Altair® AcuSolve® 2025

AcuReport Reference Manual

Updated: 11/13/2024

Contents

AcuReport Reference Manual

1

Instruction of the AcuReport tool, a standalone post-processor batch tool used to generate a report from an AcuSolve solution database.

AcuReport includes and uses different modules for creating this report, such as Report, Adb, Acs, 2D Plot and 3D Plot. Since different problems (AcuSolve solution database) have very different information and structures, each report generation will be different, and each must be custom built, preferably by you. Therefore, you will use Python scripting for this purpose.

For this, you will write a report generator script, and then every time you solve a new problem, you will execute the following command:

```
acuReport -problem pump -file pump.rep -html
```

The source (user code) for report generation will be `pump.rep`, and the results will be `pump.html` plus the Figures directory with all of the images. You will also have `pump.tex` as a by product.

This manual explains the role of each module used in report generating.

Report

2

This module creates the `.tex` file based on the LaTex formatting system by using various methods that work on the report, such as `addFigure`, `addSection` and `addTable`.

Report has the ability to add raw LaTex text and is able to convert the created `.tex` file to PDF, `.rtf`, and HTML using `.pdflatex`, `.latex2rtf` and `.tth` (or `.htlatex`).

Adb

3

The role of this module is to create an interface to the Acudb module for opening and extracting the last AcuSolve solution database, and transferring its data to the report and plotting them.

All of the intermediate and final data generated by ACUSIM modules is stored in a working directory, named `ACUSIM.DIR` by default. The C package adb (ACUSIM database) can be used to extract the data through C programs, and Acudb is a Python counterpart to abd.

2D Plot

4

This module defines the curve method for drawing an interpolated curve and the plot method that plots the curve(s) into an image file.

There are many options available for designing the curve, such as color and size. The curves show different information and the relation between multiple values.

3D Plot

5

The 3D plot part role is used to show and change the 3D visualization view of the problem.

The functions in this part show the problem model, such as Volumes, Surfaces, Nodes and Perdiodics. You can change the different properties of the model and scene graph using these functions.

