgrans = @aimdb.find_deleted_granuleids_by_collectionid(@colls[1].collectionid)
  rv = true
  if(@delgrans.length < 2)
        rv = false
  else
    @g2 = @delgrans[0]
    @g3 = @delgrans[1]
  end
  rv.should be_truthy
end

Then(/^criteria_190 Manually Export granules in Collection C2$/) do
  @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
  rv = Sdps::Bmgt::ExportRequest.new(@mode).manual_granule_export_for_collection?(@colls[1].collectionid, nil)
  rv.should be_truthy
end

Then(/^criteria_190 Verify that the TCP proxy log shows an HTTP PUT request for each granule in Collection C2
except for logically deleted granules$/) do
  @granules = @aimdb.find_notdeleted_granuleids_by_collectionid(@colls[1].collectionid)
  @granules.each do |gran|
    is_gran_request_successful?(gran).should be_truthy
    @xmlUtil.http_put_for_granule?(@proxy_log_location, @request_time, gran).should be_truthy
  end
end

Then(/^criteria_190 Verify that the TCP proxy shows that the request contains the full granule metadata for each
granule in Collection C2 except for logically deleted granules$/) do
  @granules.each do |gran|
    gran_xml = @xmlUtil.get_granule_xml(@proxy_log_location, @request_time, gran)
    @xmlUtil.granule_xml_valid?(gran_xml).should be_truthy
  end
end
```

```
Then(/^criteria_190 Verify that the TCP proxy log shows an HTTP DELETE is exported for each of granules g2 and
g3$/) do
  is_gran_request_successful?(@g2).should be_truthy
  @xmlUtil.http_delete_for_granule?(@proxy_log_location, @request_time, @g2).should be_truthy
  is_gran_request_successful?(@g3).should be_truthy
  @xmlUtil.http_delete_for_granule?(@proxy_log_location, @request_time, @g3).should be_truthy
end

Then(/^criteria_190 Verify that the TCP proxy log shows an HTTP DELETE is also exported for any other logically
deleted granules in the collection$/) do
  @delgrans = @aimdb.find_deleted_granuleids_by_collectionid(@colls[1].collectionid)
  @delgrans.each do |gran|
    @xmlUtil.http_delete_for_granule?(@proxy_log_location, @request_time, gran).should be_truthy
  end
end

Then(/^criteria_190 Identify granule g4 in Collection C3 as a normal science granule that can be exported$/) do
  granules = @aimdb.find_notdeleted_granuleids_by_collectionid(@colls[2].collectionid)
  rv = true
  if(@delgrans.length < 1)
    rv = false
  else
    @g4 = granules[0]
  end
  rv.should be_truthy
end

Then(/^criteria_190 Identify granule g5 that is logically deleted in Collection C3$/) do
  @delgrans = @aimdb.find_deleted_granuleids_by_collectionid(@colls[2].collectionid)
  rv = true
  if(@delgrans.length < 1)
    rv = false
  else
    @g5 = @delgrans[0]
  end
  rv.should be_truthy
```

```
end

Then(/^criteria_190 Identify granule g6 which is physically deleted in Collection C3 or make up a nonexistent
granuleid in Collection C3$/) do
  @g6id = '12345'
  @g6 = "SC:#{@colls[2].shortname}.#{@colls[2].versionid}:#{@g6id}"
end

Then(/^criteria_190 Request, in a single manual operation, specifying the export of delete\-only for g4,g5,g6$/)
do
  @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
  granules = "'#{@g4}, #{@g5}, #{@g6}'"
  options = Array.new
  options.push('-del')
  rv = Sdps::Bmgt::ExportRequest.new(@mode).manual_export_for_granules?(granules, options)
  rv.should be_truthy
end

Then(/^criteria_190 Verify that the TCP proxy log shows No Export for granule g4$/) do
  @xmlUtil.http_put_for_granule?(@proxy_log_location, @request_time, @g4).should be_falsey
  @xmlUtil.http_delete_for_granule?(@proxy_log_location, @request_time, @g4).should be_falsey
end

Then(/^criteria_190 Verify that the TCP proxy log shows an HTTP DELETE is exported for granule g5$/) do
  is_gran_request_successful?(@g5).should be_truthy
  @xmlUtil.http_delete_for_granule?(@proxy_log_location, @request_time, @g5).should be_truthy
end

Then(/^criteria_190 Verify that the TCP proxy log shows an HTTP DELETE is exported for granule g6$/) do
  is_gran_request_successful?(@g6id).should be_truthy
  @xmlUtil.http_delete_for_granule?(@proxy_log_location, @request_time, @g6id).should be_truthy
end

def criteria_190_testdata_checked_out?

  @colls = Array.new
```

```
  @data = @sotestdata[@critinfo]
  @data.should_not
  cids = Array.new
  i = 0
  rv = true
  if @data.length != 3
    puts "There should be 3 collections returned for #{@critinfo}, check AutoTestSoTestDataInfo table"
    rv = false
  else
    @data.each do |esdt|
      shortname = @data[i].shortname
      versionid = @data[i].versionid
      i += 1

      @aimdb = Sdps::AimDb.new(@db_conn)
      coll = @aimdb.find_collection_by_shortname_versionid(shortname, versionid)

      if (coll == nil || !@aimdb.is_collection_installed?(shortname, versionid))
        puts "#{shortname}, #{versionid} is not installed"
        rv = false
      else
        #collection should have datamodeltype = 'ECS'
        if coll.datamodeltype != 'ECS'
          puts "collection's datamodeltype is not ECS"
          rv = false
        end
      end
      rv.should be_truthy
      @colls.push(coll)
    end
  end
  rv.should be_truthy
end
def is_gran_request_successful?(granid)
  begin
    rv = true;
```

```
    status = Sdps::Util.poll(180, 15){
      result = @bmgtdb.get_gran_export_request_status(@request_time,granid, 'MAN')
      if (result == 'PENDING' || result == 'STARTED')
        nil
      else
        result
      end
    }
    rv = false if (status != 'SUCCESS')

    return rv;
  end
end
```

Current Test Script: BE_82_01_40_V9.rb

```
Then(/^Verify that the metadata generated by the action in S-2 subclause h contains the QA values reflecting the
update which was performed$/) do
  pending # express the regexp above with the code you wish you had
end
```

Current Test Script: Ecs_Collection_Additional_Metadata_S2.rb

```
require 'sdps/sotestdata'
require 'sdps/aimdb'
require 'sdps/mode'
require 'sdps/db'
require 'sdps/util'
require 'sdps/bmgt'

Given(/^Criteria_20_2 Initial Setup$/) do
  #Check mode.
  @mode = ENV['MODE']
  @mode.should_not be_nil
  @mode_checker = Sdps::Mode.new(@mode)
  @mode_checker.mode_checked_out?().should be_truthy
  @proxy_log_location = @mode_checker.get_proxy_log_location()
  @generator_log_location = "/usr/ecs/#{@mode}/CUSTOM/logs/EcBmBMGTGenerator.log"
```

```
  @xmlUtil = Sdps::Bmgt::Xml.new()

  #set up db connection
  @db_subsystem = Sdps::Bmgt::DB_SUBSYSTEM
  @conn = Sdps::Db.connect(@db_subsystem, @mode)
  @conn.should_not be_nil

  #Make sure BMGT is up and running, proxy needs to be up too
  @mode_checker.bmgt_subsystem_checked_out?().should be_truthy, "bmgt system check failed"

  #Initialize critinfo and SoTestData
  @critinfo = "BE_82_01:20:2"
  @sotestdata = Sdps::SoTestData.new(@conn)
end

Given(/^Criteria_20_2 All test collections are installed in the mode$/) do
  @data = @sotestdata[@critinfo]
  @data.should_not be_nil

  rv = true
  if @data.length != 1
    puts "There should be one collection returned for #{@critinfo}, check AutoTestSoTestDataInfo table"
    rv = false
  else
    @shortname = @data[0].shortname
    @versionid = @data[0].versionid

    @aimdb = Sdps::AimDb.new(@conn)
    @collection =@aimdb.find_collection_by_shortname_versionid(@shortname,@versionid)

    if @collection == nil
      puts "test data #{@shortname}, #{@versionid} not installed"
      rv = false
    else
      #collection should have datamodeltype = 'ECS'
      if @collection.datamodeltype != 'ECS'
```

```
        puts "collection's datamodeltype is not ECS"
        rv = false
      end
    end
  end
  rv.should be_truthy
end

Given(/^Criteria_20_2 All test collections are configured for collection and granule metadata export$/) do
  @collectionid = @collection.collectionid
  @bmgtdb = Sdps::Bmgt::BmgtDb.new(@conn)

  if @bmgtdb.collection_export_enabled?(@collectionid) == false
    @bmgtdb.enable_collection_granule_export().should be_truthy
  end
end

Given(/^Criteria_20_2 A local copy of the collection schema is available to validate against$/) do
 puts "In the case of Criteria_20_2, no xml should be exported, therefore no need for the schema file"
end

Given(/^Criteria_20_2 A TCP proxy is capturing BMGT traffic$/) do
 #This is checked at the initial setup
 puts "If it gets here, the TCP proxy is up and running"

end

When(/^Criteria_20_2 Update the BMGT collection configuration table, setting the value of the S-2 collection's
GranuleSpatialRepresentation to an invalid value$/) do
  #There's only one collection in this test

  begin
    #note, collectionCoordinateSystem must be valid,otherwise, the error would be on invalid
collectionCoordinateSystem
    rs = @conn.exec ("update bg_collection_configuration set GranuleSpatialRepresentation='NONEUCLIDEAN',
collectionCoordinateSystem = null  where ShortName='#{@shortname}' and versionid = '#{@versionid}' returning 1")
```

```
    if rs != nil and rs.count == 1
      rv = true
    else
      rv = false
    end
  rescue => e
    puts e.backtrace
    rv = false
  end

  rv.should be_truthy
end

Then(/^Criteria_20_2 Verify that it succeeds$/) do

  puts "If it gets here, the update successful is verified"

end

When(/^Criteria_20_2 Request a Manual Export of the S-2 collection$/) do
  #record the request time
  @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
  Sdps::Bmgt::ExportRequest.new(@mode).manual_export_for_collection?(@collection.collectionid, nil).should_not
be_falsey, "Falied running Manual Export for collection"

  ##make sure the request is queued
  Sdps::Bmgt::BmgtDb.new(@conn).get_coll_export_request_status(@request_time, @collection.collectionid,
"MAN").should_not be_nil
end

Then(/^Criteria_20_2 Verify the manual export fails$/) do
  #check the EcBmBMGTManualDriver.log? key word?
  #actually the manual run log just queues the request, the actual run error appears in the generator log
  #but first have to wait till the request is processed
  #poll and wait
```

```
  export_req = Sdps::Bmgt::BmgtDb.new(@conn)
  status = Sdps::Util.poll(180, 15){
    result = export_req.get_coll_export_request_status(@request_time, @collection.collectionid, "MAN")
    if (result == 'PENDING' || result == 'STARTED')
      nil
    else
      result
    end
  }
  status.should eq 'SKIPPED'
end

Then(/^Criteria_20_2 Verify the log file records the export failure and the reason for failure$/) do
  #Check the generator log, after @request_time
  founderror = false
  open (@generator_log_location) do |log|
    # move the file pointer to the line after the timestamp
    log.find{|line| line =~ /^\d{4}-\d\d-\d\d \d\d:\d\d/ and line > @request_time}
    #find all the lines with ERROR for the collection esdt
    len =  log.grep(/ERROR.*#{@collection.esdt}.*Invalid value for ECS granuleSpatialRepresentation:
NONEUCLIDEAN/).map{|line| line.chomp}.length
    if len > 0
      founderror = true
    end
  end
  founderror.should be_truthy
end

Then(/^Criteria_20_2 Verify the log file indicates the invalid GranuleSpatialRepresentation value caused the
export failure\.$/) do
  puts "If it gets here, the failure msg is verified"
end

Then(/^Criteria_20_2 Verify that all Collection metadata in the previous steps validates against
https:\/\/api\.echo\.nasa\.gov\/ingest\/schema\/Collection\.xsd$/) do
  puts "In the case of Criteria_20_2, there is no xml file exported, therefore nothing to validate against the
```

```
schema file"
end

Then(/^Criteria_20_2 Verify that no collection metadata should have been exported$/) do
  #there is no Put request in the proxy log?
  dataset =  @collection.datasetid()
  @xmlUtil.http_put_for_collection?(@proxy_log_location, @request_time, dataset).should be_falsey
end

Current Test Script: BE_82_01_40_S2j.rb

  When(/^Unpublish a granule in the datapool$/) do
    granuleid = @test_granule.granuleid
    @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
    @export_queue = 'EVENT'
    @poll_wait_time = 300 #need to get this value from the configuration property table
    @publish_wait_time = 300 # from config

    # get hostname for unpublish
    publisher = Sdps::Publishing.new(@mode)
    publisher.should_not be_nil
    publisher.unpublish_granule(granuleid).should be_truthy

    #check if granule is unpublished
     if @aimdb.nil?
       @aimdb = Sdps::AimDb.new(@db_conn)
     end
     status = Sdps::Util.poll(@publish_wait_time, 5){
         if @aimdb.granule_published?(granuleid) == false
           true
         else
           nil
         end
     }
     status.should be_truthy, "granule #{granuleid} not unpublished"
  end
```

```
Current Test Script: Longform_Granule_Export.rb
Given(/^criteria_410 mode is checked out$/) do
  @mode = ENV['MODE']
  @mode.should_not be_nil
  #host should be checked to be f5oml01v
  @mode_checker = Sdps::Mode.new(@mode)
  @mode_checker.mode_checked_out?().should be_truthy
  @db_subsystem = Sdps::Bmgt::DB_SUBSYSTEM
  @db_conn = Sdps::Db.connect(@db_subsystem, @mode)
  @db_conn.should_not be_nil

  @xmlUtil = Sdps::Bmgt::Xml.new()
  @bmgtdb = Sdps::Bmgt::BmgtDb.new(@db_conn)

  #Initialize critinfo and SoTestData
  @critinfo = "BE_82_01:410"
  @sotestdata = Sdps::SoTestData.new(@db_conn)

  @proxy_log_location = "/tools/common/test/BE_82_01/proxy/#{@mode}/logs/tcp.log"

  @c2granules = Array.new
end

Given(/^criteria_410 bmgt subsystem is checked out$/) do
  @mode_checker.bmgt_subsystem_checked_out?().should be_truthy, "bmgt system check failed"
end

Given(/^criteria_410 testdata is checked out$/) do
  criteria_410_testdata_checked_out?().should be_truthy
end

Given(/^criteria_410 collections are installed$/) do
   @colls.each do |coll |
    @aimdb.is_collection_installed?(coll.shortname, coll.versionid).should be_truthy,
"#{coll.shortname}.#{coll.versionid} for Criteria 200 not installed."
  end
```

```
end

Given(/^criteria_410 collections are enabled for collection and granule export$/) do
  @colls.each do |coll |
    @bmgtdb.enable_collection_granule_export(coll.collectionid).should be_truthy
  end
end

Given(/^criteria_410 collections were exported to ECHO$/) do
  true
end

Given(/^criteria_410 Identify a granule g1 in Collection C1 that can be exported\.$/) do
  granules = @aimdb.find_notdeleted_granuleids_by_collectionid(@colls[0].collectionid)
  granules.should_not be_nil, "no granules for test collection #{@colls[0].esdt}"
  granules.should_not be_empty , "No granules for test colelction #{@colls[0].esdt}"
  @g1 = granules[0]
end

When(/^criteria_410 Request the long form verification of granule metadata for a single granule g1$/) do
  @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
  options = Array.new
  options.push('-vl')
  rv = Sdps::Bmgt::ExportRequest.new(@mode).manual_export_for_granules?(@g1, options)
  rv.should be_truthy
end

Then(/^criteria_410 Verify the granule g1 is exported$/) do
  is_qgran_request_successful?(@g1, 'VER').should be_truthy
end

Then(/^criteria_410 Verify that the TCP proxy shows an HTTP PUT request for granule g1$/) do
  @xmlUtil.http_put_for_granule?(@proxy_log_location, @request_time, @g1).should be_truthy
end

Then(/^criteria_410 Verify that the TCP proxy shows full granule metadata for granule g1$/) do
```

```
  @g1_xml = @xmlUtil.get_granule_xml(@proxy_log_location, @request_time, @g1)
  @xmlUtil.granule_xml_valid?(@g1_xml).should be_truthy
end

Then(/^criteria_410 Verify that the TCP proxy log shows that the request contains a query parameter
xml_diff=true, indicating it is for verification purposes$/) do
  @xmlUtil.http_verification_put_for_granule?(@proxy_log_location, @request_time, @g1).should be_truthy
end

Then(/^criteria_410 Identify granules g2, g3 in Collection C2 that are linked to browse granules br2, br3$/) do
  puts "Collectionid for retrieving browse #{@colls[1].collectionid}"
  brgrans = @aimdb.get_granules_with_browse(@colls[1].collectionid, 2)
  rv = true
  if(brgrans.length == 2)
    @g2 = brgrans[0]
    @g3 = brgrans[1]
  else
     rv = false
  end
  rv.should be_truthy
end

Then(/^criteria_410 Identify granules g4, g5 in Collection C2 which are public science granules$/) do
  rv = true;
  grans = @aimdb.get_public_granules(@colls[1].collectionid)
  @pgrans = Array.new
  grans.each do |gran|
    @pgrans.push(gran) if (gran != @g2 && gran != @g3)
  end
  puts "Public grans after : "
  puts @pgrans.inspect
  if(@pgrans.length < 4)
    rv =  false
  else
    @g4 = @pgrans[0]
    @g5 = @pgrans[1]
```

```
  end
  rv.should be_truthy
end

Then(/^criteria_410 Identify granules g6, g7 in Collection C2 which can be used to set the restriction flag$/)
do
  @g6 = @pgrans[2]
  @g7 = @pgrans[3]

  @aimdb.add_granule_restriction(@g6).should be_truthy
  @aimdb.add_granule_restriction(@g7).should be_truthy
end

Then(/^criteria_410 Request the long form verification of granule metadata for all granules in specified
collection C2$/) do
  @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
  options = Array.new
  options.push('-vl')
  rv = Sdps::Bmgt::ExportRequest.new(@mode).manual_granule_export_for_collection?(@colls[1].collectionid,
options)
  rv.should be_truthy
end

Then(/^criteria_410 Verify that the TCP proxy log shows an HTTP PUT request for each of g2\-g7 granules$/) do
  @c2granules.push(@g2)
  @c2granules.push(@g3)
  @c2granules.push(@g4)
  @c2granules.push(@g5)
  @c2granules.push(@g6)
  @c2granules.push(@g7)

  @c2granules.each do |gran|
    is_qgran_request_successful?(gran, 'VER').should be_truthy
    @xmlUtil.http_put_for_granule?(@proxy_log_location, @request_time, gran).should be_truthy
  end
```

```
end

Then(/^criteria_410 Verify that the TCP proxy shows that the request contains the full granule metadata for each
of g2\-g7 granules$/) do
  @c2granules.each do |gran|
    puts "FULL #{gran}"
    gran_xml = @xmlUtil.get_granule_xml(@proxy_log_location, @request_time, gran)
    if(gran == @g2)
      @g2_xml = gran_xml
    end
    @xmlUtil.granule_xml_valid?(gran_xml).should be_truthy
  end
end

Then(/^criteria_410 Verify that the TCP proxy shows that each of the requests g2\-g7 contain a query parameter
xml_diff=true, indicating it is for verification purposes$/) do
  @c2granules.each do |gran|
    @xmlUtil.http_verification_put_for_granule?(@proxy_log_location, @request_time, gran).should be_truthy
  end
end

Then(/^criteria_410 Verify that the verification metadata for the granules g2,g3 contain URLs for Browse
links$/) do

  g2_xml = @xmlUtil.get_granule_xml(@proxy_log_location, @request_time, @g2)
  @xmlUtil.granule_xml_contains?(g2_xml,"//OnlineResource/Type", "<Type>BROWSE</Type>").should be_truthy,
"granule - #{@g2} xml does not contains OnlineResource of type BROWSE"

  g3_xml = @xmlUtil.get_granule_xml(@proxy_log_location, @request_time, @g3)
  @xmlUtil.granule_xml_contains?(g3_xml,"//OnlineResource/Type", "<Type>BROWSE</Type>").should be_truthy,
"granule - #{@g3} xml does not contains OnlineResource of type BROWSE"

end

Then(/^criteria_410 Verify that the verification metadata for the granules g4,g5 contain Science, Metadata, and
ancillary file \(if appropriate\) URLs for public granules$/) do
```

```
  check_for_online_resources?(@g4).should be_truthy
  check_for_online_resources?(@g5).should be_truthy
end

Then(/^criteria_410 Verify that the verification metadata for the granules g6,g7 contain Restriction flag for
restricted granules\.$/) do
  g6_xml = @xmlUtil.get_granule_xml(@proxy_log_location, @request_time, @g6)
  @xmlUtil.granule_xml_contains?(g6_xml,"//Granule/RestrictionFlag",
"<RestrictionFlag>255</RestrictionFlag>").should be_truthy, "granule - #{@g6} xml does not contain
RestrictionFlag = 255"
  g7_xml = @xmlUtil.get_granule_xml(@proxy_log_location, @request_time, @g7)
  @xmlUtil.granule_xml_contains?(g7_xml,"//Granule/RestrictionFlag",
"<RestrictionFlag>255</RestrictionFlag>").should be_truthy, "granule - #{@g7} xml does not contain
RestrictionFlag = 255"
end

Then(/^criteria_410 Manually Export 2 granules from g1\-g7 in Collection C1 and C2$/) do
  @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
  @granules = "'#{@g1}, #{@g2}'"
  rv = Sdps::Bmgt::ExportRequest.new(@mode).manual_export_for_granules?(@granules, nil)
  rv.should be_truthy
end

Then(/^criteria_410 Verify that the TCP proxy log shows an HTTP PUT request for g1,g2 granules$/) do
  is_qgran_request_successful?(@g1, 'MAN').should be_truthy
  @xmlUtil.http_put_for_granule?(@proxy_log_location, @request_time, @g1).should be_truthy
  @man_g1_xml = @xmlUtil.get_granule_xml(@proxy_log_location, @request_time, @g1)

  is_qgran_request_successful?(@g2, 'MAN').should be_truthy
  @xmlUtil.http_put_for_granule?(@proxy_log_location, @request_time, @g2).should be_truthy
  @man_g2_xml = @xmlUtil.get_granule_xml(@proxy_log_location, @request_time, @g2)
end

Then(/^criteria_410 Verify the metadata files from Verification\(S2\) and Manual\(S3\) Exports are identical for
each granule exported$/) do
  @g2_xml.eql?(@man_g2_xml).should be_truthy
```

```
  #@g1_xml.eql?(@man_g1_xml).should be_truthy
end

def criteria_410_testdata_checked_out?
  @colls = Array.new
  @data = @sotestdata[@critinfo]
  @data.should_not
  cids = Array.new
  i = 0
  rv = true
  if @data.length != 2
    puts "There should be 2 collections returned for #{@critinfo}, check AutoTestSoTestDataInfo table"
    rv = false
  else
    @data.each do |esdt|
      shortname = @data[i].shortname
      versionid = @data[i].versionid

      puts "Collections #{shortname}, #{versionid}"
      i += 1

      @aimdb = Sdps::AimDb.new(@db_conn)
      coll = @aimdb.find_collection_by_shortname_versionid(shortname, versionid)

      if (coll == nil || !@aimdb.is_collection_installed?(shortname, versionid))
        puts "#{shortname}, #{versionid} is not installed"
        rv = false
      else
        #collection should have datamodeltype = 'ECS'
        if coll.datamodeltype != 'ECS'
          puts "collection's datamodeltype is not ECS"
          rv = false
        end
      end
      rv.should be_truthy
      @colls.push(coll)
```

```
    end
  end
  rv.should be_truthy
end

def check_for_online_resources?(granuleid)
  urlbase = @bmgtdb.get_gran_url_base()
  granule_xml = @xmlUtil.get_granule_xml(@proxy_log_location, @request_time, granuleid)
  onlinefilename = @aimdb.get_online_filename(granuleid)
  onlinefilename.should_not be_nil
  @onlineaccessurl = urlbase + onlinefilename
  @xmlUtil.granule_xml_contains?(granule_xml,"//OnlineAccessURL/URL", "<URL>#{@onlineaccessurl}</URL>").should
be_truthy, "granule - #{granuleid} xml does not contain OnlineAccessURL = #{@onlineaccessurl}"

  onlinemetadatafilename = @aimdb.get_online_metadata_filename(granuleid)
  onlinemetadatafilename.should_not be_nil
  @onlineresourceurl = urlbase + onlinemetadatafilename
  @xmlUtil.granule_xml_contains?(granule_xml,"//OnlineResource/URL", "<URL>#{@onlineresourceurl}</URL>").should
be_truthy, "granule - #{granuleid} xml does not contain OnlineResource = #{@onlineresourceurl}"
end

def is_qgran_request_successful?(granid, queue)
  begin
    rv = true;
    status = Sdps::Util.poll(180, 15){
      result = @bmgtdb.get_gran_export_request_status(@request_time,granid, queue)
      if (result == 'PENDING' || result == 'STARTED')
        nil
      else
        result
      end
    }
    rv = false if (status != 'SUCCESS' && status != 'WARNING')

    return rv;
  end
```

```
end
Current Test Script: Manual_Collection_Full_Update.rb
Given(/^criteria_170 mode is checked out$/) do
  @mode = ENV['MODE']
  @mode.should_not be_nil
  #host should be checked to be f5oml01v
  @mode_checker = Sdps::Mode.new(@mode)
  @mode_checker.mode_checked_out?().should be_truthy
  @db_subsystem = Sdps::Bmgt::DB_SUBSYSTEM
  @db_conn = Sdps::Db.connect(@db_subsystem, @mode)
  @db_conn.should_not be_nil

  @xmlUtil = Sdps::Bmgt::Xml.new()
  @bmgtdb = Sdps::Bmgt::BmgtDb.new(@db_conn)

  #Initialize critinfo and SoTestData
  @critinfo = "BE_82_01:170"
  @sotestdata = Sdps::SoTestData.new(@db_conn)

  @proxy_log_location = "/tools/common/test/BE_82_01/proxy/#{@mode}/logs/tcp.log"
end

Given(/^criteria_170 testdata is checked out$/) do
  criteria_170_testdata_checked_out?().should be_truthy
end

Given(/^criteria_170 bmgt subsystem is checked out$/) do
  @mode_checker.bmgt_subsystem_checked_out?().should be_truthy, "bmgt system check failed"
end

Given(/^criteria_170 collection is installed$/) do
  @aimdb.is_collection_installed?(@coll.shortname, @coll.versionid).should be_truthy, "Collection for 170 not
installed."
end

Given(/^criteria_170  collection is enabled for collection and granule export\.$/) do
```

```
  if (!@bmgtdb.collection_export_enabled?(@coll.collectionid))
    @bmgtdb.enable_collection_granule_export(@coll.collectionid).should be_truthy
  end

end

When(/^criteria_170 Request a full collection update for collection C1\.$/) do

  @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
  manual_full_coll_update_export?(@coll.collectionid).should_not be_falsey
end

Then(/^criteria_170 Verify that the request for C1 is queued and completed$/) do
  #poll and wait

  status = Sdps::Util.poll(180, 15){
    result = @bmgtdb.get_coll_export_request_status(@request_time, @coll.collectionid, 'MAN')
    if (result == 'PENDING' || result == 'STARTED')
      nil
    else
      result
    end
  }
  status.should eq 'SUCCESS'
end

Then(/^criteria_170 Verify the TCP proxy log reports a single HTTP DELETE request for collection C1$/) do
  dataset =  @coll.datasetid()
  @xmlUtil.http_delete_for_collection?(@proxy_log_location, @request_time, dataset).should be_truthy
end

Then(/^criteria_170 Verify that the TCP proxy log reports a single HTTP PUT request for collection C1$/) do
  status = Sdps::Util.poll(180, 15){
    result = @bmgtdb.get_coll_export_request_status(@request_time, @coll.collectionid, 'NEW')
    if (result == 'PENDING' || result == 'STARTED')
      nil
```

```
    else
      result
    end
  }
  status.should eq 'SUCCESS'
  dataset =  @coll.datasetid()
  @xmlUtil.http_put_for_collection?(@proxy_log_location, @request_time, dataset).should be_truthy
end

Then(/^criteria_170 Verify that the HTTP PUT request contains collection C1's full collection metadata$/) do
  @collection_xml = @xmlUtil.get_collection_xml(@proxy_log_location, @request_time, @coll)
  @xmlUtil.collection_xml_valid?(@collection_xml).should be_truthy
end

Then(/^criteria_170 Verify that the TCP proxy shows an HTTP PUT request for each granule in Collection C1$/) do

  statushash = @bmgtdb.get_gran_export_request_statushash(@request_time, @coll.collectionid, 'NEW')
  @granules = statushash.keys
  puts "Total granules retrieved :  #{@granules.length}"

  requests_processed_successfully?(@granules).should be_truthy

  @granules.each do |granid|
    puts "HTTP put for granule id #{granid}"
    @xmlUtil.http_put_for_granule?(@proxy_log_location, @request_time, granid).should be_truthy
  end
end

Then(/^Verify that the TCP proxy log shows that each request has complete granule metadata for each granule in
Collection C1$/) do

  @granules.each do |granid|
    gran_xml = @xmlUtil.get_granule_xml(@proxy_log_location, @request_time, granid)
    @xmlUtil.granule_xml_valid?(gran_xml).should be_truthy
  end
end
```

```
def criteria_170_testdata_checked_out?

  @data = @sotestdata[@critinfo]
  @data.should_not be_nil

  rv = true
  if @data.length != 1
    puts "There should be one collections returned for #{@critinfo}, check AutoTestSoTestDataInfo table"
    rv = false
  else
    @shortname = @data[0].shortname
    @versionid = @data[0].versionid

    @aimdb = Sdps::AimDb.new(@db_conn)
    @coll = @aimdb.find_collection_by_shortname_versionid(@shortname, @versionid)


    if (@coll == nil || !@aimdb.is_collection_installed?(@shortname, @versionid))
      puts "#{@shortname}, #{@versionid} is not installed"
      rv = false
    else
      #collection should have datamodeltype = 'ECS'
      if @coll.datamodeltype != 'ECS'
        puts "#{@shortname}, #{@versionid}'s datamodeltype is not ECS"
        rv = false
      end
    end
  end
  rv.should be_truthy
end


def manual_full_coll_update_export?(collectionid)
  begin
    request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
```

```
    #options = Array.new
    #options.push('-collupd')
    #.new(@mode).manual_export_for_collection?(collectionid, options).should be_truthy, "Falied running Manual
Full Collection update Export for collection"
    cdir = "/usr/ecs/#{@mode}/CUSTOM/utilities"
    Dir.chdir("#{cdir}") do
      status = system("./EcBmBMGTManualStart #{@mode} -collupd -c #{collectionid}")
    end
    #check if request was queued

    if @bmgtdb.get_coll_export_request_status(request_time, collectionid, 'MAN') == nil
      false
    else
      true
    end
  rescue =>e
    puts e.backtrace
    false
  end
end

def requests_processed_successfully?(granules)
  begin
    rv = true;
    granules.each do|granid|
      status = Sdps::Util.poll(180, 15){
        result = @bmgtdb.get_gran_export_request_status(@request_time, granid, 'NEW')
        if (result == 'PENDING' || result == 'STARTED')
          nil
        else
          result
        end
      }
      rv = false if (status != 'SUCCESS')
    end
    return rv;
```

```
  end
end
Current Test Script: ManualExport_LastUpdate.rb
Given(/^criteria_210 mode is checked out$/) do
  @mode = ENV['MODE']
  @mode.should_not be_nil
  #host should be checked to be f5oml01v
  @mode_checker = Sdps::Mode.new(@mode)
  @mode_checker.mode_checked_out?().should be_truthy
  @db_subsystem = Sdps::Bmgt::DB_SUBSYSTEM
  @db_conn = Sdps::Db.connect(@db_subsystem, @mode)
  @db_conn.should_not be_nil

  @xmlUtil = Sdps::Bmgt::Xml.new()
  @bmgtdb = Sdps::Bmgt::BmgtDb.new(@db_conn)

  #Initialize critinfo and SoTestData
  @critinfo = "BE_82_01:200"
  @sotestdata = Sdps::SoTestData.new(@db_conn)

  @proxy_log_location = "/tools/common/test/BE_82_01/proxy/#{@mode}/logs/tcp.log"
  @manual_log = "/usr/ecs/#{@mode}/CUSTOM/logs/EcBmBMGTManualDriver.log"
end

Given(/^criteria_210 bmgt subsystem is checked out$/) do
  @mode_checker.bmgt_subsystem_checked_out?().should be_truthy, "bmgt system check failed"
end

Given(/^criteria_210 testdata is checked out$/) do
  criteria_210_testdata_checked_out?().should be_truthy
end

Given(/^criteria_210 collections are installed$/) do
  @colls.each do |coll |
    @aimdb.is_collection_installed?(coll.shortname, coll.versionid).should be_truthy,
"#{coll.shortname}.#{coll.versionid} for Criteria 210 not installed."
```

```
  end
end

Given(/^criteria_210  collections are enabled for collection and granule export$/) do
  @colls.each do |coll |
    @bmgtdb.enable_collection_granule_export(coll.collectionid).should be_truthy
  end
end

Given(/^criteria_210 collections have few granules ingested$/) do
  check_for_granules?().should be_truthy
end

Given(/^criteria_210 Identify two granules within a collection and their lastupdate times$/) do
  identify_criteria210_granules()
end

When(/^criteria_210 Request export of granule metadata for collection based on lastupdate times$/) do
  options  = Array.new
  options.push('-c')
  options.push(@colls[0].collectionid)
  options.push('-lastupdate')

  time_based_manual_export?('-metg', options).should be_truthy
  gran_requests_processed_successfully?().should be_truthy
end

Then(/^criteria_210 Verify the granule at starttime is exported but not the one with the end time$/) do
  @bmgtdb.get_gran_export_request_status(@request_time, @granules[0], 'MAN').should_not be_nil
  @bmgtdb.get_gran_export_request_status(@request_time, @granules[3], 'MAN').should be_nil
end

Then(/^criteria_210 Verify that the TCP proxy shows an HTTP PUT\/DELETE request each for granules g1, g2, g3$/)
do
   i = 0;
  until i > 2 do
```

```
    if(@aimdb.granule_deleted?(@granules[i]))
      @xmlUtil.http_delete_for_granule?(@proxy_log_location, @request_time, @granules[i]).should be_truthy
    else
      @xmlUtil.http_put_for_granule?(@proxy_log_location, @request_time, @granules[i]).should be_truthy
    end
    i += 1
  end
end

Then(/^criteria_210 Verify that the TCP proxy does not show any request for granule g4$/) do
  @xmlUtil.http_put_for_granule?(@proxy_log_location, @request_time, @granules[3]).should be_falsey
  @xmlUtil.http_delete_for_granule?(@proxy_log_location, @request_time, @granules[3]).should be_falsey
end

Then(/^criteria_210 Verify that the TCP proxy shows full granule metadata for each of the non deleted granules
g1, g2, g3$/) do
  i = 0;
  until i > 2 do
    if(!@aimdb.granule_deleted?(@granules[i]))
      granule_xml = @xmlUtil.get_granule_xml(@proxy_log_location, @request_time, @granules[i])
      @xmlUtil.granule_xml_valid?(granule_xml).should be_truthy
    end
    i += 1
  end
end

Then(/^criteria_210 Identify two collections and their lastupdate times$/) do
  @starttime = @colls[0].lastupdate.strftime("%Y-%m-%d %H:%M:%S")
  @endtime = @colls[3].lastupdate.strftime("%Y-%m-%d %H:%M:%S")
end

Then(/^criteria_210 Request export of collection metadata based on lastupdate times$/) do
  options  = Array.new

  options.push('-lastupdate')
  time_based_manual_export?('-metc',options).should be_truthy
```

```
  coll_requests_processed_successfully?().should be_truthy
end

Then(/^criteria_210 Verify the collection at starttime is exported but not the one with the end time$/) do
  @bmgtdb.get_coll_export_request_status(@request_time, @colls[0].collectionid, 'MAN').should_not be_nil
  @bmgtdb.get_coll_export_request_status(@request_time, @colls[3].collectionid, 'MAN').should be_nil
end

Then(/^criteria_210 Verify that the TCP proxy shows an HTTP PUT request each for collection C1, C2, C3$/) do
   i = 0;
  until i > 2 do
    dataset =  @colls[i].datasetid()
    @xmlUtil.http_put_for_collection?(@proxy_log_location, @request_time, dataset).should be_truthy
    i += 1
  end
end

Then(/^criteria_210 Verify that the TCP proxy does not show any request for collection C4$/) do
  dataset =  @colls[3].datasetid()
  @xmlUtil.http_put_for_collection?(@proxy_log_location, @request_time, dataset).should be_falsey
  @xmlUtil.http_delete_for_collection?(@proxy_log_location, @request_time, dataset).should be_falsey
end

Then(/^criteria_210 Verify that the TCP proxy shows full collection metadata for each of the collection C1, C2,
C3$/) do
  i = 0;
  until i > 2 do
    collection_xml = @xmlUtil.get_collection_xml(@proxy_log_location, @request_time, @colls[i])
    @xmlUtil.collection_xml_valid?(collection_xml).should be_truthy
    i += 1
  end
end

def criteria_210_testdata_checked_out?

