Specific TELEMAC Toolbox

Global dictionnaries to avoid having to read these more than once.

The keys are the full path to the dictionnaries and therefore allows for *root* and *version* to change

```
def getDICO(cfg,code):

   dicoFile = path.join(cfg['MODULES'][code]['path'],code+'.dico')
   if dicoFile not in DICOS.keys():
      print '    +> register this DICO file: ' + dicoFile
      frgb,dico = scanDICO(dicoFile)
      idico,odico = getIOFilesSubmit(frgb,dico)
      globals()['DICOS'].update({dicoFile:{ 'frgb':frgb, 'dico':dico,
'input':idico, 'output':odico }})

   return dicoFile

# _____                    ____________________________________________
# ____/ General XML Toolbox /___________________________________________/
#
"""
   Will read the xml's XML keys based on the template do.
   +: those with None are must have
   +: those without None are optional and reset if there
"""
def getXMLKeys(xml,do):

   xcpt = []                              # try all keys for full report
   done = do.copy()                       # shallow copy is here sufficient
   for key in done.keys():
      if not key in xml.keys():
         if done[key] == None:
            xcpt.append({'name':'getXMLKeys','msg':'cannot find the key: '
+key})
      else:
         done[key] = xml.attrib[key]
   if xcpt != []: raise Exception(xcpt) # raise full report

   return done

def setSafe(casFile,cas,idico,odico,safe):

   copyFile(casFile,safe)   # TODO: look at relative paths
   wDir = path.dirname(casFile)

   # ~~> process sortie files if any
   sacFile = path.join(safe,casFile)
   sortieFiles = getLatestSortieFiles(sacFile)
```

```
   # ~~> process input / output
   iFS = []; oFS = []
   for k,v in zip(*cas):
      if idico.has_key(k):
         copyFile(path.join(wDir,eval(v[0])),safe)
         ifile = path.join(safe,eval(v[0]))
         iFS.append([k,[ifile],idico[k]])
         #if not path.isfile(ifile):
         #   print '... file does not exist ',ifile
         #   sys.exit()
      if odico.has_key(k):
         ofile = path.join(safe,eval(v[0]))
         oFS.append([k,[ofile],odico[k]])

   return sortieFiles,iFS,oFS

def findTargets(dido,src):
   layers = {}
   oneFound = False
   for cfg in dido.keys():
      cfgFound = False
      if dido[cfg].has_key(src):
         layers.update({ cfg:[dido[cfg][src],'',src] })
         cfgFound = True
      if not cfgFound and dido[cfg].has_key('input'):
         for i,j,k in dido[cfg]['input']:
            k = k.split(';')
            if src in k[1]:               # filename, fileForm, fileType
               # /!\ Temporary fix because TOMAWAC's IOs names are not yet
standard TELEMAC
               if k[5] =='SCAL': k[5] = k[1]
               # \!/
               layers.update({ cfg:[j,k[3],k[5]] })
               cfgFound  = True
      if not cfgFound and dido[cfg].has_key('output'):
         if dido[cfg]['code'] == 'postel3d':
            for file in dido[cfg]['output']:
               if src == path.basename(file) :
                  layers.update({ cfg:[[file],'','SELAFIN'] })
                  cfgFound = True
         else :
            for i,j,k in dido[cfg]['output']:
               k = k.split(';')
               if src in k[1]:               # filename, fileForm, fileType
                  # /!\ Temporary fix because TOMAWAC's IOs names are not
yet standard TELEMAC
                  if k[5] =='SCAL': k[5] = k[1]
                  # \!/
                  layers.update({ cfg:[j,k[3],k[5]] })
                  cfgFound = True
```

```
      if not cfgFound and dido[cfg].has_key('type'):
         if dido[cfg]['type'] == src:
            layers.update({ cfg:[[dido[cfg]['target']],'',src] })
            cfgFound = True
      oneFound = oneFound or cfgFound
   if not oneFound:
      raise Exception([{'name':'findTargets','msg':'did not find the file to
group: '+src}])
   return layers

def findDecos():
   pass

# _____                                                    _____
# ____/ Primary Class: ACTION /____________________________________/
#
"""
   In the classes below, the double quoted keys refer to the keys
      of the XML file. Contrarily, the single quoted keys in
      because they are internal to the python scripts.
   In the XML file, you can have multiple actions and each action
      will be associated with multiple configurations:
      did.keys() => array [xref] for each action
      did[xref].keys() => array [cfgname] for each possible configuration
      did[xref][cfgname].keys() =>
         - 'target', basename of the CAS file
         - 'cas', scanned CAS file
         - 'code', name of the module
         - 'title', title of the action (xref)
         - 'cmaps', refer to the directory 'ColourMaps' for colour plotting
         - 'deprefs', refer to the files that need copying from path to safe
         - 'outrefs', refer to the files that need copying from safe to path
         - 'where' will not default to xref anymore, allowing files to
      be shared and located at the same place between actions
"""
class ACTION:

   # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   #                                                      General Methods
   # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   availacts = ''
   availkeys = { 'path':'','safe':'','cfg':'',
         "target": '', "xref": '', "do": '', "rank":'',
         "title": '', "deprefs":'', "outrefs":'', "where":'' }

   def __init__(self,title='',bypass=True):
      self.active = {}
      if title != '': self.active["title"] = title
      self.bypass = bypass
      self.dids = {}
```

```
   def addAction(self,actions,rank=''):
      self.active.update(deepcopy(self.availkeys))
      try:
         self.active = getXMLKeys(actions,self.active)
      except Exception as e:
         raise Exception([filterMessage({'name':'ACTION::addACTION'},e,self.
bypass)])  # only one item here
      if self.dids.has_key(self.active["xref"]):
         raise Exception([{'name':'ACTION::addACTION','msg':'you are getting
me confused, this xref already exists: '+self.active["xref"]}])
      self.dids.update({ self.active["xref"]:{} })
      if self.active["rank"] == '': self.active["rank"] = rank
      if self.active["rank"] == '': self.active["rank"] = '953'
      self.active["rank"] = int(self.active["rank"])
      if isinstance(self.active["deprefs"], StringTypes):
         deprefs = {}
         if self.active["deprefs"] != '':
            for depref in self.active["deprefs"].split(';'):
               if ':' in depref: ref,dep = depref.split(':')
               else: # ref becomes the name itself if no dependencies to
other actions
                  ref = depref
                  dep = depref
               deprefs.update({ref:dep})
         self.active["deprefs"] = deprefs
      if isinstance(self.active["outrefs"], StringTypes):
         outrefs = {}
         if self.active["outrefs"] != '':
            for outref in self.active["outrefs"].split(';'):
               ref,out = outref.split(':')
               outrefs.update({ref:out})
         self.active["outrefs"] = outrefs
      return self.active["target"]

   def addCFG(self,cfgname,cfg):
      self.active['cfg'] = cfgname
      if self.active["where"] != '':
         self.active['safe'] = path.join( path.join(self.active['path'],self.
active["where"]),cfgname )
      else: self.active['safe'] = path.join( path.join(self.active['path'],
self.active["xref"]),cfgname )
      self.dids[self.active["xref"]].update( { cfgname: {
         'target': self.active["target"],
         'safe': self.active['safe'],
         'path': self.active['path'],
         'title': self.active["title"],
         'cmaps': path.join(cfg['PWD'],'ColourMaps')
         } } )

   def updateCFG(self,d): self.dids[self.active["xref"]][self.active['cfg']].
update( d )
```

```
# _____                                                    _____
# ____/ Primary Class: GROUPS /___________________________________/
#
"""
   In the classes below, the double quoted keys refer to the keys
      of the XML file. Contrarily, the single quoted keys in
      because they are internal to the python scripts.