Example of .Rep File

The `.rep` file may look something like the example shown below.

```
##************************************************************************
##  Copyright (c) 1994-2024 ALTAIR Engineering, Inc.
##  All rights reserved.
##  This source code is confidential and may not be disclosed.
##************************************************************************

#=========================================================================
#
# "WindTunnel.rep": Generate a report for the WindTunnel problem
#
#   To get a report, execute:
#   acuReport -problem IM -file WindTunnel.rep -pdf
#
#=========================================================================

import  acuCnf
import  acuUnit
import  string
import  glob
import  os
import  os.path
import  copy
import  re
import  acupu
import  time
import  math
import  inspect
import  numpy
import  sys
import  shutil
import  subprocess
from    acuFvx              import  AcuFvx, AcuPlot
from    acuXmlParReader     import  AcuXmlParReader
from    acuUtil             import  cnvStr2Val

#-------------------------------------------------------------------------
# getMaxFileInd:
#-------------------------------------------------------------------------
def getMaxFileInd( prefix, suffix ):

    ind     = -1
    maxInd  = -1

    prefix = prefix + "."
    suffix = "." + suffix

    searchStr   = str(prefix) + '*[0-9]' + str(suffix)
    fileList    = glob.glob(        searchStr               )

    if len( fileList ) == 0: return -1

    for mFile in fileList:
        preFile = string.split(     mFile,      str(prefix) )
        if len( preFile ) == 2:
```

```
            sufFile = string.split( preFile[1], str(suffix) )
            if len( sufFile ) == 2:
                try:
                    ind = int( sufFile[0] )
                    if ind > maxInd:
                        maxInd = ind
                except:
                    pass
    return maxInd

#-------------------------------------------------------------------------
# getListVals: Get the entries of a list
#-------------------------------------------------------------------------

def getListVals( lstStrVal ):
    """ Get the array value.
        Argument:
            aryStrVal   - a sting contains array value
        Output:
            aryVal      - Tha array values
    """

    parValue    = []
    strLst      = lstStrVal.split(',')
    for item in strLst:
        item    = \
         item.split('(')[-1].split(')')[0].strip().split('\"')[1].strip()
        parValue.append( item )
    return parValue

    #---------------------------------------------------------------------
    # Check the end of the array values and put "," after the last value
    # if does not exist. It needed for findall()
    #---------------------------------------------------------------------
    if not _endAryReg.match( aryStrVal ):
        indx        = aryStrVal.find(           "]"                     )
        if indx == -1:
            indx    = aryStrVal.find(           ")"                     )

        aryStrVal   = aryStrVal[:indx] + "," + aryStrVal[indx:]

    parValue        = []
    lstVals         = re.findall(        _aryVals,   aryStrVal          )

    for val in lstVals:
        value       = val #float(               val[0]                  )
        parValue.append(                        value                   )

    return parValue

#-------------------------------------------------------------------------
# getArrayVals: Get the array Value
#-------------------------------------------------------------------------

def getArrayVals( aryStrVal, _endAryReg, _aryVals ):
    """ Get the array value.
        Argument:
            aryStrVal   - a sting contains array value
        Output:
            aryVal      - Tha array values
    """

    #---------------------------------------------------------------------
```

```
    # Check the end of the array values and put "," after the last value
    # if does not exist. It needed for findall()
    #-------------------------------------------------------------------
    if not _endAryReg.match( aryStrVal ):
        indx        = aryStrVal.find(           "]"                     )
        if indx == -1:
            indx    = aryStrVal.find(           ")"                     )

        aryStrVal   = aryStrVal[:indx] + "," + aryStrVal[indx:]

    parValue        = []
    lstVals         = re.findall(       _aryVals,   aryStrVal           )

    for val in lstVals:
        value       = float(                    val[0]                  )
        unit        = val[1].strip(                                     )
        if unit != "":
            value   = acuUnit.convert(          value,      unit        )

        parValue.append(                        value                   )

    return parValue

#=============================================================================
# dict2Val
#=============================================================================

def dict2Val( dict1 ):

    return cnvStr2Val( dict1['type'], dict1['value'] )

#=============================================================================
# Read Vars
#=============================================================================

def readVars( fileName, wtComps ):

    hmDic       = {}
    parDic      = {}
    fsiDic      = {}

    if not os.path.exists( fileName ):
        for compName in wtComps:
            hmDic[compName] = compName
        return ( hmDic, parDic )

    reader  = AcuXmlParReader( None, fileName = fileName )

    parDic  = reader.getElementsDic( "ACS_PARS" )['ACS_PARS']

    if 'ACS_FLEXIBLE_BODIES' in parDic:
        for comp in parDic['ACS_FLEXIBLE_BODIES']:
            component = dict2Val(
                        parDic['ACS_FLEXIBLE_BODIES'][comp]['component'])
            fsiDic[comp] = []
            for item in component:
                fsiDic[comp].append( item )

    return ( hmDic, parDic, fsiDic )

#=============================================================================
# Get Time History Info
#=============================================================================
```

```
def getMonitorPoints( parDic ):

    othDir      = {}
    defVar      = []
    defLgndRng  = []

    if 'time_history_variables' in parDic['ACS_GLOBAL']:
        defVar = dict2Val(parDic['ACS_GLOBAL']['time_history_variables'])
    if 'time_history_legend_range' in parDic['ACS_GLOBAL']:
        defLgndRng = dict2Val(
                        parDic['ACS_GLOBAL']['time_history_legend_range'])

    if 'ACS_TIME_HISTORY_OUTPUTS' in parDic:
        for item in parDic['ACS_TIME_HISTORY_OUTPUTS']:
            othDir[item]            = {}
            othDir[item]['coords']  = dict2Val(