  @colls = Array.new
```

```
  @data = @sotestdata[@critinfo]
  @data.should_not be_nil
  i = 0
  rv = true
  if @data.length != 4
    puts "There should be 4 collections returned for #{@critinfo}, check AutoTestSoTestDataInfo table"
    rv = false
  else
    @data.each do |esdt|
      shortname = @data[i].shortname
      versionid = @data[i].versionid
      i = i + 1

      @aimdb = Sdps::AimDb.new(@db_conn)
      coll = @aimdb.find_collection_by_shortname_versionid(shortname, versionid)

      if (coll == nil || !@aimdb.is_collection_installed?(shortname, versionid))
        puts "#{shortname}, #{versionid} is not installed"
        rv = false
      else
        #collection should have datamodeltype = 'ECS'
        if coll.datamodeltype != 'ECS'
          puts "collection's datamodeltype is not ECS"
          rv = false
        end
      end
      rv.should be_truthy
      @colls.push(coll)
    end
    @colls.sort_by!(&:lastupdate)
  end
  rv.should be_truthy
end

def identify_criteria210_granules()
  @granules = @bmgtdb.get_granules_by_time(@colls[0].collectionid, 'lastupdate', 4 )
```

```
  last  = @granules.length - 1
  @starttime = @aimdb.find_granule_by_granuleid(@granules[0]).lastupdate.strftime("%Y-%m-%d %H:%M:%S")
  @endtime = @aimdb.find_granule_by_granuleid(@granules[last]).lastupdate.strftime("%Y-%m-%d %H:%M:%S")
end

def check_for_granules?()
  rv = false
  granules = @aimdb.find_notdeleted_granuleids_by_collectionid(@colls[0].collectionid)
  if(granules.length > 0)
    rv = true;
  else
    rv = false;
  end

  return rv

end

def time_based_manual_export?(type, options)
  @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
  rv = Sdps::Bmgt::ExportRequest.new(@mode).manual_time_export?(type, @starttime, @endtime, options)
  rv.should be_truthy
  return rv
end


def coll_requests_processed_successfully?()
  begin
    rv = true;
    i = 0
    until i > 2 do
      status = Sdps::Util.poll(180, 15){
        result = @bmgtdb.get_coll_export_request_status(@request_time,@colls[i].collectionid, 'MAN')
        if (result == 'PENDING' || result == 'STARTED')
          nil
        else
```

```
          result
        end
      }
      rv = false if (status != 'SUCCESS')
      i += 1
    end
    return rv;
  end
end
def gran_requests_processed_successfully?()
  begin
    rv = true;
    i = 0
    until i > 2 do
      status = Sdps::Util.poll(180, 15){
        result = @bmgtdb.get_gran_export_request_status(@request_time,@granules[i], 'MAN')
        if (result == 'PENDING' || result == 'STARTED')
          nil
        else
          result
        end
      }
      rv = false if (status != 'SUCCESS')
      i += 1
    end
    return rv;
  end
end
Current Test Script: Ecs_Collection_Additional_Metadata_S1.rb
require 'sdps/sotestdata'
require 'sdps/aimdb'
require 'sdps/mode'
require 'sdps/db'
require 'sdps/util'
require 'sdps/bmgt'
```

```
Given(/^Criteria_20_1 Initial Setup$/) do
  #Check mode.
  @mode = ENV['MODE']
  @mode.should_not be_nil
  @mode_checker = Sdps::Mode.new(@mode)
  @mode_checker.mode_checked_out?().should be_truthy
  @proxy_log_location = @mode_checker.get_proxy_log_location()
  @generator_log_location = "/usr/ecs/#{@mode}/CUSTOM/logs/EcBmBMGTGenerator.log"
  @xmlUtil = Sdps::Bmgt::Xml.new()

  #set up db connection
  @db_subsystem = Sdps::Bmgt::DB_SUBSYSTEM
  @conn = Sdps::Db.connect(@db_subsystem, @mode)
  @conn.should_not be_nil

  #Make sure BMGT is up and running, proxy needs to be up too
  @mode_checker.bmgt_subsystem_checked_out?().should be_truthy, "bmgt system check failed"

  #Initialize critinfo and SoTestData
  @critinfo = "BE_82_01:20:1"
  @sotestdata = Sdps::SoTestData.new(@conn)
end

Given(/^Criteria_20_1 All test collections are installed in the mode$/) do
  @data = @sotestdata[@critinfo]
  @data.should_not be_nil

  rv = true
  if @data.length != 1
    puts "There should be one collection returned for #{@critinfo}, check AutoTestSoTestDataInfo table"
    rv = false
  else
    @shortname = @data[0].shortname
    @versionid = @data[0].versionid

    @aimdb = Sdps::AimDb.new(@conn)
```

```
    @collection =@aimdb.find_collection_by_shortname_versionid(@shortname,@versionid)

    if @collection == nil
      puts "test data #{@shortname}, #{@versionid} not installed"
      rv = false
    else
      #collection should have datamodeltype = 'ECS'
      if @collection.datamodeltype != 'ECS'
        puts "collection's datamodeltype is not ECS"
        rv = false
      end
    end
  end
  rv.should be_truthy
end

Given(/^Criteria_20_1 All test collections are configured for collection and granule metadata export$/) do
  @collectionid = @collection.collectionid
  @bmgtdb = Sdps::Bmgt::BmgtDb.new(@conn)

  if @bmgtdb.collection_export_enabled?(@collectionid) == false
    @bmgtdb.enable_collection_granule_export(@collectionid).should be_truthy
  end
end

Given(/^Criteria_20_1 A local copy of the collection schema is available to validate against$/) do
 puts "In the case of Criteria_20_1, no xml should be exported, therefore no need for the schema file"
end

Given(/^Criteria_20_1 A TCP proxy is capturing BMGT traffic$/) do
 #This is checked at the initial setup
 puts "If it gets here, the TCP proxy is up and running"

end

When(/^Criteria_20_1 Update the BMGT collection configuration table, setting the value of the S-1 collection's
```

```
CoordinateSystem to an invalid value$/) do
  #There's only one collection in this test

  begin
    rs = @conn.exec ("update bg_collection_configuration set CollectionCoordinateSystem='POLAR' where
ShortName='#{@shortname}' and versionid = '#{@versionid}' returning 1")

    if rs != nil and rs.count == 1
      rv = true
    else
      rv = false
    end
  rescue => e
    puts e.backtrace
    rv = false
  end

  rv.should be_truthy
end

Then(/^Criteria_20_1 Verify that it succeeds$/) do

  puts "If it gets here, the update successful is verified"

end

When(/^Criteria_20_1 Request a Manual Export of the S-1 collection$/) do
  #record the request time
  @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
  Sdps::Bmgt::ExportRequest.new(@mode).manual_export_for_collection?(@collection.collectionid, nil).should
be_truthy, "Falied running Manual Export for collection"

  ##make sure the request is queued
  Sdps::Bmgt::BmgtDb.new(@conn).get_coll_export_request_status(@request_time, @collection.collectionid,
"MAN").should_not be_nil, "Failed get_coll_export_request_status"
end
```

```
Then(/^Criteria_20_1 Verify the manual export fails$/) do
  #check the EcBmBMGTManualDriver.log? key word?
  #actually the manual run log just queues the request, the actual run error appears in the generator log
  #but first have to wait till the request is processed
  #poll and wait
  export_req = Sdps::Bmgt::BmgtDb.new(@conn)
  status = Sdps::Util.poll(180, 15){
    result = export_req.get_coll_export_request_status(@request_time, @collection.collectionid, "MAN")
    if (result == 'PENDING' || result == 'STARTED')
      nil
    else
      result
    end
  }
  status.should eq 'SKIPPED'
end

Then(/^Criteria_20_1 Verify the log file records the export failure$/) do
  #Check the generator log, after @request_time
  founderror = false
  #puts @generator_log_location
  open (@generator_log_location) do |log|
    # move the file pointer to the line after the timestamp
    log.find{|line| line =~ /^\d{4}-\d\d-\d\d \d\d:\d\d/ and line > @request_time}
    #find all the lines with ERROR for the collection esdt
    len =  log.grep(/ERROR.*#{@collection.esdt}.*Invalid value for ECS collectionCoordinateSystem:
POLAR/).map{|line| line.chomp}.length
    if len > 0
      founderror = true
    end
  end
  founderror.should be_truthy
end

Then(/^Criteria_20_1 Verify the log file indicates the invalid CoordinateSystem value caused the export
```

```
failure\.$/) do
  puts "If it gets here, the failure msg is verified"
end

Then(/^Criteria_20_1 Verify that all Collection metadata in the previous steps validates against
https:\/\/api\.echo\.nasa\.gov\/ingest\/schema\/Collection\.xsd$/) do
  puts "In the case of Criteria_20_1, there is no xml file exported, therefore nothing to validate against the
schema file"
end

Then(/^Criteria_20_1 Verify that no collection metadata should have been exported$/) do
  #there is no Put request in the proxy log?
  dataset =  @collection.datasetid()
  @xmlUtil.http_put_for_collection?(@proxy_log_location, @request_time, dataset).should be_falsey
end

Current Test Script: Ecs_Collection_Additional_Metadata_S12.rb
require 'sdps/sotestdata'
require 'sdps/aimdb'
require 'sdps/mode'
require 'sdps/db'
require 'sdps/util'
require 'sdps/bmgt'

Given(/^Criteria_20_12 Initial Setup$/) do
  #Check mode.
  @mode = ENV['MODE']
  @mode.should_not be_nil
  @mode_checker = Sdps::Mode.new(@mode)
  @mode_checker.mode_checked_out?().should be_truthy
  @proxy_log_location = @mode_checker.get_proxy_log_location()
  @generator_log_location = "/usr/ecs/#{@mode}/CUSTOM/logs/EcBmBMGTGenerator.log"
  @xmlUtil = Sdps::Bmgt::Xml.new()

  #set up db connection
  @db_subsystem = Sdps::Bmgt::DB_SUBSYSTEM
```

```
  @conn = Sdps::Db.connect(@db_subsystem, @mode)
  @conn.should_not be_nil

  #Make sure BMGT is up and running, proxy needs to be up too
  @mode_checker.bmgt_subsystem_checked_out?().should be_truthy, "bmgt system check failed"

  #Initialize critinfo and SoTestData
  @critinfo = "BE_82_01:20:12"
  @sotestdata = Sdps::SoTestData.new(@conn)
end

Given(/^Criteria_20_12 All test collections are installed in the mode$/) do
  @data = @sotestdata[@critinfo]
  @data.should_not be_nil

  rv = true
  if @data.length != 1
    puts "There should be one collection returned for #{@critinfo}, check AutoTestSoTestDataInfo table"
    rv = false
  else
    @shortname = @data[0].shortname
    @versionid = @data[0].versionid

    @aimdb = Sdps::AimDb.new(@conn)
    @collection =@aimdb.find_collection_by_shortname_versionid(@shortname,@versionid)

    if @collection == nil
      puts "test data #{@shortname}, #{@versionid} not installed"
      rv = false
    else
      #collection should have datamodeltype = 'ECS'
      if @collection.datamodeltype != 'ECS'
        puts "collection's datamodeltype is not ECS"
        rv = false
      end
    end
```

```
  end
  rv.should be_truthy
end

Given(/^Criteria_20_12 All test collections are configured for collection and granule metadata export$/) do
  @collectionid = @collection.collectionid
  @bmgtdb = Sdps::Bmgt::BmgtDb.new(@conn)

  if @bmgtdb.collection_export_enabled?(@collectionid) == false
    @bmgtdb.enable_collection_granule_export(@collectionid).should be_truthy
  end
end

Given(/^Criteria_20_12 A local copy of the collection schema is available to validate against$/) do
  #verify Sdps::Bmgt::ECHO10_COLLECTION_SCHEMA exists
  schemaFile = Sdps::Bmgt::ECHO10_COLLECTION_SCHEMA
  File.exists?(schemaFile).should be_truthy
end

Given(/^Criteria_20_12 A TCP proxy is capturing BMGT traffic$/) do
 #This is checked at the initial setup
 puts "If it gets here, the TCP proxy is up and running"
end

When(/^Criteria_20_12 S-12 Attempt to configure the CoordinateSystem for an ECS collection to null \(i\.e\.
automatically populate\)$/) do
  #There's only one collection in this test

  begin
    rs = @conn.exec ("update bg_collection_configuration set
CollectionCoordinateSystem=null,GranuleSpatialRepresentation=null where ShortName='#{@shortname}' and versionid
= '#{@versionid}' returning 1")

    if rs != nil and rs.count == 1
      rv = true
    else
```

```
      rv = false
    end
  rescue => e
    puts e.backtrace
    rv = false
  end

  rv.should be_truthy
end

When(/^Criteria_20_12 Request a Manual Export of this collection$/) do
  #record the request time
  @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
  Sdps::Bmgt::ExportRequest.new(@mode).manual_export_for_collection?(@collection.collectionid, nil).should
be_truthy, "Falied running Manual Export for collection"

  ##make sure the request is queued
  Sdps::Bmgt::BmgtDb.new(@conn).get_coll_export_request_status(@request_time, @collection.collectionid,
"MAN").should_not be_nil
end

Then(/^Criteria_20_12 Manual Export succeeds$/) do
  #poll and wait
  export_req = Sdps::Bmgt::BmgtDb.new(@conn)
  status = Sdps::Util.poll(180, 15){
    result = export_req.get_coll_export_request_status(@request_time, @collection.collectionid, "MAN")
    if (result == 'PENDING' || result == 'STARTED')
      nil
    else
      result
    end
  }
  status.should eq 'SUCCESS'
end

Then(/^Criteria_20_12 V-12 Verify that all Collection metadata in the previous steps validates against
```

```
https:\/\/api\.echo\.nasa\.gov\/ingest\/schema\/Collection\.xsd$/) do
  #Verify the S-12 collection's exported XML validates against the schema:
  #The following method makes use of the local copy of the collection.xsd file:
Sdps::Bmgt::ECHO10_COLLECTION_SCHEMA
  @collection_xml = @xmlUtil.get_collection_xml(@proxy_log_location, @request_time, @collection)
  #puts "xml is #{@collection_xml}"
  @xmlUtil.collection_xml_valid?(@collection_xml).should be_truthy
end

Then(/^Criteria_20_12 V-13 Verify that the collection metadata generated in S-12 contains the default
CoordinateSystem value$/) do
  #Query the BMGT configuration table to find the default CoordinateSystem value
  begin
    rs = @conn.exec ("select propertyvalue from  bg_configuration_property where propertyname =
'BMGT.Common.CoordinateSystemDefault'")

    if rs != nil and rs.count == 1
      @dfv = rs[0]['propertyvalue']
      rv = true
    else
      rv = false
    end
  rescue => e
    puts e.backtrace
    rv = false
  end
  rv.should be_truthy
  #puts "dfv is #{@dfv}"

  #Verify @collection_xml contains <CoordinateSystem>"#{@dfv}"</CoordinateSystem>
  @xmlUtil.collection_xml_contains?(@collection_xml,
"//Collection/Spatial/HorizontalSpatialDomain/Geometry/CoordinateSystem",
"<CoordinateSystem>#{@dfv}</CoordinateSystem>").should be_truthy
end
```

Current Test Script: BE_82_01_40_S2f.rb

```
When(/^Restrict a granule$/) do
   @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
   @export_queue = 'EVENT'
   @poll_wait_time = 300 #need to get this value from the configuration property table
   @restriction_flag = 6
   @description = "Testing BE_82_01 Crit 40 S2f"

   status = @aimdb.add_restriction(@restriction_flag, @description);
   status.should be_truthy
   status = @aimdb.restrict_granule(@test_granule.granuleid,@restriction_flag);
   status.should be_truthy
end
```

Current Test Script: BE_82_01_40_S2n.rb

```
When(/^Move a collection$/) do

   #get esdt for test collection
   #collection1 is the test collection
    @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
    @export_queue = 'EVENT'
    @poll_wait_time = 300 #need to get this value from the configuration property table
    @movecollection_wait_time = 300 # from config

    # get hostname for MoveCollectionUtility
    movecollectionutility = Sdps::MoveCollection.new(@mode)
    movecollectionutility.should_not be_nil
    #get all active filesystems
    filesystems = @aimdb.get_active_filesystems()
    #get current filesystem
    @current_filesystem = @collection1.filesystemlabel
    current_absfilesystem_path = @aimdb.get_absolutefilesystempath(@current_filesystem)
    puts current_absfilesystem_path
    @current_filesystem_path = @aimdb.get_filesystempath(@current_filesystem).gsub("/","")
    puts @current_filesystem_path
    #get the files in the current filesystem
    source_listing = movecollectionutility.get_files_in_filesystem(@collection1, current_absfilesystem_path)
```

```
    puts "source:"
    puts source_listing
    @target_filesystem = nil
    #pick a filesystem that is not the current filesystem
    filesystems.each { |filesystem|
          if filesystem != @current_filesystem
               @target_filesystem = filesystem
               break
         end
    }
    @target_filesystem.should_not be_nil
    puts "target:" + @target_filesystem

    target_absfilesystem_path = @aimdb.get_absolutefilesystempath(@target_filesystem)
    puts target_absfilesystem_path
    @target_filesystem_path = @aimdb.get_filesystempath(@target_filesystem).gsub("/","")
    puts @target_filesystem_path

    movecollectionutility.move_collection(@collection1, @target_filesystem).should be_truthy

    puts "done move collection"

    #check if collection was moved
    status = Sdps::Util.poll(@movecollection_wait_time, 5){
          new_collection1 = @aimdb.find_collection_by_collectionid(@collection1.collectionid)
          puts new_collection1.filesystemlabel
          if new_collection1.filesystemlabel == @target_filesystem
               #check the files moved
               puts "files moved"
               dest_listing = movecollectionutility.get_files_in_filesystem(new_collection1,
target_absfilesystem_path)
               #compare the two. folder name should have new filesystem . filenames should match
               source_folders = source_listing.select {|filename| filename.include?(":")}
               source_folders.map! { |folder| folder.gsub!(@current_filesystem_path, @target_filesystem_path)}
               dest_folders = dest_listing.select{|filename| filename.include?(":")}
               match = source_folders & dest_folders
```

```
            puts "source_folders:"
            puts source_folders
            puts "dest_folders:"
            puts dest_folders
            puts "folder match:"
            puts match
            if match.nil? or match.empty? or match.size < source_folders.size
                  "no folder match"
                  nil
            else
                #check files
                source_files = source_listing.delete_if {|filename| filename.include?(":")}
                dest_files = dest_listing.delete_if {|filename| filename.include?(":")}
                puts "source_files:"
                puts source_files
                puts "dest_files:"
                puts dest_files
                match = source_files & dest_files
                puts "file match:"
                puts match
                if match.nil? or match.empty? or match.size < source_files.size
                    puts "no file match"
                    nil
                else
                    "SUCCESS"
                end
            end
        else
            puts "collection not moved "
            nil
      end
    }
    status.should be_truthy, "move collection failed"

end
Current Test Script: AutoExport_Collection_Insert.rb
```

```
Given(/^criteria_10_S2_a mode is checked out$/) do
  @mode = ENV['MODE']
  @mode.should_not be_nil
  #host should be checked to be f5oml01v
  @mode_checker = Sdps::Mode.new(@mode)
  @mode_checker.mode_checked_out?().should be_truthy
  @db_subsystem = Sdps::Bmgt::DB_SUBSYSTEM
  @db_conn = Sdps::Db.connect(@db_subsystem, @mode)
  @db_conn.should_not be_nil

  @xmlUtil = Sdps::Bmgt::Xml.new()
  @bmgtdb = Sdps::Bmgt::BmgtDb.new(@db_conn)

  #Initialize critinfo and SoTestData
  @critinfo = "BE_82_01:10:2_A"
  @sotestdata = Sdps::SoTestData.new(@db_conn)

  @proxy_log_location = "/tools/common/test/BE_82_01/proxy/#{@mode}/logs/tcp.log"
end

Given(/^criteria_10_S2_a testdata is checked out$/) do
  criteria_10_s2_a_testdata_checked_out?().should be_truthy
end

Given(/^criteria_10_S2_a bmgt subsystem is checked out$/) do
  @mode_checker.bmgt_subsystem_checked_out?().should be_truthy, "bmgt system check failed"
  stop_auto_driver?().should be_truthy
end

Given(/^criteria_10_S2_a collection is not installed$/) do
 # @coll.installed.should be_false, "Collection for 10_S2_a is already installed."
 @coll = nil
 @coll.should be_nil

end
```

```
When(/^criteria_10_S2_a Insert a new collection into the ECS inventory$/) do
  install_collection()
end

When(/^criteria_10_S2_a insert collection into collection configuration$/) do

  insert_collection_into_collconfig(@aimcoll.collectionid)

  @coll = @aimdb.find_collection_by_shortname_versionid(@shortname, @versionid)

end

When(/^criteria_10_S2_a  enable the collection for collection export\.$/) do
  if(!@bmgtdb.collection_export_enabled?(@coll.collectionid))
    @bmgtdb.enable_collection_export(@coll.collectionid).should be_truthy
  end
  @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
  start_auto_driver?().should be_truthy

end

Then(/^criteria_10_S2_a Verify the bg_export_request table shows a successful export for collection C1$/) do

  #poll and wait
  status = Sdps::Util.poll(300, 30){
    result = @bmgtdb.get_coll_export_request_status(@request_time, @coll.collectionid, 'EVENT')
    if (result == 'PENDING' || result == 'STARTED')
      nil
    else
      result
    end
  }
  status.should eq 'SUCCESS'
end

Then(/^criteria_10_S2_a Verify that the TCP proxy log reports a single HTTP PUT request for collection C1$/) do
```

```
  dataset =  @coll.datasetid()
  @xmlUtil.http_put_for_collection?(@proxy_log_location, @request_time, dataset).should be_truthy
end

Then(/^criteria_10_S2_a Verify that the HTTP PUT request contains collection C1's full collection metadata\.$/)
do
  @collection_xml = @xmlUtil.get_collection_xml(@proxy_log_location, @request_time, @coll)
  @xmlUtil.collection_xml_valid?(@collection_xml).should be_truthy
end

Then(/^criteria_10_S2_a Verify that the metadata exported validates against the ECHO collection metadata schema
\(https:\/\/api\.echo\.nasa\.gov\/ingest\/schema\/Collection\.xsd\)$/) do
  @xmlUtil.collection_xml_valid?(@collection_xml).should be_truthy
end

Then(/^criteria_10_S2_a Verify that the metadata exported has the element Visible = true$/) do
  @xmlUtil.collection_xml_contains?(@collection_xml,"//Collection/Visible", "<Visible>true</Visible>").should
be_truthy
end

Then(/^criteria_10_S2_a Verify that the metadata exported has the element Orderable = false$/) do
  @xmlUtil.collection_xml_contains?(@collection_xml,"//Collection/Orderable",
"<Orderable>false</Orderable>").should be_truthy
end

Then(/^criteria_10_S2_a Verify that the metadata exported has the element InsertTime = The insert time of the
collection recorded in the AIM database\.$/) do

  insertTime = @coll.inserttime
  insertTime.should_not be_nil
  insertTime_in_db = DateTime.parse(insertTime.to_s).strftime('%Y-%m-%dT%H:%M:%S.%3NZ')
  @xmlUtil.collection_xml_contains?(@collection_xml, "//Collection/InsertTime",
"<InsertTime>#{insertTime_in_db}</InsertTime>").should be_truthy

end
```

```
Then(/^criteria_10_S2_a Verify that the metadata exported has the element LastUpdate = The last update time of
the collection recorded in the AIM database$/) do

  lastUpdate = @coll.lastupdate
  lastUpdate.should_not be_nil
  lastUpdate_in_db = DateTime.parse(lastUpdate.to_s).strftime('%Y-%m-%dT%H:%M:%S.%3NZ')

  @xmlUtil.collection_xml_contains?(@collection_xml, "//Collection/LastUpdate",
"<LastUpdate>#{lastUpdate_in_db}</LastUpdate>").should be_truthy

end

Then(/^criteria_10_S2_a Verify that the metadata exported contains version numbers with leading zeroes$/) do
  vId = "%03d" %  @coll.versionid
  @xmlUtil.collection_xml_contains?(@collection_xml, "//Collection/VersionId",
"<VersionId>#{vId}</VersionId>").should be_truthy
end

def criteria_10_s2_a_testdata_checked_out?

  @data = @sotestdata[@critinfo]
  @data.should_not be_nil

  rv = true
  if @data.length != 1
    puts "There should be one collection returned for #{@critinfo}, check AutoTestSoTestDataInfo table"
    rv = false
  else
    @shortname = @data[0].shortname
    @versionid = @data[0].versionid

    delete_from_coll_config(@shortname, @versionid)

    @aimdb = Sdps::AimDb.new(@db_conn)
    @coll = @aimdb.find_collection_by_shortname_versionid(@shortname, @versionid)
    if(@coll != nil || @aimdb.is_collection_installed?(@shortname, @versionid))
```

```
     if(@bmgtdb.collection_export_enabled?(@coll.collectionid))
         puts "#{@shortname}, #{@versionid} is installed"
        rv = false
     end

    end
  end
  rv.should be_truthy
end

  def stop_auto_driver?()
    begin
      cdir = "/usr/ecs/#{@mode}/CUSTOM/utilities"
      Dir.chdir("#{cdir}") do
        status = system("./EcBmBMGTAutoStop #{@mode} ")
        status.should be_truthy
      end
    rescue =>e
      puts e.backtrace
      false
    end
  end

  def install_collection()
    #TODO
    #install collection method from ESDT Maintenance GUI.
    @acoll = Sdps::AimDb.new(@db_conn)
    @aimcoll = @acoll.find_collection_by_shortname_versionid(@shortname, @versionid)

    insert_row_event_history(@aimcoll)

  end

  def insert_collection_into_collconfig(id)

    if(!collection_exists_in_collconfig?(id))
```

```
      @bmgtdb.insert_collection_into_collectionconfig(id)
    end
    @bmgtdb.enable_collection_export(id).should be_truthy
  end

  def start_auto_driver?()
    begin
      cdir = "/usr/ecs/#{@mode}/CUSTOM/utilities"
      Dir.chdir("#{cdir}") do
        status = system("./EcBmBMGTAutoStart #{@mode} ")
        status.should be_truthy
      end
    rescue =>e
      puts e.backtrace
      false
    end
  end

  def insert_row_event_history(coll)

    cid = coll.collectionid
    sname = coll.shortname
    vid = coll.versionid

    puts "cid, sname, vid  #{cid}, #{sname} #{vid}"
    rs = @db_conn.exec("insert into DsMdGrEventHistory (dbid, eventtime, eventid, eventType, shortname,
versionid, collectionid )
                select collectionId, now(), 31, 'CLINSERT', shortname, versionId, #{cid} from amcollection where
collectionId = #{cid}")
    if rs != nil and rs.count > 0
      true
    else
      false
    end
  rescue =>e
    puts e.backtrace
```

```
    false
  end

  def delete_from_coll_config(shortname, versionid)
    @db_conn.exec("delete from bg_collection_configuration where shortname = '#{shortname}' and versionid =
#{versionid}");
  end

  def collection_exists_in_collconfig?(id)

    rs = @db_conn.exec("select * from bg_collection_configuration where collectionId = #{id}")
    if rs != nil and rs.count > 0
      true
    else
      false
    end
  rescue =>e
    puts e.backtrace
    false

  end

Current Test Script: BE_82_01_40_V10.rb
Then(/^Verify that the metadata generated by the action in S-2 subclause i contains URLs for the science and
metadata files, as well as any other ancillary files like browse associated with the affected granule$/) do

    @test_granules_xml.keys.each do |granuleid|
      granule_xml = @test_granules_xml[granuleid]

      onlinefilename = @aimdb.get_online_filename(granuleid)
      puts onlinefilename
      onlinefilename.should_not be_nil
      @onlineaccessurl = "http://f4ftl01/" + onlinefilename
      puts "xml contains " + @xml_utils.granule_xml_getvalue(granule_xml,"//OnlineAccessURL/URL")
      @xml_utils.granule_xml_contains?(granule_xml,"//OnlineAccessURL/URL",
"<URL>#{@onlineaccessurl}</URL>").should be_truthy, "granule - #{granuleid} xml does not contain OnlineAccessURL
```

```
= #{@onlineaccessurl}"


      onlinemetadatafilename = @aimdb.get_online_metadata_filename(granuleid)
      puts onlinemetadatafilename
      onlinemetadatafilename.should_not be_nil
      @onlineresourceurl = "http://f4ftl01/" + onlinemetadatafilename
      puts "xml contains " + @xml_utils.granule_xml_getvalue(granule_xml,"//OnlineResource/URL")
      @xml_utils.granule_xml_contains?(granule_xml,"//OnlineResource/URL",
"<URL>#{@onlineresourceurl}</URL>").should be_truthy, "granule - #{granuleid} xml does not contain
OnlineResource = #{@onlineresourceurl}"
    end
end
Current Test Script: Longform_Collection_Export.rb
Given(/^criteria_420 mode is checked out$/) do
  @mode = ENV['MODE']
  @mode.should_not be_nil
  #host should be checked to be f5oml01v
  @mode_checker = Sdps::Mode.new(@mode)
  @mode_checker.mode_checked_out?().should be_truthy
  @db_subsystem = Sdps::Bmgt::DB_SUBSYSTEM
  @db_conn = Sdps::Db.connect(@db_subsystem, @mode)
  @db_conn.should_not be_nil

  @xmlUtil = Sdps::Bmgt::Xml.new()
  @bmgtdb = Sdps::Bmgt::BmgtDb.new(@db_conn)

  #Initialize critinfo and SoTestData
  @critinfo = "BE_82_01:420"
  @sotestdata = Sdps::SoTestData.new(@db_conn)

  @proxy_log_location = "/tools/common/test/BE_82_01/proxy/#{@mode}/logs/tcp.log"

end

Given(/^criteria_420 bmgt subsystem is checked out$/) do
```

```
  @mode_checker.bmgt_subsystem_checked_out?().should be_truthy, "bmgt system check failed"
end

Given(/^criteria_420 testdata is checked out$/) do
  criteria_420_testdata_checked_out?().should be_truthy
end

Given(/^criteria_420 collections are installed$/) do
    @aimdb.is_collection_installed?(@coll.shortname, @coll.versionid).should be_truthy,
"#{@coll.shortname}.#{@coll.versionid} for Criteria 420 not installed."
end

Given(/^criteria_420  collections are enabled for collection and granule export$/) do
    @bmgtdb.enable_collection_granule_export(@coll.collectionid).should be_truthy
end

When(/^criteria_420 Request the long form verification of metadata for a specified collection C1$/) do
  @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
  options = Array.new
  options.push('-vl')
  rv = Sdps::Bmgt::ExportRequest.new(@mode).manual_export_for_collection?(@coll.collectionid, options)
  rv.should be_truthy
end

Then(/^criteria_420 Verify the collection C1 is exported$/) do
   #poll and wait
  status = Sdps::Util.poll(180, 15){
    result = @bmgtdb.get_coll_export_request_status(@request_time, @coll.collectionid, 'VER')
    if (result == 'PENDING' || result == 'STARTED')
      nil
    else
      result
    end
  }
  status.should eq 'SUCCESS'
end
```

```
Then(/^criteria_420 Verify that the TCP proxy shows an HTTP PUT request for Collection C1$/) do
  dataset =  @coll.datasetid()
  @xmlUtil.http_put_for_collection?(@proxy_log_location, @request_time, dataset).should be_truthy
end

Then(/^criteria_420 Verify that the TCP proxy shows full collection metadata of the requested collection$/) do
  @collection_xml = @xmlUtil.get_collection_xml(@proxy_log_location, @request_time, @coll)
  @xmlUtil.collection_xml_valid?(@collection_xml).should be_truthy
end

Then(/^criteria_420 Verify that the TCP proxy shows that the request contains a query parameter xml_diff=true,
indicating it is for verification purposes$/) do
  dataset =  @coll.datasetid()
  @xmlUtil.http_verification_put_for_collection?(@proxy_log_location, @request_time, dataset).should be_truthy
end

Then(/^criteria_420 Request the manual export of collection metadata for collection C1$/) do
  rv = Sdps::Bmgt::ExportRequest.new(@mode).manual_export_for_collection?(@coll.collectionid, nil)
  rv.should be_truthy
end

Then(/^criteria_420 Verify the manual collection C1 is exported$/) do
   #poll and wait
  status = Sdps::Util.poll(180, 15){
    result = @bmgtdb.get_coll_export_request_status(@request_time, @coll.collectionid, 'MAN')
    if (result == 'PENDING' || result == 'STARTED')
      nil
    else
      result
    end
  }
  status.should eq 'SUCCESS'
end

Then(/^criteria_420 Save the exported metadata to an XML file$/) do
```

```
  @man_collection_xml = @xmlUtil.get_collection_xml(@proxy_log_location, @request_time, @coll)
  @xmlUtil.collection_xml_valid?(@man_collection_xml).should be_truthy
end

Then(/^criteria_420 Verify the metadata files from S1 and S2 are identical$/) do
  @collection_xml.eql?(@man_collection_xml).should be_truthy
end

def criteria_420_testdata_checked_out?

  @data = @sotestdata[@critinfo]
  @data.should_not be_nil

  rv = true
  if @data.length != 1
    puts "There should be 1 collections returned for #{@critinfo}, check AutoTestSoTestDataInfo table"
    rv = false
  else
    @shortname = @data[0].shortname
    @versionid = @data[0].versionid

    @aimdb = Sdps::AimDb.new(@db_conn)
    @coll = @aimdb.find_collection_by_shortname_versionid(@shortname, @versionid)

    if (@coll == nil || !@aimdb.is_collection_installed?(@shortname, @versionid))
      puts "#{@shortname}, #{@versionid} is not installed"
      rv = false
    else
      if @coll.datamodeltype != 'ECS'
        puts "#{@shortname}, #{@versionid} datamodeltype is not ECS"
        rv = false
      end
    end
  end
  rv.should be_truthy
endCurrent Test Script: BE_82_01_40_V13.rb
```

```
Then(/^Verify that the metadata generated by the action in S-2 subclause l contains no browse linkage URLs$/) do

    @test_granules_xml.keys.each do |granuleid|
      granule_xml = @test_granules_xml[granuleid]
      puts "xml contains " +
@xml_utils.granule_xml_getvalue(granule_xml,"//OnlineResource[Type='BROWSE']/URL/text()")
      @xml_utils.granule_xml_contains?(granule_xml,"//OnlineResource/Type", "<Type>BROWSE</Type>").should_not
be_truthy, "granule - #{granuleid} xml contains OnlineResource of type BROWSE"
     end
end
Current Test Script: BE_82_01_40_S2k.rb
When(/^Link a granule to a browse granule$/) do
   #ingest the browse test granule from the PDR
   @ingest =  Sdps::Ingest.build(@db_conn)
   polling_location = Conf.get_property("#{@mode.downcase}_ingest", "polling_location")
   puts polling_location
   #get the browse granule pdr
   @so_test_data_granules.each {|so_test_data_granule|
     if so_test_data_granule.shortname == "Browse"
       pdr_path =  so_test_data_granule.pathname
       puts pdr_path
       request =  @ingest.submit(polling_location, File.dirname(pdr_path), File.basename(pdr_path))
       @poll_wait_time = 300
       @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
       @export_queue = 'EVENT'
       puts "request = "
       puts request
       #poll and wait on status
       status = Sdps::Util.poll(600,30) {
       puts request.status
       if request.status.nil?
         nil
       elsif request.successful?
      #elsif request.status == :Successful
         true
      else
```

```
         puts request.status
         nil
      end
      }
      status.should be_truthy, "ingest failed"

     if status == true
       #browse_granule_ids = request.granuleids
       browse_granule_ids = request.browseids
       @test_granules2 = Array.new()
       browse_granule_ids.each do |browseid|
          browse_granules = @aimdb.find_browse_by_browseid(browseid)
          @test_granules2.push(browse_granules)
        end
      end
     end
   }
end
```

Current Test Script: BE_82_01_40_V12.rb

```
Then(/^Verify that the metadata generated by the action in S-2 subclause k contains a browse linkage URL, in
addition to any datapool URLs$/) do

    @test_granules_xml.keys.each do |granuleid|
      granule_xml = @test_granules_xml[granuleid]
      puts "xml contains " +
@xml_utils.granule_xml_getvalue(granule_xml,"//OnlineResource[Type='BROWSE']/URL/text()")
      @xml_utils.granule_xml_contains?(granule_xml,"//OnlineResource/Type", "<Type>BROWSE</Type>").should
be_truthy, "granule - #{granuleid} xml does not contain OnlineResource of type BROWSE"
     end
end
```

Current Test Script: Ecs_Collection_Additional_Metadata_S13.rb

```
require 'sdps/sotestdata'
require 'sdps/aimdb'
require 'sdps/mode'
require 'sdps/db'
require 'sdps/util'
```

```
require 'sdps/bmgt'

Given(/^Criteria_20_13 Initial Setup$/) do
  #Check mode.
  @mode = ENV['MODE']
  @mode.should_not be_nil
  @mode_checker = Sdps::Mode.new(@mode)
  @mode_checker.mode_checked_out?().should be_truthy
  @proxy_log_location = @mode_checker.get_proxy_log_location()
  @generator_log_location = "/usr/ecs/#{@mode}/CUSTOM/logs/EcBmBMGTGenerator.log"
  @xmlUtil = Sdps::Bmgt::Xml.new()

  #set up db connection
  @db_subsystem = Sdps::Bmgt::DB_SUBSYSTEM
  @conn = Sdps::Db.connect(@db_subsystem, @mode)
  @conn.should_not be_nil

  #Make sure BMGT is up and running, proxy needs to be up too
  @mode_checker.bmgt_subsystem_checked_out?().should be_truthy, "bmgt system check failed"

  #Initialize critinfo and SoTestData
  @critinfo = "BE_82_01:20:12"
  @sotestdata = Sdps::SoTestData.new(@conn)
end

Given(/^Criteria_20_13 All test collections are installed in the mode$/) do
  @data = @sotestdata[@critinfo]
  @data.should_not be_nil

  rv = true
  if @data.length != 1
    puts "There should be one collection returned for #{@critinfo}, check AutoTestSoTestDataInfo table"
    rv = false
  else
    @shortname = @data[0].shortname
    @versionid = @data[0].versionid
```

```
    @aimdb = Sdps::AimDb.new(@conn)
    @collection =@aimdb.find_collection_by_shortname_versionid(@shortname,@versionid)

    if @collection == nil
      puts "test data #{@shortname}, #{@versionid} not installed"
      rv = false
    else
      #collection should have datamodeltype = 'ECS'
      if @collection.datamodeltype != 'ECS'
        puts "collection's datamodeltype is not ECS"
        rv = false
      end
    end
  end
  rv.should be_truthy
end

Given(/^Criteria_20_13 All test collections are configured for collection and granule metadata export$/) do
  @collectionid = @collection.collectionid
  @bmgtdb = Sdps::Bmgt::BmgtDb.new(@conn)

  if @bmgtdb.collection_export_enabled?(@collectionid) == false
    @bmgtdb.enable_collection_granule_export(@collectionid).should be_truthy
  end
end

Given(/^Criteria_20_13 A local copy of the collection schema is available to validate against$/) do
  #verify Sdps::Bmgt::ECHO10_COLLECTION_SCHEMA exists
  schemaFile = Sdps::Bmgt::ECHO10_COLLECTION_SCHEMA
  File.exists?(schemaFile).should be_truthy
end

Given(/^Criteria_20_13 A TCP proxy is capturing BMGT traffic$/) do
 #This is checked at the initial setup
 puts "If it gets here, the TCP proxy is up and running"
```

```
end

When(/^Criteria_20_13 S-13 Attempt to configure the GranuleSpatialRepresentation for an ECS collection to null
\(i\.e\. automatically populate\)$/) do
  #There's only one collection in this test

  begin
    rs = @conn.exec ("update bg_collection_configuration set GranuleSpatialRepresentation=null,
CollectionCoordinateSystem=null where ShortName='#{@shortname}' and versionid = '#{@versionid}' returning 1")

    if rs != nil and rs.count == 1
      rv = true
    else
      rv = false
    end
  rescue => e
    puts e.backtrace
    rv = false
  end

  rv.should be_truthy
end

When(/^Criteria_20_13 Request a Manual Export of the collection$/) do
  #record the request time
  @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
  Sdps::Bmgt::ExportRequest.new(@mode).manual_export_for_collection?(@collection.collectionid, nil).should
be_truthy, "Falied running Manual Export for collection"

  ##make sure the request is queued
  Sdps::Bmgt::BmgtDb.new(@conn).get_coll_export_request_status(@request_time, @collection.collectionid,
"MAN").should_not be_nil
end

Then(/^Criteria_20_13 Manual export succeeds$/) do
```

```
  #poll and wait
  export_req = Sdps::Bmgt::BmgtDb.new(@conn)
  status = Sdps::Util.poll(180, 15){
    result = export_req.get_coll_export_request_status(@request_time, @collection.collectionid, "MAN")
    if (result == 'PENDING' || result == 'STARTED')
      nil
    else
      result
    end
  }
  status.should eq 'SUCCESS'
end