   In the XML file, you can have multiple actions and each action
      will be associated with multiple configurations:
      did.keys() => array [xref] for each action
      did[xref].keys() => array [cfgname] for each possible configuration
      did[xref][cfgname].keys() =>
         - 'target', basename of the CAS file
         - 'cas', scanned CAS file
         - 'code', name of the module
         - 'title', title of the action (xref)
         - 'cmaps', refer to the directory 'ColourMaps' for colour plotting
         - 'deprefs', refer to the files that need copying from path to safe
         - 'outrefs', refer to the files that need copying from safe to path
         - 'where' will not default to xref anymore, allowing files to
      be shared and located at the same place between groups
"""

class GROUPS:

   # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   #                                                    General Methods
   # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   availacts = ''
   availkeys = { "xref": None, "deco": '' }
   groupkeys = { }
   avaylkeys = { }

   def __init__(self,title='',bypass=True):
      self.active = deepcopy(self.availkeys)
      if title != '': self.active["title"] = title
      self.bypass = bypass
      self.dids = {}
      # those you need to see in the XML file
      self.active["target"] = None
      self.active["code"] = None
      self.active["do"] = None
      self.active["type"] = None
      # additional entities
      self.tasks = {}

   def addGroupType(self,group):
      self.dids.update({group:{}})
      self.active['type'] = group
```

```
   def addGroup(self,group):
      tasks = deepcopy(self.availkeys)
      try:
         self.tasks = getXMLKeys(group,tasks)
      except Exception as e:
         raise Exception([filterMessage({'name':'GROUP::addGroup'},e,self.
bypass)])
      self.active['xref'] = self.tasks["xref"]
      if self.dids[self.active['type']].has_key(self.tasks["xref"]):
         raise Exception([{'name':'GROUP::addGroup','msg':'you are getting
me confused, this xref already exists: '+self.tasks["xref"]}])
      self.dids[self.active['type']].update({self.tasks["xref"]:self.tasks})

   def update(self,d): self.dids[self.active['type']][self.active['xref']].
update( d )

   def addSubTask(self,layer,nametask='layers'):
      # ~~> set default from the upper grouper
      subtasks = {}
      for k in self.groupkeys: subtasks.update({k:self.tasks[k]})
      for k in self.avaylkeys: subtasks.update({k:self.avaylkeys[k]})
      # ~~> reset from layer
      try:
         subtasks = getXMLKeys(layer,subtasks)
      except Exception as e:
         raise Exception([filterMessage({'name':'GROUP::addSubTask'},e,self.
bypass)])
      # ~~> filling-in remaining gaps
      subtasks = self.distributeMeta(subtasks)
      # ~~> adding subtask to the list of tasks
      if self.tasks.has_key(nametask): self.tasks[nametask].append(subtasks)
      else: self.tasks.update({nametask:[subtasks]})
      return len(self.tasks[nametask])-1,nametask

   def targetSubTask(self,target,index=0,nametask='layers'):
      self.tasks[nametask][index].update({ 'fileName': target })
      if not self.dids[self.active['type']][self.active['xref']].has_key(
nametask): self.dids[self.active['type']][self.active['xref']].update({
nametask:[]})
      self.dids[self.active['type']][self.active['xref']][nametask].append(
self.tasks[nametask][index])

   def distributeMeta(self,subtask): return subtask

   def decotasks(self,deco={},index=0,nametask='layers'):
      # ~~> set default
      self.tasks[nametask][index]['deco'] = deco



# _____                                                    _____
# ____/ Secondary Class: META /___________________________________/
```

```
#
class groupMETA(GROUPS):

   availkeys = deepcopy(GROUPS.availkeys)
   availkeys.update({ "roi":'', "deco": 'line', "size":'[15;10]',
         'outFormat': 'png', 'grid': True })
   groupkeys = deepcopy(GROUPS.groupkeys)

   def __init__(self,xmlFile,title='',bypass=True):
      GROUPS.__init__(self,title,bypass)
      # those you reset
      self.active['path'] = path.dirname(xmlFile)
      # those you need to see in the XML file
      self.active["deco"] = {}

   def addDraw(self,meta):
      GROUPS.addGroup(self,meta)

   def addLookTask(self,layer,nametask='look'):
      #self.avaylkeys = deepcopy(GROUPS.avaylkeys)
      self.avaylkeys = {}
      for key in layer.attrib.keys():
         self.avaylkeys.update({key:layer.attrib[key]})
         if key == 'grid':
            if layer.attrib['grid'] == 'no': layer.attrib['grid'] = False
            else: layer.attrib['grid'] = True
      return GROUPS.addSubTask(self,layer,'look')

   def addDataTask(self,layer,nametask='data'):
      self.avaylkeys = deepcopy(GROUPS.avaylkeys)
      self.avaylkeys.update({ "title":'', "contact":'', "author":'' })
      return GROUPS.addSubTask(self,layer,'data')

# ______                                  _____________________________________
# ____/ Secondary Class actionRUN /_____________________________________/
#
# actionRUN is to do with the modules of the TELEMAC system and
#    other execution (pre- and post-processes).
#    . It understands what a PRINCI and what a CAS file and will do
#    the necessary steps to run modules accordingly.
#    . It includes specific methods to do with CAS and PRINCI
#    . It will organise the tranfer of files (inputs and outputs)
#

class actionRUN(ACTION):

   availkeys = deepcopy(ACTION.availkeys)
   availkeys.update({ 'dico':'', "ncsize":'', "code": '' })

   def __init__(self,xmlFile,title='',bypass=True):
      ACTION.__init__(self,title,bypass)
```

```
      # those you reset
      self.active['path'] = path.dirname(xmlFile)
      # those you need to see in the XML file
      self.active["target"] = None
      self.active["code"] = None
      self.active["xref"] = None
      self.active["do"] = None

   def addAction(self,actions,rank=''):
      target = ACTION.addAction(self,actions,rank)
      self.code = self.active["code"]
      return target

   def addCFG(self,cfgname,cfg):
      if not ( self.active["code"] == 'exec' or
               self.active["code"] in cfg['MODULES'].keys() ):
         print '... do not know about: ' + self.active["code"] + ' in
configuration: ' + cfgname
         sys.exit()
      ACTION.addCFG(self,cfgname,cfg)
      ACTION.updateCFG(self,{ "links": {},
         'code': self.active["code"],
         'deprefs': self.active['deprefs'],
         'outrefs': self.active['outrefs'] })

   # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   #                                              CAS related Methods
   # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

   # ~~ todo: difference of CAS file with default keys ~~~~~~~~~~~~~~
   def diffCAS(self): return

   # ~~ Translate the CAS file ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   def translateCAS(self,rebuild):
      if not "translate" in self.availacts.split(';'): return
      xref = self.active["xref"]; cfgname = self.active['cfg']
      active = self.dids[xref][cfgname]
      # ~~> principal CAS file
      casFile = path.join(active['path'],active["target"])
      sacFile = path.join(active['safe'],active["target"])
      oneup = path.dirname(active['safe'])            # copied one level up
      if matchSafe(casFile,active["target"]+'.??',oneup,rebuild):
         print '      ~> translate cas file: ' + active["target"]
         casfr,casgb = translateCAS(sacFile,DICOS[active['dico']]['frgb'])
#/!\ removes comments at end of lines
         moveFile(casfr,oneup)
         moveFile(casgb,oneup)
      # ~~> associated CAS files
      for mod in active["links"]:
         link = active["links"][mod]
         casFile = path.join(active['path'],link['target'])
```

```
         sacFile = path.join(active['safe'],link['target'])
         if matchSafe(casFile,link['target']+'.??',oneup,rebuild):
            print '      ~> translate cas file: ' + link['target']
            casfr,casgb = translateCAS(sacFile,DICOS[link['dico']]['frgb'])
#/!\ removes comments at end of lines
            moveFile(casfr,oneup)
            moveFile(casgb,oneup)

   # ~~ Run the CAS file ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   def runCAS(self,options,cfg,rebuild):
      if not "run" in self.availacts.split(';'): return

      # ~~> prepare options as if run from command line
      specs = Values()
      setattr(specs,'configName',options.configName)
      setattr(specs,'configFile', options.configFile)
      setattr(specs,'sortieFile',True)
      setattr(specs,'tmpdirectory',True)
      setattr(specs,'rootDir', options.rootDir)
      setattr(specs,'version', options.version)
      setattr(specs,'wDir', options.wDir)
      setattr(specs,'compileonly', False)
      if options.hosts != '': setattr(specs,'hosts', options.hosts)
      else: setattr(specs,'hosts', gethostname().split('.')[0])
      setattr(specs,'split', options.split)
      setattr(specs,'run', options.run)
      setattr(specs,'merge', options.merge)
      if options.ncsize != '' and self.active["ncsize"] != '': self.active[
"ncsize"] = options.ncsize
      setattr(specs,'ncsize', self.active["ncsize"])
      setattr(specs,'nctile', '1')    # default but should not be used for
validation
      setattr(specs,'bypass',self.bypass)

      # ~~> check on sorties and run
      casFile = path.join(self.active['path'],self.active["target"])
      sacFile = path.join(self.active['safe'],self.active["target"])
      sortieFiles = getLatestSortieFiles(sacFile)
      outputs = self.dids[self.active["xref"]][self.active['cfg']]['output']
      if matchSafe(casFile,self.active["target"]+'_*??h??min??s*.sortie',
self.active['safe'],rebuild):
         print '     +> running cas file: ' + self.active["target"]
         for k in outputs: matchSafe('',path.basename(k[1][0]),self.active[
'safe'],2)
         try:
            sortieFiles = runCAS(self.active['cfg'],cfg,self.active["code"],
sacFile,specs)
         except Exception as e:
            raise Exception([filterMessage({'name':'ACTION::runCAS'},e,self.