                parDic['ACS_TIME_HISTORY_OUTPUTS'][item]['coordinates'])
            othDir[item]['output']  = {}
            othDir[item]['legend_range'] = {}
            lgndRng = []
            if 'legend_range' in parDic['ACS_TIME_HISTORY_OUTPUTS'][item]:
                lgndRng = dict2Val(
                parDic['ACS_TIME_HISTORY_OUTPUTS'][item]['legend_range'])
            if 'variables' in parDic['ACS_TIME_HISTORY_OUTPUTS'][item]:
                vars = dict2Val(
                    parDic['ACS_TIME_HISTORY_OUTPUTS'][item]['variables'])
                for var in vars:
                    othDir[item]['output'][var] = []
                    try:
                        if lgndRng != []:
                            if isinstance(lgndRng[0], list):
                                othDir[item]['legend_range'][var] = \
                                            [lgndRng[vars.index(var)][0],
                                            lgndRng[vars.index(var)][1]]
                            else:
                                othDir[item]['legend_range'][var] = \
                                                [lgndRng[0],lgndRng[1]]
                    except:
                        othDir[item]['legend_range'][var] = \
                            ['Auto','Auto']
            else:
                for var in defVar:
                    othDir[item]['output'][var] = []
                    try:
                        if defLgndRng != []:
                            if isinstance(defLgndRng[0], list):
                                othDir[item]['legend_range'][var] = \
                                        [defLgndRng[defVar.index(var)][0],
                                        defLgndRng[defVar.index(var)][1]]
                            else:
                                othDir[item]['legend_range'][var] = \
                                    [defLgndRng[0],defLgndRng[1]]
                    except:
                        othDir[item]['legend_range'][var] = \
                            ['Auto','Auto']
    return othDir

#===============================================================================
# Get Drag Info
#===============================================================================

def createResultData(   rep,    adb,    compMap,
```

```
                    parDic, fsiDic, othDir,
                    rho,    Aref,   fDir=0,
                    tDir=1, cDir=2
                ):

try:
    os.mkdir('Figures')
except:
    pass

fsiFlag     = False
if 'fsi' in parDic['ACS_GLOBAL']:
    if dict2Val(parDic['ACS_GLOBAL']['fsi']):
        fsiFlag = True

osiNames    = adb.getOsiNames()
osiVars     = adb.getOsiVarNames()

thPrts      = adb.getOthNames()
othVars     = adb.getOthVarNames()

bdyPrts     = []
whlPrts     = []
prsPrts     = []

fsiPrts     = {}
nFsiComps   = 0

if 'ACS_COMPONENTS' in parDic:
    for comp in parDic['ACS_COMPONENTS']:
        if comp in osiNames or \
           comp + " tri3 Fluid tet4" in osiNames:
            if dict2Val(
             parDic['ACS_COMPONENTS'][comp]['wtcomp']).find('WHEEL')\
             != -1:
                whlPrts.append( comp )
            if dict2Val(
             parDic['ACS_COMPONENTS'][comp]['wtcomp']).find('BODY')\
             != -1:
                bdyPrts.append( comp )
            if dict2Val(
             parDic['ACS_COMPONENTS'][comp]['wtcomp']).find('WT_INLET')\
             != -1:
                inPrt = comp
            if dict2Val(
             parDic['ACS_COMPONENTS'][comp]['wtcomp']).find('POROUS')\
             != -1:
                comp = "%s tri3 Fluid tet4" % comp
                prsPrts.append( comp )
        else:
            if 'surface_output' in parDic['ACS_COMPONENTS'][comp]:
                srfout = dict2Val(
                    parDic['ACS_COMPONENTS'][comp]['surface_output'])
                if not srfout:
                    print "%s surface output is not active" % comp
                else:
                    print "%s surface output does not exist" % comp

if 'ACS_FLEXIBLE_BODIES' in parDic:
    for comp in parDic['ACS_FLEXIBLE_BODIES']:
        component = dict2Val(
                    parDic['ACS_FLEXIBLE_BODIES'][comp]['component'])
        fsiPrts[comp] = []
```

```
            for item in component:
                fsiPrts[comp].append( item )

    nBdyPrts    = len( bdyPrts )
    nWhlPrts    = len( whlPrts )
    nPrsPrts    = len( prsPrts )

    nFsiPrts    = 0
    #fsiPrts    = fsiDic
    for item in fsiPrts:
        nFsiPrts= nFsiPrts + len( fsiPrts[item] )

    Uin     = adb.getOsiValues( inPrt ,"velocity")
    Ux      = Uin[:,0]
    Uy      = Uin[:,1]
    Uz      = Uin[:,2]
    Usum    = Ux*Ux + Uy*Uy + Uz*Uz
    U       = numpy.sqrt( Usum )
    Dr      = 0.5 * rho * U * U * Aref

    nSteps   = adb.get('nOsiSteps',1)
    osiSteps = adb.get('osiSteps',1)
    osiTimes = adb.get('osiTimes',1)

    DragBody        = {}
    DragAreaBody    = {}
    LiftBody        = {}
    CrossBody       = {}
    DragBodyAvg     = {}
    DragAreaBodyAvg = {}
    LiftBodyAvg     = {}
    CrossBodyAvg    = {}

    TotDragBody     = numpy.zeros( nSteps, dtype='d' )
    TotDragAreaBody = numpy.zeros( nSteps, dtype='d' )
    TotLiftBody     = numpy.zeros( nSteps, dtype='d' )
    TotCrossBody    = numpy.zeros( nSteps, dtype='d' )

    TotDrag         = numpy.zeros( nSteps, dtype='d' )
    TotDragArea     = numpy.zeros( nSteps, dtype='d' )
    TotLift         = numpy.zeros( nSteps, dtype='d' )
    TotCross        = numpy.zeros( nSteps, dtype='d' )

    As  = 1
    if nSteps > 50:
        if 'avg_iter_coefficient' in parDic['ACS_GLOBAL']:
            As  = dict2Val(parDic['ACS_GLOBAL']['avg_iter_coefficient'])
        else:
            As  = 50

    for bdyPrt in bdyPrts:
        trc                     = adb.getOsiValues( bdyPrt, "traction" )
        DragBody[bdyPrt]        = trc[:,fDir] / Dr
        DragAreaBody[bdyPrt]    = DragBody[bdyPrt] * Aref
        LiftBody[bdyPrt]        = trc[:,tDir] / Dr
        CrossBody[bdyPrt]       = trc[:,cDir] / Dr
        TotDragBody             = TotDragBody + DragBody[bdyPrt]
        TotDragAreaBody         = TotDragAreaBody + DragAreaBody[bdyPrt]
        TotLiftBody             = TotLiftBody + LiftBody[bdyPrt]
        TotCrossBody            = TotCrossBody + CrossBody[bdyPrt]
        TotDrag                 = TotDrag + DragBody[bdyPrt]
        TotDragArea             = TotDragArea + DragAreaBody[bdyPrt]
        TotLift                 = TotLift + LiftBody[bdyPrt]
```

```
        TotCross                = TotCross + CrossBody[bdyPrt]
        DragBodyAvg[bdyPrt]     = DragBody[bdyPrt][-As:].mean()
        DragAreaBodyAvg[bdyPrt] = DragAreaBody[bdyPrt][-As:].mean()
        LiftBodyAvg[bdyPrt]     = LiftBody[bdyPrt][-As:].mean()