Then(/^Criteria_20_13 V-12 Verify that all Collection metadata in the previous steps validates against
https:\/\/api\.echo\.nasa\.gov\/ingest\/schema\/Collection\.xsd$/) do
  @collection_xml = @xmlUtil.get_collection_xml(@proxy_log_location, @request_time, @collection)
  @xmlUtil.collection_xml_valid?(@collection_xml).should be_truthy
end

Then(/^Criteria_20_13 V-14 Verify that the collection metadata generated in S-13 contains the proper
GranuleSpatialMetadata value derived from AmCollection.SpatialSearchType using the standard mapping$/) do

  #first create a mapping hash
  SearchTypeMapping = Hash.new()
  SearchTypeMapping["GPolygon"]="GEODETIC"
  SearchTypeMapping["Orbit"]="ORBIT"
  SearchTypeMapping["Point"]="GEODETIC"
  SearchTypeMapping["Rectangle"]="CARTESIAN"
  SearchTypeMapping["NotSupported"]="NO_SPATIAL"


  #Query AmCollection for each S-13 collection's SpatialSearchType
  begin
    rs = @conn.exec ("select SpatialSearchType from AmCollection where ShortName='#{@shortname}' and versionid =
'#{@versionid}'")
```

```
    if rs != nil and rs.count == 1
      @SpatialSearchType = rs[0]['spatialsearchtype']
      rv = true
    else
      rv = false
    end
  rescue => e
    puts e.backtrace
    rv = false
  end
  rv.should be_truthy

  #get the default value of the GranuleSpatialPresentation
  @dfv = SearchTypeMapping["#{@SpatialSearchType}"]

  #Verify @collection_xml contains <GranuleSpatialRepresentation>"#{@dfv}"</GranuleSpatialRepresentation>
  @xmlUtil.collection_xml_contains?(@collection_xml, "//Collection/Spatial/GranuleSpatialRepresentation",
"<GranuleSpatialRepresentation>#{@dfv}</GranuleSpatialRepresentation>").should be_truthy

end

Current Test Script: BE_82_01_40_S2a.rb
When(/^Ingest a new granule into the ECS inventory$/) do
   @ingest =  Sdps::Ingest.build(@db_conn)
   polling_location = Conf.get_property("#{@mode.downcase}_ingest", "polling_location")
   puts polling_location
   pdr_path =  @so_test_data_granules[0].pathname
   puts pdr_path
   request =  @ingest.submit(polling_location, File.dirname(pdr_path), File.basename(pdr_path))
   @poll_wait_time = 300
   @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
   @export_queue = 'EVENT'
   puts "request = "
   puts request

   #poll and wait on status
```

```
   status = Sdps::Util.poll(600,30) {
      puts request.status
      if request.status.nil?
         nil
      elsif request.successful?
      #elsif request.status == :Successful
         true
      else
         puts request.status
         nil
      end
   }
   status.should be_truthy, "ingest failed"

   if status == true
     ingest_granules = request.granuleids
     @test_granules1 = Array.new()
     ingest_granules.each do |granuleid|
        granule = @aimdb.find_granule_by_granuleid(granuleid)
        if granule != nil
           @test_granules1.push(granule)
        end
     end
   end
end

Current Test Script: nominal_collection_export.rb
# $LOAD_PATH << '../../lib'

require 'nokogiri'
require 'rspec'
# require 'pgx'

require 'sdps/sotestdata'
require 'sdps/mode'
require 'sdps/db'
```

```
require 'sdps/util'
require 'sdps/bmgt'

Given(/^criteria_10_1 mode is checked out$/) do
  #Need to add mode scheck out script
  @mode = ENV['MODE']
  @mode.should_not be_nil
  #host should be checked to be f5oml01v
  @mode_checker = Sdps::Mode.new(@mode)
  @mode_checker.mode_checked_out?().should be_truthy
  @db_subsystem = Sdps::Bmgt::DB_SUBSYSTEM
  @db_conn = Sdps::Db.connect(@db_subsystem, @mode)
  @db_conn.should_not be_nil

  @xmlUtil = Sdps::Bmgt::Xml.new()
  @bmgtdb = Sdps::Bmgt::BmgtDb.new(@db_conn)
  @proxy_log_location = "/tools/common/test/BE_82_01/proxy/#{@mode}/logs/tcp.log"

  #Initialize critinfo and SoTestData
  @critinfo = "BE_82_01:10:1"
  @sotestdata = Sdps::SoTestData.new(@db_conn)
end

Given(/^criteria_10_1 testdata is checked out$/) do
  criteria_10_1_testdata_checked_out?().should be_truthy
end

Given(/^criteria_10_1 bmgt subsystem is checked out$/) do
  @mode_checker.bmgt_subsystem_checked_out?().should be_truthy, "bmgt system check failed"
end

Given(/^criteria_10_1 two collections with ECS metadata and which share the same short name, but have different
version IDs$/) do
  (@coll1.shortname == @coll2.shortname).should be_truthy
  (@coll1.versionid == @coll2.versionid).should_not be_truthy
end
```

```
Given(/^criteria_10_1 that both collections are enabled for collection and granule export\.$/) do
  if (!@bmgtdb.collection_export_enabled?(@coll1.collectionid))
    @bmgtdb.enable_collection_export(@coll1.collectionid).should be_truthy
  end

  if (!@bmgtdb.collection_export_enabled?(@coll2.collectionid))
    @bmgtdb.enable_collection_granule_export(@coll2.collectionid).should be_truthy
  end
end

When(/^criteria_10_1 Request the manual export of collection metadata for one of these collections$/) do
  @request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
  manual_coll_export_for_collection?(@coll1.collectionid).should_not be_falsey
end


Then(/^criteria_10_1 Verify the bg_export_error table shows no errors referring to collection C1 on or after the
time of the manual export\.$/) do

  #poll and wait
  status = Sdps::Util.poll(180, 15){
    result = @bmgtdb.get_coll_export_request_status(@request_time, @coll1.collectionid, 'MAN')
    if (result == 'PENDING' || result == 'STARTED')
      nil
    else
      result
    end
  }
  status.should eq 'SUCCESS'
end

Then(/^criteria_10_1 Verify that the TCP proxy log reports a single HTTP PUT request for collection C1\.$/) do
  dataset =  @coll1.datasetid()
  @xmlUtil.http_put_for_collection?(@proxy_log_location, @request_time, dataset).should be_truthy
end
```

```
Then(/^criteria_10_1 Verify that the HTTP PUT request contains collection C1's full collection metadata\.$/) do
  @collection_xml = @xmlUtil.get_collection_xml(@proxy_log_location, @request_time, @coll1)
  @xmlUtil.collection_xml_valid?(@collection_xml).should be_truthy
end

Then(/^criteria_10_1 Verify the TCP proxy log reports no PUTs for collection C2\.$/) do
  dataset =  @coll2.datasetid()
  @xmlUtil.http_put_for_collection?(@proxy_log_location, @request_time, dataset).should be_falsey
end

Then(/^criteria_10_1 Verify that the metadata exported validates against the ECHO collection metadata schema
\(https:\/\/api\.echo\.nasa\.gov\/ingest\/schema\/Collection\.xsd\)$/) do

  @xmlUtil.collection_xml_valid?(@collection_xml).should be_truthy

end

Then(/^criteria_10_1 Verify that the metadata exported has the element Visible = true$/) do

  @xmlUtil.collection_xml_contains?(@collection_xml,"//Collection/Visible", "<Visible>true</Visible>").should
be_truthy

end

Then(/^criteria_10_1 Verify that the metadata exported has the element Orderable = false$/) do

  @xmlUtil.collection_xml_contains?(@collection_xml,"//Collection/Orderable",
"<Orderable>false</Orderable>").should be_truthy

end

Then(/^criteria_10_1 Verify that the metadata exported has the element InsertTime = The insert time of the
collection recorded in the AIM database\.$/) do

  insertTime = @coll1.inserttime
```

```
  insertTime.should_not be_nil
  insertTime_in_db = DateTime.parse(insertTime.to_s).strftime('%Y-%m-%dT%H:%M:%S.%3NZ')
  @xmlUtil.collection_xml_contains?(@collection_xml, "//Collection/InsertTime",
"<InsertTime>#{insertTime_in_db}</InsertTime>").should be_truthy

end

Then(/^criteria_10_1 Verify that the metadata exported has the element LastUpdate = The last update time of the
collection recorded in the AIM database$/) do

  lastUpdate = @coll1.lastupdate
  lastUpdate.should_not be_nil
  lastUpdate_in_db = DateTime.parse(lastUpdate.to_s).strftime('%Y-%m-%dT%H:%M:%S.%3NZ')

  @xmlUtil.collection_xml_contains?(@collection_xml, "//Collection/LastUpdate",
"<LastUpdate>#{lastUpdate_in_db}</LastUpdate>").should be_truthy

end

Then(/^criteria_10_1 Verify that the metadata exported contains version numbers with leading zeroes$/) do

  vId = "%03d" %  @coll1.versionid #changed to reflect NCR 8052154's bmgt version id conversion from int to
3-char string; Georgia Sandoval
  @xmlUtil.collection_xml_contains?(@collection_xml, "//Collection/VersionId",
"<VersionId>#{vId}</VersionId>").should be_truthy

end

def criteria_10_1_testdata_checked_out?
  puts "step1"
  @data = @sotestdata[@critinfo]
  @data.should_not be_nil

  rv = true
  if @data.length != 2
    puts "There should be two collections returned for #{@critinfo}, check AutoTestSoTestDataInfo table"
```

```
  rv = false
else
  @shortname = @data[0].shortname
  @versionid = @data[0].versionid

  @aimdb = Sdps::AimDb.new(@db_conn)
  @coll1 = @aimdb.find_collection_by_shortname_versionid(@shortname, @versionid)

  if (@coll1 == nil || !@aimdb.is_collection_installed?(@shortname, @versionid))
    puts "#{@shortname}, #{@versionid} is not installed"
    rv = false
  else
    if @coll1.datamodeltype != 'ECS'
      puts "#{@shortname}, #{@versionid} datamodeltype is not ECS"
      rv = false
    end
  end

  @shortname = @data[1].shortname
  @versionid = @data[1].versionid

  @coll2 = @aimdb.find_collection_by_shortname_versionid(@shortname, @versionid)

  if (@coll2 == nil  || !@aimdb.is_collection_installed?(@shortname, @versionid))
    puts "#{@shortname}, #{@versionid} is not installed"
    rv = false
  else
    #collection should have datamodeltype = 'ECS'
    if @coll2.datamodeltype != 'ECS'
      puts "#{@shortname}, #{@versionid} datamodeltype is not ECS"
      rv = false
    end
  end

end
rv.should be_truthy
```

```
end

# custom code application functions

def manual_coll_export_for_collection?(collectionid)
  begin
    request_time = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
    Sdps::Bmgt::ExportRequest.new(@mode).manual_export_for_collection?(collectionid, nil).should be_truthy,
"Falied running Manual Export for collection"

    #check if request was queued
    if @bmgtdb.get_coll_export_request_status(request_time, collectionid, 'MAN') == nil
      false
    else
      true
    end
  rescue =>e
    puts e.backtrace
    false
  end
end
Current Test Script: env.rb
# puts ">>> #{__FILE__}"

#user = `/usr/bin/whoami`.strip
#unless user == 'cmshared'
#  $stderr.puts "\n#{user}: Please run features as cmshared.\n\n"
#  exit 1
#end

def silence_warnings
  begin
    old_verbose, $VERBOSE = $VERBOSE, nil
    yield
  ensure
    $VERBOSE = old_verbose
```

```
  end
end

silence_warnings do
  #require 'logger'
  require 'nokogiri'
  require 'test-unit'
  require 'minitest'
  require 'rspec'
  require_relative '../../spec/spec_helper.rb'
  #require 'watir'
  require 'capybara'
  require 'capybara/cucumber'



  @mode = ENV['MODE']
  require 'sdps'
  @config = Conf.load("config/properties.yml")
  Capybara.configure do |config|
      config.default_driver = :selenium
  #    config.app_host = 'http://www.google.com'
  end

end

# puts "<<< #{__FILE__}"
Current Test Script: hooks.rb
# Is this needed?
### require_relative './env.rb'

# Before all scenarios
#    Code included here will be run before all tests.

# After all scenarios
at_exit do
```

```
  # @headless.destroy
end

AfterConfiguration do |config|

end

# This runs before each Scenario
Before do |scenario|
  # # 'skip_invoke!' can be used to skip a Scenario.
  # scenario.skip_invoke!

#  # Feature name
#  case scenario
#  when Cucumber::Core::Ast::Scenario
#    @feature_name = scenario.feature.name
#  when Cucumber::Core::Ast::ScenarioOutline
#    @feature_name = scenario.scenario_outline.feature.name
#  end
#
#  # Scenario name
#  case scenario
#  when Cucumber::Core::Ast::Scenario
#    @scenario_name = scenario.name
#  when Cucumber::Core::Ast::ScenarioOutline
#    @scenario_name = scenario.name
#  end

  # Tags (as an array)
  @scenario_tags = scenario.source_tag_names
end

# This runs after each Scenario
After do |scenario|
  # Stop on first error.  Only do this if absolutely necessary.
  #Cucumber.wants_to_quit = true if scenario.failed?
```

```
  #if scenario.passed? and is_final_scenario?(scenario)
  #  print "\n#{'*'*72}\n"
  #  print "* The end.\n"
  #  print "#{'*'*72}\n\n"
  #end
end

# Returns true if the scenario is the final one in the current feature.
def is_final_scenario?(scenario)
  scenario.line == final_scenario(scenario).line
end

# Returns the line and title of the final scenario in the current
# feature.  Ignores skipped scenarios, such as those marked @wip during
# a standard run.
def final_scenario(scenario)
  scenarios = scenario.feature.to_sexp.select { |sexp|
    sexp.is_a?(Array) and sexp.first == :scenario
  }
  Struct.new(:line, :title)[scenarios.last[1], scenarios.last[3]]
end

# Runs after each step of a Scenario tagged with @pause
AfterStep('@pause') do
  print "Press Return to continue"
  STDIN.getc
end

Current Test Script: oneline.rb
# ----------------------------------------
# Cucumber formatter that outputs the file
# and line number for each test.
#
# Useful for Hudson CI which only flushes
# the output to the log after a line return
```

```
# so that you can see what's failed before
# a long build finishes.
# ----------------------------------------
#
# Usage
#
#   - Put the contents of this file into "features/support/oneline.rb"
#
#   - Make sure cucumber has oneline.rb in the load list (which is the
#     default unless you have "-r" parameters in your cucumber.yml)
#
#   - Add "--format Oneline" to your cucumber.yml or at the command line
#
# Example output
#
# [Thu, 12 Aug 2010 15:30:08 -0400]      Pass      features/admin/_login.feature:L#8
# [Thu, 12 Aug 2010 15:30:09 -0400] **** FAIL **** features/admin/_login.feature:L#12
# [Thu, 12 Aug 2010 15:30:10 -0400]      Pass      features/admin/_login.feature:L#19
# [Thu, 12 Aug 2010 15:30:11 -0400]      Pass      features/admin/_login.feature:L#29
#

require 'cucumber/formatter/io'
require 'cucumber/formatter/console'

class Oneline
  include Cucumber::Formatter::Io
  include Cucumber::Formatter::Console

  attr_reader :step_mother

  def initialize(step_mother, path_or_io, options)
    @step_mother, @io, @options = step_mother, ensure_io(path_or_io, "oneline"), options
    @file_names = []
    @file_colon_lines = Hash.new{|h,k| h[k] = []}
  end
```

```
def after_step_result(keyword, step_match, multiline_arg, status, exception, source_indent, background)
  if status != :skipped
    @step_details << "#{keyword} #{step_match.format_args}"
  end

  if ![:passed, :skipped].include?(status)
    @step_details << ""
    @step_details << "#{exception}"
    @step_details << "#{step_match.backtrace_line}"
  end
end

def before_feature_element(feature_element)
  reset_scenario_status
end

def after_feature_element(feature_element)
  # Don't print the sum result of an outline scenario
  unless @scenario_outline
    file, line = feature_element.file_colon_line.split(':')
    progress(@scenario_status, "#{file}:L##{line}")
    if @scenario_status != :passed && @step_details.size > 0
      @io.print("\n")
      @io.print(format_string("\n\n                Scenario steps:\n", @scenario_status))
      @step_details.each do |step|
        @io.print(format_string("\n                #{step}", @scenario_status))
      end
      @io.print("\n")
      @io.flush
    end
  end

  @scenario_outline = false
end

def step_name(keyword, step_match, status, source_indent, background)
```

```
  update_scenario_status(status)
end

def after_table_row(table_row)
  if table_row && table_row.respond_to?("passed?")
    begin
      table_row.passed?
      file, line = table_row.backtrace_line.split(':')
      progress(table_row.status, "#{file}:L##{line}")

      update_scenario_status(table_row.status)
    rescue Cucumber::Ast::OutlineTable::ExampleRow::InvalidForHeaderRowError
      # Header row for a scenario outline
      @scenario_outline = true
    end
  end
  reset_scenario_status
end

def after_features(features)
  @io.puts
  @io.puts
  print_summary(features)
end

private

def reset_scenario_status
  @scenario_status = :passed
  @step_details    = []
end

def update_scenario_status(status)
  @scenario_status = status  if     status == :failed
  @scenario_status = status  unless [:failed, :pending, :undefined].include?(@scenario_status)
end
```

```
  def progress(status, message = "")
    char = CHARS[status]
    # @io.print(format_string("\n[#{Time.now.rfc822}] #{char} #{message}", status))
    @io.print(format_string("\n[#{Time.now}] #{char} #{message}", status))
    @io.flush
  end

  CHARS = {
    :passed    => "     Pass     ",
    :failed    => "**** FAIL ****",
    :undefined => "**** Undefined",
    :pending   => "**** Pending  ",
    :skipped   => "     Skipped  "
  }

  def print_summary(features)
    print_steps(:pending)
    print_steps(:failed)
    print_stats(features, @options.custom_profiles)
    print_snippets(@options)
    print_passing_wip(@options)
  end
end
```

Current Test Script: formatter.rb

```
require 'cucumber/formatter/pretty'

# The formatter used for <tt>--format PrettyNoSkip</tt>.
# Hides skipped steps, to prevent thousands of useless lines printing after a
# failure.
class PrettyNoSkip < Cucumber::Formatter::Pretty

  def before_step_result(keyword, step_match, multiline_arg, status, exception, source_indent, background,
file_colon_line)
    result = super
    if status == :skipped
```

```
      @hide_this_step = true
    end
    result
  end

end
Current Test Script: esi_parameter_subset.rb
#When(/^Submit a ESI "(.*?)" request for each of the SMAP test granules "(.*?)" for each band parameter and
"(.*?)" format$/) do |agent, localgranuleid, fmt|
#     #puts fmt
#     @response_granule_xmls  = Hash.new()
#     @esi = Sdps::Da::Esi.new(@mode)
#             @service_agent = agent.strip
#             @format = fmt.strip
#             puts @format
#             puts @service_agent
#             granuleid = @test_granule.granuleid
#             esdt  = @test_collection.esdt
#             dataset_id = @test_collection.datasetid
#             shortname = @test_collection.shortname
#             @response_band_xmls = Hash.new()
#             @test_bands.each { |band|
#                 @start_time = Time.now
#                 #prepend band with shortname
#                 #band1 = "/#{shortname}#{band}"
#                 response_xml = @esi.submit_request(esdt, granuleid, @service_agent, format:@format,
band_parameter:band)
#                 @response_band_xmls[band] = response_xml
#                  #@end_time = Time.now
#                  puts @esi.esi_service_request_url
#                  #puts "#{@start_time} - #{@end_time}"
#                  #puts "elapsed=#{@end_time-@start_time}"
#
#             }
#end
When(/^Submit a ESI "([^"]*)" request for the test granule for "([^"]*)" for each band parameter and "([^"]*)"
```

```
format$/) do |agent, esdt, fmt|
  #puts fmt
     @response_granule_xmls  = Hash.new()
     @esi = Sdps::Da::Esi.new(@mode)
             @service_agent = agent.strip
             @format = fmt.strip
             puts @format
             puts @service_agent
             puts @mode
             granuleid = @test_granule.granuleid
             esdt  = @test_collection.esdt
             dataset_id = @test_collection.datasetid
             shortname = @test_collection.shortname
             @response_band_xmls = Hash.new()
             @test_bands.each { |band|
                 @start_time = Time.now
                 #prepend band with shortname
                 #band1 = "/#{shortname}#{band}"
                 response_xml = @esi.submit_request(esdt, granuleid, @service_agent, format:@format,
band_parameter:band)
                 @response_band_xmls[band] = response_xml
                  #@end_time = Time.now
                  puts @esi.esi_service_request_url
                  #puts "#{@start_time} - #{@end_time}"
                  #puts "elapsed=#{@end_time-@start_time}"

             }
end

#Then(/^the ESI service request succeeds for the SMAP  test granule "(.*?)" and band parameter and "(.*?)"
format$/) do |localgranuleid, format|
#        collection = @test_collection
#        esdt = collection.esdt
#        @response_results = Hash.new()
#        failed_requests = Array.new
#        @response_band_xmls.keys.each do |band|
```

```
#           response_xml = @response_band_xmls[band]
#           response = @esi.parse_response(response_xml)
#           msg = response.message.to_s
#           puts "#{esdt}, #{@format}, #{band}, #{msg}"
#           @response_results[band] = response.urls
#           if msg.eql?("FAILURE")
#               failed_requests.push ("#{esdt}, #{@format}, #{band}, #{msg}")
#           end
#
#        end
#        failed_requests.should be_empty, failed_requests.to_s
#end

Then(/^the ESI "([^"]*)" request should succeed for the test granule for "([^"]*)" and band parameter and
"([^"]*)" format$/) do |agent, esdt, format|
      collection = @test_collection
        esdt = collection.esdt
        @response_results = Hash.new()
        failed_requests = Array.new
        @response_band_xmls.keys.each do |band|
           response_xml = @response_band_xmls[band]
           response = @esi.parse_response(response_xml)
           msg = response.message.to_s
           puts "#{esdt}, #{@format}, #{band}, #{msg}"
           @response_results[band] = response.urls
           if msg.eql?("FAILURE")
               failed_requests.push ("#{esdt}, #{@format}, #{band}, #{msg}")
           end

        end
        failed_requests.should be_empty, failed_requests.to_s
end


#Then(/^the ESI converted files for each band parameter for the test granule "(.*?)" should appear in the
results in the requested "(.*?)" format$/) do |arg1, arg2|
```

```
#          @formatted_outputs = Hash.new
#          @response_results.keys.each do | band |
#          response = @response_results[band]
#          #puts band
#          #puts response
#         # outputs1 = Array.new
#          output = response.strip
#          #puts output
#
#          if !output.nil? and output.length > 0
#                      File.file?(output).should be_truthy, "#{output} file does not exist"
#          end
#
#          if @format.nil? or @format.empty?
#                extension = File.extname(@test_hdf)
#          else
#                extension = Sdps::Da::FORMAT_EXTENSIONS[@format]
#          end
#          #puts extension
#
#          File.extname(output).should eql extension
#
#
#         @formatted_outputs[band] = output
#         end
#        puts @formatted_outputs
#end

Then(/^the ESI converted files for each band parameter for the test granule for "([^"]*)" should appear in the
results in the requested "([^"]*)" format$/) do |arg1,
 arg2|
  @formatted_outputs = Hash.new
          @response_results.keys.each do | band |
          response = @response_results[band]
          #puts band
          #puts response
```

```
        # outputs1 = Array.new
         output = response.strip
         #puts output

         if !output.nil? and output.length > 0
                     File.file?(output).should be_truthy, "#{output} file does not exist"
         end

         if @format.nil? or @format.empty?
               extension = File.extname(@test_hdf)
         else
               extension = Sdps::Da::FORMAT_EXTENSIONS[@format]
         end
         #VIIRS outputs .he5 for no reformat
         if extension.eql?(".h5") and @test_collection.shortname.include?("VNP")
           extension = ".he5"
         end

         File.extname(output).should eql extension


        @formatted_outputs[band] = output
        end
       puts @formatted_outputs
end


Then(/^the output file for each band for the test granule for "([^"]*)" from ESI is valid for the requested band
parameter$/) do |arg1|
    # #Then(/^the output file for each band for the test granule "(.*?)" from
    # ESI is valid for the requested band parameter$/) do |arg1|
    @saved_outputs = Hash.new

    if @format.nil? or @format.empty?
        @format = 'NO_REFORMAT'
        extension = File.extname(@test_hdf)
```

```
else
    extension = Sdps::Da::FORMAT_EXTENSIONS[@format]
end

if @format.nil? or @format.empty?
    format1 = "NO_REFORMAT"
else
    format1 = @format.gsub("-", "_")
end

if @projection.nil? or @projection.empty?
    projection1 = "NO_CHANGE"
else
    projection1 = @projection.gsub(" ", "_")
end

if @interpolation.nil? or @interpolation.empty?
    interpolation1 = "NO_INTERPOLATION"
else
    interpolation1 = @interpolation
end

if @spatial_subset.nil? or @spatial_subset.empty?
    subset = nil
else
    subset = @spatial_subset.tr('-', '_')
    arr = subset.split(',')
    arr[0] = "#{arr[0]}W"
    arr[1] = "#{arr[1]}S"
    arr[2] = "#{arr[2]}E"
    arr[3] = "#{arr[3]}N"
    subset = "#{arr[0]}#{arr[1]}#{arr[2]}#{arr[3]}"
    puts "spatial_subset dir = #{subset}"
end
runtime = Time.now.strftime("%Y_%m_%d_%H_%M_%9N")
failed_validation_results = Array.new
```

```
    @formatted_outputs.keys.each  do | band |
        output_file = @formatted_outputs[band]
        # #puts output_files #puts @format #output_files.each do |output_file |
        # #replace slash with underscore
        band1 = band.gsub("/", "_")

        if subset.nil?
            dest_path =
"/workingdata/cucumber/#{@mode}/#{@service_agent}/#{@test_collection.esdt}/#{format1}/#{projection1}/#{interpola
tion1}/NO_SUBSET/#{band1}/#{runtime}"
        else
            dest_path =
"/workingdata/cucumber/#{@mode}/#{@service_agent}/#{@test_collection.esdt}/#{format1}/#{projection1}/#{interpola
tion1}/#{subset}/#{band1}/#{runtime}"
        end
        utils =  Sdps::Da::DaUtil.new()
        utils.copy_with_path(output_file, dest_path)
        dest_file = "#{dest_path}/#{File.basename(output_file)}"
        puts dest_file
        if !File.extname(dest_file).eql?(extension)
            failed_validation_results.push(dest_file)
        end
        @saved_outputs[band] = dest_file
        # #end
    end
    failed_validation_results.should be_empty, "failed validation - #{failed_validation_results.to_s}"
end

Then(/^the attributes in the test granule file for each band parameter should appear in the output file for the
requested band parameter$/) do

failed_validation_results = Array.new
                 utils =  Sdps::Da::DaUtil.new()
                case @format
                when "GeoTIFF","KML"
                    puts"format=#{@format} - no attribute validation"
```

```
                when "ASCII"
                    @test_bands.each do | band |
                      dest_file  = @saved_outputs[band]
                    # no attributes in ascii output

                    validation_result = utils.validate_ascii_output_attributes_for_band(@test_hdf,
dest_file,band)
                    #if validation_result.nil? or validation_result == false
                    #   failed_validation_results.push( "#{output_file} : #{band}")
                       puts ("band =#{band}, expected attr=#{utils.input_arr[0]}, result
attr=#{utils.output_arr}")
                      if validation_result.nil? or validation_result == false
                         failed_validation_results.push( "#{dest_file} : #{band}")
                      end
                   end
                when "NetCDF-3"
                    @test_bands.each do | band |
                    output_file = @saved_outputs[0]

                    validation_result = utils.validate_netcdf3_output_attributes_for_band(@test_hdf,
output_file,band)
                       puts ("band =#{band}, expected attr=#{utils.input_arr[0]}, result
attr=#{utils.output_arr}")
                      if validation_result.nil? or validation_result == false
                         failed_validation_results.push( "#{dest_file} : #{band}")
                      end
                   end
                when "HDF-EOS5", "NO_REFORMAT", "","NetCDF4-CF"
                   #attributes for each band
                   @test_bands.each do | band |
                      dest_file  = @saved_outputs[band]
                      validation_result = utils.validate_h5_output_attributes_for_band(@test_hdf,
dest_file,band)
                      if validation_result.nil? or validation_result == false
                         failed_validation_results.push( "#{dest_file} : #{band}, #{@error_msg}")
                         puts ( "#{dest_file} : #{band}, #{@error_msg}")
```

```
                      else
                          puts ("band =#{band}, expected attr=#{utils.input_arr[0]}, result
attr=#{utils.output_arr}")
                      end
                  end
               end
              #skipping this for now. to proceed to next step
              # failed_validation_results.should be_empty, "failed validation}"
end

Then(/^the data in the "([^"]*)" output file for each band parameter should match the data in the test granule
for "([^"]*)"$/) do |arg1, arg2|
        failed_validation_results = Array.new
       puts @format
        puts @test_hdf
         utils =  Sdps::Da::DaUtil.new()
        if @format == "GeoTIFF" or @format == "KML"
                    puts"format=#{@format} - no data validation"
        end
        @test_bands.each do | band |
                dest_file  = @saved_outputs[band]
                 #replace slash with underscore
                band1 = band.gsub("/", "_")
                puts dest_file
                puts band
               case @format
                when "ASCII"
                    validation_result = utils.validate_ascii_output_data(@test_hdf, dest_file, band)
                   #puts ("band =#{band}, expected=#{utils.input_arr}, result=#{dautils.output_arr}")
                    if validation_result.nil? or validation_result == false
                        puts "FAILED VALIDATION"
                        puts "band =#{band}"
                        puts "expected=#{utils.input_arr}"
                        puts "result=#{utils.output_arr}"
                       failed_validation_results.push( "dest_file:#{dest_file} : band =#{band},
expected=#{utils.input_arr}, result=#{utils.output_arr}")
```

```
                    end
                when "NetCDF-3"
                    validation_result = utils.validate_netcdf3_output_data(@test_hdf,dest_file, band)
                    #puts ("band =#{band}, expected=#{utils.input_arr}, result=#{utils.output_arr}")
                    if validation_result.nil? or validation_result == false
                        puts "FAILED VALIDATION"
                        puts "band =#{band}"
                        puts "expected=#{utils.input_arr}"
                        puts "result=#{utils.output_arr}"
                        failed_validation_results.push( "dest_file:#{dest_file} : band =#{band},
expected=#{utils.input_arr}, result=#{utils.output_arr}")
                    end
                when "NO_REFORMAT",  "", "NetCDF4-CF","HDF-EOS5"
                    validation_result = utils.validate_h5_output_data(@test_hdf, dest_file, band)
                   #puts ("band =#{band}, expected=#{utils.input_arr}, result=#{utils.output_arr}")
                    if validation_result.nil? or validation_result == false
                        puts "FAILED VALIDATION"
                        puts "band =#{band}"
                        puts "expected=#{utils.input_arr}"
                        puts "result=#{utils.output_arr}"
                        failed_validation_results.push( "dest_file:#{dest_file} : band =#{band},
expected=#{utils.input_arr}, result=#{utils.output_arr}")
                    end
                when ''
                   validation_result = utils.validate_h5_output_data(@test_hdf, dest_file, band)
                   #puts ("band =#{band}, expected=#{utils.input_arr}, result=#{utils.output_arr}")
                    if validation_result.nil? or validation_result == false
                       puts "FAILED VALIDATION"
                       failed_validation_results.push( "dest_file:#{dest_file} : band =#{band},
expected=#{utils.input_arr}, result=#{utils.output_arr}")
                    end
                end
    end

      failed_validation_results.should be_empty, "failed validation - #{failed_validation_results.to_s}"
end
```

Current Test Script: esi_temporal_subset.rb

```
When(/^Submit a ESI "([^"]*)" request for the h5 test granule "([^"]*)" and "([^"]*)" format and temporalsubset
"([^"]*)"$/) do |agent, granule, fmt, temporalsubset|
             p Time.now
             @service = "esi"
             @format = fmt.strip
             @service_agent = agent.strip
             @temporal_subset = temporalsubset.strip

             @response_granule_xmls  = Hash.new()
             @esi = Sdps::Da::Esi.new(@mode)
             granuleid = @test_granule.granuleid
             esdt  = @test_collection.esdt
             @response_band_xmls = Hash.new()
             puts "bands excluded=#{@bands_excluded}"
             if !@bands_excluded.nil?  and  !@bands_excluded.empty?
                  band_str = @test_bands.join(",")
                  @response_xml = @esi.submit_request(esdt, granuleid, @service_agent, format:@format,
temporal:@temporal_subset,band_parameter:band_str)
             else
               @response_xml = @esi.submit_request(esdt, granuleid, @service_agent, format:@format,
temporal:@temporal_subset)
             end
             puts @esi.esi_service_request_url
                 #@response_band_xmls[band] = response_xml

#             }
             p Time.now
end


Then(/^the ESI "([^"]*)" request should succeed for the h5 test granule "([^"]*)" and "([^"]*)" format and
temporalsubset "([^"]*)"$/) do |agent, granule, fmt, temporalsubset|
         p Time.now
        collection = @test_collection
```

```
        esdt = collection.esdt
        #@response_results = Hash.new()
        failed_requests = Array.new
        #@response_band_xmls.keys.each do |band|
           #response_xml = @response_band_xmls[band]
           response = @esi.parse_response(@response_xml)
           p response
           msg = response.message.to_s
           puts "#{granule}, #{fmt},  #{temporalsubset}, #{msg}"
           #@response_results[band] = response.urls
           p response.urls
           if msg.eql?("FAILURE")
               failed_requests.push ("#{esdt}, #{fmt}, #{temporalsubset}, #{msg}")
           else
               @test_results = response.urls
           end
        #end
        failed_requests.should be_empty, failed_requests.to_s
         p Time.now
end


#for ICESAT temporal subset - all band request
Then(/^the output file for each temporalband "([^"]*)" for the test granule "([^"]*)" from ESI is valid for the
requested temporalsubset "([^"]*)"$/) do |temporalband, granule,temporalsubset|
     p Time.now
     temporal_bands_arr = temporalband.split(",")
     @temporal_bands = Array.new()
     @test_bands.each do | band |
          temporal_bands_arr.each do |temporalband|
             if band.include?(temporalband)
                 @temporal_bands.push(band)
             end
         end
     end
```

```
    puts "temporal bands:#{@temporal_bands}"
    failed_validation_results = Array.new
    puts "subset=#{temporalsubset}"
    puts "agent=#{@service_agent}"
    epoch = ""
    if @service_agent.eql?("ICESAT2") or @service_agent.eql?("GLAS")
        #get the epoch
        epoch = Sdps::Da::EPOCH[@service_agent]
    end
    puts "epoch=#{epoch}"
    utils =  Sdps::Da::DaUtil.new()
    @zero_length_temporal_bands = Array.new
    bands_to_exclude = Array.new
    @saved_outputs.each {|dest_file|
          case @format
          when "HDF-EOS5", "NO_REFORMAT", ""
            puts "NO_REFORMAT"
            temporal_subset_hash = Hash.new

            subset_arr = @temporal_subset.split(',')
            temporal_subset_hash[:START] = subset_arr[0]
            temporal_subset_hash[:END] = subset_arr[1]

            #h5dump deltatime
              puts "expected=#{@temporal_subset}"
                start_delta_time = utils.get_epoch_delta(epoch, temporal_subset_hash[:START])
                end_delta_time = utils.get_epoch_delta(epoch, temporal_subset_hash[:END])
              puts "start=#{start_delta_time}, end=#{end_delta_time}"
              validation_result = false
              #get the temporal bands

             @temporal_bands.each do | band |
                 if utils.h5output_contains_band(dest_file, band)
                   #validation_result = utils.validate_h5_temporal_subset_data(@test_hdf, dest_file, band,
temporal_subset_hash, epoch)
                   if @service_agent.eql?("ICESAT2")
```

```
                        #temporal validation for icesat only validates the first and last entry
                        validation_result = utils.validate_icesat_temporal_subset_data(@test_hdf, dest_file,
band, start_delta_time, end_delta_time)
                    else
                       validation_result = utils.validate_h5_temporal_subset_data(@test_hdf, dest_file, band,
start_delta_time, end_delta_time)
                    end
                    #the data was not in the temporal range  requested
                    #this should not be in the output
                    if utils.output_arr.nil? or utils.output_arr.empty?
                       len=0
                       puts ("temporal band =#{band}, data-len: #{len}")
                        @zero_length_temporal_bands.push(band)
                        #get the group name
                        band_group = band.slice(0..band.rindex("/"))
                        bands_to_exclude += @test_bands.select { |group| group.include?(band_group)}
                   else
                       len= utils.output_arr.length
                   end
                    if validation_result.nil? or validation_result == false
                        puts "Failed temporal subset validation for #{band} :  #{@error_msg}, validation_result
= #{validation_result}"
                        if utils.output_arr.nil?
                             puts "dest_file:#{dest_file} : band =#{band}, len=#{len}"
                        failed_validation_results.push( "dest_file:#{dest_file} : band =#{band}, len=#{len}")
                        else
                        puts "dest_file:#{dest_file} : band =#{band}, len=#{len},
result=#{utils.output_arr[0..10]}"
                        failed_validation_results.push( "dest_file:#{dest_file} : band =#{band}, len=#{len},
result=#{utils.output_arr[0..10]}")
                        end
                    else
                        puts "Success - temporal subset validation: #{band}"
                    end
                  else #band not in output
                       puts ("temporal band =#{band} excluded from output")
```

```
                        @zero_length_temporal_bands.push(band)
                        #get the group name
                        band_group = band.slice(0..band.rindex("/"))
                       bands_to_exclude += @test_bands.select { |group| group.include?(band_group)}
                   end

               end #do band
               puts "Bands excluded in output due to data not being in temporal subset range=#{bands_to_exclude}"
               @test_bands = @test_bands - bands_to_exclude
            end #case
     }
     #puts failed_validation_results.to_s
     failed_validation_results.should be_empty
      p Time.now
end


Then(/^the data in the output file from "([^"]*)" service should match the data in the test granule for the
requested "([^"]*)" temporalsubset$/) do | agent, temporalsubset |
    p Time.now
        #get the number of bands with coordinates attribute
        # those bands should all be subsetted and it should match the geolocation band range

        failed_validation_results = Array.new
       #@test_bands.each do | band |
       #    @formatted_outputs.each do |output_file |
                 #replace slash with underscore
       #          band1 = band.gsub("/", "_")
                #dest_path =
"/workingdata/cucumber/#{@mode}/#{@service}_#{@agent}/#{@test_collection.esdt}/#{band1}/#{@format}/"
                #@utils.copy_with_path(output_file, dest_path)
                #dest_file = "#{dest_path}#{File.basename(output_file)}"
                #puts dest_file
                case @format
#                when "ASCII"
#                    output_file = @saved_outputs[0]
```