bypass)])  # only one item here
      if sortieFiles != []: self.updateCFG({ 'sortie': sortieFiles })
```

```
   # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   #                                          PRINCI related Methods
   # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

   # ~~ Highligh user PRINCI differences ~~~~~~~~~~~~~~~~~~~~~~~~~~
   def diffPRINCI(self,options,cfg,rebuild):
      if not "princi" in self.availacts.split(';'): return
      xref = self.active["xref"]; cfgname = self.active['cfg']
      active = self.dids[xref][cfgname]
      oneup = path.dirname(active['safe'])            # copied one level up
      # ~~> principal PRINCI file
      value,default = getKeyWord('FICHIER FORTRAN',active['cas'],DICOS[
active['dico']]['dico'],DICOS[active['dico']]['frgb'])
      princiFile = ''
      if value != []:
         princiFile = path.join(active['path'],eval(value[0]))
         if path.exists(princiFile):
            htmlFile = path.join(oneup,path.splitext(path.basename(
princiFile))[0]+'.html')
            if matchSafe(htmlFile,path.basename(htmlFile),oneup,rebuild):
               # ~~> Scans the principal user PRINCI file
               print '      ~> scanning your PRINCI file: ',path.basename(
princiFile)
               pFiles = getPrincipalWrapNames(princiFile)
               if pFiles == []:
                  raise Exception([{'name':'ACTION::diffPRINCI','msg':'I
could not recognised entities in your PRINCI: '+princiFile}])
               else:
                  print '        +> found:'
                  for pFile in pFiles: print '           - ',pFile
               # ~~> Scans the entire system
               oFiles = {}
               for mod in cfg['MODULES']:
                  for dirpath,dirnames,filenames in walk(cfg['MODULES'][mod]
['path']) : break
                  for file in filenames:
                     n,e = path.splitext(file)
                     # Only looking for fortran files
                     if e.lower() not in ['.f','.f90']: continue
                     for pFile in pFiles:
                        if pFile.lower() == n:
                           oFiles.update( filterPrincipalWrapNames( [pFile],
[path.join(dirpath,file)] ) )
               if oFiles == {}:
                  raise Exception([{'name':'ACTION::diffPRINCI','msg':'I
could not relate your PRINCI with the system: '+princiFile}])
               else:
                  print '        +> found:'
                  for oFile in oFiles: print '           - ',oFile
               # ~~> Save temporarily for subsequent difference
```

```
                oriFile = path.splitext(princiFile)[0]+'.original'+path.
splitext(princiFile)[1]
                putFileContent(oriFile,[])
                for p in pFiles:
                   if p in oFiles.keys(): addFileContent(oriFile,
getFileContent(oFiles[p]))
                # ~~> Process difference and write output into an HTML file
                diff = diffTextFiles(oriFile,princiFile,options)
                remove(oriFile)
                of = open(htmlFile,'wb')
                of.writelines( diff )
                of.close()
                print '       ~> comparison successful ! created: ' + path.
basename(htmlFile)
             else:
                raise Exception([{'name':'ACTION::diffPRINCI','msg':'I could not
find your PRINCI file: '+princiFile}])
          # ~~> associated PRINCI file
          # TODO: case of coupling with multiple PRINCI files

      # ~~ Compile the PRINCI file ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
      def compilePRINCI(self,cfg,rebuild):
         if not "compile" in self.availacts.split(';'): return
         xref = self.active["xref"]; cfgname = self.active['cfg']
         active = self.dids[xref][cfgname]
         confirmed = False
         # ~~> principal PRINCI file
         value,default = getKeyWord('FICHIER FORTRAN',active['cas'],DICOS[
active['dico']]['dico'],DICOS[active['dico']]['frgb'])
         princiFile = ''; princiSafe = ''
         if value != []:       # you do not need to compile the default
executable
            princiFile = path.join(active['path'],eval(value[0]))
            if path.exists(princiFile):
               exeFile = path.join(active['safe'],path.splitext(eval(value[0]))
[0] + cfg['SYSTEM']['sfx_exe'])
               if not path.exists(exeFile) or cfg['REBUILD'] == 0:
                  print '     +> compiling princi file: ' + path.basename(
princiFile)
                  copyFile(princiFile,active['safe'])
                  print '*********copying '+princiFile+' '+active['safe']
                  princiSafe = path.join(active['safe'],path.basename(
princiFile))
                  confirmed = True
            else:
               raise Exception([{'name':'ACTION::compilePRINCI','msg':'I could
not find your PRINCI file: '+princiFile}])
         # ~~> associated PRINCI file
         for mod in active["links"]:
            link = active["links"][mod]
            value,default = getKeyWord('FICHIER FORTRAN',link['cas'],DICOS[link
```

```
['dico']]['dico'],DICOS[link['dico']]['frgb'])
         princiFilePlage = ''
         if value != []:       # you do not need to compile the default
executable
            princiFilePlage = path.join(active['path'],eval(value[0]))
            if path.exists(princiFilePlage):
               if princiSafe != '':
                  print '*********adding content of '+path.basename(
princiFilePlage)+' to '+princiSafe
                  putFileContent(princiSafe,getFileContent(princiSafe)+['']
+getFileContent(princiFilePlage))
               else:
                  print '     +> compiling princi file: ' + path.basename(
princiFilePlage)
                  exeFile = path.join(active['safe'],path.splitext(eval(
value[0]))[0] + cfg['SYSTEM']['sfx_exe'])
                  princiSafe = path.join(active['safe'],path.basename(
princiFilePlage))
                  print '*********copying '+path.basename(princiFilePlage)+
' ' + active['safe']
                  copyFile(princiFilePlage,active['safe'])
               confirmed = True
            else:
               raise Exception([{'name':'ACTION::compilePRINCI','msg':'I
could not find your PRINCI file: '+princiFilePlage}])
      if confirmed:
         try:
            compilePRINCI(princiSafe,active["code"],self.active['cfg'],cfg,
self.bypass)
         except Exception as e:
            raise Exception([filterMessage({'name':'ACTION::compilePRINCI'},
e,self.bypass)])  # only one item here
         #moveFile(exeFile,active['safe'])
         print '       ~> compilation successful ! created: ' + path.