        CrossBodyAvg[bdyPrt]    = CrossBody[bdyPrt][-As:].mean()

    DragPorous          = {}
    DragAreaPorous      = {}
    LiftPorous          = {}
    CrossPorous         = {}
    DragPorousAvg       = {}
    DragAreaPorousAvg   = {}
    LiftPorousAvg       = {}
    CrossPorousAvg      = {}

    TotDragPorous       = numpy.zeros( nSteps, dtype='d' )
    TotDragAreaPorous   = numpy.zeros( nSteps, dtype='d' )
    TotLiftPorous       = numpy.zeros( nSteps, dtype='d' )
    TotCrossPorous      = numpy.zeros( nSteps, dtype='d' )

    for prsPrt in prsPrts:
        trc                     = adb.getOsiValues( prsPrt,
                                                    "traction"          )
        mtm                     = adb.getOsiValues( prsPrt,
                                                    "momentum_flux"     )
        DragPorous[prsPrt]      = (trc[:,fDir] + mtm[:,fDir]) / Dr
        DragAreaPorous[prsPrt]  = DragPorous[prsPrt] * Aref
        LiftPorous[prsPrt]      = (trc[:,tDir] + mtm[:,tDir]) / Dr
        CrossPorous[prsPrt]     = (trc[:,cDir] + mtm[:,cDir]) / Dr
        TotDragPorous           = TotDragPorous + DragPorous[prsPrt]
        TotDragAreaPorous       = TotDragAreaPorous \
                                            + DragAreaPorous[prsPrt]
        TotLiftPorous           = TotLiftPorous + LiftPorous[prsPrt]
        TotCrossPorous          = TotCrossPorous + CrossPorous[prsPrt]
        TotDrag                 = TotDrag  + DragPorous[prsPrt]
        TotDragArea             = TotDragArea + DragAreaPorous[prsPrt]
        TotLift                 = TotLift  + LiftPorous[prsPrt]
        TotCross                = TotCross + CrossPorous[prsPrt]
        DragPorousAvg[prsPrt]   = DragPorous[prsPrt][-As:].mean()
        DragAreaPorousAvg[prsPrt] = DragAreaPorous[prsPrt][-As:].mean()
        LiftPorousAvg[prsPrt]   = LiftPorous[prsPrt][-As:].mean()
        CrossPorousAvg[prsPrt]  = CrossPorous[prsPrt][-As:].mean()

    DragWheel       = {}
    DragAreaWheel   = {}
    LiftWheel       = {}
    CrossWheel      = {}
    DragWheelAvg    = {}
    DragAreaWheelAvg= {}
    LiftWheelAvg    = {}
    CrossWheelAvg   = {}

    TotDragWheel    = numpy.zeros( nSteps, dtype='d' )
    TotDragAreaWheel= numpy.zeros( nSteps, dtype='d' )
    TotLiftWheel    = numpy.zeros( nSteps, dtype='d' )
    TotCrossWheel   = numpy.zeros( nSteps, dtype='d' )

    for whlPrt in whlPrts:
        trc                     = adb.getOsiValues( whlPrt, "traction" )
        DragWheel[whlPrt]       = trc[:,fDir] / Dr
        DragAreaWheel[whlPrt]   = DragWheel[whlPrt] * Aref
        LiftWheel[whlPrt]       = trc[:,tDir] / Dr
        CrossWheel[whlPrt]      = trc[:,cDir] / Dr
```

```
        TotDragWheel            = TotDragWheel  + DragWheel[whlPrt]
        TotDragAreaWheel        = TotDragAreaWheel \
                                            + DragAreaWheel[whlPrt]
        TotLiftWheel            = TotLiftWheel  + LiftWheel[whlPrt]
        TotCrossWheel           = TotCrossWheel + CrossWheel[whlPrt]
        TotDrag                 = TotDrag + DragWheel[whlPrt]
        TotDragArea             = TotDragArea + DragAreaWheel[whlPrt]
        TotLift                 = TotLift + LiftWheel[whlPrt]
        TotCross                = TotCross + CrossWheel[whlPrt]
        DragWheelAvg[whlPrt]    = DragWheel[whlPrt][-As:].mean()
        DragAreaWheelAvg[whlPrt]= DragAreaWheel[whlPrt][-As:].mean()
        LiftWheelAvg[whlPrt]    = LiftWheel[whlPrt][-As:].mean()
        CrossWheelAvg[whlPrt]   = CrossWheel[whlPrt][-As:].mean()

    DragFsi             = {}
    DragAreaFsi         = {}
    LiftFsi             = {}
    CrossFsi            = {}
    DragFsiAvg          = {}
    DragAreaFsiAvg      = {}
    LiftFsiAvg          = {}
    CrossFsiAvg         = {}

    TotDragFsi          = {}
    TotDragAreaFsi      = {}
    TotLiftFsi          = {}
    TotCrossFsi         = {}
    MeshXDisplacement   = {}
    MeshYDisplacement   = {}
    MeshZDisplacement   = {}

    for item in fsiPrts:
        TotDragFsi[item]            = numpy.zeros( nSteps, dtype='d' )
        TotDragAreaFsi[item]        = numpy.zeros( nSteps, dtype='d' )
        TotLiftFsi[item]            = numpy.zeros( nSteps, dtype='d' )
        TotCrossFsi[item]           = numpy.zeros( nSteps, dtype='d' )
        MeshXDisplacement[item]     = numpy.zeros( nSteps, dtype='d' )
        MeshYDisplacement[item]     = numpy.zeros( nSteps, dtype='d' )
        MeshZDisplacement[item]     = numpy.zeros( nSteps, dtype='d' )

        for fsiPrt in fsiPrts[item]:
            trc                     = adb.getOsiValues( fsiPrt,
                                                        "traction"          )
            disp                    = adb.getOsiValues( fsiPrt,
                                                        "mesh_displacement" )
            DragFsi[fsiPrt]         = trc[:,fDir] / Dr
            DragAreaFsi[fsiPrt]     = DragFsi[fsiPrt] * Aref
            LiftFsi[fsiPrt]         = trc[:,tDir] / Dr
            CrossFsi[fsiPrt]        = trc[:,cDir] / Dr
            TotDragFsi[item]        = TotDragFsi[item] + DragFsi[fsiPrt]
            TotDragAreaFsi[item]    = TotDragAreaFsi[item] \
                                                + DragAreaFsi[fsiPrt]
            TotLiftFsi[item]        = TotLiftFsi[item] + LiftFsi[fsiPrt]
            TotCrossFsi[item]       = TotCrossFsi[item] \
                                                + CrossFsi[fsiPrt]
            TotDrag                 = TotDrag + DragFsi[fsiPrt]
            TotDragArea             = TotDragArea + DragAreaFsi[fsiPrt]
            TotLift                 = TotLift + LiftFsi[fsiPrt]
            TotCross                = TotCross + CrossFsi[fsiPrt]
            DragFsiAvg[fsiPrt]      = DragFsi[fsiPrt][-As:].mean()
            DragAreaFsiAvg[fsiPrt]  = DragAreaFsi[fsiPrt][-As:].mean()
            LiftFsiAvg[fsiPrt]      = LiftFsi[fsiPrt][-As:].mean()
            CrossFsiAvg[fsiPrt]     = CrossFsi[fsiPrt][-As:].mean()
```

```
            MeshXDisplacement[item] = MeshXDisplacement[item] + disp[:,0]
            MeshYDisplacement[item] = MeshYDisplacement[item] + disp[:,1]
            MeshZDisplacement[item] = MeshZDisplacement[item] + disp[:,2]

    print "plotting time history"
    timeHistory_Disp                    = {}

    othVarNms = othVars
    if 'velocity' in othVars:
        othVarNms = othVarNms + [ 'x_velocity', 'y_velocity',
                          'z_velocity', 'velocity_magnitude' ]
    if 'mesh_displacement' in othVars:
        othVarNms = othVarNms + [ 'x_mesh_displacement',
                          'y_mesh_displacement', 'z_mesh_displacement' ]

    def which(lst, var):
        for element in lst:
            if element in var:
                return element
        return False

    coordDic    = { "x_" : 0, "y_" : 1, "z_" : 2 }
    scalarVar   = [ '_magnitude' ]

    for entryName, entry in othDir.iteritems():
        if entryName not in thPrts:
            print "monitor point %s not in database!" % entryName
            continue

        if 'velocity' in entry['output']:
            if 'velocity' in othVars:
                if 'x_velocity' not in entry['output']:
                    entry['output'][u'x_velocity'] = []
                    entry['legend_range'][u'x_velocity'] = \
                        entry['legend_range'][u'velocity']

                if 'y_velocity' not in entry['output']:
                    entry['output'][u'y_velocity'] = []
                    entry['legend_range'][u'y_velocity'] = \
                        entry['legend_range'][u'velocity']

                if 'z_velocity' not in entry['output']:
                    entry['output'][u'z_velocity'] = []
                    entry['legend_range'][u'z_velocity'] = \
                        entry['legend_range'][u'velocity']

            else:
                if 'x_velocity' in entry['output']:
                    entry['output'].pop(u'x_velocity')
                    entry['legend_range'].pop(u'x_velocity')

                if 'y_velocity' in entry['output']:
                    entry['output'].pop(u'y_velocity')
                    entry['legend_range'].pop(u'y_velocity')

                if 'z_velocity' in entry['output']:
                    entry['output'].pop(u'z_velocity')
                    entry['legend_range'].pop(u'z_velocity')

            entry['output'].pop(u'velocity')
            entry['legend_range'].pop(u'velocity')

        if 'mesh_displacement' in entry['output']:
```

```
            if 'mesh_displacement' in othVars:
                if 'x_mesh_displacement' not in entry['output']:
                    entry['output'][u'x_mesh_displacement'] = []
                    entry['legend_range'][u'x_mesh_displacement'] = \
                        entry['legend_range'][u'mesh_displacement']

                if 'y_mesh_displacement' not in entry['output']:
                    entry['output'][u'y_mesh_displacement'] = []
                    entry['legend_range'][u'y_mesh_displacement'] = \
                        entry['legend_range'][u'mesh_displacement']

                if 'z_mesh_displacement' not in entry['output']:
                    entry['output'][u'z_mesh_displacement'] = []
                    entry['legend_range'][u'z_mesh_displacement'] = \
                        entry['legend_range'][u'mesh_displacement']

            else:
                if 'x_mesh_displacement' in entry['output']:
                    entry['output'].pop(u'x_mesh_displacement')
                    entry['legend_range'].pop(u'x_mesh_displacement')

                if 'y_mesh_displacement' in entry['output']:
                    entry['output'].pop(u'y_mesh_displacement')
                    entry['legend_range'].pop(u'y_mesh_displacement')

                if 'z_mesh_displacement' in entry['output']:
                    entry['output'].pop(u'z_mesh_displacement')
                    entry['legend_range'].pop(u'z_mesh_displacement')

            entry['output'].pop(u'mesh_displacement')
            entry['legend_range'].pop(u'mesh_displacement')

        for output,outputArray in entry['output'].iteritems():
            if output not in othVarNms:
                print "given output %s not a valid output" % output
                continue

            arrayInd = which(coordDic, output)
            if arrayInd:
                outputVar   = output.split(arrayInd,1)[1]
                if outputVar != 'magnitude' and outputVar != 'viscosity':
                    index   = coordDic[arrayInd]
                    entry['output'][output] = numpy.asarray(
                      adb.getOthValues(entryName, 0, outputVar)[:,index])
                    continue

            scalarInd   = which( scalarVar, output )

            if scalarInd:
                outputVar   = output.split(scalarInd,1)[0]
                xValue      = numpy.asarray(
                              adb.getOthValues(entryName,0,outputVar)[:,0])
                yValue      = numpy.asarray(
                              adb.getOthValues(entryName,0,outputVar)[:,1])
                zValue      = numpy.asarray(
                              adb.getOthValues(entryName,0,outputVar)[:,2])
                value       = numpy.sqrt(
                              xValue**2 + yValue**2 + zValue**2           )
            else:
                value       = numpy.asarray(
                              adb.getOthValues( entryName, 0, output)[:,0])
            entry['output'][output]  = value
```

```
    ###############################################################
    # Add tables to the report
    ###############################################################
    Cd          = 0.0
    Cl          = 0.0
    Cc          = 0.0
    Ca          = 0.0

    global dataTable
    dataTable   = [ (   "Surface", "Drag Coefficient",
                        "Lift Coefficient", "Cross Coefficient" )   ]

    global fsiTable
    fsiTable    = [ (   "Surface", "Drag Coefficient",
                        "Lift Coefficient", "Cross Coefficient")    ]

    for bdyPrt in bdyPrts:
        sName   = str(bdyPrt)
        sName   = string.replace( sName, "_", "\_" )
        dCoeff  = "%6.5f" % DragBodyAvg[bdyPrt]
        lCoeff  = "%6.5f" % LiftBodyAvg[bdyPrt]
        cCoeff  = "%6.5f" % CrossBodyAvg[bdyPrt]
        dataTable.append( ( sName, dCoeff, lCoeff, cCoeff ) )
        Cd      = Cd + DragBodyAvg[bdyPrt]
        Cl      = Cl + LiftBodyAvg[bdyPrt]
        Cc      = Cc + CrossBodyAvg[bdyPrt]

    for whlPrt in whlPrts:
        sName   = str(whlPrt)
        sName   = string.replace( sName, "_", "\_" )
        dCoeff  = "%6.5f" % DragWheelAvg[whlPrt]
        lCoeff  = "%6.5f" % LiftWheelAvg[whlPrt]
        cCoeff  = "%6.5f" % CrossWheelAvg[whlPrt]
        dataTable.append( ( sName, dCoeff, lCoeff, cCoeff ) )
        Cd      = Cd + DragWheelAvg[whlPrt]
        Cl      = Cl + LiftWheelAvg[whlPrt]
        Cc      = Cc + CrossWheelAvg[whlPrt]

    for item in sorted(fsiPrts.iterkeys()):
        fCd = 0.0
        fCl = 0.0
        fCc = 0.0
        fCa = 0.0
        for fsiPrt in fsiPrts[item]:
            fCd = fCd + DragFsiAvg[fsiPrt]
            fCl = fCl + LiftFsiAvg[fsiPrt]
            fCc = fCc + CrossFsiAvg[fsiPrt]
            fCa = fCa + DragAreaFsiAvg[fsiPrt]
        sName   = (str(item)).replace(  "_" ,   " "     )
        fCd     = "%6.5f" % fCd
        fCl     = "%6.5f" % fCl
        fCc     = "%6.5f" % fCc
        fCa     = "%6.5f" % fCa