```
#                    @test_bands.each do | band |
#                    validation_result = @dautil.validate_ascii_output_data(@test_hdf, output_file, band)
#                   puts ("band =#{band}, expected data=#{@dautil.input_arr[0]}, result
data=#{@dautil.output_arr[0]}")
#                    if validation_result.nil? or validation_result == false
#                       failed_validation_results.push( "#{output_file} : #{band}")
#                    end
#                    end
#                when "NetCDF-3"
#                    output_file = @saved_outputs[0]
#                    @test_bands.each do | band |
#                    validation_result = @dautil.validate_netcdf3_output_data(@test_hdf, output_file, band)
#                     puts ("band =#{band}, expected data=#{@dautil.input_arr[0]}, result
data=#{@dautil.output_arr[0]}")
#                    if validation_result.nil? or validation_result == false
#                       failed_validation_results.push( "#{output_file} : #{band}")
#                    end
#                    end
#              TO BE UPDATED...
                when "NetCDF4-CF", "HDF-EOS5",  "NO_FORMAT", ""
#                    check_spatially_subsetted_bands
#                    check_spatially_unsubsetted_bands
#                    output_file = @saved_outputs[0]
#                    @test_bands.each do | band |
#                    validation_result = @dautil.validate_h5_output_data(@test_hdf, output_file, band)
#                    puts ("band =#{band}, expected data=#{@dautil.input_arr[0]}, result
data=#{@dautil.output_arr[0]}")
#                    if validation_result.nil? or validation_result == false
#                       failed_validation_results.push( "#{output_file} : #{band}")
#                    end
                end
#           end
      failed_validation_results.should be_empty, "failed validation - #{failed_validation_results.to_s}"
      p Time.now
end
```

```
When(/^Submit a ESI "([^"]*)" request for the h5 test granule "([^"]*)" for each band parameter and "([^"]*)"
format and temporalsubset "([^"]*)"$/) do | agent, localgranuleid, fmt, temporalsubset |
             @service = "esi"
             format = fmt.strip
             service_agent = agent.strip
             @temporal_subset = temporalsubset.strip
             @response_granule_xmls  = Hash.new()
             @esi = Sdps::Da::Esi.new(@mode)
             granuleid = @test_granule.granuleid
             esdt  = @test_collection.esdt
             @response_band_xmls = Hash.new()
             @test_bands.each { |band|
                  #prepend band with shortname
                  band1 = "#{band}"
                 #remove leading slash and replace / with :
                  if band1[0] ==  '/'
                       band1 = band[1..-1]
                  end
                  band1.gsub("/", ":")
                  response_xml = @esi.submit_request(esdt, granuleid, service_agent, format:format,
temporal:@temporal_subset, band_parameter:band)
                  puts @esi.esi_service_request_url
                 @response_band_xmls[band] = response_xml

             }
end

Then(/^the ESI "([^"]*)" request should succeed for the h5 test granule "([^"]*)" and band parameter and
"([^"]*)" format and temporalsubset "([^"]*)"$/) do | agent, localgranuleid, format, temporalsubset |
        #collection = @test_collection
        #esdt = collection.esdt
        @response_results = Hash.new()
        failed_requests = Array.new
        @response_band_xmls.keys.each do |band|
           response_xml = @response_band_xmls[band]
           response = @esi.parse_response(response_xml)
```

```
            p response
            msg = response.message.to_s
            puts "#{localgranuleid}, #{format},  #{@temporal_subset}, #{band}, #{msg}"
            @response_results[band] = response.urls
            p response.urls
            if msg.eql?("FAILURE")
                failed_requests.push ("#{localgranuleid}, #{format}, #{@temporal_subset}, #{band}, #{msg}")
            end
        end
        failed_requests.should be_empty, failed_requests.to_s
end

Then(/^the "([^"]*)" output file for each band for the test granule "([^"]*)" from ESI is valid for the
requested temporalsubset "([^"]*)"$/) do | format, localgranuleid, temporalsubset |
 if @temporal_subset.nil? or @temporal_subset.empty?
            subset = nil
         else
                subset = @spatial_subset.tr('-', '_')
                arr = subset.split(',')
                arr[0] = "#{arr[0]}W"
                arr[1] = "#{arr[1]}S"
                arr[2] = "#{arr[2]}E"
                arr[3] = "#{arr[3]}N"
                subset = "#{arr[0]}#{arr[1]}#{arr[2]}#{arr[3]}"
                puts "spatial_subset dir = #{subset}"
      end
      utils =  Sdps::Da::DaUtil.new()
      runtime = Time.now.strftime("%Y_%m_%d_%H_%M_%9N")
      failed_validation_results = Array.new delta = 0.05
      failed_subset = Array.new()
      subset_arr = @spatial_subset.split(',')
          case @format
            when "GeoTIFF"
               @gdal = Sdps::Da::Gdal.new(@mode)
              @formatted_outputs.keys.each  do | band |
                     output_file = @formatted_outputs[band]
```

```
                # #puts "output from gdalinfo...."
                spatial_extent = @gdal.get_spatial_extent(output_file)
                #puts "#{spatial_extent} - #{output_file}"
                #projection_info =  @gdal.get_projection_info(output_file)
                proj4_params = @gdal.get_proj4_params(output_file)
                proj4_params.should_not be_nil
               puts "params: #{proj4_params}"
               if(proj4_params.include?("cea"))
                 if (utils.spatial_extent_match?(subset_arr, spatial_extent,delta) == false)
                   #add to the failed list
                   puts  "Failure : #{output_file} : input=#{subset}, output=#{spatial_extent.to_s}"
                   failed_subset.push( "#{output_file} : input=#{subset}, output=#{spatial_extent.to_s}")
                else
                  puts  "Success : #{output_file} : input=#{subset}, output=#{spatial_extent.to_s}"
                end
              elsif (proj4_params.include?("laea"))
                 #get modified input wsen
                 modified_input_wsen = utils.get_modified_input_subset(subset_arr, proj4_params)
                 #convert back to degrees
                 if(utils.spatial_extent_match?(modified_input_wsen, spatial_extent, delta) == false)
                   # #add to the failed list
                    puts  "Failure : #{output_file} : orig input=#{subset}, modified
input=#{modified_input_wsen}, output=#{spatial_extent.to_s}"
                    failed_subset.push( "#{output_file} : input=#{subset}, output=#{spatial_extent.to_s}")
                else
                    puts  "Success : #{output_file} :orig input=#{subset}, modified input=#{modified_input_wsen},
output=#{spatial_extent.to_s}"
                end
              else
                puts "proj4 params could not be determined"
              end
      end
          when "HDF-EOS5", "NO_REFORMAT", ""
             #puts "NO_REFORMAT"
             #h5dump lat/lon
```

```
            spatial_subset_hash = Hash.new
            subset_arr = @spatial_subset.split(',')
            spatial_subset_hash[:w] = subset_arr[0] .to_f
            spatial_subset_hash[:s] = subset_arr[1] .to_f
            spatial_subset_hash[:e] = subset_arr[2] .to_f
            spatial_subset_hash[:n] = subset_arr[3] .to_f
            puts "expected : #{@spatial_subset}"
            @saved_outputs.keys.each  do | band |
              dest_file = @saved_outputs[band]
              if band.include?("lat") or band.include?("lon")
                  validation_result = utils.validate_h5_spatial_subset_data(@test_hdf, dest_file,
band,spatial_subset_hash)
                  if utils.output_arr.nil?
                      len=0
                  else
                      len= utils.output_arr.length
                  end
                  puts ("geolocation band =#{band}, data-len: #{len}, result=#{utils.output_arr}")
                   if validation_result.nil? or validation_result == false
                       puts "Failed Spatial Validation"
                       failed_validation_results.push( "dest_file:#{dest_file}, band=#{band},datalen: #{len},
result=#{utils.output_arr}")
                   else
                       puts 'Success Spatial Validation'
                   end
              else
                  length = utils.get_h5_data_len_for_band(dest_file, band)
                  puts ("not a geolocation band =#{band}, data len=#{length}")
              end
            end
            failed_validation_results.should be_empty, "failed subsets: #{failed_validation_results.to_s}"
          end

end
Current Test Script: esi_format.rb
When(/^Submit a ESI "([^"]*)" request without the excluded bands for the h5 test granule for the "([^"]*)" and
```

```
"([^"]*)"$/) do |agent,esdt,format|
  @service = "esi"
  @format = format.strip
  @service_agent = agent.strip

  @response_granule_xmls  = Hash.new()
  @esi = Sdps::Da::Esi.new(ENV['MODE'])
  granuleid = @test_granule.granuleid
  esdt  = @test_collection.esdt

  @response_no_subset_xml = @esi.submit_request(esdt, granuleid, @service_agent, format:@format)
  puts @esi.esi_service_request_url
end

When(/^Submit a ESI "([^"]*)" request for the test granule for the "([^"]*)" and "([^"]*)"$/) do
|agent,esdt,format|
  @service = "esi"
  @format = format.strip
  @service_agent = agent.strip
  @mode =  ENV['MODE']
  @response_granule_xmls  = Hash.new()
  @esi = Sdps::Da::Esi.new(@mode)
  granuleid = @test_granule.granuleid
  esdt  = @test_collection.esdt

  @response_no_subset_xml = @esi.submit_request(esdt, granuleid, @service_agent, format:@format)
  puts @esi.esi_service_request_url
end

Then(/^the ESI "([^"]*)" request should succeed for the test granule for "([^"]*)" and "([^"]*)" format$/) do
|agent, esdt, format|
  @esi_response = @esi.parse_response(@response_no_subset_xml)
  #p @esi_response
  msg =  @esi_response.message.to_s

  msg.should eql "SUCCESS"
```

```
  @test_results = @esi_response.urls
end

Then(/^the ESI service request succeeds for the SMAP  test granule "([^"]*)" and "([^"]*)"$/) do |testgranule,
format|

  @esi_response = @esi.parse_response(@response_no_subset_xml)
  #p @esi_response
  msg =  @esi_response.message.to_s

  msg.should eql "SUCCESS"
  @test_results = @esi_response.urls
end

Then(/^the ESI "([^"]*)" request should succeed for the h5 test granule for "([^"]*)" and "([^"]*)" format$/) do
|agent, esdt, format|

  @esi_response = @esi.parse_response(@response_no_subset_xml)
  #p @esi_response
  msg =  @esi_response.message.to_s

  msg.should eql "SUCCESS"
  @test_results = @esi_response.urls
end

Then(/^the ESI processed files for the test granule should appear in the results in the requested format$/) do
  outputs1 = Array.new
  outputs = @test_results.split("\n")
  outputs.each do |output|
    output.strip!
    if !output.nil? and output.length > 0
      outputs1.push(output)
      File.file?(output).should be_truthy, "#{output} file does not exist"
    end
  end
  @test_outputs = outputs1
```

```
end

Then(/^the format of the output file for the test granule from ESI is valid$/) do
  utils =  Sdps::Da::DaUtil.new()
  failed_validation_results = Array.new
  puts "format=#{@format}"
  # format is no reformatting
  if @format.nil? or @format.empty?
    extension = File.extname(@test_hdf)
    puts "extension=#{extension}"
  else
    #assuming the format is GTiff. so the output file should be tif
    extension = Sdps::Da::FORMAT_EXTENSIONS[@format]
    #puts extension
  end
  #p @test_outputs
  @formatted_outputs = @test_outputs.select{ |outputfile| File.extname(outputfile).eql?(extension)}
  #p @formatted_outputs
  #formatted outputs - check for Gdal and for tif or kml outputs.. = no. of bands..
  #@formatted_outputs.length.should eq @test_outputs.length        #, "outputs are not for requested format"

  if @format.nil? or @format.empty?
    format1 = "NO_REFORMAT"
  else
    format1 = @format.gsub("-", "_")
  end

  if @projection.nil? or @projection.empty?
    projection1 = "NO_CHANGE"
  else
    projection1 = @projection.gsub(" ", "_")
  end

  if @interpolation.nil? or @interpolation.empty?
    interpolation1 = "NO_INTERPOLATION"
  else
```

```
    interpolation1 = @interpolation
  end

  if @spatial_subset.nil? or @spatial_subset.empty?
    subset = nil
  else
    subset = @spatial_subset.tr('-', '_')
    arr = subset.split(',')
    arr[0] = "#{arr[0]}W"
    arr[1] = "#{arr[1]}S"
    arr[2] = "#{arr[2]}E"
    arr[3] = "#{arr[3]}N"
    subset = "#{arr[0]}#{arr[1]}#{arr[2]}#{arr[3]}"
    puts "spatial_subset dir = #{subset}"
  end

  runtime = Time.now.strftime("%Y_%m_%d_%H_%M_%9N")
  if subset.nil?
    dest_path =
"/workingdata/cucumber/#{@mode}/#{@service_agent}/#{@test_collection.esdt}/#{format1}/#{projection1}/#{interpola
tion1}/NO_SUBSET/#{runtime}"
  else
    dest_path =
"/workingdata/cucumber/#{@mode}/#{@service_agent}/#{@test_collection.esdt}/#{format1}/#{projection1}/#{interpola
tion1}/#{subset}/#{runtime}"
  end


  @saved_outputs = Array.new
  @formatted_outputs.each  { | output_file |
    #puts "#{output_file}"
    #puts "#{@dest_path}"
    utils.copy_with_path(output_file, dest_path)
    dest_file = "#{dest_path}/#{File.basename(output_file)}"
    #if @format.eql?("GeoTIFF")
    #    @gdal = Sdps::Da::Gdal.new(@mode)
```

```
    #    #create a png file.
    #    @gdal.run_gdal_translate(dest_file, "PNG")
    #end
    if !File.extname(dest_file).eql?(extension)
      failed_validation_results.push(dest_file)
    end
    @saved_outputs.push(dest_file)
    puts "destination: #{dest_file}"
  }
end


Then(/^all the bands in the test granule that are not excluded should appear in the output file$/) do
  utils =  Sdps::Da::DaUtil.new()
  failed_validation_results = Array.new
  case @format
  when "KML"
  when "GeoTIFF"
    @test_bands.each do | band |
      puts band
    end
    @saved_outputs.each do |output_file |
      #geotiff has a file for each band output
      #validation_result = utils.geotiff_contains_bands(@saved_outputs, @test_bands)
      puts output_file
    end

  when "ASCII"
    output_file = @saved_outputs[0]
    validation_result = utils.opendap_ascii_contains_bands(output_file, @test_bands)
    #utils.opendap_ascii_contains_band(output_file, band)
    if validation_result == false
      failed_validation_results.push(utils.error_msg)
    end
  when "NetCDF-3"
    output_file = @saved_outputs[0]
```

```
    #utils.opendap_netcdf3_contains_band(output_file, band)
    validation_result = utils.opendap_netcdf3_contains_bands(output_file, @test_bands)
    if validation_result == false
      failed_validation_results.push(utils.error_msg)
    end
  when "HDF-EOS5",  "NO_FORMAT",  "" ,"NetCDF4-CF", "NetCDF-4"
    output_file = @saved_outputs[0]
    #validation_result = utils.h5output_contains_band(output_file, band)
    #ICE SAT spatial subset
    output_file.should_not be_nil
    output_file.should_not be_empty
    puts @test_bands
    if utils.output_contains_bands(output_file, @test_bands) == false
      failed_validation_results.push(utils.error_msg)
    end

  end

  failed_validation_results.should be_empty, "Missing bands in output - #{failed_validation_results.to_s}"
end

Then(/^all the attributes in the test granule file should appear in the output file$/) do
  utils =  Sdps::Da::DaUtil.new()
  failed_validation_results = Array.new
  case @format
  when "ASCII"
    output_file = @saved_outputs[0]
    # no attributes in ascii output
    validation_result = utils.validate_ascii_output_attributes(@test_hdf, output_file)
    puts ("expected attr =#{utils.input_arr.length}, result attr =#{utils.output_arr.length}")
    if validation_result.nil? or validation_result == false
      failed_validation_results.push("expected attr len=#{utils.input_arr.length}, result attr
len=#{utils.output_arr.length}, input file = #{@test_hdf}, output file= #{output_file}, #{utils.error_msg}")
    end
  when "NetCDF-3"
    output_file = @saved_outputs[0]
```

```
    validation_result = utils.validate_netcdf3_output_attributes(@test_hdf, output_file)
    puts ("expected attr =#{utils.input_arr.length}, result attr =#{utils.output_arr.length}")
    if validation_result.nil? or validation_result == false
      failed_validation_results.push("expected attr len=#{utils.input_arr.length}, result attr
len=#{utils.output_arr.length}, input file = #{@test_hdf}, output file= #{output_file}, #{utils.error_msg}")
    end

  when  "HDF-EOS5",  "NO_FORMAT",  "","NetCDF4-CF", "HDF-EOS"
    output_file = @saved_outputs[0]
    output_file.should_not be_nil, "no outputs generated"
    #if @service_agent.eql?("ICESAT2")
    #zero_length_bands = utils.get_zerolength_bands(@test_hdf)
    # if zero_length_bands.nil? or zero_length_bands.empty?
    #                     #all attributes
    validation_result = utils.validate_h5_output_attributes(@test_hdf, output_file)
    if !utils.input_arr.nil? and !utils.output_arr.nil?
       puts ("expected attr =#{utils.input_arr.length}, result attr =#{utils.output_arr.length}")
    end
    if validation_result.nil? or validation_result == false
      if !utils.input_arr.nil? and !utils.output_arr.nil?
        failed_validation_results.push("expected attr len=#{utils.input_arr.length}, result attr
len=#{utils.output_arr.length}, input file = #{@test_hdf}, output file= #{output_file}, #{utils.error_msg}")
      end
  else
    #if there are missing bands in the output, the number of attributes in the output will not match..
    #attributes for each band
    #TBD later. works. but has an issue when we add custom attributes in the output
    #@test_bands.each do | band |
    #  validation_result = utils.validate_h5_output_attributes_for_band(@test_hdf, output_file,band)
    #  if validation_result.nil? or validation_result == false
    #    puts ("band =#{band}, expected attr=#{utils.input_arr[0]}, result attr=#{utils.output_arr[0]}")
    #    failed_validation_results.push("#{output_file} : #{band} - utils.error_msg")
    #  end
   # end
  end
  end
```

```
  #else
  #HEG  or other subsetterss

  # end
  #end
  failed_validation_results.should be_empty, "failed validation - #{failed_validation_results.to_s}"
end

Then(/^the data in the output file should match the data in the test granule$/) do
  utils =  Sdps::Da::DaUtil.new()
  failed_validation_results = Array.new
  #@test_bands.each do | band |
  #    @formatted_outputs.each do |output_file |
  #replace slash with underscore
  #          band1 = band.gsub("/", "_")
  #dest_path =
"/workingdata/cucumber/#{@mode}/#{@service}_#{@agent}/#{@test_collection.esdt}/#{band1}/#{@format}/"
  #@utils.copy_with_path(output_file, dest_path)
  #dest_file = "#{dest_path}#{File.basename(output_file)}"
  #puts dest_file
  case @format
  when "ASCII"
    output_file = @saved_outputs[0]
    @test_bands.each do | band |
      validation_result = utils.validate_ascii_output_data(@test_hdf, output_file, band)
      puts ("band =#{band}, expected data=#{utils.input_arr.to_s}, result data=#{utils.output_arr.to_s}")
      if validation_result.nil? or validation_result == false
        failed_validation_results.push( "#{output_file} : #{band}")
      end
    end
  when "NetCDF-3"
    output_file = @saved_outputs[0]
    @test_bands.each do | band |
      validation_result = utils.validate_netcdf3_output_data(@test_hdf, output_file, band)
      puts ("band =#{band}, expected data=#{utils.input_arr.to_s}, result data=#{utils.output_arr.to_s}")
      if validation_result.nil? or validation_result == false
```

```
        failed_validation_results.push( "#{output_file} : #{band}")
      end
    end

  when "NetCDF4-CF", "HDF-EOS5",  "NO_FORMAT", "", "HDF_EOS","NetCDF-4"
    output_file = @saved_outputs[0]
    @test_bands.each do | band |
      validation_result = utils.validate_output_data(@test_hdf, output_file, band)
      puts ("band =#{band}, expected data=#{utils.input_arr.to_s}, result data=#{utils.output_arr.to_s}")
      if validation_result.nil? or validation_result == false
        failed_validation_results.push( "#{output_file} : #{band}")
      end
    end
  end
  failed_validation_results.should be_empty, "failed validation - #{failed_validation_results.to_s}"
end


Then(/^the size of the output file is comparable to the input file$/) do
  utils =  Sdps::Da::DaUtil.new()
  #case @format
  #when "HDF-EOS5", "NO_FORMAT", ""
  output_file = @saved_outputs[0]
  #assuming output is not greater than 90-100% of the input
  utils.filesizes_comparable?(@test_hdf, output_file, 0.9).should be_truthy, "#{utils.error_msg}"
  puts "#{utils.error_msg}"
  #end
endCurrent Test Script: esi_smap_spatial_subset_geotiff.rb
When(/^Submit a ESI "(.*?)" request for the SMAP test granule "(.*?)" for spatial subset "(.*?)" and GeoTIFF
format$/) do |agent, testgranule, spatialsubset|
    # #TODO check input parameters, - remove spaces, remove spaces in spatial
    # subset params
    @service = "esi"
    @format = "GeoTIFF"
    @service_agent = agent.strip
    @spatial_subset = spatialsubset.strip
```

```
    # #puts "agent=#{@agent}" #puts "service=#{@service}" #puts
    # "format=#{@format}" #puts "subset=#{@spatial_subset}"

    @response_granule_xmls  = Hash.new()
    @esi = Sdps::Da::Esi.new(@mode)
    granuleid = @test_granule.granuleid
    esdt  = @test_collection.esdt
    dataset_id = @test_collection.datasetid
    shortname = @test_collection.shortname
    #@response_spatial_subset_xml = @esi.submit_service_request(granuleid, esdt, dataset_id, "",
@spatial_subset, @format, @agent)
    @response_spatial_subset_xml = @esi.submit_request(esdt, granuleid, @service_agent, format:@format,
bbox:@spatial_subset)
    puts @esi.esi_service_request_url
end

Then(/^the ESI service "(.*?)" request succeeds for the SMAP  test granule "(.*?)" for spatial subset "(.*?)"
and GeoTIFF format$/) do |agent, testgranule, spatialsubset|
    collection = @test_collection
    esdt = collection.esdt
    @esi_response = @esi.parse_response(@response_spatial_subset_xml)
    # p @esi_response
    msg =  @esi_response.message.to_s

    msg.should eql "SUCCESS"
    @test_results = @esi_response.urls
    p @test_results
     @test_results.should_not be_nil, "No Download Urls"
end

Then(/^the ESI converted files for each band for the test granule "(.*?)" should appear in the results in the
requested GeoTIFF format$/) do |arg1|
    outputs1 = Array.new
    outputs = @test_results.split("\n")
    outputs.each do |output|
```

```
        output.strip!
        if !output.nil? and output.length > 0
            outputs1.push(output)
            File.file?(output).should be_truthy, "#{output} file does not exist"
        end
    end
    @test_outputs = outputs1

    extension = Sdps::Da::FORMAT_EXTENSIONS[@format]

    @formatted_outputs = @test_outputs.select{ |outputfile| File.extname(outputfile).eql?(extension)}

end

Then(/^the GeoTIFF output file for each band for the test granule "(.*?)" from ESI is valid for the requested
"(.*?)"$/) do |testgranule, spatialsubset|
    @gdal = Sdps::Da::Gdal.new(@mode)
    # #puts "output from gdalinfo...."
    delta = 0.05
    failed_subset = Array.new()
    success_subset = Array.new()
    subset = @spatial_subset.tr('-', '_')
    arr = subset.split(',')
    arr[0] = "#{arr[0]}W"
    arr[1] = "#{arr[1]}S"
    arr[2] = "#{arr[2]}E"
    arr[3] = "#{arr[3]}N"
    subset = "#{arr[0]}#{arr[1]}#{arr[2]}#{arr[3]}"

    @formatted_outputs.each  { | output_file |
        spatial_extent = @gdal.get_spatial_extent(output_file)
        #puts "#{spatial_extent} - #{output_file}"
        subset_arr = @spatial_subset.split(',')
        #projection_info =  @gdal.get_projection_info(output_file)
        proj4_params = @gdal.get_proj4_params(output_file)
        proj4_params.should_not be_nil
```

```
        puts "params: #{proj4_params}"
        if(proj4_params.include?("cea"))

            if (@dautil.spatial_extent_match?(subset_arr, spatial_extent,delta) == false)
                #add to the failed list
                puts  "Failure : #{output_file} : input=#{subset}, output=#{spatial_extent.to_s}"
                failed_subset.push( "#{output_file} : input=#{subset}, output=#{spatial_extent.to_s}")
            else
                puts  "Success : #{output_file} : input=#{subset}, output=#{spatial_extent.to_s}"
            end
        elsif (proj4_params.include?("laea"))

               #get modified input wsen
               modified_input_wsen = @dautil.get_modified_input_subset(subset_arr, proj4_params)
               #convert back to degrees

              if(@dautil.spatial_extent_match?(modified_input_wsen, spatial_extent, delta) == false)
                # #add to the failed list
                puts  "Failure : #{output_file} : orig input=#{subset}, modified input=#{modified_input_wsen},
output=#{spatial_extent.to_s}"
                failed_subset.push( "#{output_file} : input=#{subset}, output=#{spatial_extent.to_s}")
            else
                puts  "Success : #{output_file} :orig input=#{subset}, modified input=#{modified_input_wsen},
output=#{spatial_extent.to_s}"
            end
        else
            puts "proj4 params could not be determined"
        end
    }
    # #copy the files into the workingdata location for manual validation #puts
    # subset
    @output_format = @format
    # #puts @format #puts @output_format
    dest_path =
"/workingdata/cucumber/#{@mode}/#{@service}_#{@agent}/#{@test_collection.esdt}/#{subset}/#{@output_format}/"
```

```
    @formatted_outputs.each  { | output_file |
        @dautil.copy_with_path(output_file, dest_path)
        dest_file = "#{dest_path}#{File.basename(output_file)}"
        #create a png file.

        #@gdal.run_gdal_translate(dest_file, "PNG")
    }
    failed_subset.should be_empty, "failed subsets: #{failed_subset.to_s}"
end
Current Test Script: esi_reprojection.rb
When(/^Submit a ESI "([^"]*)" request without the excluded bands for each test granule "([^"]*)" and "([^"]*)"
format and spatialsubset "([^"]*)" and "([^"]*)" reprojection and "([^"]*)" param$/) do |agent, testgranule,
fmt, spatialsubset, reprojection, param|
  @service = "esi"
  @format = fmt.strip
  @service_agent = agent.strip
  @spatial_subset = spatialsubset.strip
  @projection = reprojection.strip
  @projection_param = param.strip
  #puts "format=#{@format}, projection=#{@projection}, param=#{param}"

  @response_granule_xmls  = Hash.new()
  @esi = Sdps::Da::Esi.new(ENV['MODE'])
  granuleid = @test_granule.granuleid
  esdt  = @test_collection.esdt
  puts "bands excluded=#{@bands_excluded}"
  if !@bands_excluded.nil?  and  !@bands_excluded.empty?
    band_str = @test_bands.join(",")
    @response_no_subset_xml = @esi.submit_request(esdt, granuleid, @service_agent, format:@format,
bbox:@spatial_subset, band_parameter:band_str,  projection:@projection, projection_param:@projection_param)
  else
    @response_no_subset_xml = @esi.submit_request(esdt, granuleid, @service_agent, format:@format,
bbox:@spatial_subset, projection:@projection, projection_param:@projection_param)
  end
  puts @esi.esi_service_request_url
```

```
end

Then(/^the ESI "([^"]*)" request should succeed for the test granule "([^"]*)" and "([^"]*)" format and
spatialsubset "([^"]*)"  and "([^"]*)"  reprojection and "([^"]*)" param$/) do |agent, testgranule, format,
spatialsubset, projection, param|
  esdt = @test_collection.esdt
  @esi_response = @esi.parse_response(@response_no_subset_xml)
  #p @esi_response
  msg =  @esi_response.message.to_s
  msg.should eql "SUCCESS"
  @test_results = @esi_response.urls
  p @test_results
  @test_results.should_not be_nil, "No Download Urls"
end


When(/^Submit a ESI "([^"]*)" request for each test granule "([^"]*)" for each band parameter and "([^"]*)"
format and spatialsubset "([^"]*)" and "([^"]*)" reprojection and "([^"]*)" param$/) do |agent, testgranule,
fmt, spatialsubset, reprojection, param|
  @service = "esi"
  @format = fmt.strip
  @service_agent = agent.strip
  @spatial_subset = spatialsubset.strip
  @projection = reprojection.strip
  @projection_param = param.strip
  @response_granule_xmls  = Hash.new()
  mode = ENV['MODE']
  @esi = Sdps::Da::Esi.new(mode)
  granuleid = @test_granule.granuleid
  esdt  = @test_collection.esdt
  @response_band_xmls = Hash.new()
  puts "testbands=#{@test_bands}"

  @test_bands.each { |band|
    #prepend band with shortname
    band1 = "#{band}"
```

```
    #remove leading slash and replace / with :
    if band1[0] ==  '/'
      band1 = band[1..-1]
    end
    band1.gsub("/", ":")
    response_xml = @esi.submit_request(esdt, granuleid, @service_agent, format:@format, bbox:@spatial_subset,
band_parameter:band, projection:@projection, projection_param:@projection_param)
    puts @esi.esi_service_request_url
    @response_band_xmls[band] = response_xml
    #puts response_xml
    puts @response_band_xmls[band]

  }
end

Then(/^the ESI "([^"]*)" should succeed for the test granule "([^"]*)" and band parameter and "([^"]*)" format
and spatialsubset "([^"]*)" and "([^"]*)" reprojection and "([^"]*)" param$/) do |arg1, arg2, arg3, arg4, arg5,
arg6|
  #Then(/^the ESI "([^"]*)" should succeed for the test granule "([^"]*)" and band parameter and "([^"]*)"
format and  spatialsubset "([^"]*)" and "([^"]*)" reprojection and "([^"]*)" param$/) do |agent, testgranule,
format, spatialsubset, projection, param|
  collection = @test_collection
  esdt = collection.esdt
  @response_results = Hash.new()
  failed_requests = Array.new
  @response_band_xmls.keys.each do |band|
    response_xml = @response_band_xmls[band]
    response = @esi.parse_response(response_xml)
    p response
    msg = response.message.to_s
    puts "#{esdt}, #{@format},  #{@projection}, #{@projection_param}, #{band}, #{msg}"
    @response_results[band] = response.urls
    puts response.urls
    puts @response_results[band]
    if msg.eql?("FAILURE")
      failed_requests.push ("#{esdt}, #{@format}, #{@projection}, #{@projection_param}, #{band}, #{msg}")
```

```
    end
    puts band
    puts @response_results[band]
  end
  failed_requests.should be_empty, failed_requests.to_s
end

Then(/^the ESI formatted and reprojected files for each band parameter for the test granule "(.*?)" should
appear in the results in the requested "(.*?)" format$/) do |arg1, arg2|
  @formatted_outputs = Hash.new
  @response_results.keys.each do | band |
    response = @response_results[band]
    puts band
    puts "response=#{response}"
    outputs1 = Array.new
    outputs = response.split("\n")
    puts outputs
    outputs.should_not be_empty

    outputs.each do |output|
      output.strip!
      if !output.nil? and output.length > 0
        outputs1.push(output)
        File.file?(output).should be_truthy, "#{output} file does not exist"
      else
        puts "output=#{output} exists"
      end
    end
    test_outputs = outputs1

    if @format.nil? or @format.empty?
      extension = File.extname(@test_hdf)
    else
      extension = Sdps::Da::FORMAT_EXTENSIONS[@format]
    end
    puts extension
```

```
    format_outputs = test_outputs.select{ |outputfile| File.extname(outputfile).eql?(extension)}
    @formatted_outputs[band] = format_outputs
  end
  puts @formatted_outputs


end

Then(/^the "([^"]*)" format of the formatted and reprojected output file for the test granule "([^"]*)" from ESI
is valid$/) do |format, testgranule|
  #puts "format=#{@format}"
  # format is no reformatting
  if @format.nil? or @format.empty?
    extension = File.extname(@test_hdf)
    #puts "extension=#{extension}"
  else
    #assuming the format is GTiff. so the output file should be tif
    extension = Sdps::Da::FORMAT_EXTENSIONS[@format]
    #puts extension
  end
  #p @test_outputs
  @formatted_outputs = @test_outputs.select{ |outputfile| File.extname(outputfile).eql?(extension)}
  #p @formatted_outputs
  #formatted outputs - check for Gdal and for tif or kml outputs.. = no. of bands..
  #@formatted_outputs.length.should eq @test_outputs.length        #, "outputs are not for requested format"

  if @format.nil? or @format.empty?
    format1 = "NO_REFORMAT"
  else
    format1 = @format.gsub("-", "_")
  end

  if @projection.nil? or @projection.empty?
    projection1 = "NO_CHANGE"
  else
    projection1 = @projection.gsub(" ", "_")
```

```
  end

  if @projection_param.nil? or @projection_param.empty?
    projection_param1 = "NO_PARAM"
  else
    projection_param1 = @projection_param.gsub(" ", "_")
  end

  if @interpolation.nil? or @interpolation.empty?
    interpolation1 = "NO_INTERPOLATION"
  else
    interpolation1 = @interpolation
  end

  if @spatial_subset.nil? or @spatial_subset.empty?
    subset = nil
  else
    subset = @spatial_subset.tr('-', '_')
    arr = subset.split(',')
    arr[0] = "#{arr[0]}W"
    arr[1] = "#{arr[1]}S"
    arr[2] = "#{arr[2]}E"
    arr[3] = "#{arr[3]}N"
    subset = "#{arr[0]}#{arr[1]}#{arr[2]}#{arr[3]}"
    puts "spatial_subset dir = #{subset}"
  end
  mode = ENV['MODE']
  runtime = Time.now.strftime("%Y_%m_%d_%H_%M_%6N")
  if subset.nil?
    dest_path =
"/workingdata/cucumber/#{mode}/#{@test_collection.esdt}/#{format1}/#{projection1}/#{interpolation1}/#{projection
_param1}/NO_SUBSET/#{runtime}"
  else
    dest_path =
"/workingdata/cucumber/#{mode}/#{@test_collection.esdt}/#{format1}/#{projection1}/#{interpolation1}/#{projection
_param1}/#{subset}/#{runtime}"
```

```
  end

  #dest_path =
"/workingdata/cucumber/#{mode}/#{runtime}_#{@service}_#{@service_agent}_#{@test_collection.esdt}/NO_SUBSET_#{for
mat1}_#{projection1}"
  #puts dest_path
  utils = Sdps::Da::DaUtil.new()
  @formatted_outputs.each  { | output_file |
    #puts "#{output_file}"
    #puts "#{dest_path}"
    utils.copy_with_path(output_file, dest_path)
    dest_file = "#{dest_path}/#{File.basename(output_figure)}"
    puts "#{output_file}"
    puts "#{dest_file}"
    #if @format.eql?("GeoTIFF")
    #    @gdal = Sdps::Da::Gdal.new(@mode)
    #    #create a png file.
    #    @gdal.run_gdal_translate(dest_file, "PNG")
    # end
  }
endCurrent Test Script: esi_spatial_subset.rb

When(/^Submit a ESI "(.*?)" request for  "(.*?)" and "(.*?)"$/) do |agent, fmt, spatialsubset|
      #TODO
     # check input parameters, - remove spaces, remove spaces in spatial subset params
     @service = "esi"
     @format = fmt.strip
     @agent = agent.strip
     @spatial_subset = spatialsubset.strip

     #puts "agent=#{@agent}"
     #puts "service=#{@service}"
     #puts "format=#{@format}"
     #puts "subset=#{@spatial_subset}"

     @response_granule_xmls  = Hash.new()
```

```
     @esi = Sdps::Da::Esi.new(@mode)
             granuleid = @test_granule.granuleid
             esdt  = @test_collection.esdt
             dataset_id = @test_collection.datasetid
             shortname = @test_collection.shortname
             @response_no_subset_xml = @esi.submit_service_request(granuleid, esdt, dataset_id, "",
@spatial_subset, @format, @agent)
end

Then(/^the ESI service request succeeds for the SMAP  test granule and format with spatial subset$/) do
     collection = @test_collection
     esdt = collection.esdt
        @esi_response = @esi.parse_response(@response_no_subset_xml)
       # p @esi_response
        msg =  @esi_response.message.to_s
        msg.should eql "SUCCESS"
        @test_results = @esi_response.urls
       #p @test_results
end

#for ICESAT spatial subset - all band request
Then(/^the output file for each "([^"]*)" for the test granule "([^"]*)" from ESI is valid for the requested
"([^"]*)"$/) do |geolocationband, granule,spatialsubset|

     geolocation_bands_arr = geolocationband.split(",")
     @geolocation_bands = Array.new()
     @test_bands.each do | band |
          geolocation_bands_arr.each do |geoband|
             if band.include?(geoband)
                 @geolocation_bands.push(band)
             end
         end
     end
     puts "geolocation bands:#{@geolocation_bands}"
     failed_validation_results = Array.new
     puts "subset=#{@spatial_subset}"
```

```
     puts "agent=#{@agent}"
        utils =  Sdps::Da::DaUtil.new()
     @zero_length_geolocation_bands = Array.new
     bands_to_exclude = Array.new
     @saved_outputs.each {|dest_file|
           case @format
           when "HDF-EOS5", "NO_REFORMAT", ""
             puts "NO_REFORMAT"
             spatial_subset_hash = Hash.new

             subset_arr = @spatial_subset.split(',')
             spatial_subset_hash[:w] = subset_arr[0] .to_f
             spatial_subset_hash[:s] = subset_arr[1] .to_f
             spatial_subset_hash[:e] = subset_arr[2] .to_f
             spatial_subset_hash[:n] = subset_arr[3] .to_f
             #h5dump lat/lon
               puts "expected=#{@spatial_subset}"
               #get the geolocation bands

              @geolocation_bands.each do | band |
              #get lat-lon bands first
              #if band.include?("lat") or band.include?("lon")
              #    geolocation_bands.push(band)
               if utils.h5output_contains_band(dest_file, band)
                   validation_result = utils.validate_h5_spatial_subset_data(@test_hdf, dest_file, band,
spatial_subset_hash)
                   if utils.output_arr.nil? or utils.output_arr.empty?
                       len=0
                       puts ("geolocation band =#{band}, data-len: #{len}")
                        @zero_length_geolocation_bands.push(band)
                        #get the group name
                        band_group = band.slice(0..band.rindex("/"))
                        bands_to_exclude += @test_bands.select { |group| group.include?(band_group)}
                   else
                       len= utils.output_arr.length
#                       puts ("geolocation band =#{band}, data-len: #{len}, result=#{utils.output_arr[0..10]}")
```

```
                    end
                    if validation_result.nil? or validation_result == false
                         puts "Failed spatial subset validation #{band} :  #{@error_msg}"
                         failed_validation_results.push( "dest_file:#{dest_file} : band =#{band}, len=#{len},
result=#{utils.output_arr[0..10]}")
                     else
                         puts "Success - spatial subset validation: #{band}"
                     end
               end #do band
               end
               puts "Bands EXCLUDED due to spatial subset=#{bands_to_exclude}"
               @test_bands = @test_bands - bands_to_exclude
            end #case
     }
     puts failed_validation_results.to_s
     failed_validation_results.should be_empty
end