basename(exeFile)
      #else: you may wish to retrieve the executable for later analysis

   # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   #                                                Direct Executions
   # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

   def runCommand(self,rebuild):

      print '    +> executing your command:\n      ', self.active["do"]
      mes = MESSAGES(size=10)

      # ~~> copy of inputs
      copyFile(path.join(self.active['path'],self.active["target"]),self.
active['safe'])
      for iFile in self.active["deprefs"]: copyFile(path.join(self.active[
'path'],iFile),self.active['safe'])
```

```
      # ~~> execute command locally
      os.chdir(self.active['safe'])
      try:
         tail,code = mes.runCmd(self.active["do"],True) #self.bypass)
      except Exception as e:
         raise Exception([filterMessage({'name':'runCommand','msg':
'something went wrong when executing you command.'},e,True)])
      if code != 0: raise Exception([{'name':'runCommand','msg':'Could run
your command ('+self.active["do"]+').\n      '+tail}])

      # ~~> copy of outputs /!\ you are replacing one config by another
      for oFile in self.active["outrefs"]:
         if path.exists(path.join(self.active['safe'],self.active["outrefs"]
[oFile])):
            try:
               copyFile(path.join(self.active['safe'],self.active["outrefs"]
[oFile]),self.active['path'])
            except Exception as e:
               raise Exception([filterMessage({'name':'runCommand','msg':'I
can see your file '+oFile+': '+self.active["outrefs"][oFile]+'but cannot
copy it'},e,True)])
         else: raise Exception([{'name':'runCommand','msg':'I cannot see
your output file '+oFile+': '+self.active["outrefs"][oFile]}])

# _____                             ____________________________________
# ____/ Secondary Class actionGET /___________________________________/
#
# actionGET is to do with loading data in memory for future use.
#

class actionGET(ACTION):

   availkeys = deepcopy(ACTION.availkeys)
   availkeys.update({ 'type':'' })

   def __init__(self,xmlFile,title='',bypass=True):
      ACTION.__init__(self,title,bypass)
      # those you reset
      self.active['path'] = path.dirname(xmlFile)
      # those you need to see in the XML file
      self.active["target"] = None
      self.active["xref"] = None
      self.active["type"] = None

   def addCFG(self,cfgname,cfg):
      ACTION.addCFG(self,cfgname,cfg)
      ACTION.updateCFG(self,{ "type": self.active["type"],
         "target":path.join(self.active['path'],self.active["target"]) })

# _____                             ____________________________________
```

```
# _____/ Secondary Class groupPLOT /_____________________________________/
#
# groupPLOT is to do with plotting and producing PNG (mainly)
#

class groupPLOT(GROUPS):

   availkeys = deepcopy(GROUPS.availkeys)
   availkeys.update({ 'path':'','safe':'','cfg':'', "size":'[15;10]',
         "time": '[-1]', "extract": '', "vars": '', 'outFormat': 'png',
         "target": '', "do": '', "rank":'',
         "title": '', "deprefs":'', "outrefs":'', "where":'',
         "type":'', "config": 'distinct' })
   groupkeys = deepcopy(GROUPS.groupkeys)
   groupkeys.update({ "vars":'', "time":'', "extract":'', "config":'' })
   avaylkeys = deepcopy(GROUPS.avaylkeys)
   avaylkeys.update({ "title":'', "target":'', "deco":'' })

   def __init__(self,xmlFile,title='',bypass=True):
      GROUPS.__init__(self,title,bypass)
      # those you reset
      self.active['path'] = path.dirname(xmlFile)

   def addDraw(self,draw,rank=''):
      GROUPS.addGroup(self,draw)
      if self.dids[self.active['type']][self.tasks["xref"]]['rank'] == '':
self.dids[self.active['type']][self.tasks["xref"]]['rank'] = rank
      if self.dids[self.active['type']][self.tasks["xref"]]['rank'] == '':
self.dids[self.active['type']][self.tasks["xref"]]['rank'] = '953'
      self.dids[self.active['type']][self.tasks["xref"]]['rank'] = int(self.
dids[self.active['type']][self.tasks["xref"]]['rank'])
      #self.active['deco'] = self.tasks["deco"]

   def distributeMeta(self,subtask):
      # ~~> distribute decoration
      vars = subtask["vars"].split(';')
      for i in range(len(vars)):
         if ':' not in vars[i]: vars[i] = vars[i] + ':' + self.tasks["deco"]
      subtask["vars"] = ';'.join(vars)
      return subtask

   def decolooks(self,deco={},index=0,nametask='layers'):
      # ~~> set default
      GROUPS.decotasks(self,deco,index,nametask)
      # ~~> set the look
      if deco.has_key('look'):
         for key in deco['look'][0].keys():
            self.tasks[nametask][index]['deco'].update({key:deco['look'][0][
key]})

# _____                                 ____________________________________
```

```
# _____/ Secondary Class groupGET /_____________________________________/
#
class groupGET(GROUPS):

   availkeys = deepcopy(GROUPS.availkeys)
   availkeys.update({ 'path':'','safe':'','cfg':'',
         "time": '[-1]', "extract": '', "vars": '',
         'outFormat': 'csv', "target": '', "do": '', "rank":'',
         "title": '', "deprefs":'', "outrefs":'', "where":'',
         "type":'', "config": 'distinct' })
   groupkeys = deepcopy(GROUPS.groupkeys)
   groupkeys.update({ "vars":'', "time":'', "extract":'', "config":'' })
   avaylkeys = deepcopy(GROUPS.avaylkeys)
   avaylkeys.update({ "title":'', "target":'', "deco":'' })

   def __init__(self,xmlFile,title='',bypass=True):
      GROUPS.__init__(self,title,bypass)
      # those you reset
      self.active['path'] = path.dirname(xmlFile)

   def distributeMeta(self,subtask):
      # ~~> distribute decoration
      vars = subtask["vars"].split(';')
      for i in range(len(vars)):
         if ':' not in vars[i]: vars[i] = vars[i] + ':xyz'   # :xyz is not
used
      subtask["vars"] = ';'.join(vars)
      return subtask

# _____                          ____________________________________
# ____/ Secondary Class CRITERIA /___________________________________/
#

class CRITERIA(GROUPS):

   availkeys = deepcopy(GROUPS.availkeys)
   availkeys.update({ 'path':'','safe':'','cfg':'',
         "target": '', "do": '', "rank":'',
         "title": '', "deprefs":'', "outrefs":'', "where":'',
         "type":'', "config": 'distinct' })
   groupkeys = deepcopy(GROUPS.groupkeys)
   groupkeys.update({ "vars":'', "time":'', "extract":'', "config":'' })
   avaylkeys = deepcopy(GROUPS.avaylkeys)
   #avaylkeys.update({ "title":'', "target":'' })

   def __init__(self,xmlFile,title='',bypass=True):
      GROUPS.__init__(self,title,bypass)
      # those you reset
      self.active['path'] = path.dirname(xmlFile)
      # ~~~~~~~~~~~~~
      self.variabling = {}; self.conditionning = {}
```

```
   def addCriteria(self,criteria):
      try:
         i = getXMLKeys(criteria,self.active)
      except Exception as e:
         raise Exception([filterMessage({'name':'ACTION::addCriteria'},e,
self.bypass)])  # only one item here
      else:
         self.active = i
      if self.dids.has_key(self.active["xref"]):
         raise Exception([{'name':'CRITERIA::addCriteria','msg':'you are
getting me confused, this xref already exists: '+self.active["xref"]}])
      self.dids.update({ self.active["xref"]:{} })
      print '\n    +> Validation Criteria :', self.active["xref"]
      return

   def addCFG(self,cfgname,cfg):
      self.active['cfg'] = cfgname
      self.dids[self.active["xref"]].update( { cfgname: {'variables':{},
'condition':{} } } )

   def addvariable(self,variable):
      variabling = {"vars": None, "target":None, "path":''}
      cfgname = self.active['cfg']
      try:
         self.variabling = getXMLKeys(variable,variabling)
      except Exception as e:
         raise Exception([filterMessage({'name':'CRITERIA::addvariable'},e,
self.bypass)])  # only one item here

      self.dids[self.active["xref"]][self.active['cfg']]['variables'].update
({self.variabling["vars"]:{}})
      active = self.dids[self.active["xref"]][self.active['cfg']][
'variables']

      folder,file = self.variabling["target"].split(':')
      pathfolder = path.join(os.getcwd(),folder)
      pathcfg = path.join(pathfolder,cfgname)
      pathfile = path.join(pathcfg,file)

      active[self.variabling["vars"]]['path'] = [pathfile]
      self.variabling['path'] = [pathfile]
      return

   def addcondition(self,condition,NBR):
      conditionning = {"do": None, "success":None, "failure":None, "warning":
'',
                       "result":[]}
      cfgname = self.active['cfg']
      try:
         self.conditionning = getXMLKeys(condition,conditionning)
```

```
      except Exception as e:
         raise Exception([filterMessage({'name':'CRITERIA::addcondition'},e,
self.bypass)])  # only one item here

      self.dids[self.active["xref"]][self.active['cfg']]['condition'][NBR] =
self.conditionning
      return self.conditionning

   def updateCFG(self,d): self.dids[self.active["xref"]][self.active['cfg']].