        fsiTable.append(  ( sName,  fCd,     fCl,    fCc     )  )

    for prsPrt in prsPrts:
        sName   = str(prsPrt)
        if sName.endswith(' tri3 Fluid tet4'):
            sName = sName[:-16]
        sName   = string.replace( sName, "_", "\_" )
        dCoeff  = "%6.5f" % DragPorousAvg[prsPrt]
        lCoeff  = "%6.5f" % LiftPorousAvg[prsPrt]
        cCoeff  = "%6.5f" % CrossPorousAvg[prsPrt]
```

```
        dataTable.append( ( sName, dCoeff, lCoeff, cCoeff ) )
        Cd      = Cd + DragPorousAvg[prsPrt]
        Cl      = Cl + LiftPorousAvg[prsPrt]
        Cc      = Cc + CrossPorousAvg[prsPrt]

    dataTable.append( ( "Total ", Cd, Cl, Cc ) )

    global areaDataTable
    areaDataTable   = [ ( "Surface", "Drag Area" ) ]

    for bdyPrt in bdyPrts:
        sName   = str(bdyPrt)
        sName   = string.replace( sName, "_", "\_" )
        aCoeff  = "%6.5f" % DragAreaBodyAvg[bdyPrt]
        areaDataTable.append( ( sName, aCoeff ) )
        Ca      = Ca + DragAreaBodyAvg[bdyPrt]

    for whlPrt in whlPrts:
        sName   = str(whlPrt)
        sName   = string.replace( sName, "_", "\_" )
        aCoeff  = "%6.5f" % DragAreaWheelAvg[whlPrt]
        areaDataTable.append( ( sName, aCoeff ) )
        Ca      = Ca + DragAreaWheelAvg[whlPrt]

    for item in fsiPrts:
        for fsiPrt in fsiPrts[item]:
            sName   = str(fsiPrt)
            sName   = string.replace( sName, "_", "\_" )
            aCoeff  = "%6.5f" % DragAreaFsiAvg[fsiPrt]

    for prsPrt in prsPrts:
        sName   = str(prsPrt)
        if sName.endswith(' tri3 Fluid tet4'):
            sName = sName[:-16]
        sName   = string.replace( sName, "_", "\_" )
        aCoeff  = "%6.5f" % DragAreaPorousAvg[prsPrt]
        areaDataTable.append( ( sName, aCoeff ) )
        Ca      = Ca + DragAreaPorousAvg[prsPrt]

    areaDataTable.append( ( "Total ", Ca ) )

    ##############################################################
    # Add curves to the report
    ##############################################################

    if sys.platform == "win32":
        lnTyp = "dotted"
    else:
        lnTyp = "solid"

    if 'simulationType' in parDic['ACS_GLOBAL']:
        if dict2Val(parDic['ACS_GLOBAL']['simulationType'])=='transient':
            xLabel  = "Time (s)"
            xTimes  = osiTimes
        else:
            xLabel  = "Iterations"
            xTimes  = osiSteps
    else:
        xLabel = "Iterations"
        xTimes = osiSteps
    yLabel  = "Coefficients"

    crvList1= []
```

```
    curve1  = Curve(xTimes, TotDrag, name="Drag Coeff", color="Green",
                    lineType = lnTyp) #, symbol = "circle")
    crvList1.append( curve1 )
    curve2  = Curve(xTimes, TotLift, name="Lift Coeff", color="Red",
                    lineType = lnTyp) #, symbol = "square")
    crvList1.append( curve2 )
    curve3  = Curve(xTimes, TotCross, name="Cross Coeff", color="Blue",
                    lineType = lnTyp) #, symbol = "diamond")
    crvList1.append( curve3 )
    curve4  = Curve(xTimes, TotDragArea, name="Drag Area", color="Black",
                    lineType = lnTyp) #, symbol = "diamond")
    crvList1.append( curve4 )


    yMin    = -0.1
    yMax    = 0.5
    xMin    = xTimes[0]
    xMax    = xTimes[-1]

    if nSteps > 10:
        yMin    = min( TotDrag[10:,].min(),TotLift[10:,].min(),
                TotCross[10:,].min(),TotDragArea[10:,].min()) * 0.8
        yMax    = max( TotDrag[10:,].max(),TotLift[10:,].max(),
                TotCross[10:,].max(),TotDragArea[10:,].max()) * 1.2

    if sys.platform == "win32":
        x_data                  = {}
        y_data                  = {}
        x_data['all']           = xTimes
        y_data['Drag Coeff']    = TotDrag
        y_data['Lift Coeff']    = TotLift
        y_data['Cross Coeff']   = TotCross
        y_data['Drag Area']     = TotDragArea

        fvPlots                 = AcuPlot()
        fvPlots.addPlot(x_data  = x_data,       y_data  = y_data,
                        x_label = xLabel,       y_label = yLabel,
                        title   = "Drag Plots", xRange  = (xMin, xMax),
                        yRange  = (yMin, yMax))

        if len(othDir) != 0:
            for entry in othDir:
                for output in othDir[entry]['output']:
                    try:
                        yMin = othDir[entry]['legend_range'][output][0]
                        yMax = othDir[entry]['legend_range'][output][1]
                    except:
                        yMin = 'Auto'
                        yMax = 'Auto'
                    if yMin == 'Auto':
                        try:
                            yMin = othDir[entry]['output'][output].min()
                        except:
                            yMin = min(othDir[entry]['output'][output])
                    else:
                        yMin = float(yMin)
                    if yMax == 'Auto':
                        try:
                            yMax = othDir[entry]['output'][output].max()
                        except:
                            yMax = max(othDir[entry]['output'][output])
                    else:
                        yMax = float(yMax)
```

```
                    fvPlots.addPlot(
                        x_data  = x_data,
                        y_data  = {output:othDir[entry]['output'][output]},
                        x_label = xLabel,
                        y_label = output,
                        title   = '%s_%s' % ( entry, output ),
                        xRange  = (xMin, xMax),
                        yRange  = (yMin, yMax) )

        fvPlots.createPlots()
    else:
        global plot1
        plot1   = Plot2D(crvList1,  width = 800,height  =   600,
                        xLabel  =   xLabel,     yLabel  =   yLabel,
                        yRange  =   [yMin,yMax],gridFlag=   True )

        if len(othDir) != 0:
            for entry in othDir:
                for output in othDir[entry]['output']:
                    try:
                        yMin = othDir[entry]['legend_range'][output][0]
                        yMax = othDir[entry]['legend_range'][output][1]
                    except:
                        yMin = 'Auto'
                        yMax = 'Auto'
                    if yMin == 'Auto':
                        try:
                            yMin = othDir[entry]['output'][output].min()
                        except:
                            yMin = min(othDir[entry]['output'][output])
                    else:
                        yMin = float(yMin)
                    if yMax == 'Auto':
                        try:
                            yMax = othDir[entry]['output'][output].max()
                        except:
                            yMax = max(othDir[entry]['output'][output])