 #update it for SMAP and HEG services
Then(/^the "([^"]*)" output file for the test granule "([^"]*)" from ESI is valid for the requested
spatialsubset "([^"]*)"$/) do |fmt, granule,spatialsubset|
     #geolocation_bands = Array.new
     failed_validation_results = Array.new
     puts "subset=#{@spatial_subset}"
     puts "agent=#{@agent}"
     @saved_outputs.each {|dest_file|
            utils =  Sdps::Da::DaUtil.new()
            case @format
            when "HDF-EOS5", "NO_REFORMAT", ""
              puts "NO_REFORMAT"
              spatial_subset_hash = Hash.new

              subset_arr = @spatial_subset.split(',')
              spatial_subset_hash[:w] = subset_arr[0] .to_f
              spatial_subset_hash[:s] = subset_arr[1] .to_f
              spatial_subset_hash[:e] = subset_arr[2] .to_f
```

```
             spatial_subset_hash[:n] = subset_arr[3] .to_f
             #h5dump lat/lon
               puts "expected=#{@spatial_subset}"
              @test_bands.each do | band |
              #get lat-lon bands first
              if band.include?("lat") or band.include?("lon")
                  #geolocation_bands.push(band)
                   validation_result = utils.validate_h5_spatial_subset_data(@test_hdf, dest_file, band,
spatial_subset_hash)
                   if utils.output_arr.nil?
                       len=0
                   else
                       len= utils.output_arr.length
                   end
                   puts ("geolocation band =#{band}, data-len: #{len}, result=#{utils.output_arr}")
                    if validation_result.nil? or validation_result == false
                        puts "Failed spatial subset validation"
                        failed_validation_results.push( "dest_file:#{dest_file} : band =#{band}, len=#{len},
result=#{utils.output_arr}")
                    else
                        puts "Success - spatial subset validation"
                    end
               else
                   length = utils.get_h5_data_len_for_band(dest_file, band)
                   puts ("not a geolocation band =#{band}, data len=#{length}")
               end  #lat-lon check
              end #do band
           end #case
     }
     #remove zero length bands from the testbands
     @test_bands
     failed_validation_results.should be_empty
end

When(/^Submit a ESI "([^"]*)" request for the test granule "([^"]*)" for spatial subset "([^"]*)" and "([^"]*)"
format$/) do |agent, granule, spatialsubset, fmt|
```

```
                    @service = "esi"
            @format = fmt.strip
            @service_agent = agent.strip
            @spatial_subset = spatialsubset.strip
            @response_granule_xmls  = Hash.new()
            @esi = Sdps::Da::Esi.new(@mode)
            granuleid = @test_granule.granuleid
            esdt  = @test_collection.esdt
            @response_band_xmls = Hash.new()
#            @test_bands.each { |band|
#                 #prepend band with shortname
#                 band1 = "#{band}"
#                #remove leading slash and replace / with :
#                 if band1[0] ==  '/'
#                      band1 = band[1..-1]
#                 end
#                 band1.gsub("/", ":")
                 @response_xml = @esi.submit_request(esdt, granuleid, @service_agent, format:@format,
bbox:@spatial_subset)
                 puts @esi.esi_service_request_url
                #@response_band_xmls[band] = response_xml

#            }
end

When(/^Submit a ESI "([^"]*)" request for the h5 test granule "([^"]*)" for each band parameter and "([^"]*)"
format and spatialsubset "([^"]*)"$/) do |agent, granule, fmt, spatialsubset|
            @service = "esi"
            @format = fmt.strip
            @service_agent = agent.strip
            @spatial_subset = spatialsubset.strip
            @response_granule_xmls  = Hash.new()
            @esi = Sdps::Da::Esi.new(@mode)

             puts "testing - #{@mode}"
            granuleid = @test_granule.granuleid
```

```
             esdt  = @test_collection.esdt
             @response_band_xmls = Hash.new()
             @test_bands.each { |band|
                  #prepend band with shortname
                  band1 = "#{band}"
                 #remove leading slash and replace / with :
                  if band1[0] ==  '/'
                       band1 = band[1..-1]
                  end
                  band1.gsub("/", ":")
                  response_xml = @esi.submit_request(esdt, granuleid, @service_agent, format:@format,
bbox:@spatial_subset, band_parameter:band)
                  puts @esi.esi_service_request_url
                 @response_band_xmls[band] = response_xml

             }
end

Then(/^the ESI service "([^"]*)" request should succeed for the test granule "([^"]*)" for spatial subset
"([^"]*)" and "([^"]*)" format$/) do |agent, granule, spatialsubset, fmt|
      collection = @test_collection
        esdt = collection.esdt
        #@response_results = Hash.new()
        failed_requests = Array.new
        #@response_band_xmls.keys.each do |band|
           #response_xml = @response_band_xmls[band]
           response = @esi.parse_response(@response_xml)
           p response
           msg = response.message.to_s
           puts "#{granule}, #{@format},  #{@spatial_subset}, #{msg}"
           #@response_results[band] = response.urls
           p response.urls
           if msg.eql?("FAILURE")
               failed_requests.push ("#{esdt}, #{@format}, #{@spatial_subset}, #{msg}")
           else
               @test_results = response.urls
```

```
          end
        #end
        failed_requests.should be_empty, failed_requests.to_s

end



Then(/^the ESI "([^"]*)" request should succeed for the h5 test granule "([^"]*)" and band parameter and
"([^"]*)" format and spatialsubset "([^"]*)"$/) do |agent, granule, fmt, spatialsubset|
        collection = @test_collection
        esdt = collection.esdt
        @response_results = Hash.new()
        failed_requests = Array.new
        @response_band_xmls.keys.each do |band|
           response_xml = @response_band_xmls[band]
           response = @esi.parse_response(response_xml)
           p response
           msg = response.message.to_s
           puts "#{granule}, #{@format},  #{@spatialsubset}, #{band}, #{msg}"
           @response_results[band] = response.urls
           p response.urls
           if msg.eql?("FAILURE")
               failed_requests.push ("#{esdt}, #{@format}, #{@spatialsubset}, #{band}, #{msg}")
           end
        end
        failed_requests.should be_empty, failed_requests.to_s

end

Then(/^the "([^"]*)" output file for each band for the test granule "([^"]*)" from ESI is valid for the
requested spatialsubset "([^"]*)"$/) do |fmt, granule, spatialsubset|
      utils =  Sdps::Da::DaUtil.new()
     if @spatial_subset.nil? or @spatial_subset.empty?
           subset = nil
         else
```

```
         subset = @spatial_subset.tr('-', '_')
         arr = subset.split(',')
         arr[0] = "#{arr[0]}W"
         arr[1] = "#{arr[1]}S"
         arr[2] = "#{arr[2]}E"
         arr[3] = "#{arr[3]}N"
         subset = "#{arr[0]}#{arr[1]}#{arr[2]}#{arr[3]}"
         puts "spatial_subset dir = #{subset}"
end
runtime = Time.now.strftime("%Y_%m_%d_%H_%M_%9N")
failed_validation_results = Array.new delta = 0.05
failed_subset = Array.new()
subset_arr = @spatial_subset.split(',')
    case @format
      when "GeoTIFF"
         @gdal = Sdps::Da::Gdal.new(@mode)
        @formatted_outputs.keys.each  do | band |
               output_file = @formatted_outputs[band]

         # #puts "output from gdalinfo...."
         spatial_extent = @gdal.get_spatial_extent(output_file)
         #puts "#{spatial_extent} - #{output_file}"
         #projection_info =  @gdal.get_projection_info(output_file)
         proj4_params = @gdal.get_proj4_params(output_file)
         proj4_params.should_not be_nil
        puts "params: #{proj4_params}"
        if(proj4_params.include?("cea"))
          if (utils.spatial_extent_match?(subset_arr, spatial_extent,delta) == false)
            #add to the failed list
            puts  "Failure : #{output_file} : input=#{subset}, output=#{spatial_extent.to_s}"
            failed_subset.push( "#{output_file} : input=#{subset}, output=#{spatial_extent.to_s}")
         else
           puts  "Success : #{output_file} : input=#{subset}, output=#{spatial_extent.to_s}"
         end
       elsif (proj4_params.include?("laea"))
          #get modified input wsen
```

```
                modified_input_wsen = utils.get_modified_input_subset(subset_arr, proj4_params)
                #convert back to degrees
                if(utils.spatial_extent_match?(modified_input_wsen, spatial_extent, delta) == false)
                  # #add to the failed list
                   puts  "Failure : #{output_file} : orig input=#{subset}, modified
input=#{modified_input_wsen}, output=#{spatial_extent.to_s}"
                   failed_subset.push( "#{output_file} : input=#{subset}, output=#{spatial_extent.to_s}")
               else
                   puts  "Success : #{output_file} :orig input=#{subset}, modified input=#{modified_input_wsen},
output=#{spatial_extent.to_s}"
               end
             else
               puts "proj4 params could not be determined"
             end
      end
          when "HDF-EOS5", "NO_REFORMAT", ""
             #puts "NO_REFORMAT"
             #h5dump lat/lon
             spatial_subset_hash = Hash.new
             subset_arr = @spatial_subset.split(',')
             spatial_subset_hash[:w] = subset_arr[0] .to_f
             spatial_subset_hash[:s] = subset_arr[1] .to_f
             spatial_subset_hash[:e] = subset_arr[2] .to_f
             spatial_subset_hash[:n] = subset_arr[3] .to_f
             puts "expected : #{@spatial_subset}"
             @saved_outputs.keys.each  do | band |
               dest_file = @saved_outputs[band]
               if band.include?("lat") or band.include?("lon")
                   validation_result = utils.validate_h5_spatial_subset_data(@test_hdf, dest_file,
band,spatial_subset_hash)
                   if utils.output_arr.nil?
                       len=0
                   else
                       len= utils.output_arr.length
                   end
                   puts ("geolocation band =#{band}, data-len: #{len}, result=#{utils.output_arr}")
```

```
                    if validation_result.nil? or validation_result == false
                        puts "Failed Spatial Validation"
                        failed_validation_results.push( "dest_file:#{dest_file}, band=#{band},datalen: #{len},
result=#{utils.output_arr}")
                    else
                        puts 'Success Spatial Validation'
                    end
               else
                   length = utils.get_h5_data_len_for_band(dest_file, band)
                   puts ("not a geolocation band =#{band}, data len=#{length}")
               end
             end
             failed_validation_results.should be_empty, "failed subsets: #{failed_validation_results.to_s}"
          end
end

When(/^Submit a ESI "([^"]*)" request for the h5 test granule "([^"]*)" and "([^"]*)" format and spatialsubset
"([^"]*)"$/) do |agent, granule, fmt, spatialsubset|
             @service = "esi"
             @format = fmt.strip
             @service_agent = agent.strip
             @spatial_subset = spatialsubset.strip
             @response_granule_xmls  = Hash.new()
             @esi = Sdps::Da::Esi.new(@mode)
             granuleid = @test_granule.granuleid
             esdt  = @test_collection.esdt
             @response_band_xmls = Hash.new()
             puts "bands excluded=#{@bands_excluded}"
             if !@bands_excluded.nil?  and  !@bands_excluded.empty?
                  band_str = @test_bands.join(",")
                  @response_xml = @esi.submit_request(esdt, granuleid, @service_agent, format:@format,
band_parameter:band_str,bbox:@spatial_subset)
             else
                    @response_xml = @esi.submit_request(esdt, granuleid, @service_agent, format:@format,
bbox:@spatial_subset)
             end
```

```
                  puts @esi.esi_service_request_url
                 #@response_band_xmls[band] = response_xml
end


Then(/^the ESI "([^"]*)" request should succeed for the h5 test granule "([^"]*)" and "([^"]*)" format and
spatialsubset "([^"]*)"$/) do |agent, granule, fmt, spatialsubset|

   collection = @test_collection
        esdt = collection.esdt
        #@response_results = Hash.new()
        failed_requests = Array.new
        #@response_band_xmls.keys.each do |band|
           #response_xml = @response_band_xmls[band]
           response = @esi.parse_response(@response_xml)
           p response
           msg = response.message.to_s
           puts "#{granule}, #{@format},  #{@spatial_subset}, #{msg}"
           #@response_results[band] = response.urls
           p response.urls
           if msg.eql?("FAILURE")
               failed_requests.push ("#{esdt}, #{@format}, #{@spatial_subset}, #{msg}")
           else
               @test_results = response.urls
           end
        #end
        failed_requests.should be_empty, failed_requests.to_s

end

Then(/^the data in the output file from "([^"]*)" service should match the data in the test granule for the
requested "([^"]*)" spatialsubset$/) do | agent, spatialsubset |
        #get the number of bands with coordinates attribute
        # those bands should all be subsetted and it should match the geolocation band range
           utils =  Sdps::Da::DaUtil.new()
        failed_validation_results = Array.new
```

```
       #@test_bands.each do | band |
       #    @formatted_outputs.each do |output_file |
                 #replace slash with underscore
       #          band1 = band.gsub("/", "_")
                #dest_path =
"/workingdata/cucumber/#{@mode}/#{@service}_#{@agent}/#{@test_collection.esdt}/#{band1}/#{@format}/"
                #@utils.copy_with_path(output_file, dest_path)
                #dest_file = "#{dest_path}#{File.basename(output_file)}"
                #puts dest_file
                case @format
#                when "ASCII"
#                    output_file = @saved_outputs[0]
#                    @test_bands.each do | band |
#                    validation_result = utils.validate_ascii_output_data(@test_hdf, output_file, band)
#                   puts ("band =#{band}, expected data=#{utils.input_arr[0]}, result
data=#{utils.output_arr[0]}")
#                    if validation_result.nil? or validation_result == false
#                       failed_validation_results.push( "#{output_file} : #{band}")
#                    end
#                    end
#                when "NetCDF-3"
#                    output_file = @saved_outputs[0]
#                    @test_bands.each do | band |
#                    validation_result = utils.validate_netcdf3_output_data(@test_hdf, output_file, band)
#                     puts ("band =#{band}, expected data=#{utils.input_arr[0]}, result
data=#{utils.output_arr[0]}")
#                    if validation_result.nil? or validation_result == false
#                       failed_validation_results.push( "#{output_file} : #{band}")
#                    end
#                    end
#              TO BE UPDATED...
                when "NetCDF4-CF", "HDF-EOS5",  "NO_FORMAT", ""
#                    check_spatially_subsetted_bands
#                    check_spatially_unsubsetted_bands
#                    output_file = @saved_outputs[0]
#                    @test_bands.each do | band |
```

```
#                     validation_result = utils.validate_h5_output_data(@test_hdf, output_file, band)
#                     puts ("band =#{band}, expected data=#{utils.input_arr[0]}, result
data=#{utils.output_arr[0]}")
#                     if validation_result.nil? or validation_result == false
#                        failed_validation_results.push( "#{output_file} : #{band}")
#                     end
                 end
#            end
      failed_validation_results.should be_empty, "failed validation - #{failed_validation_results.to_s}"
end
Current Test Script: esi_parameter_subset_reprojection.rb
Then(/^the "([^"]*)" format of the formatted and reprojected output file for each band parameter of the test
granule "([^"]*)" from ESI is valid for the requested band parameter$/) do |fmt, localgranuleid|
          @formatted_outputs.should_not be_nil
          @formatted_outputs.should_not be_empty
          # puts @format
          # puts output_files
          # puts @projection
         if @format.nil? or @format.empty?
            format1 = "NO_REFORMAT"
         else
            format1 = @format.gsub("-", "_")
         end
         if @projection.nil? or @projection.empty?
           projection1 = "NO_CHANGE"
        else
           projection1 = @projection.gsub(" ", "_")
        end

        if @interpolation.nil? or @interpolation.empty?
           interpolation1 = "NO_INTERPOLATION"
        else
            interpolation1 = @interpolation
        end

         if @spatial_subset.nil? or @spatial_subset.empty?
```

```
          subset = nil
        else
              subset = @spatial_subset.tr('-', '_')
              arr = subset.split(',')
              arr[0] = "#{arr[0]}W"
              arr[1] = "#{arr[1]}S"
              arr[2] = "#{arr[2]}E"
              arr[3] = "#{arr[3]}N"
              subset = "#{arr[0]}#{arr[1]}#{arr[2]}#{arr[3]}"
              puts "spatial_subset dir = #{subset}"
        end
          #puts format1
          #puts projection1
          runtime = Time.now.strftime("%Y_%m_%d_%H_%M_%6N")
          #puts runtime
         @dest_files = Array.new
         @formatted_outputs.keys.each  do | band |
         output_files = @formatted_outputs[band]
         output_files.each do |output_file |
                #replace slash with underscore
                band1 = band.gsub("/", "_")
                if subset.nil?
                       dest_path =
"/workingdata/cucumber/#{@mode}/#{@test_collection.esdt}/#{format1}/#{projection1}/#{interpolation1}/#{band1}/#{
runtime}"
                       #dest_path =
"/workingdata/cucumber/#{@mode}/#{runtime}_#{@service}_#{@service_agent}_#{@test_collection.esdt}/#{band1}_#{for
mat1}_#{projection1}"
                else
                       dest_path =
"/workingdata/cucumber/#{@mode}/#{@test_collection.esdt}/#{format1}/#{projection1}/#{interpolation1}/#{band1}/#{
subset}/#{runtime}"
                end
                utils = Sdps::Da::DaUtil.new()
                utils.copy_with_path(output_file, dest_path)
                dest_file = "#{dest_path}/#{File.basename(output_file)}"
```

```
                puts output_file
                puts dest_file
                @dest_files.push(dest_file)
                #case @format
                #when "ASCII"
                   # validation_result = @utils.validate_ascii_output(@test_hdf, output_file, band)
                    #if validation_result.nil? or validation_result == false
                     #  failed_validation_results.push( "#{output_file} : #{band}")
                    #end
                #when "NetCDF-3"
                    #@utils.validate_netcdf3_output(@test_hdf, output_file, band)
                #when "HDF-EOS5"
                   #@dautil.validate_hdfeos5_output(@test_hdf, output_file, band)
                #when "NetCDF4-CF"
                  #  @dautil.validate_netcdf4_output(@test_hdf, output_file, band)
                #when "GeoTIFF"
                 #      @gdal = Sdps::Da::Gdal.new(@mode)
                #      #create a png file.
                #       @gdal.run_gdal_translate(dest_file, "PNG")
                #end
            end
        end
        #failed_validation_results.should be_empty, "failed validation - #{failed_validation_results.to_s}"
end

Then(/^the ESI formatted and reprojected files for the test granule "([^"]*)" should appear in the results in
the requested "([^"]*)" format$/) do |testgranule, format|
         outputs1 = Array.new
         outputs = @test_results.split("\n")
         outputs.should_not be_nil, ("no outputs")
         outputs.should_not be_empty, ("no outputs")

         outputs.each do |output|
                output.strip!
                if !output.nil? and output.length > 0
                      outputs1.push(output)
```

```
                    File.file?(output).should be_truthy, "#{output} file does not exist"
                end
        end
        @test_outputs = outputs1
end
```

```
Then(/^the "([^"]*)" projection for the output file for each band parameter of the test granule "([^"]*)" is
valid$/) do |arg1, arg2|
  pending # Write code here that turns the phrase above into concrete actions
end

Then(/^the "(.*?)" output file for each band for the test granule "(.*?)" from ESI is valid for the requested
spatial subset "(.*?)"$/) do |fmt, testgranule, spatialsubset|
    if @spatial_subset.nil? or @spatial_subset.empty?
           subset = nil
    else
        if @format.eql?("GeoTIFF")
    @gdal = Sdps::Da::Gdal.new(@mode)
    # #puts "output from gdalinfo...."
    delta = 0.05
    failed_subset = Array.new()
    success_subset = Array.new()
    subset = @spatial_subset.tr('-', '_')
    arr = subset.split(',')
    arr[0] = "#{arr[0]}W"
    arr[1] = "#{arr[1]}S"
    arr[2] = "#{arr[2]}E"
    arr[3] = "#{arr[3]}N"
    subset = "#{arr[0]}#{arr[1]}#{arr[2]}#{arr[3]}"

    @dest_files.each  { | output_file |
```

```
        spatial_extent = @gdal.get_spatial_extent(output_file)
        #puts "#{spatial_extent} - #{output_file}"
        subset_arr = @spatial_subset.split(',')
        #projection_info =  @gdal.get_projection_info(output_file)
        proj4_params = @gdal.get_proj4_params(output_file)
        proj4_params.should_not be_nil
        puts "params: #{proj4_params}"
        if(proj4_params.include?("cea"))

            if (@dautil.spatial_extent_match?(subset_arr, spatial_extent,delta) == false)
                #add to the failed list
                puts  "Failure : #{output_file} : input=#{subset}, output=#{spatial_extent.to_s}"
                failed_subset.push( "#{output_file} : input=#{subset}, output=#{spatial_extent.to_s}")
            else
                puts  "Success : #{output_file} : input=#{subset}, output=#{spatial_extent.to_s}"
            end
        elsif (proj4_params.include?("laea"))

               #get modified input wsen
               modified_input_wsen = @dautil.get_modified_input_subset(subset_arr, proj4_params)
               #convert back to degrees

              if(@dautil.spatial_extent_match?(modified_input_wsen, spatial_extent, delta) == false)
                # add to the failed list
                puts  "Failure : #{output_file} : orig input=#{subset}, modified input=#{modified_input_wsen},
output=#{spatial_extent.to_s}"
                failed_subset.push( "#{output_file} : input=#{subset}, output=#{spatial_extent.to_s}")
            else
                puts  "Success : #{output_file} :orig input=#{subset}, modified input=#{modified_input_wsen},
output=#{spatial_extent.to_s}"
            end
        elsif (proj4_params.include?("longlat"))
             #GEOGRAPHIC projection

        else
            puts "proj4 params could not be determined"
```

```
        end
    }

    failed_subset.should be_empty, "failed subsets: #{failed_subset.to_s}"
        end
    end
end
Current Test Script: da_common.rb

Given(/^prefixed bands containing "([^"]*)" with "([^"]*)"$/) do |bands_updated_by_design, prefix|
  bands_str = bands_updated_by_design.strip
  test_bands1 = bands_str.split(",")
  test_bands1.each { | band1|
    band2 = band1.strip
    if @test_bands.include?(band2)
      index = @test_bands.index(band2)
      @test_bands[index] = prefix+band2
    end
  }
  #puts @test_bands.to_s
end

Given(/^does not include bands containing "([^"]*)"$/) do | bands_excluded_by_design |
  test_bands1 = bands_excluded_by_design.split(",")
  @bands_excluded = test_bands1
  test_bands1.each { | band1|
    @test_bands.delete_if { |band| band.include?(band1.strip)}
  }
  #puts @test_bands.to_s
end

Given(/^includes  "([^"]*)"$/) do |bands_required|
  #puts bands_required
  required_bands_arr = bands_required.strip.split(",")
  #p required_bands_arr
  @required_bands = Array.new()
```

```
  required_bands_arr.each { | band1|
    band2 =  @test_bands.select { |band| band.eql?(band1.strip)

    }
    #puts "band1=#{band1}"
    #puts "band=#{band}"
    #puts "band2=#{band2}"
    #puts band2[0]
    @required_bands.push(band2[0])
  }

  #p @required_bands
end

Given(/^not excluded bands retrieved successfully for granule "(.*?)" for ESI "(.*?)" and "(.*?)" request$/)  do
|localgranuleid, service_agent, format|

  @test_bands = Hash.new()
  filename = @test_hdf
  utils =  Sdps::Da::DaUtil.new()
  if service_agent.eql?("ICESAT2")
    all_bands = utils.get_non_zerolength_bands(@test_hdf)
  else
    all_bands= utils.get_band_params(filename)
  end
  #puts all_bands
  @bands_excluded = Array.new
  mode_checker = Sdps::Mode.new(ENV['MODE'])
  db_conn = mode_checker.get_db_connection()
  db_conn.should_not be_nil

  dadb = Sdps::Da::DaDb.new(db_conn, @mode)
  @bands_excluded = dadb.get_objects_excluded_for_collection_service_format(@test_collection.shortname,
@test_collection.versionid, service_agent, format)
  #puts bands_excluded
  @test_bands = Array.new
```

```
  @test_bands = all_bands
  if !@bands_excluded.nil?
    #@test_bands = all_bands.select { |band| !@bands_excluded.include?(band.strip) }
    @bands_excluded.each { | band1|
      @test_bands.delete_if { |band| band.include?(band1.strip)}
    }
    #puts "test"
    puts @test_bands
    #        else
    #             @test_bands = all_bands
  end

  #puts filename
  #puts @test_bands
end


# has to compound format and projection
Given(/^not excluded bands retrieved successfully for granule "([^"]*)" for ESI "([^"]*)" and "([^"]*)" and
"([^"]*)" reprojection$/) do |localgranuleid, service_agent, format,reprojection|

  format = format.strip
  service_agent = service_agent.strip
  reprojection = reprojection.strip

  #shortname and versionid ...??
  @test_bands = Hash.new()
  filename = @test_hdf
  utils =  Sdps::Da::DaUtil.new()
  if service_agent.eql?("ICESAT2")
    all_bands = @dautil.get_non_zerolength_bands(@test_hdf)
  else
    all_bands= utils.get_band_params(filename)
  end
  #puts "all=#{all_bands}"
  @bands_excluded = Array.new
```

```
  excluded1 = Array.new
  excluded2 = Array.new
  if !format.nil? and !format.empty?
    excluded1 = @dadb.get_objects_excluded_for_collection_service_format(@test_collection.shortname,
@test_collection.versionid, service_agent, format)
    #puts format
    #puts "excluded1=#{excluded1}"
  end
  if !reprojection.nil? and !reprojection.empty?
    excluded2 = (@dadb.get_objects_excluded_for_collection_service_projection(@test_collection.shortname,
@test_collection.versionid, service_agent, reprojection))
    #puts reprojection
    #puts "excluded2=#{excluded2}"
  end
  @bands_excluded = excluded1 + excluded2

  #puts "excluded=#{@bands_excluded}"
  @test_bands = Array.new
  @test_bands = all_bands
  if !@bands_excluded.nil?
    #@test_bands = all_bands.select { |band| !@bands_excluded.include?(band) }
    @bands_excluded.each { | band1|
      @test_bands.delete_if { |band| band.include?(band1.strip)}
    }
    #        else
    #             @test_bands = all_bands
  end
  #puts "test bands= #{@test_bands}"
end


When(/^not excluded bands retrieved successfully for esdt "([^"]*)" for the service "([^"]*)" and "([^"]*)"
format and "([^"]*)" groups$/) do |esdt, agent,fmt,unsubsettable|
  format = fmt.strip
  service_agent = agent.strip
  unsubsettable_group = unsubsettable.strip
```

```
  if !unsubsettable_group.nil?
    unsubsettable_arr = unsubsettable_group.split(',')
  end
  @test_bands = Hash.new()
  filename = @test_hdf
  utils =  Sdps::Da::DaUtil.new()
  if service_agent.eql?("ICESAT2")
    all_bands = utils.get_non_zerolength_bands(@test_hdf)
    #remove any bands that are in the unsubsettable group
    if !unsubsettable_arr.nil? && !unsubsettable_arr.empty?
      unsubsettable_arr.each { | band1|
        all_bands.delete_if { |band| band.include?(band1.strip)}}
    end
  else
    all_bands= utils.get_band_params(filename)
  end
  @bands_excluded = Array.new
  db_subsystem = Sdps::Da::DB_SUBSYSTEM

  db_conn = Sdps::Db.connect(db_subsystem, @mode)
  db_conn.should_not be_nil
  dadb = Sdps::Da::DaDb.new(db_conn, @mode)

  if !format.nil? and !format.empty?
    @bands_excluded = dadb.get_objects_excluded_for_collection_service_format(@test_collection.shortname,
@test_collection.versionid, service_agent, format)
    puts format
    puts "excluded1=#{@bands_excluded}"
  end
  @test_bands = Array.new
  @test_bands = all_bands
  if !@bands_excluded.nil? and !@bands_excluded.empty?
    @bands_excluded.each { | band1|
      @test_bands.delete_if { |band| band.include?(band1.strip)}
    }
  end
```

```
end


Given(/^bands retrieved successfully for granule "(.*?)" for ESI request$/) do |testgranule|
  @test_bands = Array.new()
  filename = @test_hdf
  dautil =  Sdps::Da::DaUtil.new()
  @test_bands= @dautil.get_band_params(filename)
  #puts filename
  #puts @test_bands
  @test_bands.should_not be_nil
end

Then(/^Cleanup$/) do
  if !@db_conn.nil?
    @db_conn.close
  end
endCurrent Test Script: egi_smap_nosubset_format.rb

When(/^Submit a EGI "(.*?)" request for each of the SMAP  test granules "(.*?)" for "(.*?)"$/) do
|agent,localgranuleid, format|
     @format = format
     @service_agent = agent
     @egi = Sdps::Da::Egi.new(@mode)
     #puts @format
     @response_xml = @egi.submit_service_request(@test_granule.granuleid, @test_collection.esdt,
@test_collection.datasetid, "", @format, @service_agent)

end

Then(/^the EGI "(.*?)" service request should succeed for the SMAP  test granule "(.*?)" and "(.*?)"$/) do
|agent, localgranuleid, format|
       @response = @egi.parse_response(@response_xml)
       @response.should_not be_nil
       result = @response.message.eql?("SUCCESS") ? true:false
       puts " #{@service_agent}, #{@test_collection.esdt}, #{@format}, #{@response.message}"
```

```
       #response.should eql("SUCCESS")
       result.should be_truthy, "#{@service_agent}, #{@test_collection.esdt}, #{@format}, #{@response.message}"
end
Current Test Script: egi_smap_parameter_subset.rb


#the bands should be configured for band subsetting
#Then(/^the number of output files should match the number of bands in the input granule$/) do
#    @test_granule_hdf_fields = Hash.new()
#    @test_outputs.keys.each do | granuleid|
#         #number of bands in the input granule ?
#         hdf_fields = @dadb.get_hdf_objects_for_collection_and_service(@test_granules[granuleid].shortname,
@test_granules[granuleid].versionid, Sdps::Da::Gdal::SERVICE_NAME)
#         @test_granule_hdf_fields[granuleid] = hdf_fields
#         outputs = @test_outputs[granuleid]
#         outputs.length.should eq(hdf_fields.length) #should calculate the bands
#    end
#end

When(/^Submit a EGI "(.*?)" request for each of the SMAP  test granules "(.*?)" for each band parameter and
"(.*?)"$/) do |agent, arg2, fmt|
     @format = fmt.strip
     @service_agent = agent
     @response_granule_xmls  = Hash.new()
     @egi = Sdps::Da::Egi.new(@mode)
             granuleid = @test_granule.granuleid
             esdt  = @test_collection.esdt
             dataset_id = @test_collection.datasetid
             shortname = @test_collection.shortname
             @response_band_xmls = Hash.new()

             @test_bands.each { |band|
                  #prepend band with shortname
                  band1 = "/#{shortname}#{band}"
                  response_xml = @egi.submit_request(esdt, granuleid, @service_agent, format:@format,
band_parameter:band1)
```

```
                  @response_band_xmls[band] = response_xml
                  puts @egi.egi_service_request_url
             }
end

When(/^Submit a EGI "([^"]*)" request for each of the SMAP  test granule for "([^"]*)" for each band parameter
with required bands and "([^"]*)"$/) do |agent, arg2, fmt|
      @format = fmt.strip
     @service_agent = agent
     @response_granule_xmls  = Hash.new()
     @egi = Sdps::Da::Egi.new(@mode)
             granuleid = @test_granule.granuleid
             esdt  = @test_collection.esdt
             dataset_id = @test_collection.datasetid
             shortname = @test_collection.shortname
             @response_band_xmls = Hash.new()
             #p @test_bands
             #p @required_bands
             @test_bands.each { |band|
                  #prepend band with shortname
                  band1 = "/#{shortname}#{band}"
                  #include required bands
                  @required_bands.each {|required|
                        if band != required
                           band2 = "/#{shortname}#{required}"
                          band1 = "#{band1},#{band2}"
                        end
                  }
                  response_xml = @egi.submit_request(esdt, granuleid, @service_agent, format:@format,
band_parameter:band1)
                  @response_band_xmls[band] = response_xml
                  puts @egi.egi_service_request_url
             }
end

When(/^Submit a EGI "([^"]*)" request for each of the SMAP  test granules "([^"]*)" for each band parameter with
```

```
required bands and "([^"]*)"$/) do |agent, arg2, fmt|
    @format = fmt.strip
     @service_agent = agent
     @response_granule_xmls  = Hash.new()
     @egi = Sdps::Da::Egi.new(@mode)
             granuleid = @test_granule.granuleid
             esdt  = @test_collection.esdt
             dataset_id = @test_collection.datasetid
             shortname = @test_collection.shortname
             @response_band_xmls = Hash.new()
             #p @test_bands
             #p @required_bands
             @test_bands.each { |band|
                  #prepend band with shortname
                  band1 = "/#{shortname}#{band}"
                  #include required bands
                  @required_bands.each {|required|
                        if band != required
                           band2 = "/#{shortname}#{required}"
                          band1 = "#{band1},#{band2}"
                        end
                  }
                  response_xml = @egi.submit_request(esdt, granuleid, @service_agent, format:@format,
band_parameter:band1)
                  @response_band_xmls[band] = response_xml
                  puts @egi.egi_service_request_url
             }
end


When(/^the EGI service request succeeds for the SMAP  test granule for "([^"]*)" and band parameter and
"([^"]*)"$/) do |arg1, arg2|
         collection = @test_collection
       esdt = collection.esdt
        @response_results = Hash.new()
        @bands_to_not_exclude = Array.new
```

```
        @bands_to_exclude = Array.new

        @response_band_xmls.keys.each do |band|
           response_xml = @response_band_xmls[band]
           response = @egi.parse_response(response_xml)
           response.should_not be_nil
           result = response.message.eql?("SUCCESS") ? true:false
           puts "#{esdt}, #{@format}, #{band}, #{response.message}"
           @response_results[band] = response
           if response.message.eql? "SUCCESS"
              @bands_to_not_exclude.push (band)
           else
             @bands_to_exclude.push(band)
           end
        end
end

When(/^the EGI service request succeeds for the SMAP  test granule "(.*?)" and band parameter and "(.*?)"$/) do
|arg1, arg2|
       collection = @test_collection
       esdt = collection.esdt
        @response_results = Hash.new()
        @bands_to_not_exclude = Array.new
        @bands_to_exclude = Array.new

        @response_band_xmls.keys.each do |band|
           response_xml = @response_band_xmls[band]
           response = @egi.parse_response(response_xml)
           response.should_not be_nil
           result = response.message.eql?("SUCCESS") ? true:false
           puts "#{esdt}, #{@format}, #{band}, #{response.message}"
           @response_results[band] = response
           if response.message.eql? "SUCCESS"
              @bands_to_not_exclude.push (band)
           else
             @bands_to_exclude.push(band)
```

```
          end
        end
end

Then(/^the band parameter and "([^"]*)" should not be configured for exclusion for the collection "([^"]*)" and
"([^"]*)"$/) do |arg1, arg2, arg3|
         @bands_excluded = Array.new
      @bands_excluded = @dadb.get_objects_excluded_for_collection_service_format(@test_collection.shortname,
@test_collection.versionid, @service_agent, @format)
      #puts @bands_excluded
      @incorrect_excluded_bands = Array.new
      if !@bands_excluded.nil?
         @bands_to_not_exclude.each do |band|
              if @bands_excluded.include?(band)
                   @incorrect_excluded_bands.push ("#{band}")
              end
         end
     end
     @incorrect_excluded_bands.should be_empty, "#{@incorrect_excluded_bands.to_s} should not be excluded for
#{@test_collection.esdt} , #{@service_agent}, #{@format}"
end

Then(/^the band parameter and "(.*?)" should not be configured for exclusion for the test collection for "(.*?)"
and "(.*?)"$/) do |arg1, arg2, arg3|
      @bands_excluded = Array.new
      @bands_excluded = @dadb.get_objects_excluded_for_collection_service_format(@test_collection.shortname,
@test_collection.versionid, @service_agent, @format)
      #puts @bands_excluded
      @incorrect_excluded_bands = Array.new
      if !@bands_excluded.nil?
         @bands_to_not_exclude.each do |band|
              if @bands_excluded.include?(band)
                   @incorrect_excluded_bands.push ("#{band}")
              end
         end
     end
```

```
     @incorrect_excluded_bands.should be_empty, "#{@incorrect_excluded_bands.to_s} should not be excluded for
#{@test_collection.esdt} , #{@service_agent}, #{@format}"
end


When(/^the EGI service request fails for the SMAP test granule for "([^"]*)" and band parameter and "([^"]*)"$/)
do |arg1, arg2|

    #puts @bands_excluded
    @incorrect_not_excluded_bands = Array.new
    if !@bands_to_exclude.nil?
           @bands_to_exclude.each do |band|
               if @bands_excluded.nil? or  !@bands_excluded.include?(band)
                     @incorrect_not_excluded_bands.push("#{band}")
               end
           end
   end
end

When(/^the EGI service request fails for the SMAP test granule "(.*?)" and band parameter and "(.*?)"$/) do
|arg1, arg2|
    #puts @bands_excluded
    @incorrect_not_excluded_bands = Array.new
    if !@bands_to_exclude.nil?
           @bands_to_exclude.each do |band|
               if @bands_excluded.nil? or  !@bands_excluded.include?(band)
                     @incorrect_not_excluded_bands.push("#{band}")
               end
           end
   end

 end


Then(/^the band parameter and "([^"]*)" should be configured for exclusion for the collection "([^"]*)" and
"([^"]*)"$/) do |arg1, arg2, arg3|
```

```
    # #p @bands_to_exclude.to_s
    @incorrect_not_excluded_bands.should be_empty, "#{@incorrect_not_excluded_bands.to_s} should be excluded for
   #{@test_collection.esdt} , #{@service_agent}, #{@format}"
end

Then(/^the band parameter and "(.*?)" should be configured for exclusion for the test collection for "(.*?)" and
"(.*?)"$/) do |arg1, arg2,arg3|
    # #p @bands_to_exclude.to_s
    @incorrect_not_excluded_bands.should be_empty, "#{@incorrect_not_excluded_bands.to_s} should be excluded for
#{@test_collection.esdt} , #{@service_agent}, #{@format}"
end

Current Test Script: egi_request.rb
When(/^Submit a EGI "([^"]*)" request for each of the SMAP test granule "([^"]*)" and "([^"]*)" format and
"([^"]*)" spatialsubset and "([^"]*)" projection  with "([^"]*)" param$/) do |agent, localgranuleid, fmt,
spatialsubset, reprojection, param|

     @format = fmt.strip
     @service_agent = agent.strip
     @spatial_subset = spatialsubset.strip
     @projection = reprojection.strip
     @response_granule_xmls  = Hash.new()
     @egi = Sdps::Da::Egi.new(@mode)
             granuleid = @test_granule.granuleid
             esdt  = @test_collection.esdt
             dataset_id = @test_collection.datasetid
             shortname = @test_collection.shortname
             #@response_xml = @egi.submit_service_request(granuleid, esdt, dataset_id, '', @format,
@service_agent,@projection)
             @response_xml = @egi.submit_sync_request(esdt, granuleid, @service_agent, format:@format,
bbox:@spatial_subset, projection:@projection)
             puts @egi.egi_service_request_url
end

Then(/^the EGI "([^"]*)" request should succeed for the SMAP test granule "([^"]*)" and "([^"]*)" format and
```