update( d )

   # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   #   Available Operations for criteria
   # ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

   def compare(self,condition):
      variables = self.dids[self.active["xref"]][self.active['cfg']][
'variables']

      ######### Success condition ###################################
      NameVar1,operator,NameVar2 = condition["success"].split(':')

      print '         +> comparing :', NameVar1, '&', NameVar2

      if NameVar1.isalpha() == False : var1 = float(NameVar1)
      else :
         PathVar1 = variables[NameVar1]['path'][0]
         var1 = [] #getVariableCSV(PathVar1,NameVar1)
      if NameVar2.isalpha() == False : var2 = float(NameVar2)
      else :
         PathVar2 = variables[NameVar2]['path'][0]
         var2 = [] #getVariableCSV(PathVar2,NameVar2)

      if operator == 'LE' :
         if var1 <= var2 : condition["result"].append('success')
      elif operator == 'LT' :
         if var1 < var2 : condition["result"].append('success')
      elif operator == 'GE' :
         if var1 >= var2 : condition["result"].append('success')
      elif operator == 'GT' :
         if var1 > var2 : condition["result"].append('success')


      ######### Failure condition ###################################
      NameVar1,operator,NameVar2 = condition["failure"].split(':')

      if NameVar1.isalpha() == False : var1 = float(NameVar1)
      else :
         PathVar1 = variables[NameVar1]['path'][0]
         var1 = [] #getVariableCSV(PathVar1,NameVar1)
      if NameVar2.isalpha() == False : var2 = float(NameVar2)
```

```
      else :
         PathVar2 = variables[NameVar2]['path'][0]
         var2 = [] #getVariableCSV(PathVar2,NameVar2)

      if operator == 'LE' :
         if var1 <= var2 : condition["result"].append('failed')
      elif operator == 'LT' :
         if var1 < var2 : condition["result"].append('failed')
      elif operator == 'GE' :
         if var1 >= var2 : condition["result"].append('failed')
      elif operator == 'GT' :
         if var1 > var2 : condition["result"].append('failed')

      ######### Warning condition ##################################
      if condition["result"] == [] : condition["result"].append('warning')

      return

   def CalcL2error(self,task):
      xref = self.active["xref"]; cfgname = self.active['cfg']
      active = self.dids[xref][cfgname]
      file = self.tasking["target"]
      print '       +> Calculating L2 error :', self.tasking["title"],'\n'
      if self.tasking["reference"] != '':
         file = path.join(self.dids[self.tasking["reference"]][cfgname][
'tasks']['safe'],file)
      var1 = self.tasking["ModelData"].lower()
      var2 = self.tasking["ExactData"].lower()
      VarModel,VarExperiment = get2VariablesCSV(file,var1,var2)
      L2error = (la.norm(VarModel-VarExperiment))/(la.norm(VarExperiment))
      columns = [[self.tasking["title"],'None',L2error]]
      file2 = path.join(self.active['safe'], xref +'.csv')
      if path.exists(file2):
         columns = [[self.tasking["title"],'None',L2error]]
         #addColumnCSV(file2,columns[0])
      else : pass #putDataCSV(file2,columns)
      return
```

XML Parser Toolbox

```
def runXML(xmlFile,xmlConfig,bypass):

   xcpt = []                              # try all keys for full report

   # ~~ Parse xmlFile ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   import xml.etree.ElementTree as XML
   print '... reading XML test specification file: ' + path.basename(xmlFile)
   f = open(xmlFile,'r')
```

```
   xmlTree = XML.parse(f)  # may need to try first and report error
   xmlRoot = xmlTree.getroot()
   f.close()
   # ~~ Default Ranking ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   rank = ''
   if "rank" in xmlRoot.keys(): rank = xmlRoot.attrib["rank"]

   # ~~ Meta data process ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   #
   #    This needs to be developed further
   #
   title = ""
   dc = groupMETA(xmlFile,title,bypass)
   dc.addGroupType("meta")
   for metaing in xmlRoot.findall("meta"):

      # ~~ Step 1. Common check for keys ~~~~~~~~~~~~~~~~~~~~~~~~
      try:
         dc.addGroup(metaing)
      except Exception as e:
         xcpt.append(filterMessage({'name':'runXML','msg':'add meta object
to the list'},e,bypass))
         continue   # bypass the rest of the for loop

      # ~~ Step 2. Cumul looks ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
      if len(metaing.findall("look")) > 1:
         xcpt.append({'name':'runXML','msg':'you can only have one look in
meta referenced: '+dc.tasks["xref"]})
      if len(metaing.findall("look")) > 0:
         look = metaing.findall("look")[0]
         try:
            dc.addLookTask(look)
         except Exception as e:
            xcpt.append(filterMessage({'name':'runXML','msg':'add look to
the list'},e,bypass))
            continue   # bypass the rest of the for loop
      # ~~ Step 2. Cumul metas ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
      if len(metaing.findall("data")) > 1:
         xcpt.append({'name':'runXML','msg':'you can only have one data in
meta referenced: '+dc.tasks["xref"]})
      if len(metaing.findall("data")) > 0:
         data = metaing.findall("data")[0]
         try:
            dc.addDataTask(data)
         except Exception as e:
            xcpt.append(filterMessage({'name':'runXML','msg':'add meta to
the list'},e,bypass))
            continue   # bypass the rest of the for loop

      dc.update(dc.tasks)
```

```
   if xcpt != []: raise Exception({'name':'runXML','msg':'looking at meta in
xmlFile: '+xmlFile,'tree':xcpt})
   display = False

   # ~~ Main action process ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   #
   #    Whether an action is carried out or not, it is known through:
   #       xmlConfig[cfgname]['options'].todos['act']['todo']
   #    PRINCAS(ACTION) will still be loaded to register various files
   #       for possible subsequent extraction, plotting or analysis
   #    TODO: limit the number of path / safe duplication
   #
   do = actionRUN(xmlFile,title,bypass)
   for action in xmlRoot.findall("action"):

      # ~~ Step 1. Common check for keys and driving file ~~~~~~~~~~~
      try:
         targetFile = do.addAction(action,rank)
      except Exception as e:
         xcpt.append(filterMessage({'name':'runXML','msg':'add todo to the
list'},e,bypass))
         continue    # bypass rest of the loop
      else:
         if not path.isfile(path.join(do.active['path'],targetFile)):
            xcpt.append({'name':'runXML','msg':'could not find your target
file'+targetFile})
            continue    # bypass rest of the loop

      # ~~ Step 2. Loop over configurations ~~~~~~~~~~~~~~~~~~~~~~~~~
      for cfgname in xmlConfig.keys():
         cfg = xmlConfig[cfgname]['cfg']
         do.addCFG(cfgname,cfg) #if not : continue

         # ~~> Temper with rank but still gather intelligence
         dodo = True
         rankdo = do.active['rank']
         print '>>',rankdo
         rankdont = xmlConfig[cfgname]['options'].todos['act']['rank']
         if rankdont != rankdo*int( rankdont / rankdo ): dodo = False
         do.updateCFG({'dodo':dodo})

         # ~~> Create the safe
         createDirectories(do.active['safe'])

         # ~~ Step 3a. Deals with TELEMAC launchers ~~~~~~~~~~~~~~~~~
         if do.active["code"] in cfg['MODULES'].keys():

            do.availacts = "translate;run;compile;princi"

            # ~~> Manage targetFile and other inputs