                    else:
                        yMax = float(yMax)

                    crvls = []
                    crvnm = '%s_%s' % (entry, output)
                    crvnm = crvnm.replace(' ', '_')
                    curvv = Curve(xTimes, othDir[entry]['output'][output],
                                  name = crvnm, color = "Red",
                                  lineType = lnTyp) #, symbol = "circle")
                    crvls.append( curvv )
                    curfn = 'Figures/%s' % crvnm
                    optDir[crvnm] = Plot2D(
                                crvls, width = 800, height = 600,
                                xLabel = xLabel, yLabel = output,
                                xRange = [xMin, xMax],
                                yRange = [yMin, yMax], gridFlag = True,
                                fileName = curfn)

    fsiFlag = False
    if 'fsi' in parDic['ACS_GLOBAL']:
        if dict2Val(parDic['ACS_GLOBAL']['fsi']):
                fsiFlag = True
    return
    if not fsiFlag:
        return
```

```
    ###############################################################
    # MESH DISPLACEMENTS - already return called
    ###############################################################

    colortable      = [ 'Green', 'Red', 'Blue', 'Yellow', 'Black' ]
    colortable      = colortable + colortable
    #symboltable     = [ 'circle', 'square']

    #for i in range(8):
        #symboltable.append('diamond')

    xDisplacements  = []
    yDisplacements  = []
    zDisplacements  = []

    for item,ind in zip( sorted( fsiPrts.iterkeys() ),
      range( len( fsiPrts ) ) ):
        tmpCrv      = Curve(xTimes,
                            MeshXDisplacement[item],
                            name    = "mesh_x_displacement " + item,
                            color   = colortable[ind],
                            lineType= lnTyp,
                            #symbol  = symboltable[ind]
                        )
        xDisplacements.append( tmpCrv )

        tmpCrv      = Curve(xTimes,
                            MeshYDisplacement[item],
                            name    ="mesh_y_displacement " + item,
                            color   =colortable[ind],
                            lineType= lnTyp,
                            #symbol  = symboltable[ind]
                        )
        yDisplacements.append( tmpCrv )

        tmpCrv      = Curve(xTimes,
                            MeshZDisplacement[item],
                            name    ="mesh_z_displacement " + item,
                            color   =colortable[ind],
                            lineType= lnTyp,
                            #symbol  = symboltable[ind]
                        )
        zDisplacements.append( tmpCrv )

    ### x displacement plot
    yMin        = 1000
    yMax        = -1000
    for item in MeshXDisplacement:
        yMin    = min( yMin, MeshXDisplacement[item].min() )
        yMax    = max( yMax, MeshXDisplacement[item].max() )

    yLabel      = "mesh_x_displacement in [m]"
    plotX       = Plot2D(
                        xDisplacements,     width   = 800,  height = 600,
                        xLabel  = xLabel,   yLabel  = yLabel,
                        yRange  = [ yMin * 0.8 , yMax * 1.2 ],
                        gridFlag= True
                    )

    ### y displacement plot
    yMin        = 1000
    yMax        = -1000
    for item in MeshYDisplacement:
```

```
        yMin    = min( yMin, MeshYDisplacement[item].min() )
        yMax    = max( yMax, MeshYDisplacement[item].max() )

    yLabel      = "mesh_y_displacement in [m]"
    plotY       = Plot2D(
                        yDisplacements,     width   = 800,  height = 600,
                        xLabel    = xLabel, yLabel  = yLabel,
                        yRange    = [ yMin * 0.8 , yMax * 1.2 ],
                        gridFlag= True
                    )

    ### z displacement plot
    yMin        = 1000
    yMax        = -1000
    for item in MeshZDisplacement:
        yMin    = min( yMin, MeshZDisplacement[item].min() )
        yMax    = max( yMax, MeshZDisplacement[item].max() )

    yLabel      = "mesh_z_displacement in [m]"
    plotZ       = Plot2D(
                        zDisplacements,     width   = 800,  height = 600,
                        xLabel  = xLabel,   yLabel  = yLabel,
                        yRange  = [ yMin * 0.8 , yMax * 1.2 ],
                        gridFlag= True                                  )

    rep.beginBullet()
    rep.addItem("The mesh displacement in x-direction for the different"\
                " FSI components is plotted in Figure"\
                " \\ref{fig:xDisplacements}.", name = ""                )
    rep.endBullet()
    rep.addFigure(  plotX, "center", "x-displacement",
                    1.0/imgFct, ref="fig:xDisplacements"                )

    rep.beginBullet()
    rep.addItem("The mesh displacement in y-direction for the different"\
                " FSI components is plotted in Figure"\
                " \\ref{fig:yDisplacements}.", name = ""                )
    rep.endBullet()
    rep.addFigure(  plotY, "center", "y-displacement",
                    1.0/imgFct, ref="fig:yDisplacements"                )

    rep.beginBullet()
    rep.addItem("The mesh displacement in z-direction for the different"\
                " FSI components is plotted in Figure"\
                " \\ref{fig:zDisplacements}.", name = ""                )
    rep.endBullet()

    rep.addFigure(  plotZ, "center", "z-displacement",
                    1.0/imgFct, ref="fig:zDisplacements"                )

#===============================================================================
# Residual and Solution ratio
#===============================================================================

def createResidualSolutionRatios( rep, adb ):

    '''
    #tStp   = adb.getSteps()
    timeSecs= adb.get('times')

    pressure=  adb.getResRatioData( "pressure", type = "final" )
    eddy    =  adb.getResRatioData( "eddy-viscosity", type = "final" )
    velocity=  adb.getResRatioData( "velocity", type = "final" )
```

```
    pTim    = pressure[1]
    vTim    = velocity[1]
    eTim    = eddy[1]

    pRes    = pressure[2]
    vRes    = velocity[2]
    eRes    = eddy[2]

    if len(pTim) != len(pRes):
        pRes = pRes[:len(pRes)-1]
        vRes = vRes[:len(vRes)-1]
        eRes = eRes[:len(eRes)-1]

    if sys.platform == "win32":
        lnTyp = "dotted"
    else:
        lnTyp = "solid"

    curve1  = Curve( pTim, pRes, name = "Pressure", color = "blue",
                     symbol="circle", lineType = lnTyp )
    curve2  = Curve( vTim, vRes, name = "Velocity", color = "red",