```
"([^"]*)" spatialsubset and "([^"]*)" projection  with "([^"]*)" param$/) do |agent, localgranuleid, fmt,
spatialsubset, reprojection, param|


#Then(/^the EGI "([^"]*)" request should succeed for the SMAP test granule "([^"]*)" and "([^"]*)" format and
"([^"]*)" spatialsubset and "([^"]*)"  projection  with "([^"]*)" param$/) do |agent, localgranuleid, fmt,
spatialsubset, reprojection, param|
          collection = @test_collection
          esdt = collection.esdt
          puts @response_xml
           response = @egi.parse_response(@response_xml)
           puts response
           response.should_not be_nil
           result = response.message.eql?("SUCCESS") ? true:false
           #puts "#{esdt}, #{@format}, #{band}, #{response.message}"
           #@response_results[band] = response
           result.should be_truthy, "#{localgranuleid} egi request failed"
end

When(/^Submit a EGI "([^"]*)" agent request for the h5 test granule for "([^"]*)" granule and "([^"]*)" format
and "([^"]*)" spatialsubset and "([^"]*)" projection  with "([^"]*)" param$/) do |agent, localgranuleid, fmt,
spatialsubset, reprojection, param|

     @format = fmt.strip
     @service_agent = agent.strip
     @spatial_subset = spatialsubset.strip
     @projection = reprojection.strip
     @projection_param = param.strip
     @response_granule_xmls  = Hash.new()
     @egi = Sdps::Da::Egi.new(@mode)
     granuleid = @test_granule.granuleid
     esdt  = @test_collection.esdt
     dataset_id = @test_collection.datasetid
     shortname = @test_collection.shortname

     puts "bands excluded=#{@bands_excluded}"
```

```
     puts "test bands=#{@test_bands}"
     #@test_bands1 = Array.new
     if !@bands_excluded.nil?  and  !@bands_excluded.empty?
#            @test_bands.each { |band|
#                  #prepend band with shortname
#                  #band1 = "/#{shortname}#{band}"
#                  band1 = band
#                 @test_bands1.push(band1)
#            }
            band_str = @test_bands.join(",")
            @response_xml = @egi.submit_request(esdt, granuleid, @service_agent, format:@format,
bbox:@spatial_subset,band_parameter:band_str,projection:@projection, projection_param:@projection_param)
      else
            @response_xml = @egi.submit_request(esdt, granuleid, @service_agent, format:@format,
bbox:@spatial_subset,projection:@projection, projection_param:@projection_param)
     end

             puts @egi.egi_service_request_url
end

When(/^Submit a EGI "([^"]*)" agent request for the h5 test granule for "([^"]*)" esdt and "([^"]*)" format and
"([^"]*)" spatialsubset and "([^"]*)" projection  with "([^"]*)" param$/) do |agent, esdt, fmt, spatialsubset,
reprojection, param|

     @format = fmt.strip
     @service_agent = agent.strip
     @spatial_subset = spatialsubset.strip
     @projection = reprojection.strip
     @projection_param = param.strip
     @response_granule_xmls  = Hash.new()
     @egi = Sdps::Da::Egi.new(@mode)
     granuleid = @test_granule.granuleid
     esdt  = @test_collection.esdt
     dataset_id = @test_collection.datasetid
     shortname = @test_collection.shortname
```

```
     puts "bands excluded=#{@bands_excluded}"
     puts "test bands=#{@test_bands}"
     #@test_bands1 = Array.new
     if !@bands_excluded.nil?  and  !@bands_excluded.empty?
#            @test_bands.each { |band|
#                  #prepend band with shortname
#                  #band1 = "/#{shortname}#{band}"
#                  band1 = band
#                 @test_bands1.push(band1)
#            }
            band_str = @test_bands.join(",")
            @response_xml = @egi.submit_request(esdt, granuleid, @service_agent, format:@format,
bbox:@spatial_subset,band_parameter:band_str,projection:@projection, projection_param:@projection_param)
      else
            @response_xml = @egi.submit_request(esdt, granuleid, @service_agent, format:@format,
bbox:@spatial_subset,projection:@projection, projection_param:@projection_param)
     end

             puts @egi.egi_service_request_url
end


Then(/^the EGI "([^"]*)" request should succeed for the  h5 test granule for "([^"]*)" esdt and "([^"]*)" format
and "([^"]*)" spatialsubset and "([^"]*)" projection  with "([^"]*)" param$/) do |agent, esdt, fmt,
spatialsubset, reprojection, param|

          collection = @test_collection
          esdt = collection.esdt
          puts @response_xml
           response = @egi.parse_response(@response_xml)
           puts response
           response.should_not be_nil
           result = response.message.eql?("SUCCESS") ? true:false
           #puts "#{esdt}, #{@format}, #{band}, #{response.message}"
           #@response_results[band] = response
           result.should be_truthy, "#{esdt} egi request failed"
```

```
end

When(/^Submit a EGI "([^"]*)" agent request for the h5 test granule for "([^"]*)" esdt and "([^"]*)" format and
"([^"]*)" spatialsubset$/) do |agent, esdt, fmt, spatialsubset|

     @format = fmt.strip
     @service_agent = agent.strip
     @spatial_subset = spatialsubset.strip

     @response_granule_xmls  = Hash.new()
     @egi = Sdps::Da::Egi.new(@mode)
     granuleid = @test_granule.granuleid
     esdt  = @test_collection.esdt
     dataset_id = @test_collection.datasetid
     shortname = @test_collection.shortname

     puts "bands excluded=#{@bands_excluded}"
     puts "test bands=#{@test_bands}"
     #@test_bands1 = Array.new
     if !@bands_excluded.nil?  and  !@bands_excluded.empty?
#            @test_bands.each { |band|
#                  #prepend band with shortname
#                  #band1 = "/#{shortname}#{band}"
#                  band1 = band
#                 @test_bands1.push(band1)
#            }
            band_str = @test_bands.join(",")
            @response_xml = @egi.submit_request(esdt, granuleid, @service_agent, format:@format,
bbox:@spatial_subset,band_parameter:band_str)
      else
            @response_xml = @egi.submit_request(esdt, granuleid, @service_agent, format:@format,
bbox:@spatial_subset)
     end

             puts @egi.egi_service_request_url
end
```

```
Then(/^the EGI "([^"]*)" request should succeed for the  h5 test granule for "([^"]*)" esdt and "([^"]*)" format
and "([^"]*)" spatialsubset$/) do |agent, esdt, fmt, spatialsubset|

          collection = @test_collection
          esdt = collection.esdt
          puts @response_xml
           response = @egi.parse_response(@response_xml)
           puts response
           response.should_not be_nil
           result = response.message.eql?("SUCCESS") ? true:false
           #puts "#{esdt}, #{@format}, #{band}, #{response.message}"
           #@response_results[band] = response
           result.should be_truthy, "#{esdt} egi request failed"
end




When(/^Submit a EGI "([^"]*)" request for the hdf test granule for the "([^"]*)" and "([^"]*)"$/) do |agent,
esdt, fmt|
     @format = fmt.strip
     @service_agent = agent
     @response_granule_xmls  = Hash.new()
     @egi = Sdps::Da::Egi.new(@mode)
     granuleid = @test_granule.granuleid
     @response_xml = @egi.submit_request(esdt, granuleid, @service_agent, format:@format)
     puts @egi.egi_service_request_url
end

Then(/^the EGI "([^"]*)" request should succeed for the  hdf test granule for "([^"]*)" and "([^"]*)"$/) do
|agent, esdt, format|
     collection = @test_collection
     esdt = collection.esdt
     response = @egi.parse_response(@response_xml)
```

```
     response.should_not be_nil
     result = response.message.eql?("SUCCESS") ? true:false
     result.should be_truthy, "#{@test_granule.localgranuleid} EGI #{agent} request failed"
end

When(/^Submit a EGI "([^"]*)" request for the hdf test granule for the "([^"]*)" and "([^"]*)" and "([^"]*)"
reprojection$/) do |agent, esdt, fmt, reprojection|
     @format = fmt.strip
     @service_agent = agent
     @projection = reprojection.strip
     @response_granule_xmls  = Hash.new()
     @egi = Sdps::Da::Egi.new(@mode)
     granuleid = @test_granule.granuleid
     @response_xml = @egi.submit_request(esdt, granuleid, @service_agent, format:@format,
projection:@projection)
     puts @egi.egi_service_request_url
end


When(/^Submit a EGI "([^"]*)" agent request for the h5 test granule "([^"]*)" and "([^"]*)" format and "([^"]*)"
projection with "([^"]*)" param$/) do |agent, localgranuleid, fmt, reprojection, param|
     @format = fmt.strip
     @service_agent = agent
     @projection = reprojection.strip
     @projection_param = param.strip
     @response_granule_xmls  = Hash.new()
     @egi = Sdps::Da::Egi.new(@mode)
     granuleid = @test_granule.granuleid
     #if there are excluded bands, they need to be removed in the request
     shortname = @test_collection.shortname
     esdt = @test_collection.esdt
     puts "bands excluded=#{@bands_excluded}"
     puts "test bands=#{@test_bands}"
     @test_bands1 = Array.new
     if !@bands_excluded.nil?  and  !@bands_excluded.empty?
#            @test_bands.each { |band|
```

```
#                   #prepend band with shortname
#                   #band1 = "/#{shortname}#{band}"
#                   band1 = band
#                  @test_bands1.push(band1)
#             }
            band_str = @test_bands.join(",")
            @response_xml = @egi.submit_sync_request(esdt, granuleid, @service_agent, format:@format,
band_parameter:band_str,projection:@projection, projection_param:@projection_param)
      else
            @response_xml = @egi.submit_request(esdt, granuleid, @service_agent, format:@format,
projection:@projection, projection_param:@projection_param)
     end
     puts @egi.egi_service_request_url
end


When(/^Submit a EGI "([^"]*)" agent request for the test granule for the "([^"]*)" esdt and "([^"]*)" format and
"([^"]*)" projection and "([^"]*)" param$/) do |agent, esdt, fmt, reprojection, param|
     @format = fmt.strip
     @service_agent = agent
     @projection = reprojection.strip
     @projection_param = param.strip
     @response_granule_xmls  = Hash.new()
     @egi = Sdps::Da::Egi.new(@mode)
     granuleid = @test_granule.granuleid
     #if there are excluded bands, they need to be removed in the request
     shortname = @test_collection.shortname
     puts "bands excluded=#{@bands_excluded}"
     puts "test bands=#{@test_bands}"
     @test_bands1 = Array.new
     if !@bands_excluded.nil?  and  !@bands_excluded.empty?
#            @test_bands.each { |band|
#                  #prepend band with shortname
#                  #band1 = "/#{shortname}#{band}"
#                  band1 = band
#                 @test_bands1.push(band1)
```

```
#            }
            band_str = @test_bands.join(",")
            @response_xml = @egi.submit_sync_request(esdt, granuleid, @service_agent, format:@format,
band_parameter:band_str,projection:@projection, projection_param:@projection_param)
      else
            @response_xml = @egi.submit_request(esdt, granuleid, @service_agent, format:@format,
projection:@projection, projection_param:@projection_param)
     end
     puts @egi.egi_service_request_url
end


Then(/^the EGI "([^"]*)" agent request should succeed for the test granule for the "([^"]*)" esdt and "([^"]*)"
format and "([^"]*)" projection and "([^"]*)" param$/) do |agent, esdt, fmt, reprojection, param|
      collection = @test_collection
     esdt = collection.esdt
     puts @response_xml
     response = @egi.parse_response(@response_xml)
     puts response
     response.should_not be_nil
     result = response.message.eql?("SUCCESS") ? true:false
     puts "status: #{response.status} msg:#{response.message} urls:#{response.urls}"
     result.should be_truthy, "#{@test_granule.localgranuleid} EGI #{agent} request failed"
end


When(/^Submit a EGI "([^"]*)" request for the hdf  test granule for the "([^"]*)" for each band parameter and
"([^"]*)"$/) do |agent, esdt, fmt|
     @format = fmt.strip
     @service_agent = agent
     @response_granule_xmls  = Hash.new()
     @egi = Sdps::Da::Egi.new(@mode)
             granuleid = @test_granule.granuleid
             esdt  = @test_collection.esdt
             dataset_id = @test_collection.datasetid
             shortname = @test_collection.shortname
```

```
            @response_band_xmls = Hash.new()

            @test_bands.each { |band|
                 #prepend band with shortname
                 #band1 = "/#{shortname}#{band}"
                 band1 = "#{band}"
                 response_xml = @egi.submit_sync_request(esdt, granuleid, @service_agent, format:@format,
band_parameter:band1)
                 @response_band_xmls[band] = response_xml
                 puts @egi.egi_service_request_url
            }
end

When(/^Submit a EGI "([^"]*)" request without the "([^"]*)" for the h5 test granule for the "([^"]*)" and
"([^"]*)"$/) do |agent, excluded_bands, esdt, fmt|
     bands_excluded_by_design = excluded_bands.strip
     @format = fmt.strip
     @service_agent = agent
     @response_granule_xmls  = Hash.new()
     @egi = Sdps::Da::Egi.new(@mode)
     granuleid = @test_granule.granuleid
     esdt  = @test_collection.esdt
     dataset_id = @test_collection.datasetid
     shortname = @test_collection.shortname

     puts "bands excluded=#{@bands_excluded}"
     if !bands_excluded_by_design.nil? and !bands_excluded_by_design.empty?
       @bands_excluded += bands_excluded_by_design.split()
     end
     @test_bands1 = Array.new
     if !@bands_excluded.nil?  and  !@bands_excluded.empty?
            @test_bands.each { |band|
                 #prepend band with shortname
                 #band1 = "/#{shortname}#{band}"
                 band1 = band
                @test_bands1.push(band1)
```

```
            }
            band_str = @test_bands1.join(",")
            @response_xml = @egi.submit_sync_request(esdt, granuleid, @service_agent, format:@format,
band_parameter:band_str)
      else
            @response_xml = @egi.submit_sync_request(esdt, granuleid, @service_agent, format:@format)

     end
     puts @egi.egi_service_request_url

end


When(/^Submit a EGI "([^"]*)" request for the h5 test granule for the "([^"]*)" for each band parameter with
required bands and "([^"]*)"$/) do |agent, esdt, fmt|
     @format = fmt.strip
     @service_agent = agent
     @response_granule_xmls  = Hash.new()
     @egi = Sdps::Da::Egi.new(@mode)
             granuleid = @test_granule.granuleid
             esdt  = @test_collection.esdt
             dataset_id = @test_collection.datasetid
             shortname = @test_collection.shortname
             @response_band_xmls = Hash.new()
             #p @test_bands
             #p @required_bands
             @test_bands.each { |band|
                  #prepend band with shortname
                  #band1 = "/#{shortname}#{band}"
                  band1 = band
                  #include required bands
                  @required_bands.each {|required|
                        if band != required
                           #band2 = "/#{shortname}#{required}"
                           band2 = required
                          band1 = "#{band1},#{band2}"
```

```
                        end
                    }
                    response_xml = @egi.submit_sync_request(esdt, granuleid, @service_agent, format:@format,
band_parameter:band1)
                    @response_band_xmls[band] = response_xml
                    puts @egi.egi_service_request_url
                }

end

When(/^Submit a EGI "([^"]*)" request for the h5 test granule for the "([^"]*)" for each band parameter and
"([^"]*)"$/) do |agent, esdt, fmt|
    @format = fmt.strip
     @service_agent = agent
     @response_granule_xmls  = Hash.new()
     @egi = Sdps::Da::Egi.new(@mode)
             granuleid = @test_granule.granuleid
             esdt  = @test_collection.esdt
             dataset_id = @test_collection.datasetid
             shortname = @test_collection.shortname
             @response_band_xmls = Hash.new()

             @test_bands.each { |band|
                  #prepend band with shortname
                 # band1 = "/#{shortname}#{band}"
                 response_xml = @egi.submit_sync_request(esdt, granuleid, @service_agent, format:@format,
band_parameter:band)
                 # response_xml = @egi.submit_request(esdt, granuleid, @service_agent, format:@format,
band_parameter:band1)
                 #response_xml = @egi.submit_post_request(esdt, granuleid, @service_agent, format:@format,
band_parameter:band1)
                 @response_band_xmls[band] = response_xml
                  puts @egi.egi_service_request_url

             }
end
```

```
When(/^the EGI "([^"]*)" request succeeds for the hdf test granule for "([^"]*)" and band parameter and
"([^"]*)"$/) do |arg1, arg2, arg3|

        @response_results = Hash.new()
        @bands_to_not_exclude = Array.new
        @bands_to_exclude = Array.new

        @response_band_xmls.keys.each do |band|
           response_xml = @response_band_xmls[band]
           response = @egi.parse_response(response_xml)
           response.should_not be_nil
           result = response.message.eql?("SUCCESS") ? true:false
           puts "#{@test_collection.esdt}, #{@format}, #{band}, #{response.message}"
           @response_results[band] = response
           if response.message.eql? "SUCCESS"
              @bands_to_not_exclude.push (band)
           else
             @bands_to_exclude.push(band)
           end
        end
end

When(/^the EGI "([^"]*)" request succeeds for the h5 test granule for "([^"]*)" and band parameter and
"([^"]*)"$/) do |arg1, arg2, arg3|
    @response_results = Hash.new()
        @bands_to_not_exclude = Array.new
        @bands_to_exclude = Array.new

        @response_band_xmls.keys.each do |band|
           response_xml = @response_band_xmls[band]
           response = @egi.parse_response(response_xml)
           response.should_not be_nil
           result = response.message.eql?("SUCCESS") ? true:false
           puts "#{@test_collection.esdt}, #{@format}, #{band}, #{response.message}"
           @response_results[band] = response
```

```
          if response.message.eql? "SUCCESS"
             @bands_to_not_exclude.push (band)
          else
            @bands_to_exclude.push(band)
          end
        end
end

Then(/^the band parameter and "([^"]*)" should not be configured for exclusion for "([^"]*)" and "([^"]*)"$/) do
|arg1, arg2, arg3|
      @bands_excluded = Array.new

        db_subsystem = Sdps::Da::DB_SUBSYSTEM
        db_conn = Sdps::Db.connect(db_subsystem, @mode)
        db_conn.should_not be_nil
        dadb = Sdps::Da::DaDb.new(db_conn, @mode)

      @bands_excluded = dadb.get_objects_excluded_for_collection_service_format(@test_collection.shortname,
@test_collection.versionid, @service_agent, @format)
      #puts @bands_excluded
      @incorrect_excluded_bands = Array.new
      if !@bands_excluded.nil?
         @bands_to_not_exclude.each do |band|
              if @bands_excluded.include?(band)
                   @incorrect_excluded_bands.push ("#{band}")
              end
         end
     end
     @incorrect_excluded_bands.should be_empty, "#{@incorrect_excluded_bands.to_s} should not be excluded for
#{@test_collection.esdt} , #{@service_agent}, #{@format}"
   if !db_conn.nil?
          db_conn.close
        end
   db_conn = nil

end
```

```
When(/^the EGI "([^"]*)" request fails for the hdf test granule for "([^"]*)" and band parameter and
"([^"]*)"$/) do |arg1, arg2, arg3|
  #puts @bands_excluded
    @incorrect_not_excluded_bands = Array.new
    if !@bands_to_exclude.nil?
           @bands_to_exclude.each do |band|
               if @bands_excluded.nil? or  !@bands_excluded.include?(band)
                     @incorrect_not_excluded_bands.push("#{band}")
               end
           end
   end
end

When(/^the EGI "([^"]*)" request fails for the h5 test granule for "([^"]*)" and band parameter and "([^"]*)"$/)
do |arg1, arg2, arg3|
    #puts @bands_excluded
    @incorrect_not_excluded_bands = Array.new
    if !@bands_to_exclude.nil?
           @bands_to_exclude.each do |band|
               if @bands_excluded.nil? or  !@bands_excluded.include?(band)
                     @incorrect_not_excluded_bands.push("#{band}")
               end
           end
   end
end

Then(/^the band parameter and "([^"]*)" should be configured for exclusion for "([^"]*)" and "([^"]*)"$/) do
|arg1, arg2, arg3|
    #p @bands_to_exclude.to_s
    @incorrect_not_excluded_bands.should be_empty, "#{@incorrect_not_excluded_bands.to_s} should be excluded for
#{@test_collection.esdt} , #{@service_agent}, #{@format}"
end

When(/^Submit a EGI "([^"]*)" request for the hdf test granule for the "([^"]*)" and "([^"]*)" and spatial
subset "([^"]*)"$/) do |agent, esdt, fmt, spatialsubset|
```

```
     @format = fmt.strip
     @service_agent = agent.strip
     @spatial_subset = spatialsubset.strip
     @response_granule_xmls  = Hash.new()
     @egi = Sdps::Da::Egi.new(@mode)
             granuleid = @test_granule.granuleid
             esdt  = @test_collection.esdt
             dataset_id = @test_collection.datasetid
             shortname = @test_collection.shortname
             #@response_xml = @egi.submit_service_request(granuleid, esdt, dataset_id, '', @format,
@service_agent,@projection)
             @response_xml = @egi.submit_request(esdt, granuleid, @service_agent, format:@format,
bbox:@spatial_subset)
           puts @egi.egi_service_request_url
end

Then(/^the EGI "([^"]*)" request should succeed for the  hdf test granule for "([^"]*)" and "([^"]*)" and
spatial subset "([^"]*)"$/) do |arg1, arg2, arg3, arg4|
          collection = @test_collection
          esdt = collection.esdt
           response = @egi.parse_response(@response_xml)
           response.should_not be_nil
           result = response.message.eql?("SUCCESS") ? true:false
           #puts "#{esdt}, #{@format}, #{band}, #{response.message}"
           #@response_results[band] = response
           result.should be_truthy, "#{esdt} spatial subset : #{@spatial_subset}  request failed"
end

Then(/^the EGI "([^"]*)" request without the "([^"]*)" should succeed for the  h5 test granule for "([^"]*)" and
"([^"]*)"$/) do |agent, excluded_bands, esdt, fmt|
          collection = @test_collection
          esdt = collection.esdt
          puts @response_xml
           response = @egi.parse_response(@response_xml)
           puts response
           response.should_not be_nil
```

```
          result = response.message.eql?("SUCCESS") ? true:false
          #puts "#{esdt}, #{@format}, #{band}, #{response.message}"
          #@response_results[band] = response
          result.should be_truthy, "#{esdt} format : #{@format}  request failed"
end


When(/^Submit a EGI "([^"]*)" agent request for the h5 test granule for "([^"]*)" esdt for each band parameter
and "([^"]*)" format and "([^"]*)" projection with "([^"]*)" param$/) do
 |agent,esdt,fmt,reprojection,param|

    @format = fmt.strip
     @service_agent = agent
     @projection = reprojection.strip
     @projection_param = param.strip
     @response_granule_xmls  = Hash.new()
     @egi = Sdps::Da::Egi.new(@mode)

     granuleid = @test_granule.granuleid
     #if there are excluded bands, they need to be removed in the request
     shortname = @test_collection.shortname
     puts "bands excluded=#{@bands_excluded}"

     @response_granule_xmls  = Hash.new()
     @egi = Sdps::Da::Egi.new(@mode)
             esdt  = @test_collection.esdt
             shortname = @test_collection.shortname
             @response_band_xmls = Hash.new()

             @test_bands.each { |band|
                  #prepend band with shortname
                  #band1 = "/#{shortname}#{band}"
                  band1 = band

                  response_xml = @egi.submit_sync_request(esdt, granuleid, @service_agent, format:@format,
band_parameter:band1,projection:@projection, projection_param:@projection_param)
```

```
                  @response_band_xmls[band] = response_xml
                  puts @egi.egi_service_request_url
              }
end

Then(/^the EGI "([^"]*)" request should succeed for the  h5 test granule for "([^"]*)" esdt for band parameter
and "([^"]*)" format and "([^"]*)" projection with "([^"]*)" param$/) do
 |agent,esdt,format,projection,param|
        @response_results = Hash.new()
        @bands_to_not_exclude = Array.new
        @bands_to_exclude = Array.new

        @response_band_xmls.keys.each do |band|
           response_xml = @response_band_xmls[band]
           response = @egi.parse_response(response_xml)
           response.should_not be_nil
           result = response.message.eql?("SUCCESS") ? true:false
           puts "#{@test_collection.esdt}, #{@format}, #{band}, #{@projection},
#{@projection_param},#{response.message}"
           @response_results[band] = response
           if response.message.eql? "SUCCESS"
              @bands_to_not_exclude.push (band)
           else
             @bands_to_exclude.push(band)
           end
        end
        @bands_to_exclude.should be_empty
end

When(/^the EGI "([^"]*)" request succeeds for the h5 test granule for "([^"]*)" and band parameter and "([^"]*)"
format and "([^"]*)" projection with "([^"]*)" param$/) do |arg1, arg2, arg3, arg4, arg5|
        @response_results = Hash.new()
        @bands_to_not_exclude = Array.new
        @bands_to_exclude = Array.new
```

```
        @response_band_xmls.keys.each do |band|
           response_xml = @response_band_xmls[band]
           response = @egi.parse_response(response_xml)
           response.should_not be_nil
           result = response.message.eql?("SUCCESS") ? true:false
           puts "#{@test_collection.esdt}, #{@format}, #{@projection}, #{@projection_param}, #{band},
#{response.message}"
           @response_results[band] = response
           if response.message.eql? "SUCCESS"
              @bands_to_not_exclude.push (band)
           else
             @bands_to_exclude.push(band)
           end
        end
end

Then(/^the band parameter and "([^"]*)" and "([^"]*)" should not be configured for exclusion for "([^"]*)" and
"([^"]*)"$/) do |arg1, arg2, arg3, arg4|
 @bands_excluded = Array.new
        db_subsystem = Sdps::Da::DB_SUBSYSTEM
        db_conn = Sdps::Db.connect(db_subsystem, @mode)
        db_conn.should_not be_nil
        dadb = Sdps::Da::DaDb.new(db_conn, @mode)
      @bands_excluded = dadb.get_objects_excluded_for_collection_service_format(@test_collection.shortname,
@test_collection.versionid, @service_agent, @format)
      #puts @bands_excluded
      @incorrect_excluded_bands = Array.new
      if !@bands_excluded.nil?
         @bands_to_not_exclude.each do |band|
              if @bands_excluded.include?(band)
                   @incorrect_excluded_bands.push ("#{band}")
              end
         end
     end
     @incorrect_excluded_bands.should be_empty, "#{@incorrect_excluded_bands.to_s} should not be excluded for
#{@test_collection.esdt} , #{@service_agent}, #{@format}, #{@projection}"
```

```
     if !db_conn.nil?
           db_conn.close
        end
      db_conn = nil
end

When(/^the EGI "([^"]*)" request fails for the h5 test granule for "([^"]*)" and band parameter and "([^"]*)"
and "([^"]*)"$/) do |arg1, arg2, arg3, arg4|
 #puts @bands_excluded
    @incorrect_not_excluded_bands = Array.new
    if !@bands_to_exclude.nil?
           @bands_to_exclude.each do |band|
               if @bands_excluded.nil? or  !@bands_excluded.include?(band)
                     @incorrect_not_excluded_bands.push("#{band}")
               end
           end
   end
end

Then(/^the band parameter and "([^"]*)" and "([^"]*)" should be configured for exclusion for "([^"]*)" and
"([^"]*)"$/) do |arg1, arg2, arg3, arg4|
  #p @bands_to_exclude.to_s
    @incorrect_not_excluded_bands.should be_empty, "#{@incorrect_not_excluded_bands.to_s} should be excluded for
#{@test_collection.esdt} , #{@service_agent}, #{@format}, #{@projection}"
end
Current Test Script: egi_heg_request.rb
When(/^Submit a EGI "([^"]*)" request for each of the AMSR test granule "([^"]*)" and "([^"]*)"$/) do | agent,
localgranuleid, fmt |
     @format = fmt.strip
     @service_agent = agent
     @response_granule_xmls  = Hash.new()
     @egi = Sdps::Da::Egi.new(@mode)
             granuleid = @test_granule.granuleid
             esdt  = @test_collection.esdt
             dataset_id = @test_collection.datasetid
             shortname = @test_collection.shortname
```

```
     @response_xml = @egi.submit_request(esdt, granuleid, @service_agent, format:@format)
      puts @egi.egi_service_request_url
end

Then(/^the EGI "([^"]*)" request should succeed for the AMSR test granule "([^"]*)" and "([^"]*)"$/) do | agent,
localgranuleid, fmt |
      collection = @test_collection
      esdt = collection.esdt
           response = @egi.parse_response(@response_xml)
           response.should_not be_nil
           result = response.message.eql?("SUCCESS") ? true:false
           #puts "#{esdt}, #{@format}, #{band}, #{response.message}"
           #@response_results[band] = response
           result.should be_truthy, "#{localgranuleid} HEG request failed"
end

When(/^Submit a EGI "([^"]*)" request for each of the AMSR test granule "([^"]*)" and "([^"]*)" and "([^"]*)"
reprojection$/) do | agent, localgranuleid, fmt, proj |
     @format = fmt.strip
     @service_agent = agent.strip
     @projection = proj.strip
     @response_granule_xmls  = Hash.new()
     @egi = Sdps::Da::Egi.new(@mode)
             granuleid = @test_granule.granuleid
             esdt  = @test_collection.esdt
             dataset_id = @test_collection.datasetid
             shortname = @test_collection.shortname
             #@response_xml = @egi.submit_service_request(granuleid, esdt, dataset_id, '', @format,
@service_agent,@projection)
             @response_xml = @egi.submit_request(esdt, granuleid, @service_agent, format:@format,
projection:@projection,interpolation:"NN")
              puts @egi.egi_service_request_url
end

Then(/^the EGI "([^"]*)" request should succeed for the AMSR test granule "([^"]*)" and "([^"]*)" and "([^"]*)"
reprojection$/) do | agent, localgranuleid, fmt, proj |
```

```
          collection = @test_collection
          esdt = collection.esdt
           response = @egi.parse_response(@response_xml)
           response.should_not be_nil
           result = response.message.eql?("SUCCESS") ? true:false
           #puts "#{esdt}, #{@format}, #{band}, #{response.message}"
           #@response_results[band] = response
           result.should be_truthy, "#{localgranuleid} HEG request failed"
end


When(/^Submit a EGI "([^"]*)" request for each of the AMSR test granule "([^"]*)" and "([^"]*)" and
spatialsubset "([^"]*)"$/) do |agent,localgranuleid,fmt,spatialsubset|
     @format = fmt.strip
     @service_agent = agent.strip
     @spatial_subset = spatialsubset.strip
     @response_granule_xmls  = Hash.new()
     @egi = Sdps::Da::Egi.new(@mode)
             granuleid = @test_granule.granuleid
             esdt  = @test_collection.esdt
             dataset_id = @test_collection.datasetid
             shortname = @test_collection.shortname
             #@response_xml = @egi.submit_service_request(granuleid, esdt, dataset_id, '', @format,
@service_agent,@projection)
             @response_xml = @egi.submit_request(esdt, granuleid, @service_agent, format:@format,
bbox:@spatial_subset)
 puts @egi.egi_service_request_url
end

Then(/^the EGI "([^"]*)" request should succeed for the AMSR test granule "([^"]*)" and "([^"]*)" and
spatialsubset "([^"]*)"$/) do |agent,localgranuleid,fmt,spatialsubset|
          collection = @test_collection
          esdt = collection.esdt
           response = @egi.parse_response(@response_xml)
           response.should_not be_nil
           result = response.message.eql?("SUCCESS") ? true:false
```

```
            #puts "#{esdt}, #{@format}, #{band}, #{response.message}"
            #@response_results[band] = response
            result.should be_truthy, "#{localgranuleid} HEG request failed"
end

When(/^Submit a EGI "([^"]*)" request for each of the AMSR  test granule "([^"]*)" for each band parameter and
"([^"]*)"$/) do |agent,localgranuleid, fmt|
     @format = fmt.strip
     @service_agent = agent
     @response_granule_xmls  = Hash.new()
     @egi = Sdps::Da::Egi.new(@mode)
             granuleid = @test_granule.granuleid
             esdt  = @test_collection.esdt
             dataset_id = @test_collection.datasetid
             shortname = @test_collection.shortname
             @response_band_xmls = Hash.new()

             @test_bands.each { |band|
                  #prepend band with shortname
                  band1 = "/#{shortname}#{band}"
                  response_xml = @egi.submit_request(esdt, granuleid, @service_agent, format:@format,
band_parameter:band1)
                  @response_band_xmls[band] = response_xml
                  puts @egi.egi_service_request_url
             }

end

When(/^the EGI "(.*?)" request succeeds for the AMSR test granule "([^"]*)" and band parameter and "([^"]*)"$/)
do |agent,localgranuleid,format|
       collection = @test_collection
       esdt = collection.esdt
        @response_results = Hash.new()
        @bands_to_not_exclude = Array.new
        @bands_to_exclude = Array.new
```

```
        @response_band_xmls.keys.each do |band|
           response_xml = @response_band_xmls[band]
           response = @egi.parse_response(response_xml)
           response.should_not be_nil
           result = response.message.eql?("SUCCESS") ? true:false
           puts "#{esdt}, #{@format}, #{band}, #{response.message}"
           @response_results[band] = response
           if response.message.eql? "SUCCESS"
              @bands_to_not_exclude.push (band)
           else
             @bands_to_exclude.push(band)
           end
        end
end

When(/^the EGI "(.*?)" request fails for the AMSR test granule "([^"]*)" and band parameter and "([^"]*)"$/) do
|agent,localgranuleid,format|
    #puts @bands_excluded
    @incorrect_not_excluded_bands = Array.new
    if !@bands_to_exclude.nil?
           @bands_to_exclude.each do |band|
               if @bands_excluded.nil? or  !@bands_excluded.include?(band)
                     @incorrect_not_excluded_bands.push("#{band}")
               end
           end
   end
end

When(/^Submit a EGI "([^"]*)" request for each of the AMSR test granule "([^"]*)" and "([^"]*)" format and
spatialsubset "([^"]*)" and "([^"]*)" reprojection$/) do   |agent, localgranuleid, fmt, spatialsubset,
reprojection|
#When(/^Submit a EGI "([^"]*)" request for each of the AMSR test granule "([^"]*)" and "([^"]*)" and
spatialsubset "([^"]*)" and "([^"]*)" reproje
#ction$/) do |agent, localgranuleid, format, spatialsubset, reprojection|

     @format = fmt.strip
```

```
     @service_agent = agent.strip
     @spatial_subset = spatialsubset.strip
     @projection = reprojection.strip
     @response_granule_xmls  = Hash.new()
     @egi = Sdps::Da::Egi.new(@mode)
             granuleid = @test_granule.granuleid
             esdt  = @test_collection.esdt
             dataset_id = @test_collection.datasetid
             shortname = @test_collection.shortname
             #@response_xml = @egi.submit_service_request(granuleid, esdt, dataset_id, '', @format,
@service_agent,@projection)
             @response_xml = @egi.submit_request(esdt, granuleid, @service_agent, format:@format,
bbox:@spatial_subset, projection:@projection,interpolation:"NN")
              puts @egi.egi_service_request_url
end

Then(/^the EGI "([^"]*)" request should succeed for the AMSR test granule "([^"]*)" and "([^"]*)" format and
spatialsubset "([^"]*)"  and "([^"]*)"  reprojection$/) do  |agent, localgranuleid, fmt, spatialsubset,
reprojection|

#Then(/^the EGI "([^"]*)" request should succeed for the AMSR test granule "([^"]*)" and "([^"]*)" and
spatialsubset "([^"]*)"  and "([^"]*)"  r
#eprojection$/)  do  |agent, localgranuleid, format, spatialsubset, reprojection|
 collection = @test_collection
          esdt = collection.esdt
           response = @egi.parse_response(@response_xml)
           response.should_not be_nil
           result = response.message.eql?("SUCCESS") ? true:false
           #puts "#{esdt}, #{@format}, #{band}, #{response.message}"
           #@response_results[band] = response
           result.should be_truthy, "#{localgranuleid} HEG request failed"

end

Current Test Script: egi_service_steps.rb
# global variables : @response_xml
```

```
# @test_granule, @test_collection
# @egi
When(/^Submit a EGI "([^"]*)" request for the h5 test granule for the "([^"]*)" esdt and "([^"]*)" format$/) do
|agent, esdt, fmt|
     format = fmt.strip
     service_agent = agent.strip
     @egi = Sdps::Da::Egi.new(ENV['MODE'])
     granuleid = @test_granule.granuleid
     esdt  = @test_collection.esdt
     @response_xml = @egi.submit_sync_request(esdt, granuleid, service_agent, format:format)
     puts @egi.egi_service_request_url
end

Then(/^the EGI "([^"]*)" request should succeed for the  h5 test granule for "([^"]*)" esdt and "([^"]*)"
format$/) do |agent, esdt, fmt|
     format = fmt.strip
     service_agent = agent.strip
     collection = esdt.strip
     puts @response_xml
     response = @egi.parse_response(@response_xml)
     puts response
     response.should_not be_nil
     result = response.message.eql?("SUCCESS") ? true:false
     result.should be_truthy, "#{collection} format : #{format}  #{service_agent} request failed"
end

When(/^Submit a EGI "([^"]*)" request without the excluded_bands for the h5 test granule for the "([^"]*)" esdt
and "([^"]*)" format$/) do |agent, esdt, fmt|
     format = fmt.strip
     service_agent = agent.strip
     @egi = Sdps::Da::Egi.new(ENV['MODE'])
     granuleid = @test_granule.granuleid
     esdt  = @test_collection.esdt

     puts "bands excluded=#{@bands_excluded}"
     puts "test bands=#{@test_bands}"
```

```
     test_bands1 = Array.new
     if !@bands_excluded.nil?  and  !@bands_excluded.empty?
            @test_bands.each { |band|
                 test_bands1.push(band)
            }
            band_str = test_bands1.join(",")
            @response_xml = @egi.submit_request(esdt, granuleid, service_agent,
format:format,band_parameter:band_str)
      else
            @response_xml = @egi.submit_request(esdt, granuleid, service_agent, format:format)
     end
      puts @egi.egi_service_request_url
end

Then(/^the EGI "([^"]*)" request without the excluded_bands should succeed for the  h5 test granule for
"([^"]*)" esdt and "([^"]*)" format$/) do |agent, esdt, fmt|
     format = fmt.strip
     service_agent = agent.strip
     collection = esdt.strip
     puts @response_xml
     response = @egi.parse_response(@response_xml)
     puts response
     response.should_not be_nil
     result = response.message.eql?("SUCCESS") ? true:false
     result.should be_truthy, "#{collection} format : #{format}  #{service_agent} request failed"
end

When(/^Submit a EGI "([^"]*)" request for the test granule for the "([^"]*)" esdt for each band parameter and
"([^"]*)" format$/) do |agent, esdt, fmt|
      format = fmt.strip
     service_agent = agent
     @response_granule_xmls  = Hash.new()
     @egi = Sdps::Da::Egi.new(@mode)
             granuleid = @test_granule.granuleid
             esdt  = @test_collection.esdt
             dataset_id = @test_collection.datasetid
```

```
             shortname = @test_collection.shortname
             @response_band_xmls = Hash.new()