            casFile = path.join(do.active['path'],targetFile)
```

```
            # ~~> Parse DICO File and its IO Files default (only once)
            dicoFile = getDICO(cfg,do.active["code"])
            do.updateCFG({'dico':dicoFile})
            dico = DICOS[dicoFile]['dico']
            frgb = DICOS[dicoFile]['frgb']
            cas = readCAS(scanCAS(casFile),dico,frgb)
            if do.active["ncsize"] != '': cas = setKeyValue('PROCESSEURS
PARALLELES',cas,frgb,int(do.active["ncsize"]))
            do.updateCFG({'cas':cas})
            if not checkConsistency(cas,dico,frgb,cfg): continue
            idico = DICOS[dicoFile]['input']
            odico = DICOS[dicoFile]['output']

            # ~~> Define config-split storage
            sortieFiles,iFS,oFS = setSafe(casFile,cas,idico,odico,do.active[
'safe'])   # TODO: look at relative paths
            if sortieFiles != []: do.updateCFG({ 'sortie': sortieFiles })
            do.updateCFG({ 'input':iFS })
            do.updateCFG({ 'output':oFS })

            # ~~> Case of coupling
            cplages,defaut = getKeyWord('COUPLING WITH',cas,dico,frgb)
            links = {}
            for cplage in cplages:
               for mod in cfg['MODULES'].keys():
                  if mod in cplage.lower():
                     # ~~> Extract the CAS File name
                     casFilePlage,defaut = getKeyWord(mod.upper()+' STEERING
FILE',cas,dico,frgb)
                     if casFilePlage == []: casFilePlage = defaut[0]
                     else: casFilePlage = eval(casFilePlage[0])
                     casFilePlage = path.join(path.dirname(casFile),
casFilePlage)
                     if not path.isfile(casFilePlage): raise Exception([{
'name':'runCAS','msg':'missing coupling CAS file for '+mod+': '+casFilePlage
}])
                     # ~~> Read the DICO File
                     dicoFilePlage = getDICO(cfg,mod)
                     dicoPlage = DICOS[dicoFilePlage]['dico']
                     frgbPlage = DICOS[dicoFilePlage]['frgb']
                     # ~~> Read the coupled CAS File
                     casPlage = readCAS(scanCAS(casFilePlage),dicoPlage,
frgbPlage)
                     # ~~> Fill-in the safe
                     idicoPlage = DICOS[dicoFilePlage]['input']
                     odicoPlage = DICOS[dicoFilePlage]['output']
                     sortiePlage,iFSPlage,oFSPlage = setSafe(casFilePlage,
casPlage,idicoPlage,odicoPlage,do.active['safe'])   # TODO: look at relative
paths
                     links.update({mod:{}})
                     links[mod].update({ 'code':mod, 'target':path.basename(
```

```
casFilePlage),
                        'cas':casPlage, 'frgb':frgbPlage, 'dico'
:dicoFilePlage,
                        'iFS':iFSPlage, 'oFS':oFSPlage, 'sortie':sortiePlage
})
                    if sortiePlage != []: links[mod].update({ 'sortie'
:sortiePlage })
            if links != {}: do.updateCFG({ "links":links })

            # ~~> Complete all actions
            # options.todos['act']['todo'] takes: translate;run;compile and
none
            doable = xmlConfig[cfgname]['options'].todos['act']['todo']
            if doable == '': doable = do.active["do"]
            if doable == '' or doable == 'all': doable = do.availacts
            display = display or xmlConfig[cfgname]['options'].display

            # ~~> Action type A. Translate the CAS file
            if "translate" in doable.split(';') and dodo:
               try:
                  # - exchange keywords between dictionaries
                  do.translateCAS(cfg['REBUILD'])
               except Exception as e:
                  xcpt.append(filterMessage({'name':'runXML','msg':'   +>
translate'},e,bypass))

            # ~~> Action type B. Analysis of the CAS file
            # TODO:
            # - comparison with DEFAULT values of the DICTIONARY
            #if "cas" in doable.split(';') and dodo:
            # - comparison of dictionnaries betwen configurations
            #if "dico" in doable.split(';') and dodo:

            # ~~> Action type C. Analysis of the PRINCI file
            if "princi" in doable.split(';') and dodo:
               #try:
                  # - comparison with standard source files
                  specs = Values()
                  setattr(specs,'unified',False)
                  setattr(specs,'ndiff',False)
                  setattr(specs,'html',True)
                  setattr(specs,'ablines',3)
                  setattr(specs,'context',False)
                  do.diffPRINCI(specs,cfg,cfg['REBUILD'])
               #except Exception as e:
               #   xcpt.append(filterMessage({'name':'runXML','msg':'   +>
diff(princi)'},e,bypass))
            # TODO: - comparison of subroutines between action items

            # ~~> Action type E. Running CAS files
            if "run" in doable.split(';') and dodo:
```

```
                try:
                   do.runCAS(xmlConfig[cfgname]['options'],cfg,cfg['REBUILD'])
                except Exception as e:
                   xcpt.append(filterMessage({'name':'runXML','msg':'   +>
run'},e,bypass))

         # ~~ Step 3b. Deals with execute launchers ~~~~~~~~~~~~~~~~
         elif do.active["code"] == 'exec':

            do.availacts = "exec"

            # ~~> Complete all actions
            # options.todos['act']['todo'] takes: exec and none
            doable = xmlConfig[cfgname]['options'].todos['act']['todo']
            if doable == '': doable = do.active["code"]
            if doable == '' or doable == 'all': doable = do.availacts

            # ~~> Action type E. Running exec
            if "exec" in doable.split(';') and dodo:
               try:
                  # - simply run the exec as stated
                  do.runCommand(cfg['REBUILD'])
               except Exception as e:
                  xcpt.append(filterMessage({'name':'runXML::runCommand',
'msg':'   +> '+do.active["do"]},e,bypass))

   if xcpt != []: raise Exception({'name':'runXML','msg':'looking at actions
in xmlFile: '+xmlFile,'tree':xcpt})

   # ~~ Load targets ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   #
   gt = actionGET(xmlFile,title,bypass)
   for load in xmlRoot.findall("load"):

      # ~~ Step 1. Common check for keys and driving file ~~~~~~~~~~
      try:
         targetFile = gt.addAction(load,rank)
      except Exception as e:
         xcpt.append(filterMessage({'name':'runXML','msg':'add load to the
list'},e,bypass))
         continue    # bypass rest of the loop
      else:
         if not path.isfile(path.join(gt.active['path'],targetFile)):
            xcpt.append({'name':'runXML','msg':'could not find your target
file'+targetFile})
            continue    # bypass rest of the loop

      # ~~ Step 2. Loop over configurations ~~~~~~~~~~~~~~~~~~~~~~~~
      for cfgname in xmlConfig.keys():
         cfg = xmlConfig[cfgname]['cfg']
         gt.addCFG(cfgname,cfg)
```

```
   if xcpt != []: raise Exception({'name':'runXML','msg':'looking at loads
in xmlFile: '+xmlFile,'tree':xcpt})

   # ~~ Extraction targets ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   # did has all the IO references and the latest sortie files
   ex = groupGET(xmlFile,title,bypass)
   for typeSave in ["save1d","save2d","save3d"]:
      ex.addGroupType(typeSave)
      for extracting in xmlRoot.findall(typeSave):
         # ~~ Step 1. Common check for keys ~~~~~~~~~~~~~~~~~~~~~~~~
         try:
            ex.addGroup(extracting)
         except Exception as e:
            xcpt.append(filterMessage({'name':'runXML','msg':'add extract
object to the list'},e,bypass))
            continue   # bypass the rest of the for loop

         # ~~> Temper with rank but still gather intelligence
         doex = True
         rankdo = ex.dids[typeSave][ex.active['xref']]['rank']
         rankdont = xmlConfig[cfgname]['options'].todos['get']['rank']
         if rankdont != rankdo*int( rankdont / rankdo ): doex = False
         if not doex: continue

         # ~~ Step 2. Cumul layers ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
         for layer in extracting.findall("layer"):
            try:
               index,namex = ex.addSubTask(layer)
               target = ex.tasks[namex][index]["target"]
            except Exception as e:
               xcpt.append(filterMessage({'name':'runXML','msg':'add layer
to the list'},e,bypass))
               continue   # bypass the rest of the for loop

            # ~~> round up targets and their configurations looking in exes
and does
            xref,src = target.split(':')
            if gt.dids.has_key(xref):
               try:
                  findlayer = findTargets(gt.dids[xref],src)
               except Exception as e:
                  xcpt.append(filterMessage({'name':'runXML','msg':'could
not find reference to extract within loads: '+xref},e))
                  ex.targetSubTask({},index,namex)
                  continue    # bypass the rest of the for loop
               else :
                  ex.targetSubTask(findlayer,index,namex)
            elif do.dids.has_key(xref):
               try:
                  findlayer = findTargets(do.dids[xref],src)
```

```
                except Exception as e:
                   xcpt.append(filterMessage({'name':'runXML','msg':'could
not find reference to extract within actions: '+xref},e))
                   ex.targetSubTask({},index,namex)
                   continue    # bypass the rest of the for loop
                else:
                   ex.targetSubTask(findlayer,index,namex)
             else : xcpt.append({'name':'runXML','msg':'could not find
reference to extract the action: '+xref})

          ex.update(ex.tasks)

   if xcpt != []: raise Exception({'name':'runXML','msg':'looking at
extractions in xmlFile: '+xmlFile,'tree':xcpt})

   # ~~ Matrix distribution by extraction types ~~~~~~~~~~~~~~~~~~~~
   for typeSave in ex.dids.keys():
      for xref in ex.dids[typeSave]:

          task = ex.dids[typeSave][xref]
          if not task.has_key("layers"): continue
          print '    +> reference: ',xref,' of type ',typeSave

          xlayers = ''   # now done with strings as arrays proved to be too
challenging
          for layer in task["layers"]:
             if layer['config'] == 'together':
                xys = []
                for x in xlayers.split('|'): xys.append( (x+';'+':'.join(
layer['fileName'].keys() )).strip(';') )
                xlayers = '|'.join(xys)
             elif layer['config'] == 'distinct':
                ylayers = layer['fileName'].keys()
                xys = []
                for i in range(len(ylayers)):
                  for x in xlayers.split('|'): xys.append( (x+';'+ylayers[i]).
strip(';') )
                xlayers = '|'.join(xys)
             elif layer['config'] == 'oneofall':
                xys = []; cfg = layer['fileName'].keys()[0]     #/!\ you are
sure to have at least one (?)
                for x in xlayers.split('|'): xys.append( (x+';'+cfg).strip(
';') )
                xlayers = '|'.join(xys)
             else:
                if layer['config'] in layer['fileName'].keys():
                   xys = []
                   for x in xlayers.split('|'): xys.append( (x+';'+layer[
'config']).strip(';') )
                xlayers = '|'.join(xys)
```

```
         nbFile = 0; alayers = xlayers.split('|')
         for cfglist in alayers:
            # ~~> Figure name
            if len(alayers) == 1:
               extractName = '.'.join([xref.replace(' ','_'),task[
'outFormat']])
            else:
               nbFile += 1
               extractName = '.'.join([xref.replace(' ','_'),str(nbFile),
task['outFormat']])
            print '       ~> saved as: ',extractName
            extractName = path.join(path.dirname(xmlFile),extractName)
            # ~~> Create Figure
            if typeSave == "save1d": figure = Figure1D(typeSave,task,
extractName)
            if typeSave == "save2d": figure = Figure2D(typeSave,task,
extractName)

            for layer,cfgs in zip(task["layers"],cfglist.split(';')):
               for cfg in cfgs.split(':'):
                  for file in layer['fileName'][cfg][0]:
                     figure.draw( layer['fileName'][cfg][2], { 'file': file,
                        'deco': {},
                        'vars': layer["vars"], 'extract':parseArrayPaires(
layer["extract"]),
                        'type': task['type'], 'time':parseArrayPaires(layer[
"time"])[0] } )

            figure.dump()

   if xcpt != []: raise Exception({'name':'runXML','msg':'looking at savings
in xmlFile: '+xmlFile,'tree':xcpt})

   """      if "L2error" in doaddtask["do"]:
               os.chdir(racine)
               try:
                  ex.CalcL2error(doaddtask)
               except Exception as e:
                  xcpt.append(filterMessage({'name':'runXML','msg':'   +>
CalcL2error'},e,bypass))"""

   # ~~ Gathering targets ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   plot = groupPLOT(xmlFile,title,bypass)
   for typePlot in ["plot1d","plot2d","plot3d","plotpv"]:
      plot.addGroupType(typePlot)
      for ploting in xmlRoot.findall(typePlot):
         # ~~ Step 1. Common check for keys ~~~~~~~~~~~~~~~~~~~~~~~~
         try:
            plot.addDraw(ploting,rank)
         except Exception as e:
            xcpt.append(filterMessage({'name':'runXML','msg':'add plot to
```

```
the list'},e,bypass))
            continue   # bypass the rest of the for loop

         # ~~> Temper with rank but still gather intelligence
         doplt = True
         rankdo = plot.dids[typePlot][plot.active['xref']]['rank']
         rankdont = xmlConfig[cfgname]['options'].todos['draw']['rank']
         if rankdont != rankdo*int( rankdont / rankdo ): doplt = False
         if not doplt: continue

         # ~~ Step 2. Cumul layers ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
         for layer in ploting.findall("layer"):
            try:
               index,namex = plot.addSubTask(layer)
               target = plot.tasks[namex][index]["target"]
            except Exception as e:
               xcpt.append(filterMessage({'name':'runXML','msg':'add layer
to the list'},e,bypass))
               continue   # bypass the rest of the for loop

            # ~~> round up targets and their configurations looking in exes
and does
            xref,src = target.split(':')
            if gt.dids.has_key(xref):
               try:
                  findlayer = findTargets(gt.dids[xref],src)
               except Exception as e:
                  xcpt.append(filterMessage({'name':'runXML','msg':'could
not find reference to draw the load: '+xref},e))
                  plot.targetSubTask({})
                  continue    # bypass the rest of the for loop
               else :
                  plot.targetSubTask(findlayer)
            elif ex.dids.has_key(xref):
               try:
                  findlayer = findTargets(ex.dids[xref],src)
               except Exception as e:
                  xcpt.append(filterMessage({'name':'runXML','msg':'could