             @test_bands.each { |band|
                  #prepend band with shortname
                 # band1 = "/#{shortname}#{band}"
                 puts esdt
                 puts granuleid
                 puts service_agent
                 puts format
                 puts band
                 response_xml = @egi.submit_sync_request(esdt, granuleid, service_agent, format:format,
band_parameter:band)
                 # response_xml = @egi.submit_request(esdt, granuleid, @service_agent, format:@format,
band_parameter:band1)
                 #response_xml = @egi.submit_post_request(esdt, granuleid, @service_agent, format:@format,
band_parameter:band1)
                 @response_band_xmls[band] = response_xml
                  puts @egi.egi_service_request_url

             }
end

When(/^the EGI "([^"]*)" request succeeds for the test granule for "([^"]*)" esdt and band parameter and
"([^"]*)" format$/) do  |agent, esdt, fmt|

       format = fmt.strip
       service_agent = agent.strip
       collection = esdt.strip
       @response_results = Hash.new()
        @bands_to_not_exclude = Array.new
        @bands_to_exclude = Array.new

        @response_band_xmls.keys.each do |band|
           response_xml = @response_band_xmls[band]
           response = @egi.parse_response(response_xml)
```

```
            response.should_not be_nil
            result = response.message.eql?("SUCCESS") ? true:false
            puts "#{esdt}, #{format}, #{band}, #{response.message}"
            @response_results[band] = response
            if response.message.eql? "SUCCESS"
               @bands_to_not_exclude.push (band)
            else
              @bands_to_exclude.push(band)
            end
         end
end

Then(/^the band parameter and "([^"]*)" format should not be configured for exclusion for "([^"]*)" esdt and
"([^"]*)" service$/) do  |agent, esdt, fmt|
      @incorrect_excluded_bands = Array.new
      if !@bands_excluded.nil?
         @bands_to_not_exclude.each do |band|
              if @bands_excluded.include?(band)
                   @incorrect_excluded_bands.push ("#{band}")
              end
         end
     end
     @incorrect_excluded_bands.should be_empty, "#{@incorrect_excluded_bands.to_s} should not be excluded for
#{@test_collection.esdt} , #{@service_agent}, #{@format}"
end

When(/^the EGI "([^"]*)" request fails for the test granule for "([^"]*)" esdt and band parameter and "([^"]*)"
format$/) do |
agent, esdt, fmt|
   #puts @bands_excluded
    @incorrect_not_excluded_bands = Array.new
    if !@bands_to_exclude.nil?
            @bands_to_exclude.each do |band|
                if @bands_excluded.nil? or  !@bands_excluded.include?(band)
                      @incorrect_not_excluded_bands.push("#{band}")
                end
```

```
          end
   end
end

Then(/^the band parameter and "([^"]*)" format should be configured for exclusion for "([^"]*)" esdt and
"([^"]*)" service$/) do  |agent, esdt, fmt|
  #p @bands_to_exclude.to_s
    @incorrect_not_excluded_bands.should be_empty, "#{@incorrect_not_excluded_bands.to_s} should be excluded for
#{@test_collection.esdt} , #{@service_agent}, #{@format}, #{@projection}"
end

Current Test Script: egi_smap_spatial_subset.rb
When(/^Submit a EGI "(.*?)" request for each of the SMAP  test granules for  "(.*?)" with no spatial
subsetting$/) do |agent, format|
     @format = format
     @service_agent = agent
     @response_granule_xmls  = Hash.new()
     @egi = Sdps::Da::Egi.new(@mode)
             granuleid = @test_granule.granuleid
             esdt  = @test_collection.esdt
             dataset_id = @test_collection.datasetid
             shortname = @test_collection.shortname
             @response_no_subset_xml = @egi.submit_service_request(granuleid, esdt, dataset_id, "", format,
agent)

end

Then(/^the EGI service request succeeds for the SMAP  test granule and format$/) do
     collection = @test_collection
     esdt = collection.esdt
        @response_results = Hash.new()

        response = @egi.parse_response(@response_no_subset_xml)
        response.should eql "SUCCESS"
end
```

```
Then(/^the output file is in  the requested format$/) do
  pending # express the regexp above with the code you wish you had
end

Then(/^the output file is validated$/) do
  pending # express the regexp above with the code you wish you had
end

When(/^Submit a EGI service request for each of the SMAP  test granules for  the format  with "(.*?)"$/) do
|spatial_subset|
  pending
end

Then(/^the EGI service request succeeds for the SMAP  test granule and format with spatial subset$/) do
  pending # express the regexp above with the code you wish you had
end

Then(/^the output file is in the requested format$/) do
  pending # express the regexp above with the code you wish you had
end

Then(/^the output file is valid for the requested spatial subset$/) do
  pending # express the regexp above with the code you wish you had
end

Current Test Script: common_egi_steps.rb
# To change this license header, choose License Headers in Project Properties.
# To change this template file, choose Tools | Templates and open the template
# in the editor.

puts "Hello World"
Current Test Script: egi_hegservice_smap_nosubset.rb
Given(/^ECS\-ECSTC\-(\d+) testdata for "(.*?)" and "(.*?)" and "(.*?)" is checked out$/) do  |testcasenumber,
localgranuleid, shortname, versionid|
     @testcase = testcasenumber
     @localgranuleid = localgranuleid
```

```
    @shortname = shortname
    @versionid = versionid
    @db_conn.should_not be_nil
    @so_test_data = Sdps::SoTestData.new(@db_conn)
    @so_test_data.should_not be_nil

    #@so_test_data_items = @so_test_data[testcasenumber]
    #@so_test_data_items.should_not be_nil
    #@so_test_data_items.length.should_not be_zero

    @test_granules = Hash.new()
    @test_esdts = Hash.new()
    @test_hdfs = Hash.new()

    granules = @aimdb.find_granules_by_localgranuleid(localgranuleid)
       granules.each {  |granule|
       if granule.deleteeffectivedate == nil and granule.deletefromarchive != 'Y'
            filename = @aimdb.get_granule_datafile(granule.granuleid)
            filename.should_not be_nil #, "filename is null for granuleid#{{granule.granuleid}"

            if File.exists?(filename) and File.file?(filename)
                 @test_granules[granule.granuleid] = granule
                 #@test_esdts[granule.granuleid] = so_test_data_granule.esdt
                 @test_hdfs[granule.granuleid] = filename
            else
                puts "#{filename} does not exist for granule #{granule.granuleid}"
             end
       end
      }

   @test_granules.length.should_not be_zero, " no test granules"


end

When(/^Submit a EGI HegService request for each of the SMAP L3L4 test granules$/) do
```

```
     @response_xmls  = Hash.new()
     @egi = Sdps::Da::Egi.new(@mode)
     @granuleids = @test_granules.keys.join(',')
     @test_granules.keys.each do |granuleid|
             #esdt = @test_esdts[granuleid]
             hdf = @test_hdfs[granuleid]
             granule = @test_granules[granuleid]
             response_xml = @egi.submit_request(@granuleids, dataset_id, Sdps::Da::Heg::SERVICE_NAME,
"sdpstester@gmail.com")
             @response_xmls[granuleid] = response_xml
     end #each hdf
end

Then(/^the EGI HegService request should succeed for each SMAP L3L4 test granules$/) do |arg1, arg2|
  pending # express the regexp above with the code you wish you had
end

Then(/^the converted files from EGI HegService should appear in the results$/) do
  pending # express the regexp above with the code you wish you had
end

Then(/^verify that the bands in output match the input SMAP L3L4 granules$/) do |arg1, arg2|
  pending # express the regexp above with the code you wish you had
end

Then(/^verify that the attributes in the output match the input SMAP L3L4 granules$/) do |arg1, arg2|
  pending # express the regexp above with the code you wish you had
end

Then(/^verify that the fill values in the output match the input SMAP L3L4 granules$/) do |arg1, arg2|
  pending # express the regexp above with the code you wish you had
end

Current Test Script: ftppull_request.rb

Given(/^multiple type of granules exist$/) do
```

```
  pending # express the regexp above with the code you wish you had
end

Given(/^multiple types of granules can be ordered$/) do
  pending # express the regexp above with the code you wish you had
end

Then(/^order granules in "(.*?)" with ftppull$/) do
  #create grans to add
  @request.build(@gran_list)
  @request.submit
end

Then(/^verify the granule files are downloadable$/) do
  @request.verify_files_downloadable
end

Then(/^verify the request displays in gui$/) do
  @request.displayed_in_gui?
end

Then(/^verify the request ships$/) do
  @request.shipped?
end

Then(/^verify the granules are placed on order$/) do
  @request.grans.each do |granule|
    if(granule.on_order?)
      true
    end
  end
end
```

```
Current Test Script: common.rb
Given(/^OrderManager is configured and up$/) do

  @mode = ENV['MODE']
  puts "MODE:"
  puts @mode

  @db_subsystem = Sdps::Om::DB_SUBSYSTEM
  puts "SUBSYSTEM:"
  puts @db_subsystem

  @db_conn = Sdps::Db.connect(@db_subsystem, @mode)
  @omdb = Sdps::Om::OmDb.new(@db_conn)
  @aimdb = Sdps::AimDb.new(@db_conn)
  @omserver = Sdps::Om::OmServer.new(@mode,@db_conn)

  if @omserver.is_up? && @omserver.db_confg.length > 0 && @omserver.server_config.length > 0
    true
  end

endCurrent Test Script: request_granules.rb

Given(/^multiple of granules exist$/) do
  pending # express the regexp above with the code you wish you had
end

Given(/^multiple types of granules can be ordered$/) do
  pending # express the regexp above with the code you wish you had
end
Current Test Script: mail.rb
Then(/^verify DN email is sent$/) do
  pending # express the regexp above with the code you wish you had
end
```

```
Given(/^email is properly formed$/) do
  pending # express the regexp above with the code you wish you had
end
Current Test Script: ems_extract_utility.rb
$LOAD_PATH << "/workingdata/autotest/mstanley/lib/"

require 'sdps/ems'

When(/^the directory listed for the EMSEXTRACTDIR tag in the EcDbEMSdataExtractor\.CFG file contains the dataset
files$/) do
  @mode = ENV['MODE']

  if( @mode ==  nil )
    @mode = "DEV02"
  end

  @mode.should_not be_nil
  @extractor_cfg = "/usr/ecs/#{@mode}/CUSTOM/cfg/EcDbEMSdataExtractor.CFG"
  @dataset_dir = EMS.EMS_get_cfg( @extractor_cfg, "EMSEXTRACTDIR" )
end

Then(/^verify that the name given to a file is the same name listed in the EcEMSextractRecord table for that
extract run$/) do
  # assign table
  table = "EcEMSextractRecord"

  # find a filename from the directory
  @fileName = ""

  # construct a record using known data
  @record = { :ExtractFileName => @fileName }

  # use the record to find the correct record in the db
  @extractRecord = EMS.find_record_in_table( table, @record )

  if @fileName != @extractRecord[:Name]
```

```
    fail( ArgumentError.new( "Filename #{@fileName} is not extract record's name
#{@extractRecord[:ExtractName]}" ) )
  end
end

#-------------------------------------------------------------------------------------#
Then(/^verify that the name contains the Beginning date for the reporting period$/) do
  reportPeriodBeginDate = ""

  if reportPeriodBeginDate != nil && reportPeriodBeginDate != ""
    @extractRecord[:Name].include?( reportPeriodBeginDate )
  else
    fail( ArgumentError.new( "Reporting Period Begin Date is empty" ) )
  end
end

And(/^verify that the name contains the DAAC name$/) do
  filename = @extractRecord[:Name]
  if filename == nil
    fail( ArgumentError.new( "Filename is nil" ) )
  elsif !filename.include? "EDF"
    fail( ArgumentError.new( "Filename does not include DAAC name 'EDF'" ) )
  else
    true
  end
end

And(/^verify that the name contains the EMS flat file type$/) do
  if @extractRecord[:Name].include?( "DistFTP" )
    true
  elsif @extractRecord[:Name].include?( "DistHTTP" )
    true
  else
    fail( ArgumentError.new( "Extracted Record's 'Name' field does not include an EMS flat file type" ) )
  end
end
```

```
Then(/^verify that a record of the run can be found in the log that is created$/) do
  @logFile = "/usr/ecs/#{@mode}/CUSTOM/logs/EcDbEMSdataExtractor.log"
  EMS.EMS_find_record_in_log( @logFile, @mode, @startDate, @endDate, @dataSet )
end

And(/^verify that there is a record of the run process in the EcEMSextractRecord table$/) do

  @record = {   :ExecutionMode  => "#{@mode}",
                :StartDate      => "#{@startDate}",
                :EndDate        => "#{@endDate}",
                :ExtractType    => "#{@dataSet}" }


  EMS.EMS_find_record_in_table( "EcEMSextractRecord", @record )
end

And(/^verify that in extracted dataset files, there is one record per line$/) do
  EMS.EMS_one_record_per_line( datasetFile )
end

Then(/^verify that each record is divided into fields using '\|&\|' as a delimiter$/) do
  EMS.EMS_each_record_delimited( datasetFile )
end

And(/^verify that one output dataset file is created for dataset and date range$/) do
  destination = "/datapool/#{@mode}/user/FS1/EMS/DESTINATIONDIR"

  count = 0

  monthHash = { :Jan => "01",
                :Feb => "02",
                :Mar => "03",
                :Apr => "04",
                :May => "05",
                :Jun => "06",
```

```
              :Jul => "07",
              :Aug => "08",
              :Sep => "09",
              :Oct => "10",
              :Nov => "11",
              :Dec => "12" }

modifiedEndDate = nil # endDate in yyyymmdd
modifiedStartDate = nil # startDate in yyyymmdd

if( @startDate =~ /(\w+)\s+(\w+)\s+(\w+)/ )
  dateArray = @startDate.match( /(\w+)\s+(\d+)\s+(\d+)/ )
  monthSymbol = dateArray[1].to_sym
  modifiedStartDate = "#{dateArray[3]}#{monthHash[monthSymbol]}#{dateArray[2]}"
end

if( @endDate =~ /(\w+)\s+(\w+)\s+(\w+)/ )
  dateArray = @endDate.match( /(\w+)\s+(\d+)\s+(\d+)/ )
  monthSymbol = dateArray[1].to_sym
  modifiedEndDate = "#{dateArray[3]}#{monthHash[monthSymbol]}#{dateArray[2]}"
end

if modifiedEndDate != nil
  Dir.foreach( destination ) { |f|
    if f.include?( "rev" )
      # get most recent rev?
    elsif ( f.include?( @dataSet ) && f.include?( modifiedEndDate ) && f.include?( "DataPool" ) )
      count += 1
    end
  }
else
  fail( ArgumentError.new( "End date is nil" ) )
end

if count == 1
  true
```

```
  elsif count == 0
    fail( ArgumentError.new( "No output dataset file found for dataset #{@dataSet} with end date
#{modifiedEndDate}" ) )
  else
    fail( ArgumentError.new( "More than one output dataset file was created for dataset #{@dataSet} with end
date #{modifiedEndDate}" ) )
  end

  @fileName = "#{modifiedEndDate}_EDF_#{@dataSet}_#{@mode}Scp.flt"

end

And(/^verify that the file is scp'd to the configured destination$/) do
  destination = "/datapool/#{@mode}/user/FS1/EMS/DESTINATIONDIR"
  if !Dir.entries( destination ).include?( @fileName ) # verify if file is found in dir
    fail( ArgumentError.new( "'#{@filename}' not found in '#{destination}'" ) )
  end
end

And(/^verify that the file name created is the same except for ".rev1" appended to the end of the name$/) do
  @revCount = 1
  destination = "/datapool/#{@mode}/user/FS1/EMS/DESTINATIONDIR"

  #find most recently created file in the destination directory
  Dir.chdir( destination )

  @filenName = Dir.glob( "*rev*" ).max_by {|f| File.mtime(f)}

  # check to see if returned file includes a date within the start-end range

  puts @filenName
  @revCount = @filenName.match( /^.+rev(\d+)$/ )[1]

  revFileName = "#{@fileName}.rev#{@revCount}"
  if !Dir.entries( destination ).include?( revFileName )
    fail( ArgumentError.new( "'#{revFileName}' not found in '#{destination}'" ) )
```

```
  end
end

And(/^verify that for each subsequent run of the command, the rev number is incremented by 1$/) do
  @revCount += 1
  destination = "/datapool/#{@mode}/user/FS1/EMS/DESTINATIONDIR"
  revFileName = "#{@fileName}.rev#{@revCount}"
  if !Dir.entries( destination ).include?( revFileName )
    fail( ArgumentError.new( "'#{revFileName}' not found in '#{destination}'" ) )
  end
end

Given(/^the defaults for execution of the script in default mode are set about a week earlier than the current
date$/) do
  weekAgo = Time.now - 604800
  twoWeeksAgo = weekAgo - 604800

  @startDate = "#{twoWeeksAgo.month} #{twoWeeksAgo.day} #{twoWeeksAgo.year}"
  @endDate   = "#{weekAgo.month} #{weekAgo.day} #{weekAgo.year}"
end

Then(/^locate the DATASET file in the bcp extract directory and remove it$/) do
  system( "rm #{@dataset_dir}#{@fileName}" )
end

And(/^verify that the files that have had the FTPcompletionTime set to NULL and removed from the directory have
had their FTPcompletionTime updated$/) do
  # is this in the file, or the db?  Either way, it's non-trivial to figure out where to find this NULL
  @completionTime = "" #this needs to be found
end

And(/^verify that a corresponding note for this file has been place in the log$/) do
  # find location of log, then...
  @ecsHome = ENV['ECS_HOME']
  log = "#{@ecsHome}/#{@mode}/CUSTOM/logs/EcDbEMSdataExtractor.log"
  line = "update_FTPcompletionTime ( EcDbEMSdataExtractor.log, #{@completionTime} )"
```

```
  system( "grep #{line} #{log}" )
end

And(/^verify that the script finishes$/) do
  # how do you verify a script is finished after you call it? do nothing?
  true
end

Then(/^verify that output dataset files are created for each dataset$/) do
  system( "find #{@dataset_dir} -name *DistFTP* " )
  system( "find #{@dataset_dir} -name *DistHTTP* " )
end

And(/^verify the date range specified$/) do

  # make list of dates between startDate and endDate range
  dateList =  EMS.EMS_get_date_list( @startDate, @endDate )

  # check the directory to see if there's a file
  if !dateList.empty?
    dateList.foreach { |date| system( "find #{@dataset_dir} -name *#{date}* " ) }
  else
    fail( ArgumentError.new( "Date list is empty" ) )
  end
end

And(/^verify that a dataset file is created for each 24-hour period$/) do

  # make list of dates between startDate and endDate range
  dateList =  EMS.EMS_get_date_list( @startDate, @endDate )

  # check the directory to see if there's a file
  if !dateList.empty?
    dateList.foreach { |date|
      if system( "find #{@dataset_dir} -name *#{date}*DistFTP*" )
      elsif system( "find #{@dataset_dir} -name *#{date}*DistHTTP*" )
```

```
      else
        fail( ArgumentError.new( "Dataset file was not created for date '#{date}'" ) )
      end
    }
    true
  else
    fail( ArgumentError.new( "Date list is empty" ) )
  end
end

And(/^verify that no output datasets are created for this specified date range$/) do

  # make list of dates between startDate and endDate range
  dateList =  EMS.EMS_get_date_list( @startDate, @endDate )

  # check the directory to see if there's a file
  if !dateList.empty?
    dateList.foreach { |date|
      if system( "find #{@dataset_dir} -name *#{date}*" )
        fail( ArgumentError.new( "Dataset file was created for date '#{date}'" ) )
      end
    }
    true
  else
    true
  end
end

Then(/^verify that dataset files are created for that date forward$/) do
  # make list of dates between startDate and currentDate range

  @startDate.should_not be_nil

  current = Time.now

  @endDate = "#{current.month} #{current.day} #{current.year}"
```

```
  @extractor_cfg = "/usr/ecs/#{@mode}/CUSTOM/cfg/EcDbEMSdataExtractor.CFG"
  @dataset_dir = EMS.EMS_get_cfg( @extractor_cfg, "EMSEXTRACTDIR" )

  dateList = EMS.EMS_get_date_list( @startDate, @endDate )

  if !dateList.empty?
    dateList.foreach { |date|
      if( system( "find #{@dataset_dir} -name *#{date}*DistFTP*" ) || system( "find #{@dataset_dir} -name
*#{date}*DistHTTP*" ) )
        count += 1
      end
    }
  end

  # check the directory to see if there's a file
  if count == 0
    fail( ArgumentError.new( "Date list is empty" ) )
  end
end

And(/^verify that the command has specified a single flat file type for generation and export for each 24-hour
period between the start date and end date$/) do
  # for each day between @startDate and @endDate, check to see if file for each day is there

  dateList = [] # make list of dates between startDate and endDate range

  # check the directory to see if there's a file
  if !dateList.empty?
    dateList.foreach { |date|
      if system( "find #{@dataset_dir} -name *#{date}*DistFTP*" )
      elsif system( "find #{@dataset_dir} -name *#{date}*DistHTTP*" )
      else
        fail( ArgumentError.new( "Dataset file was not created for date '#{date}'" ) )
      end
    }
```

```
  else
    fail( ArgumentError.new( "Date list is empty" ) )
  end
end

And(/^verify the flat file created contains QA or PH data$/) do
  # check the file for whatever designates QA or PH data is in a dataset
  if system( "grep |&|QA|&| #{file}" )
    true
  elsif system( "grep |&|PH|&| #{file}" )
    true
  end
  fail( ArgumentError.new( "Flat file lacks both QA or PH data" ) )
end


When(/^MODE is set$/) do
  @mode = ENV['MODE']

  if( @mode ==  nil )
    @mode = "DEV02"
  end

  @mode.should_not be_nil
end

When(/^MODE is set to TS1$/) do
  @mode = "TS1"
end

And(/^DATASET is set$/) do
  @dataSet = "DistFTP"
  @dataSet.should_not be_nil
end

And(/^DATASET is set to DistFTP$/) do
```

```
  @dataSet = "DistFTP"
  @dataSet.should_not be_nil
end

Then(/^the EMS Extract Utility is ran$/) do
  EMS.EMS_data_extractor( @mode, @startDate, @endDate, @verbose, @initial, @dataSet, @override )
end

Then(/^the EMS Extract Utility is ran with the verbose flag$/) do
  @verbose = true
  EMS.EMS_data_extractor( @mode, @startDate, @endDate, @verbose, @initial, @dataSet, @override )
end

Then(/^the EMS Extract Utility is ran with the verbose and override flags$/) do
  @verbose = true
  @override = true
  EMS.EMS_data_extractor( @mode, @startDate, @endDate, @verbose, @initial, @dataSet, @override )
end

Then(/^the EMS Extract Utility is ran with the verbose and initial flags$/) do
  @verbose = true
  @initial = true
  EMS.EMS_data_extractor( @mode, @startDate, @endDate, @verbose, @initial, @dataSet, @override )
end

And(/^the EMS Extract Utility is ran with only an end date$/) do
  @endDate.should_not be_nil
  @verbose = false
  EMS.EMS_data_extractor( @mode, nil, @endDate, @verbose, @initial, @dataSet, @override )
end

And(/^the EMS Extract Utility is ran with only a start date$/) do
  @startDate.should_not be_nil
  EMS.EMS_data_extractor( @mode, @startDate, nil, @verbose, @initial, @dataSet, @override )
end
```

```
When(/^the EMS Extract Utility is ran with the verbose and initial flags and with the data set if available$/)
do
  @mode.should_not be_nil
  @startDate.should_not be_nil
  @endDate.should_not be_nil

  if @verbose && @initial
    EMS.EMS_data_extractor( @mode, @startDate, @endDate, @verbose, @initial, @dataSet, @override )
  end
end

When(/^"<start date>" is set$/) do
  @startDate = "Dec 01 2013"
end

When(/^"<start date>" to "<end date>" is set$/) do
  @startDate = "Dec 01 2013"
  @endDate = "Dec 02 2013"
end

When(/^"<start date>" to "<end date>" is greater than a 24-hour period$/) do
  @startDate = "Dec 01 2013"
  @endDate = "Dec 03 2013"
end

When(/^"<start date>" to "<end date>" is less than a 24-hour period$/) do
  @startDate = "Dec 01 2013"
  @endDate = "Dec 01 2013"
end

And(/^"<end date>" is 2 days prior to the current date$/) do
  twoDaysAgo = Time.now - 172800
  @endDate = "#{twoDaysAgo.month} #{twoDaysAgo.day} #{twoDaysAgo.year}"
end

When(/^DESTINATIONDIR is set in the configuration file$/) do
```

```
  file = "/datapool/#{@mode}/user/FS1/EMS/DESTINATIONDIR" # set config file path here
  if file != nil && file != ""
    system( "grep DESTINATIONDIR #{file}" )
  else
    fail( ArgumentError.new( "File name is nil or empty" ) )
  end
end

And(/^verify that a record for this dataset run has been entered in the EcEMSextractRecord table$/) do
  @mode.should_not be_nil
  @startDate.should_not be_nil
  @endDate.should_not be_nil
  @dataSet.should_not be_nil

  @record = {   :ExecutionMode  => "#{@mode}",
                :StartDate      => "#{@startDate}",
                :EndDate        => "#{@endDate}",
                :ExtractType    => "#{@dataSet}" }

  EMS.EMS_find_record_in_table( "EcEMSextractRecord", @record )

end

And(/^verify that the ExecutionMode for the record is recorded as override in the table$/) do

  @mode.should_not be_nil
  @startDate.should_not be_nil
  @endDate.should_not be_nil
  @dataSet.should_not be_nil

  @record = {   :ExecutionMode  => "#{@mode}",
                :StartDate      => "#{@startDate}",
                :EndDate        => "#{@endDate}",
                :ExtractType    => "#{@dataSet}" }

  result = EMS.EMS_find_record_in_table( "EcEMSextractRecord", @record )
```

```
  result.include?( "override" )

end

And(/^verify that after execution, files have been created for each DATASET up to the current date$/) do

  @startDate.should_not be_nil
  @endDate.should_not be_nil

  @extractor_cfg = "/usr/ecs/#{@mode}/CUSTOM/cfg/EcDbEMSdataExtractor.CFG"
  @dataset_dir = EMS.EMS_get_cfg( @extractor_cfg, "EMSEXTRACTDIR" )

  dateList = EMS.EMS_get_date_list( @startDate, @endDate )

  if !dateList.empty?
    dateList.foreach { |date|
    if( system( "find #{@dataset_dir} -name *#{date}*DistFTP*" ) || system( "find #{@dataset_dir} -name
*#{date}*DistHTTP*" ) )
      count += 1
    end
    }
  end

  # check the directory to see if there's a file
  if count == 0
    fail( ArgumentError.new( "Date list is empty" ) )
  elsif count < 6
    fail( ArgumentError.new( "Date list contains less than a weeks worth of files" ) )
  end
end

And(/^verify that the file is located in the extract directory$/) do
  # get directory
  extractDir = EMS.EMS_get_cfg( @extractor_cfg, "EMSEXTRACTDIR" )

  # check directory for filename
```

```
  File.exist?( extractDir + @fileName )

end

And(/^verify that the files that have had the FTPcompletionTime set to NULL have been scp'd to the sftp site$/)
do
  # get FTP site
  ftpSite = EMS.EMS_get_cfg( @extractor_cfg, "IPADDRESS" )

  destDir = EMS.EMS_get_cfg( @extractor_cfg, "DESTINATIONDIR" )

  # check if fileName exists there, since fileName was already used to designate the FTPcompletionTime-null'ed
@record
  File.exist?( ftpSite + ":" + destDir + @fileName )

end

And(/^verify that the FTPcompletionTime has been updated$/) do

  # store filename in @record
  @record = { :ExtractFileName  => "#{@fileName}" }

  # get filename's actual record
  result = EMS.EMS_find_record_in_table( "EcEMSextractRecord", @record )

  # check for NULL
  result[:FTPcompletionTime] != nil && result[:FTPcompletionTime] != ""

end

And(/^compare with record from database to ensure that the data in flat file matches the record in database$/)
do

  # create hash for record from known values
  @record = {   :ExecutionMode  => "#{@mode}",
                :StartDate      => "#{@startDate}",
```

```
                :EndDate        => "#{@endDate}",
                :ExtractType    => "#{@dataSet}" }


  # find record in flat file
  fileResult = EMS.EMS_find_record_in_flatfile( flatFile, @record )

  # find record in database
  tableResult = EMS.EMS_find_record_in_table( "EcEMSextractRecord", @record )

  # compare both
  ( fileResult != "" && fileResult != nil ) && ( tableResult != "" && tableResult != nil )

end

Then(/^verify that for command run above that "(.*?)" has been written to the EcEMSextractRecord table
ExecutionMode column for the dataset$/) do |default|

  dateList = EMS.EMS_get_date_list( @startDate, @endDate )

  if !dateList.empty?
    dateList.foreach { |date|
      @record = {       :ExecutionMode  => "#{default}",
                        :StartDate      => "#{date}",
                        :ExtractType    => "#{@dataSet}" }
      tableResult = EMS.EMS_find_record_in_table( "EcEMSextractRecord", @record )

      if tableResult == nil || tableResult == ""
        fail( ArgumentError.new( "Expected record not found" ) )
      end
    }
  end
end

And(/^verify that the starting and ending reporting period matches the start date of each dataset file$/) do
```

```
  if( @fileName =~ /(\d+)_\S+/ )
    # get dataset start date from file name (first 8 characters)
    fileDate = @fileName.match( /(\d+)_\S+/ )[1]
  end

  # verify this date is between the startDate and endDate
  EMS.EMS_get_date_list( startDate, endDate ).includes?( fileDate )
end

And(/^verify there is no entry in the EcEMSextractRecord table for this specified date range$/) do

  @startDate.should_not be_nil
  @endDate.should_not be_nil

  @record = {   :StartDate      => "#{@startDate}",
                :EndDate        => "#{@endDate}" }

  result = EMS.EMS_find_record_in_table( "EcEMSextractRecord", @record )

  ( result == "" || result == nil )

end

Then(/^Set\-up the cron by running the crontab \-e command on the server that the cron will be run$/) do
  system( "crontab -e" )
end

Given(/^a record for the file in the EcEMSextractRecord table, in this table for that record set the
FTPcompletionTime to NULL$/) do

  # create hash for record from known values
  @record = {   :ExecutionMode  => "#{@mode}",
                :StartDate      => "#{@startDate}",
                :EndDate        => "#{@endDate}",
                :ExtractType    => "#{@dataSet}" }
```

```
  # find record in database
  tableResult = EMS.EMS_find_record_in_table( "EcEMSextractRecord", @record )

  tableResult[:FTPcompletionTime] == nil || tableResult[:FTPcompletionTime] == ""
end

#####################################

And(/^includes DataPool Web accesses to QA and PH data$/) do
  pending # HOW IS THIS DETERMINED?
end

And(/^includes FTP OMS orders of QA or PH granules$/) do
  pending # HOW IS THIS DETERMINED?
end

Then(/^verify that the datasets have been run up to the time specified by the LAG configuration variable$/) do
  # parser, "LAG =<some int>", where "some int" means something, but I guess "0" means "now"?
  # 0 means up until yesterday, and -1 means the day before yesterday
  # I'm guessing 1 means up until today?
  # 2 means today included?

  # so, get the LAG value from the CFG file, and compare this value to the end date of the most recent entry in
the db is
  # whatever day the LAG value says, or maybe do the same for the flat file generated that corresponds with that
run?

  # get lag value from CFG file

  lagValue = ""
  daysAgo  = ""

  if lagValue == 0
    daysAgo = Time.now - 172800 # two days ago
  elsif  lagValue == 1
    daysAgo = Time.now - 86400  # one day ago
```

```
  elsif  lagValue == 2
    daysAgo = Time.now          # today
  else # error
    fail( ArgumentError.new( "Behavior for given LAG value is unknown" ) )
  end

  endDate = "#{daysAgo.month} #{daysAgo.day} #{daysAgo.year}"

end

When(/^"(.*?)" to "(.*?)" is set for date range that contains data populating intendedUsage$/) do |startDate,
endDate|
  pending # express the regexp above with the code you wish you had
end

Then(/^verify that the dataset contains intendedUsage data$/) do
  pending # express the regexp above with the code you wish you had
end

And(/^verify that the starting and ending reporting period matches the start date and end date entered in the
command$/) do
  # this has already been accomplished if the previous step is completed, since the values used to run the
script are also
  # used to search for the record in the database
  true
end

When(/^the current user is the one that the cron will be ran as$/) do
  # no way to test this, but the assumption can be made that whoever runs the cron is who it will be ran as?
  true
end
Current Test Script: test_data.rb
Given(/^testgranule is available for "([^"]*)" in the mode$/) do | esdt |
   begin

      db_conn = Sdps::Db.connect('aim', ENV['MODE'])
```

```
      db_conn.should_not be_nil
      aimdb = Sdps::AimDb.new(db_conn)
      @test_esdt = esdt.strip
      granules = aimdb.find_notdeleted_granules_by_esdt(@test_esdt)
      granules.each {  |granule|
        #if granule.deleteeffectivedate == nil and granule.deletefromarchive != 'Y'
             filename = aimdb.get_granule_datafile(granule.granuleid)
             filename.should_not be_nil #, "filename is null for granuleid#{{granule.granuleid}"

             if File.exists?(filename) and File.file?(filename)
                  @test_granule = granule
                  @test_hdf = filename
                  collection = aimdb.find_collection_by_shortname_versionid(granule.shortname,
granule.versionid)
                  @test_collection =  collection
                  puts "test_granuleid: #{@test_granule.granuleid}, filename=#{@test_hdf}"
                  break
             else
                 puts "#{filename} does not exist for granule #{granule.granuleid}"
              end
        #end
       }
       @test_granule.should_not be_nil, " no test granule"
    ensure
        if !db_conn.nil?
           db_conn.close
        end
        db_conn = nil
    end
end

Given(/^testdata "([^"]*)" is available$/) do | localgranuleid |
    begin
      localgranuleid1 = localgranuleid.strip
      puts "localgranuleid1=#{localgranuleid1}"
      db_conn = Sdps::Db.connect('aim', ENV['MODE'])
```

```
      db_conn.should_not be_nil
      aimdb = Sdps::AimDb.new(db_conn)
       granules = aimdb.find_granules_by_localgranuleid(localgranuleid1)
       puts "granules=#{granules}"
        granules.each {  |granule|
        if granule.deleteeffectivedate == nil and granule.deletefromarchive != 'Y'
             filename = aimdb.get_granule_datafile(granule.granuleid)
             filename.should_not be_nil #, "filename is null for granuleid#{{granule.granuleid}"

             if File.exists?(filename) and File.file?(filename)
                  @test_granule = granule
                  @test_hdf = filename
                  collection = aimdb.find_collection_by_shortname_versionid(granule.shortname,
granule.versionid)
                  @test_collection =  collection
                  @test_esdt = collection.esdt
                  break
             else
                 puts "#{filename} does not exist for granule #{granule.granuleid}"
              end
        end
       }
    @test_granule.should_not be_nil, " no test granule"
      ensure
        if !db_conn.nil?
           db_conn.close
        end
        db_conn = nil
    end
  end


Current Test Script: common_steps.rb
#global variables : @test_bands, @test_collection, @test_hdf
#@bands_excluded
```

```
Given(/^Setup$/) do
  #clear any global variables
  @test_granule = nil
  @test_hdf   = nil
  @test_collection = nil
  @response_xml = nil
  @test_granule = nil
  @test_collection = nil
  @test_esdt = nil
  @egi = nil
  @test_bands = nil
  @bands_excluded = nil
  @mode = ENV['MODE']
end

Given(/^does not include bands containing "([^"]*)" for esdt "([^"]*)" and "([^"]*)" format and "([^"]*)"
reprojection$/) do |bands_excluded_by_design, esdt, format,projection|
  test_bands2 =    bands_excluded_by_design.strip
  if !test_bands2.nil? and !test_bands2.empty?
     test_bands1 = bands_excluded_by_design.split(",")
     puts "test bands1: #{test_bands1}"
     if !@bands_excluded.nil? and !@bands_excluded.empty?
       @bands_excluded = test_bands1
     else
       @bands_excluded += test_bands1
     end
     test_bands1.each { | band1|
        @test_bands.delete_if { |band| band.include?(band1.strip)}
     }
  end
  puts "test_bands=#{@test_bands}"
end

When(/^not excluded bands retrieved successfully for esdt "([^"]*)" for the service "([^"]*)" and "([^"]*)"
format$/) do |esdt, agent,fmt|
  format = fmt.strip
```

```
  service_agent = agent.strip
  @test_bands = Hash.new()
  filename = @test_hdf
  utils =  Sdps::Da::DaUtil.new()
  puts "esdt:#{esdt}"
  puts agent
  puts fmt
  puts "test_collection:#{@test_collection.shortname},  #{@test_collection.versionid}"
  #takes too long
  #if service_agent.eql?("ICESAT2")
   # all_bands = utils.get_non_zerolength_bands(@test_hdf)
  #else
    all_bands= utils.get_band_params(filename)
  #end
  #puts "all_bands=#{all_bands}"

  @bands_excluded = Array.new
  db_conn = Sdps::Db.connect('aim', ENV['MODE'])
  db_conn.should_not be_nil
  dadb = Sdps::Da::DaDb.new(db_conn, ENV['MODE'])
  puts ENV['MODE']
  @mode = ENV['MODE']
  if !format.nil? and !format.empty?
    @bands_excluded = dadb.get_objects_excluded_for_collection_service_format(@test_collection.shortname,
@test_collection.versionid, service_agent, format)
    puts format
    puts "excluded1=#{@bands_excluded}"
  end
  puts "bands excluded=#{@bands_excluded}"
  if !db_conn.nil?
    db_conn.close
  end
  db_conn = nil

  @test_bands = Array.new
  @test_bands = all_bands
```

```
  if !@bands_excluded.nil? and !@bands_excluded.empty?
    @bands_excluded.each { | band1|
      @test_bands.delete_if { |band| band.include?(band1.strip)}
    }
  end
  puts "testbands = #{@test_bands}"
end


Given(/^not excluded bands retrieved successfully for esdt "([^"]*)" for the service "([^"]*)" and format
"([^"]*)" and "([^"]*)" reprojection$/) do |esdt, agent, fmt, projection|

  format = fmt.strip
  service_agent = agent.strip
  reprojection = projection.strip

  puts "esdt:#{esdt}"
  puts agent
  puts fmt
  puts reprojection
  puts "test_collection:#{@test_collection.shortname},  #{@test_collection.versionid}"

  #shortname and versionid ...??
  @test_bands = Hash.new()
  filename = @test_hdf
  utils =  Sdps::Da::DaUtil.new()
  if service_agent.eql?("ICESAT2")
    all_bands = utils.get_non_zerolength_bands(@test_hdf)
  else
    all_bands= utils.get_band_params(filename)
  end
  #puts "all=#{all_bands}"
  @bands_excluded = Array.new
  bands_excluded1 = Array.new
  bands_excluded2 = Array.new
```

```
  db_subsystem = Sdps::Da::DB_SUBSYSTEM
  db_conn = Sdps::Db.connect(db_subsystem, ENV['MODE'])
  db_conn.should_not be_nil
  dadb = Sdps::Da::DaDb.new(db_conn, ENV['MODE'])

  if !format.nil? and !format.empty?
    bands_excluded1 = dadb.get_objects_excluded_for_collection_service_format(@test_collection.shortname,
@test_collection.versionid, service_agent, format)
    puts format
    puts "excluded1=#{bands_excluded1}"
  end
  if !reprojection.nil? and !reprojection.empty?
    bands_excluded2 = (dadb.get_objects_excluded_for_collection_service_projection(@test_collection.shortname,
@test_collection.versionid, service_agent, reprojection))
    puts reprojection
    puts "excluded2=#{bands_excluded2}"
  end
  if !db_conn.nil?
    db_conn.close
  end
  db_conn = nil
  @bands_excluded = bands_excluded1 + bands_excluded2
  #puts "excluded=#{@bands_excluded}"
  @test_bands = Array.new
  @test_bands = all_bands
  if !@bands_excluded.nil?  && !@bands_excluded.empty?
    #@test_bands = all_bands.select { |band| !@bands_excluded.include?(band) }
    @bands_excluded.each { | band1|

      #puts band1
      #remove empty bands
      band2 = band1.strip
      if band2 == "/"
        next
      end
      #puts band2
```

```
      @test_bands.delete_if { |band| band.include?(band2)}
    }
    #puts "test=#{@test_bands}"
  end
end

When(/^does not include bands containing "([^"]*)" for esdt "([^"]*)" and "([^"]*)" format$/) do
|bands_excluded_by_design, esdt, format|

  bands_excluded_by_design = bands_excluded_by_design.strip
  puts "bands_excluded_by_design:#{bands_excluded_by_design}"
  if !bands_excluded_by_design.nil? and !bands_excluded_by_design.empty?
    test_bands1 = bands_excluded_by_design.split(',')
  end

  puts "test bands1: #{test_bands1}"

  if !@bands_excluded.nil? and !@bands_excluded.empty?
    if !test_bands1.nil? and !test_bands1.empty?
       @bands_excluded += test_bands1
    end
  else
       @bands_excluded = test_bands1
  end
  test_bands_new = @test_bands
  if !test_bands1.nil? and !test_bands1.empty?
       #@test_bands.each { | band|
       test_bands1.each { |band1|
       @test_bands.delete_if {|band| band.include?(band1.strip)}
      #if  band.include?(band1.strip)
           #puts "band=#{band}"
           #puts "band1=#{band1}" }
          # @test_bands.delete(band)
       #else
         #  puts "band=#{band} does not include band1=#{band1}"
       #end
```

```
      #}
  }
  end
  puts "test_bands=#{@test_bands}"
end

When(/^bands retrieved successfully for esdt "([^"]*)"$/) do |arg1|
  #Given(/^bands retrieved successfully for esdt "(.*?)" $/) do |arg1|
  @test_bands = Array.new()
  filename = @test_hdf
  utils =  Sdps::Da::DaUtil.new()
  @test_bands= utils.get_band_params(filename)
  puts filename
  puts  @test_bands
end

Given(/^bands retrieved successfully for granule "(.*?)" for ESI request$/) do |testgranule|
  @test_bands = Array.new()
  filename = @test_hdf
  dautil =  Sdps::Da::DaUtil.new()
  @test_bands= dautil.get_band_params(filename)
  puts filename
  puts @test_bands
  @test_bands.should_not be_nil
end
Current Test Script: geotiff2asc.rb
Given(/^User has made a request in "([^"]*)" and "([^"]*)" with "([^"]*)", "([^"]*)", "([^"]*)", "([^"]*)",
"([^"]*)", and "([^"]*)"$/) do | mode1 , mode2 , esdt , granuleid, projection, format, projection_param,
band_parameter |
        #clear any global variables
        @test_granule = nil
        @test_hdf = nil
        @test_collection = nil
        @test_bands = nil
        @response_xml = nil
```

```
        @egi1 = Sdps::Da::Egi.new(mode1)
        @tif_mode1 = @egi1.submit_pa_request(mode1, esdt, granuleid, projection, format, projection_param,
band_parameter)

        @egi2 = Sdps::Da::Egi.new(mode2)
        @tif_mode2 = @egi2.submit_pa_request(mode2, esdt, granuleid, projection, format, projection_param,
band_parameter)


        #def submit_pa_request(esdt, granuleid, subagent_id:"", projection:"", format:"", projection_param:"",
band_parameter:"" , bbox:"" , temporal:"", interpolation:"" , include_meta:"N")
         #   begin
          #      file_ids = "SC:#{esdt}:#{granuleid}"
          #      @egi_request1 = Sdps::Da::Egi::Request.new(@request_url1, file_ids, subagent_id, projection,
format, projection_param, band_parameter, bbox, temporal, interpolation , include_meta)
          #      @egi_service_request_url1 = @egi_request1.egi_request_url
          #      @tif_mode = `/tools/libcurl/bin/curl -k -s -L -o #{mode2}.tif -J
"#{@egi_service_request_url2}"`
          #  rescue=>e
          #      puts e.backtrace
          #      e.to_s
          #  end
        #end


        puts "#{@egi_service_request_url1}"
        puts "#{@egi_service_request_url2}"

        #@tif_mode1 = `/tools/libcurl/bin/curl -k -s -L -o #{mode1}.tif -J "#{url1}"`

        #@tif_mode2  = `/tools/libcurl/bin/curl -k -s -L -o #{mode2}.tif -J "#{url2}"`

end

And(/^User has converted a tif file in "([^"]*)" and "([^"]*)" to an asc file.$/) do | mode1 , mode2 |
```

```
        #cmd1 = `/tools/gdal/bin/gdal_translate -of AAIGrid @tif_dev07 test_dev07.asc`
        #@asc_dev07 = system(cmd1)

        #cmd2 = `/tools/gdal/bin/gdal_translate -of AAIGrid @tif_mode test_"<mode>".asc`
        #@asc_mode = system(cmd2)

        @lowercase_mode1 = "#{mode1}".downcase
        @lowercase_mode2 = "#{mode2}".downcase

        @asc_mode1 = `/tools/gdal/bin/gdal_translate -of AAIGrid #{@lowercase_mode1}.tif
test_#{@lowercase_mode1}.asc`
        @asc_mode2 = `/tools/gdal/bin/gdal_translate -of AAIGrid #{@lowercase_mode2}.tif
test_#{@lowercase_mode2}.asc`

end

When(/^User compares "([^"]*)" and "([^"]*)" asc conversions.$/) do | mode1 , mode2 |
        @result = `/usr/bin/diff --normal test_#{@lowercase_mode1}.asc test_#{@lowercase_mode2}.asc`
                puts "Result = #{@result}"
end

Then(/^The conversion is either correct or incorrect.$/) do
        if @asc_mode1 == @asc_mode2 && @result == 0
                puts "The conversion across modes is correct"
        end
        if @asc_mode1 != @asc_mode2
                puts "The conversion across modes is incorrect"
        end
end
Current Test Script: mode_consistency.rb
Given(/^User has made a request in "([^"]*)" and "([^"]*)" with "([^"]*)" and "([^"]*)"$/) do | mode1 , mode2 ,
url1, url2 |
        #clear any global variables
        @test_granule = nil
        @test_hdf = nil
        @test_collection = nil
```

```
        @test_bands = nil
        @response_xml = nil


        @tif_mode1 = `/tools/libcurl/bin/curl -k -s -L -o #{mode1}.tif -J "#{url1}"`

        @tif_mode2 = `/tools/libcurl/bin/curl -k -s -L -o #{mode2}.tif -J "#{url2}"`

end

And(/^User has converted a tif file in "([^"]*)" and "([^"]*)" to an asc file$/) do | mode1 , mode2 |
        #cmd1 = `/tools/gdal/bin/gdal_translate -of AAIGrid @tif_dev07 test_dev07.asc`
        #@asc_dev07 = system(cmd1)

        #cmd2 = `/tools/gdal/bin/gdal_translate -of AAIGrid @tif_mode test_"<mode>".asc`
        #@asc_mode = system(cmd2)

        @asc_mode1 = `/tools/gdal/bin/gdal_translate -of AAIGrid #{mode1}.tif test_#{mode1}.asc`
        @asc_mode2 = `/tools/gdal/bin/gdal_translate -of AAIGrid #{mode2}.tif test_#{mode2}.asc`

end

When(/^User compares "([^"]*)" and "([^"]*)" asc conversions$/) do | mode1 , mode2 |
        @result = `/usr/bin/diff --normal test_#{mode1}.asc test_#{mode2}.asc`
end

Then(/^The conversion is either correct or incorrect$/) do
        if @asc_mode1 == @asc_mode2
                puts "The conversion across modes is correct"
        end
        if @asc_mode1 != @asc_mode2
                puts "The conversion across modes is incorrect"
        end
end
Current Test Script: DP_7F_01_Criteria200_S-1i.rb
require 'sdps/sotestdata'
```

```
require 'sdps/aimdb'
require 'sdps/mode'
require 'sdps/db'
require 'sdps/util'
require 'sdps/ivt'
require 'sdps/publishing'
require 'sdps/ingest'

Given(/^DP_7F_01:200:I Initial Setup$/) do
  #Check mode.
  @mode = ENV['MODE']
  @mode.should_not be_nil
  @mode_checker = Sdps::Mode.new(@mode)
  @mode_checker.mode_checked_out?().should be_true

  #set up db connection
  @db_subsystem = Sdps::Bmgt::DB_SUBSYSTEM
  @conn = Sdps::Db.connect(@db_subsystem, @mode)
  @conn.should_not be_nil

  @aimdb = Sdps::AimDb.new(@conn)
  @ivt = Sdps::InventoryValidation.new(@conn,@mode)
  @publishing = Sdps::Publishing.new(@mode)
  @ingest =  Sdps::Ingest.build(@conn)

  #Initialize critinfo and SoTestData
  @critinfo = "DP_7F_01:200:I%"
  @sotestdata = Sdps::SoTestData.new(@conn)
end

When(/^DP_7F_01:200:I Ensure that the AIM Inventory Catalog includes at least one of browse granules in hidden
datapool, not logically deleted, that are associated with SC granules that are not logically deleted and not
DFAed$/) do
  @data = @sotestdata[@critinfo]
  @data.should_not be_nil
```

```
  @granules = @ivt.Get_CheckInventoryI_Granules(@ingest, @data)
  puts "Finished getting the granules"
  @granules.should_not be_nil
end

When(/^DP_7F_01:200:I Ensure that the AIM Inventory Catalog includes at least one of browse granules in hidden
datapool, logically deleted, that are associated with SC granules that are not logically deleted and not
DFAed$/) do
  puts "If it gets here, the granule is in the system"
end

When(/^DP_7F_01:200:I Execute a Data Pool inventory validation$/) do
  @ivt.RunIVT().should be_true
end

Then(/^DP_7F_01:200:I V\-8 Verify that browse granules that are missing from the Public Data Pool and are not
logically deleted \(S\-1i\) are logged by the utility$/) do
  @ivt.IsGranuleLogged(@granules[0]).should be_true
end

Then(/^DP_7F_01:200:I Verify that browse granules that are missing from the Public Data Pool and are logically
deleted are NOT logged by the utility$/) do
  @ivt.IsGranuleLogged(@granules[1]).should be_false
end

Current Test Script: DP_7F_01_Criteria200_S-1l.rb
require 'sdps/sotestdata'
require 'sdps/aimdb'
require 'sdps/mode'
require 'sdps/db'
require 'sdps/util'
require 'sdps/ivt'
require 'sdps/publishing'
require 'sdps/ingest'

Given(/^DP_7F_01:200:L Initial Setup$/) do
```

```
  #Check mode.
  @mode = ENV['MODE']
  @mode.should_not be_nil
  @mode_checker = Sdps::Mode.new(@mode)
  @mode_checker.mode_checked_out?().should be_true

  #set up db connection
  @db_subsystem = Sdps::Bmgt::DB_SUBSYSTEM
  @conn = Sdps::Db.connect(@db_subsystem, @mode)
  @conn.should_not be_nil

  @aimdb = Sdps::AimDb.new(@conn)
  @ivt = Sdps::InventoryValidation.new(@conn,@mode)
  @publishing = Sdps::Publishing.new(@mode)
  @ingest =  Sdps::Ingest.build(@conn)

  #Initialize critinfo and SoTestData
  @critinfo = "DP_7F_01:200:L%"
  @sotestdata = Sdps::SoTestData.new(@conn)
end

When(/^DP_7F_01:200:L Ensure that the AIM Inventory Catalog includes at least two of Hidden granules with
non\-null version numbers and their file names are not consistent with their version numbers recorded in the
Inventory Catalog$/) do
  @data = @sotestdata[@critinfo]
  @data.should_not be_nil

  @granules = @ivt.Get_CheckInventoryL_Granules(@ingest, @data)
  puts "Finished getting the granules"
  @granules.should_not be_nil
end

When(/^DP_7F_01:200:L Execute a Data Pool inventory validation$/) do
  @ivt.RunIVT().should be_true
end
```

```
Then(/^DP_7F_01:200:L Verify that granules in \(S\-1l\) are logged by the utility$/) do
  #The 2 granules should be logged
  @ivt.IsGranuleLogged(@granules[0]).should be_true
  @ivt.IsGranuleLogged(@granules[1]).should be_true
end

Current Test Script: DP_7F_01_Criteria200_S-1de.rb
require 'sdps/sotestdata'
require 'sdps/aimdb'
require 'sdps/mode'
require 'sdps/db'
require 'sdps/util'
require 'sdps/ivt'
require 'sdps/publishing'
require 'sdps/ingest'

Given(/^DP_7F_01:200:DE Initial Setup$/) do
  #Check mode.
  @mode = ENV['MODE']
  @mode.should_not be_nil
  @mode_checker = Sdps::Mode.new(@mode)
  @mode_checker.mode_checked_out?().should be_true

  #set up db connection
  @db_subsystem = Sdps::Bmgt::DB_SUBSYSTEM
  @conn = Sdps::Db.connect(@db_subsystem, @mode)
  @conn.should_not be_nil

  @aimdb = Sdps::AimDb.new(@conn)
  @ivt = Sdps::InventoryValidation.new(@conn,@mode)
  @publishing = Sdps::Publishing.new(@mode)
  @ingest =  Sdps::Ingest.build(@conn)

  #Initialize critinfo and SoTestData
  @critinfo = "DP_7F_01:200:DE%"
  @sotestdata = Sdps::SoTestData.new(@conn)
```

```
end

When(/^DP_7F_01:200:DE Ensure that the AIM Inventory Catalog includes at least two of Collections with ECS
granules that are public in the Data Pool and are currently flagged as logically deleted, deleted from archive ,
or hidden from normal users, as well as granules that do not meet any of these conditions, and Granules in the
Public Data Pool that contain Inventory Catalog Restriction entries that indicate the Granules should not be
Public, and Granules in the Public DataPool that have replacement granules in the public datapool$/) do
  @data = @sotestdata[@critinfo]
  @data.should_not be_nil

  #10 granules
  @granules = @ivt.Get_CheckInventoryF_DupGranOn_Granules(@ingest, @data)
  puts "Finished getting the granules"
  @granules.should_not be_nil

end

When(/^DP_7F_01:200:DE Turn DupGran switch ON$/) do
  pending # express the regexp above with the code you wish you had
end

When(/^DP_7F_01:200:DE Execute a Data Pool inventory validation$/) do
  @ivt.RunIVT().should be_true
end

Then(/^DP_7F_01:200:DE V\-3 Verify Public ECS granules that are logically deleted, DFAed or hidden \(S\-1d\) are
logged by the utility and that the log includes their status in the AIM inventory and the time they were
inserted into ECS$/) do
  @ivt.IsGranuleLogged(@granules[0]).should be_true
  @ivt.IsGranuleLogged(@granules[1]).should be_true
  @ivt.IsGranuleLogged(@granules[2]).should be_true
  @ivt.IsGranuleLogged(@granules[4]).should be_true
  @ivt.IsGranuleLogged(@granules[5]).should be_true
  @ivt.IsGranuleLogged(@granules[6]).should be_true
end
```

```
Then(/^DP_7F_01:200:DE V\-3 Verify Public ECS granules that are not logically deleted, DFAed or hidden are not
logged by the utility$/) do
  @ivt.IsGranuleLogged(@granules[3]).should be_false
  @ivt.IsGranuleLogged(@granules[7]).should be_false
end

Then(/^DP_7F_01:200:DE V\-4 Verify Public ECS granules that are Restricted with an option to un\-publish
\(S\-1e\) are logged by the utility$/) do
  @ivt.IsGranuleLogged(@granules[8]).should be_true
end

Then(/^DP_7F_01:200:DE Verify Public ECS granules that have replacement granules in the public datapool\(S\-1e\)
are logged by the utility$/) do
  @ivt.IsGranuleLogged(@granules[9]).should be_true
end

Current Test Script: DP_7F_01_Criteria200_S-1fk.rb
require 'sdps/sotestdata'
require 'sdps/aimdb'
require 'sdps/mode'
require 'sdps/db'
require 'sdps/util'
require 'sdps/ivt'
require 'sdps/publishing'
require 'sdps/ingest'

Given(/^DP_7F_01:200:FK Initial Setup$/) do
  #Check mode.
  @mode = ENV['MODE']
  @mode.should_not be_nil
  @mode_checker = Sdps::Mode.new(@mode)
  @mode_checker.mode_checked_out?().should be_true

  #set up db connection
  @db_subsystem = Sdps::Bmgt::DB_SUBSYSTEM
  @conn = Sdps::Db.connect(@db_subsystem, @mode)
```

```
  @conn.should_not be_nil

  @aimdb = Sdps::AimDb.new(@conn)
  @ivt = Sdps::InventoryValidation.new(@conn,@mode)
  @publishing = Sdps::Publishing.new(@mode)
  @ingest =  Sdps::Ingest.build(@conn)

  #Initialize critinfo and SoTestData
  @critinfo = "DP_7F_01:200:FK%"
  @sotestdata = Sdps::SoTestData.new(@conn)
end

When(/^DP_7F_01:200:FK Ensure that the AIM Inventory Catalog includes at least two Science Granules in the
Public Data Pool that should have been replaced by a replacement granule in the Hidden Data Pool that should be
public \(S\-1f\)$/) do
  @data = @sotestdata[@critinfo]
  @data.should_not be_nil

  @granules = @ivt.Get_CheckInventoryG_Granules(@ingest, @data)
  puts "Finished getting the granules"
  @granules.should_not be_nil
end

When(/^DP_7F_01:200:FK Ensure that the AIM Inventory Catalog includes at least two Science Granules in the
Public Data Pool that are replacements for granules in the Hidden Data Pool \(S\-1k\)$/) do
  puts "If it gets here, the granule is in the system"
end

When(/^DP_7F_01:200:FK Execute a Data Pool inventory validation$/) do
  @ivt.RunIVT().should be_true
end

Then(/DP_7F_01:200:FK V\-5 Verify Public ECS granules that have eligible replacement granules in the Hidden Data
Pool \(S\-1f\) are NOT logged by the utility$/) do
  #g1,g2
  @ivt.IsGranuleLogged(@granules[0]).should be_false
```

```
  @ivt.IsGranuleLogged(@granules[1]).should be_false
end

Then(/^DP_7F_01:200:FK V\-5 Verify the replacement granules in the Hidden Data Pool \(S\-1f\) are logged by the
utility$/) do
  #g3,g4
  @ivt.IsGranuleLogged(@granules[2]).should be_true
  @ivt.IsGranuleLogged(@granules[3]).should be_true
end

Then(/^DP_7F_01:200:FK V\-5 Verify the granules in S\-(\d+)k are not logged by the utility$/) do
  #g7,g8
  @ivt.IsGranuleLogged(@granules[6]).should be_false
  @ivt.IsGranuleLogged(@granules[7]).should be_false
end


Current Test Script: DP_7F_01_Criteria200_S-1j.rb
require 'sdps/sotestdata'
require 'sdps/aimdb'
require 'sdps/mode'
require 'sdps/db'
require 'sdps/util'
require 'sdps/ivt'
require 'sdps/publishing'
require 'sdps/ingest'

Given(/^DP_7F_01:200:J Initial Setup$/) do
  #Check mode.
  @mode = ENV['MODE']
  @mode.should_not be_nil
  @mode_checker = Sdps::Mode.new(@mode)
  @mode_checker.mode_checked_out?().should be_true

  #set up db connection
  @db_subsystem = Sdps::Bmgt::DB_SUBSYSTEM
```

```
  @conn = Sdps::Db.connect(@db_subsystem, @mode)
  @conn.should_not be_nil

  @aimdb = Sdps::AimDb.new(@conn)
  @ivt = Sdps::InventoryValidation.new(@conn,@mode)
  @publishing = Sdps::Publishing.new(@mode)
  @ingest =  Sdps::Ingest.build(@conn)

  #Initialize critinfo and SoTestData
  @critinfo = "DP_7F_01:200:J%"
  @sotestdata = Sdps::SoTestData.new(@conn)
end

When(/^DP_7F_01:200:J Ensure that the AIM Inventory Catalog includes at least one of Public Browse granules that
are associated only with hidden Science Granules$/) do
  @data = @sotestdata[@critinfo]
  @data.should_not be_nil

  @granules = @ivt.Get_CheckInventoryJ_Granules(@ingest, @data)
  puts "Finished getting the granules"
  @granules.should_not be_nil
end

When(/^DP_7F_01:200:J Ensure that the AIM Inventory Catalog includes at least one of Public Browse granules that
are NOT associated with any SC granules$/) do
  puts "If it gets here, the granule is in the system"
end

When(/^DP_7F_01:200:J Perform a Data Pool inventory validation$/) do
  @ivt.RunIVT().should be_true
end

Then(/^DP_7F_01:200:J V\-9 Verify that Public browse granules that are associated with Hidden Science granules
only \(S\-(\d+)j\) are NOT logged by the utility$/) do
  @ivt.IsGranuleLogged(@granules[0]).should be_false
end
```

```
Then(/^DP_7F_01:200:J Verify that Public Browse granules that are NOT associated with any SC granules are logged
by the utility$/) do
  @ivt.IsGranuleLogged(@granules[1]).should be_true
end

Current Test Script: DP_7F_01_Criteria200_S-1g.rb
require 'sdps/sotestdata'
require 'sdps/aimdb'
require 'sdps/mode'
require 'sdps/db'
require 'sdps/util'
require 'sdps/ivt'
require 'sdps/publishing'
require 'sdps/ingest'

Given(/^DP_7F_01:200:G Initial Setup$/) do
  #Check mode.
  @mode = ENV['MODE']
  @mode.should_not be_nil
  @mode_checker = Sdps::Mode.new(@mode)
  @mode_checker.mode_checked_out?().should be_true

  #set up db connection
  @db_subsystem = Sdps::Bmgt::DB_SUBSYSTEM
  @conn = Sdps::Db.connect(@db_subsystem, @mode)
  @conn.should_not be_nil

  @aimdb = Sdps::AimDb.new(@conn)
  @ivt = Sdps::InventoryValidation.new(@conn,@mode)
  @publishing = Sdps::Publishing.new(@mode)
  @ingest =  Sdps::Ingest.build(@conn)

  #Initialize critinfo and SoTestData
  @critinfo = "DP_7F_01:200:G%"
  @sotestdata = Sdps::SoTestData.new(@conn)
```

```
end

When(/^DP_7F_01:200:G Ensure that the AIM Inventory Catalog includes at least two of Granules that are
registered in the AIM Inventory Catalog but are not listed as Archived\.  In addition to granules from science
collections, include at least one granule each from the DAP, QA, and PH collections$/) do
  @data = @sotestdata[@critinfo]
  @data.should_not be_nil

  @granules = @ivt.Get_CheckInventoryB_Granules(@ingest, @data)
  puts "Finished getting the granules"
  @granules.should_not be_nil
end

When(/^DP_7F_01:200:G Execute a Data Pool inventory validation$/) do
  @ivt.RunIVT().should be_true
end

Then(/^DP_7F_01:200:G V\-6 Verify that granules with NULL ArchiveTime \(S\-1g\) are logged by the utility$/) do
  @granules.each {|granule|
    @ivt.IsGranuleLogged(granule).should be_true
  }
end

Current Test Script: DP_7F_01_Criteria90.rb
require 'sdps/sotestdata'
require 'sdps/aimdb'
require 'sdps/mode'
require 'sdps/db'
require 'sdps/util'
require 'sdps/bmgt'
require 'sdps/ivt'
require 'sdps/publishing'

Given(/^DP_7F_01:90 Initial Setup$/) do
  #Check mode.
```

```
  @mode = ENV['MODE']
  @mode.should_not be_nil
  @mode_checker = Sdps::Mode.new(@mode)
  @mode_checker.mode_checked_out?().should be_true
  @proxy_log_location = @mode_checker.get_proxy_log_location()
  #@generator_log_location = "/usr/ecs/#{@mode}/CUSTOM/logs/EcBmBMGTGenerator.log"
  #@xmlUtil = Sdps::Bmgt::Xml.new()

  #set up db connection
  @db_subsystem = Sdps::Bmgt::DB_SUBSYSTEM
  @conn = Sdps::Db.connect(@db_subsystem, @mode)
  @conn.should_not be_nil

  #Make sure BMGT is up and running, proxy needs to be up too
  @mode_checker.bmgt_subsystem_checked_out?().should be_true, "bmgt system check failed"

  @aimdb = Sdps::AimDb.new(@conn)
  @bmgtdb = Sdps::Bmgt::BmgtDb.new(@conn)
  @ivt = Sdps::InventoryValidation.new(@conn,@mode)
  @publishing = Sdps::Publishing.new(@mode)

  #Initialize critinfo and SoTestData
  #@critinfo = "DP_7F_01:90"
  #@sotestdata = Sdps::SoTestData.new(@conn)
end

Given(/^DP_7F_01:90 All test granules are installed in the mode$/) do
  @granules = @ivt.Get_CheckInventoryF_Granules()
  puts "Finished getting the granules"
  @granules.should_not be_nil

  #Get rid of the events in the DsMdGrEventHistory table for the granules involved.
  #This doesn't help. the granules that's are logically deleted and DFAed still won't have GRURLDELETE events in
the event history table.
end
```

```
Given(/^DP_7F_01:90 All test collections are configured for collection and granule metadata export$/) do
  #get distinct shortname, versionid for the test granules
  rv = true
  begin
    rs = @conn.exec("select distinct shortname, versionid from amgranule where granuleid in
(#{@granules[0]},#{@granules[1]},#{@granules[2]},#{@granules[3]},#{@granules[4]})")
    if rs != nil
      rs.each do |row|
        shortname = row['shortname']
        versionid = row['versionid']
        collection =@aimdb.find_collection_by_shortname_versionid(shortname,versionid)

        collectionid = collection.collectionid
        if @bmgtdb.collection_export_enabled?(collectionid) == false
          @bmgtdb.enable_collection_granule_export(collectionid).should be_true
        end
      end
    end
  rescue =>e
    puts e.backtrace
    rv = false
  end

  rv.should be_true
end

Given(/^DP_7F_01:90 A TCP proxy is capturing BMGT traffic$/) do
 #This is checked at the initial setup
 puts "If it gets here, the TCP proxy is up and running"

end

When(/^DP_7F_01:90 Perform a Data Pool inventory validation$/) do
  @ivt.RunIVT().should be_true
end
```

```
Then(/^DP_7F_01:90 Verify that the IVT reports public granules that are not eligible to be public$/) do
  #Verify that the granules[0], granules[1], granules[2] and granules[3] are reported
  #and granules[4] is not

  @ivt.IsGranuleReported(@granules[0], "InventoryDiscrp_should_be_hidden_granuleids_RepairByUnpublish").should
be_true
  @ivt.IsGranuleReported(@granules[1], "InventoryDiscrp_should_be_hidden_granuleids_RepairByUnpublish").should
be_true
  @ivt.IsGranuleReported(@granules[2], "InventoryDiscrp_should_be_hidden_granuleids_RepairByUnpublish").should
be_true
  @ivt.IsGranuleReported(@granules[3], "InventoryDiscrp_should_be_hidden_granuleids_RepairByUnpublish").should
be_true
  @ivt.IsGranuleReported(@granules[4], "InventoryDiscrp_should_be_hidden_granuleids_RepairByUnpublish").should
be_false

end

Then(/^DP_7F_01:90 Run the un-publishing utility against the output file reported by IVT$/) do
  file = @ivt.GetOutputDir() + "InventoryDiscrp_should_be_hidden_granuleids_RepairByUnpublish*"
  puts "before unpublishing"
  @eventtime = Time.new.getlocal.strftime("%Y-%m-%d %H:%M:%S")
  puts "unpublish after: #{@eventtime}"
  @publishing.unpublish_granules(file).should be_true
end

Then(/^DP_7F_01:90 Verify the granules are indeed un-published$/) do
  #The granules are no longer flagged as public in the Data Pool inventory and their files are in the correct
hidden Data Pool directory and no longer in the public Data Pool area.
  rv = true
  begin
    rs = @conn.exec("select isorderonly, shortname, granuleid from amgranule where granuleid in
(#{@granules[0]},#{@granules[1]},#{@granules[2]},#{@granules[3]})")
    rs.each do |row|
      #check isorderonly flag first
      isorderonly = row['isorderonly']
      if isorderonly != 'H' and isorderonly != 'Y'
```

```
        puts "isorderonly is: #{isorderonly}"
        rv = false
      end

      #Then make sure the granule files are in the hidden directory
      shortname = row['shortname']
      granuleid = row['granuleid']
      rs1 = @conn.exec("select absfilepath from ProcGetGrFiles('#{shortname}', #{granuleid})")
      rs1.each do |row1|
        file = row1['absfilepath']
        #make sure file contains ".orderdata" in the path
        if !file.include? ".orderdata"
          puts "file is not in hidden datapool: #{file}"
          rv = false
        end
      end
    end
  rescue =>e
    puts e.backtrace
    rv = false
  end
  rv.should be_true
end

Then(/^DP_7F_01:90 Verify that the start and completion of the un-publishing run are logged$/) do

  #Verify that for each granule that's unpublished in this test, grep the granuleid from the newest
EcDlUnpublish.log.*
  #There shouldn't be any errors and there should be a EcDlUnpublishWorker.unpublishOneGranule() start
  #and EcDlUnpublishWorker.unpublishOneGranule() end
  #as well as the un-publishing attempt/success for each of the granules reported by the Data Pool inventory
validation run

  #First get the newest EcDlUnpublish.log.* from /usr/ecs/[MODE]/logs in dpl machine.
  #then grep each granuleid unpublished from the newest EcDlUnpublish.log.* file
```

```
  rv = true
  begin
    @dplhost = Conf.get_property('publish','hostname')
    ls =   `ssh -q #{@dplhost} 'ls -lrt /usr/ecs/#{@mode}/CUSTOM/logs/EcDlUnpublish.log.*'`
    if ls.include? "No match"
     rv = false
    end

    lines = ls.split("\n")
    size = lines.length
    lastline = lines[size -1]
    puts "lastline is: #{lastline}"
    #split lastline by " ", the last element is the full path of the latest file
    elements = lastline.split(" ")
    size = elements.length
    latestfile = elements[size -1]
    puts "latest log file is: #{latestfile}"

    #Going through all the granules in @granules, except for the last one
    #to check if the file includes the necessary information and not include ERROR

    count = 0
    @granules.each do |granule|
      if count == 4
        break
      end
      grep = `ssh -q #{@dplhost} 'grep #{granule} #{latestfile}'`
      if !grep.include? "EcDlUnpublishWorker.unpublishOneGranule() start" or !grep.include?
"EcDlUnpublishWorker.unpublishOneGranule() end" or grep.include? "ERROR"
        rv = false
      end
      count = count + 1
    end
  rescue =>e
    puts "ERROR verifying unpublishing log"
    puts e.backtrace
```

```
    rv = false
  end
  rv.should be_true
end

Then(/^DP_7F_01:90 Verify that the un-publishing run exits with an exit code indicating success$/) do
  puts "If it gets here, the un-publishing run is successful"
end

Then(/^DP_7F_01:90 Verify GRURLDEL events are inserted into the DsMdGrEventHistory table for each granule
unpublished above$/) do
  rv = true
  #Note: The granule with deleteEffectiveDate != null and the granule with DeleteFromArchive = 'Y' won't have
any GRURLDEL events
  #in the DsMdGrEventHistory table because the granules are considered to be deleted already through GRDELETE
events.

  begin
    rs = @conn.exec("select dbid from DsMdGrEventHistory where dbid in
(#{@granules[0]},#{@granules[1]},#{@granules[2]},#{@granules[3]},#{@granules[4]})
                     and    eventtime > '#{@eventtime}'
                     and    eventtype = 'GRURLDEL'")

    if rs != nil and rs.count == 2
      granuleid1 = rs[0]['dbid']
      granuleid2 = rs[1]['dbid']

      if granuleid1 != @granules[2] and granuleid1 != @granules[3]
        rv = false
      else
        if granuleid2 != @granules[2] and granuleid2 != @granules[3]
          rv = false
        end
      end
    else
      rv = false
```

```
    end
  rescue =>e
    puts e.backtrace
    rv = false
  end
  rv.should be_true
end

Then(/^DP_7F_01:90 Verify that the next automatic BMGT export will include the URL deletions of the granules
that were un-published$/) do

end

Current Test Script: DP_7F_01_Criteria200_S-1h.rb
require 'sdps/sotestdata'
require 'sdps/aimdb'
require 'sdps/mode'
require 'sdps/db'
require 'sdps/util'
require 'sdps/ivt'
require 'sdps/publishing'
require 'sdps/ingest'

Given(/^DP_7F_01:200:H Initial Setup$/) do
  #Check mode.
  @mode = ENV['MODE']
  @mode.should_not be_nil
  @mode_checker = Sdps::Mode.new(@mode)
  @mode_checker.mode_checked_out?().should be_true

  #set up db connection
  @db_subsystem = Sdps::Bmgt::DB_SUBSYSTEM
  @conn = Sdps::Db.connect(@db_subsystem, @mode)
  @conn.should_not be_nil

  @aimdb = Sdps::AimDb.new(@conn)
```

```
  @ivt = Sdps::InventoryValidation.new(@conn,@mode)
  @publishing = Sdps::Publishing.new(@mode)
  @ingest =  Sdps::Ingest.build(@conn)

  #Initialize critinfo and SoTestData
  @critinfo = "DP_7F_01:200:H%"
  @sotestdata = Sdps::SoTestData.new(@conn)
end

When(/^DP_7F_01:200:H Ensure that the AIM Inventory Catalog includes at least two of Public and Hidden Science
Granules that were manually deleted from the Data Pool using the EcDlCleanupGranules utility \(both should be
represented as hidden after the utility completes\)$/) do
  @data = @sotestdata[@critinfo]
  @data.should_not be_nil

  @granules = @ivt.Get_CheckInventoryK_Granules(@ingest, @data)
  puts "Finished getting the granules"
  @granules.should_not be_nil

  #run EcDlcleanupGranules utility againast granules in @granules
end

When(/^DP_7F_01:200:H Execute a Data Pool inventory validation$/) do
  @ivt.RunIVT().should be_true
end

Then(/^DP_7F_01:200:H Verify that granules in \(S\-1h\) are not logged by the utility$/) do
  @granules.each {|granule|
    @ivt.IsGranuleLogged(granule).should be_true
  }
end

Current Test Script: DP_7F_01_Criteria200_S-1c.rb
require 'sdps/sotestdata'
require 'sdps/aimdb'
require 'sdps/mode'
```

```
require 'sdps/db'
require 'sdps/util'
require 'sdps/ivt'
require 'sdps/publishing'

Given(/^DP_7F_01:200:C Initial Setup$/) do
  #Check mode.
  @mode = ENV['MODE']
  @mode.should_not be_nil
  @mode_checker = Sdps::Mode.new(@mode)
  @mode_checker.mode_checked_out?().should be_true

  #set up db connection
  @db_subsystem = Sdps::Bmgt::DB_SUBSYSTEM
  @conn = Sdps::Db.connect(@db_subsystem, @mode)
  @conn.should_not be_nil

  @aimdb = Sdps::AimDb.new(@conn)
  @ivt = Sdps::InventoryValidation.new(@conn,@mode)
  @publishing = Sdps::Publishing.new(@mode)
  @ingest =  Sdps::Ingest.build(@conn)

  #Initialize critinfo and SoTestData
  @critinfo = "DP_7F_01:200:C%"
  @sotestdata = Sdps::SoTestData.new(@conn)

end

When(/^DP_7F_01:200:C Ensure that the AIM Inventory Catalog includes at least two granules of collections
defined in the DPL inventory and configured to be published during ingest that contain ECS granules in the AIM
inventory whose Inventory entries indicate they are members of the Hidden Data Pool\.  In addition to granules
from science collections, include at least one granule each from the QA and PH collections\. Ensure there are no
replacement granules for these granules in the public datapool$/) do
  @data = @sotestdata[@critinfo]
  @data.should_not be_nil
```

```
  @granules = @ivt.Get_CheckInventoryG_Granules(@ingest, @data)
  puts "Finished getting the granules"
  @granules.should_not be_nil

end

When(/^DP_7F_01:200:C Turn the DupGran Switch ON$/) do
  pending # express the regexp above with the code you wish you had
end

When(/^DP_7F_01:200:C Execute a Data Pool inventory validation$/) do
  @ivt.RunIVT().should be_true
end

Then(/^DP_7F_01:200:C V\-2 Verify ECS granules that are eligible to be public but that are members of the Hidden
Data Pool \(S\-1c\) are logged by the utility and that the log includes their status in the AIM inventory and
the time they were inserted into ECS$/) do
  @granules.each {|granule|
    @ivt.IsGranuleLogged(granule).should be_true
  }
end

Current Test Script: DP_7F_01_Criteria200_S-1Missing.rb
require 'sdps/sotestdata'
require 'sdps/aimdb'
require 'sdps/mode'
require 'sdps/db'
require 'sdps/util'
require 'sdps/ivt'
require 'sdps/publishing'
require 'sdps/ingest'

Given(/^DP_7F_01:200:Missing Initial Setup$/) do
  #Check mode.
  @mode = ENV['MODE']
  @mode.should_not be_nil
```

```
  @mode_checker = Sdps::Mode.new(@mode)
  @mode_checker.mode_checked_out?().should be_true

  #set up db connection
  @db_subsystem = Sdps::Bmgt::DB_SUBSYSTEM
  @conn = Sdps::Db.connect(@db_subsystem, @mode)
  @conn.should_not be_nil

  @aimdb = Sdps::AimDb.new(@conn)
  @ivt = Sdps::InventoryValidation.new(@conn,@mode)
  @publishing = Sdps::Publishing.new(@mode)
  @ingest =  Sdps::Ingest.build(@conn)

  #Initialize critinfo and SoTestData
  @critinfo = "DP_7F_01:200:Missing%"
  @sotestdata = Sdps::SoTestData.new(@conn)

end

When(/^DP_7F_01:200:Missing Ensure that the AIM Inventory Catalog includes at least two of public granules with
hidden path and hidden granules with public path$/) do
  @data = @sotestdata[@critinfo]
  @data.should_not be_nil

  @granules = @ivt.Get_CheckInventoryK_Granules(@ingest, @data)
  puts "Finished getting the granules"
  @granules.should_not be_nil
end

When(/^DP_7F_01:200:Missing Execute a Data Pool inventory validation$/) do
  @ivt.RunIVT().should be_true
end

Then(/^DP_7F_01:200:Missing Verify the granules are logged by the utility$/) do
  @granules.each {|granule|
    @ivt.IsGranuleLogged(granule).should be_true
```

```
  }
end
```