not find reference to draw the extract: '+xref},e))
                  plot.targetSubTask({})
                  continue    # bypass the rest of the for loop
               else :
                  plot.targetSubTask(findlayer)
            elif do.dids.has_key(xref):
               try:
                  findlayer = findTargets(do.dids[xref],src)
               except Exception as e:
                  xcpt.append(filterMessage({'name':'runXML','msg':'could
not find reference to draw the action: '+xref},e))
                  plot.targetSubTask({})
                  continue    # bypass the rest of the for loop
```

```
               else:
                  plot.targetSubTask(findlayer)
            else : xcpt.append({'name':'runXML','msg':'could not find
reference to draw the action: '+xref})

            # ~~> round up decos
            if dc.dids['meta'].has_key(plot.tasks[namex][index]['deco']):
               plot.decolooks(dc.dids['meta'][plot.tasks[namex][index][
'deco']],index,namex)
            else: plot.decolooks({},index,namex)

         plot.update(plot.tasks)

   if xcpt != []: raise Exception({'name':'runXML','msg':'looking at targets
in xmlFile: '+xmlFile,'tree':xcpt})

   # ~~ Matrix distribution by plot types ~~~~~~~~~~~~~~~~~~~~~~~~~~
   # /!\ configurations cannot be called "together" or "distinct" or
"oneofall"
   for typePlot in plot.dids.keys():

      for xref in plot.dids[typePlot]:

         draw = plot.dids[typePlot][xref]
         if not draw.has_key("layers"): continue
         print '    +> reference: ',xref,' of type ',typePlot

         xlayers = ''   # now done with strings as arrays proved to be too
challenging
         for layer in draw["layers"]:
            if layer['config'] == 'together':
               xys = []
               for x in xlayers.split('|'): xys.append( (x+';'+':'.join(
layer['fileName'].keys() )).strip(';') )
               xlayers = '|'.join(xys)
            elif layer['config'] == 'distinct':
               ylayers = layer['fileName'].keys()
               xys = []
               for i in range(len(ylayers)):
                 for x in xlayers.split('|'): xys.append( (x+';'+ylayers[i]).
strip(';') )
               xlayers = '|'.join(xys)
            elif layer['config'] == 'oneofall':
               xys = []; cfg = layer['fileName'].keys()[0]     #/!\ you are
sure to have at least one (?)
               for x in xlayers.split('|'): xys.append( (x+';'+cfg).strip(
';') )
               xlayers = '|'.join(xys)
            else:
               if layer['config'] in layer['fileName'].keys():
                  xys = []
```

```
                for x in xlayers.split('|'): xys.append( (x+';'+layer[
'config']).strip(';') )
                xlayers = '|'.join(xys)
         if xlayers == '':
            xcpt.append(filterMessage({'name':'runXML','msg':'could not find
reference to draw the action: '+xref},e))
            continue

         nbFile = 0; alayers = xlayers.split('|')
         for cfglist in alayers:
            # ~~> Figure name
            if len(alayers) == 1:
              figureName = '.'.join([xref.replace(' ','_'),draw['outFormat']])
            else:
               nbFile += 1
               figureName = '.'.join([xref.replace(' ','_'),str(nbFile),draw
['outFormat']])
            print '       ~> saved as: ',figureName
            figureName = path.join(path.dirname(xmlFile),figureName)
            # ~~> Figure size
            draw["size"] = parseArrayPaires(draw["size"])
            # ~~> Create Figure
            if typePlot == "plot1d": figure = Figure1D(typePlot,draw,
figureName,display)
            if typePlot == "plot2d": figure = Figure2D(typePlot,draw,
figureName,display)
            # ~~> User Deco taken from 'look'
            if layer['deco'].has_key('look'):
               for key in layer['deco']['look'][0]: layer['deco'].update({
key:layer['deco']['look'][0][key]})

            for layer,cfgs in zip(draw["layers"],cfglist.split(';')):
               for cfg in cfgs.split(':'):
                  for file in layer['fileName'][cfg][0]:
                     figure.draw( layer['fileName'][cfg][2], { 'file': file,
                        'deco': layer["deco"],
                        'vars': layer["vars"], 'extract':parseArrayPaires(
layer["extract"]),
                        'type': draw['type'], 'time':parseArrayPaires(layer[
"time"])[0] } )

            figure.show()

   if xcpt != []: raise Exception({'name':'runXML','msg':'looking at
plotting in xmlFile: '+xmlFile,'tree':xcpt})

   # ~~ Validation Criteria ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   #
   racine = path.split(xmlFile)[0]
   os.chdir(racine)
   docriteria = CRITERIA(xmlFile,title,bypass)
```

```
   for criteria in xmlRoot.findall("criteria"):

  # ~~ Step 1. Common check for keys ~~~~~~~~~~~~~~~~~~~~~~~~~~~
      try:
         doadd = docriteria.addCriteria(criteria)
      except Exception as e:
         xcpt.append(filterMessage({'name':'runXML','msg':'add todo to the
list'},e,bypass))
         continue

      # ~~> Temper with rank but still gather intelligence
      docrt = True
      rankdo = docriteria.dids['?'][docriteria.active['xref']]['rank']
      rankdont = xmlConfig[cfgname]['options'].todos['test']['rank']
      if rankdont != rankdo*int( rankdont / rankdo ): docrt = False
      if not docrt: continue

      for cfgname in xmlConfig.keys():
         criteriacfg = xmlConfig[cfgname]['cfg']
         docriteria.addCFG(cfgname,criteriacfg) #if not : continue

         for vars in criteria.findall("variable"):
            try:
               doaddvariable = docriteria.addvariable(vars)
            except Exception as e:
               xcpt.append(filterMessage({'name':'runXML','msg':'add todo to
the list'},e,bypass))
               continue

         conditionNBR = 1
         for condition in criteria.findall("condition"):
            try:
               doaddcondition = docriteria.addcondition(condition,
conditionNBR)
               conditionNBR += 1
            except Exception as e:
               xcpt.append(filterMessage({'name':'runXML','msg':'add todo to
the list'},e,bypass))
               continue

            if "compare" in doaddcondition["do"]:
               try:
                  docriteria.compare(doaddcondition)
               except Exception as e:
                  xcpt.append(filterMessage({'name':'runXML','msg':'   +>
compare'},e,bypass))


      for criteria in docriteria.dids.keys() :
         for cfg in docriteria.dids[criteria]:
            for NBR in docriteria.dids[criteria][cfg]['condition']:
```

```
                if docriteria.dids[criteria][cfg]['condition'][NBR]['result']
[0] == 'success' : continue
                else :
                   if docriteria.dids[criteria][cfg]['condition'][NBR][
'result'][0] == 'warning' :
                      print '\n!!!!!  Warning about the follwing condition %s
!!!!!!!!\n' % docriteria.dids[criteria][cfg]['condition'][NBR]['success']
                   else :
                      #raise Exception([{'name':'CRITERIA','msg':'the
following criteria failed: '+criteria}])
                      raise Exception([{'name':'CRITERIA','msg':'the
following condition failed: '+docriteria.dids[criteria][cfg]['condition'][
NBR]['success']}])


   # ~~ Error management ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   if xcpt != []:  # raise full report
      raise Exception({'name':'runXML','msg':'in xmlFile: '+xmlFile,'tree'
:xcpt})

   return
```