                     symbol="square", lineType = lnTyp )
    curve3  = Curve( eTim, eRes, name = "Eddy Viscosity", color = "green",
                     symbol="diamond", lineType = lnTyp )
    rcrvList= []
    rcrvList.append( curve1 )
    rcrvList.append( curve2 )
    rcrvList.append( curve3 )
    fname   = Plot2D(rcrvList,                  width   =1200,
                   height  =800,                resScale= 2,
                   xLabel  = "Time (s)",        yLog    = True,
                   yLabel  = "Residual Ratio", gridFlag= True,
                   fileName= "resratio",        fileType= "png" )

    pressure=  adb.getSolRatioData( "pressure", type = "final" )
    eddy    =  adb.getSolRatioData( "eddy-viscosity", type = "final" )
    velocity=  adb.getSolRatioData( "velocity", type = "final" )

    pTim    = pressure[1]
    vTim    = velocity[1]
    eTim    = eddy[1]

    pSol    = pressure[2]
    vSol    = velocity[2]
    eSol    = eddy[2]

    if len(pTim) != len(pSol):
        pSol= pSol[:len(pSol)-1]
        vSol= vSol[:len(vSol)-1]
        eSol= eSol[:len(eSol)-1]

    curve4  = Curve( pTim, pSol, name = "Pressure", color = "blue",
                     symbol="circle", lineType = lnTyp )
    curve5  = Curve( vTim, vSol, name = "Velocity", color = "red",
                     symbol="square" , lineType = lnTyp )
    curve6  = Curve( eTim, eSol, name = "Eddy Viscosity", color = "green",
                     symbol="diamond" , lineType = lnTyp )
    scrvList= []
    scrvList.append( curve4 )
    scrvList.append( curve5 )
    scrvList.append( curve6 )
    fname   = Plot2D(scrvList, width   =1200,   height  =800,
```

```
                    resScale= 2,
                    xLabel  = "Time (s)",       yLog    = True,
                    yLabel  = "Solution Ratio", gridFlag= True,
                    fileName= "solratio",       fileType= "png" )
    '''
    pass

#=============================================================================
# animation
#=============================================================================

def createVideo( lZeros ):

    homeDir     = os.getenv(    'ACUSIM_HOME', 'none'       )
    verDir      = os.getenv(    'ACUSIM_VERSION', 'none'    )
    machineDir  = os.getenv(    'ACUSIM_MACHINE', 'none'    )
    acuDir      = os.path.join( homeDir , machineDir        )

    if sys.platform == "win32":
        acuDir      = os.path.join(
                              os.path.join( acuDir, 'base'),
                              'ImageMagick'               )
        vidExec     = os.path.join(     acuDir, 'ffmpeg'    )
    else:
        acuDir      = os.path.join(
                              os.path.join( acuDir, 'base'),
                              'bin'                 )
        vidExec     = os.path.join(     acuDir, 'ffmpeg'    )

    os.chdir('Animation')

    extension   = ( "%%0%dd" % lZeros ) % 1
    testBase    = glob.glob(    '*_%s.*' % extension          )
    if not testBase:
        print "no files to convert"
        os.chdir('..')
        return

    ext         = testBase[0].rsplit('.')[-1]
    for ind,item in enumerate(testBase):
        testBase[ind]   = item.split('_%s.' % extension)[0]

    nOutputs    = len( glob.glob( testBase[0] + '*' )       )

    if nOutputs >= 50:
        frate   = 15
    elif nOutputs >= 10:
        frate   = 7
    else:
        frate   = 1

    for base in testBase:
        vidCmd  = [ vidExec, '-f', 'image2', '-r', '%d' % frate, '-i',
                    '%s_%%0%dd.%s' % (base, lZeros , ext),
                    '-vcodec','copy', '-y', '%s.mp4' % base             ]
        proc = subprocess.Popen(vidCmd , stdout=subprocess.PIPE,
                                stderr=subprocess.PIPE                  )
        proc.wait()

    os.chdir('..')

#=============================================================================
# Images
```

```
#================================================================================

def createImages(   rep, compMap, parDic, fsiDic,
                    fDir=0, tDir=1, cDir=2, rho=1.225
                ):

    print "enter images"

    if 'flow_direction' in parDic['ACS_GLOBAL']:
        fDir    = dict2Val( parDic['ACS_GLOBAL']['flow_direction']  )

    if 'top_direction' in parDic['ACS_GLOBAL']:
        tDir    = dict2Val( parDic['ACS_GLOBAL']['top_direction']   )

    if 'cross_direction' in parDic['ACS_GLOBAL']:
        cDir    = dict2Val( parDic['ACS_GLOBAL']['cross_direction'] )

    cbbox           = (0.0,1.0,0.0,1.0,0.0,1.0)
    tbbox           = (-1.0,6.0,-2.0,2.0,0.0,4.0)
    fsiBbox         = {}
    angle_axis_from = (4.0,0.7,-4.0)
    Vin             = 1.0

    if "angle_axis_from" in parDic['ACS_GLOBAL']:
        angle_axis_from = dict2Val(
                                parDic['ACS_GLOBAL']['angle_axis_from'] )

    if "zoom" in parDic['ACS_GLOBAL']:
        zoom    = dict2Val( parDic['ACS_GLOBAL']['zoom'] )

    if "inletVelocity" in parDic['ACS_GLOBAL']:
        Vin     = dict2Val( parDic['ACS_GLOBAL']['inletVelocity'] )

    if "body_bounding_box" in parDic['ACS_GLOBAL']:
        cbbox   = dict2Val( parDic['ACS_GLOBAL']['body_bounding_box'] )

    if "tunnel_bounding_box" in parDic['ACS_GLOBAL']:
        tbbox   = dict2Val( parDic['ACS_GLOBAL']['tunnel_bounding_box'] )

    if 'ACS_FLEXIBLE_BODIES' in parDic:
        for comp in parDic['ACS_FLEXIBLE_BODIES']:
            fsiBbox[comp]= dict2Val(
                    parDic['ACS_FLEXIBLE_BODIES'][comp]['bounding_box'] )

    ( Bx1, Bx2, By1, By2, Bz1, Bz2 ) = cbbox
    Bxa = 0.5 * ( Bx1+Bx2 )
    Bya = 0.5 * ( By1+By2 )
    Bza = 0.5 * ( Bz1+Bz2 )

    ( Tx1, Tx2, Ty1, Ty2, Tz1, Tz2 ) = tbbox
    Txa = 0.5 * ( Tx1+Tx2 )
    Tya = 0.5 * ( Ty1+Ty2 )
    Tza = 0.5 * ( Tz1+Tz2 )

    if tDir == 1:
        axis_up = (0,1,0)
    elif tDir == 2:
        axis_up = (0,0,1)
    elif tDir == 0:
        axis_up = (1,0,0)

    if fDir == 0:
        Bxf = Bx1
```

```
        Tyf = Ty1
        if tDir == 1:
            Byf = By2
            Bzf = Bz1
            Tyf = Ty2
            Tzf = Tz1
        if tDir == 2:
            Bzf = Bz2
            Byf = By1
            Tzf = Tz2
            Tyf = Ty1

    bdyType     = []
    botType     = []

    fsiType     = {}
    if 'ACS_FLEXIBLE_BODIES' in parDic:
        for comp in parDic['ACS_FLEXIBLE_BODIES']:
            component = dict2Val(
                        parDic['ACS_FLEXIBLE_BODIES'][comp]['component'])
            fsiType[comp] = []
            for item in component:
                fsiType[comp].append( "OSF: %s" % str(item) )

    inType      = []
    prsType     = []

    osiNames    = adb.getOsiNames()

    if 'ACS_COMPONENTS' in parDic:
        for comp in parDic['ACS_COMPONENTS']:
            if comp in osiNames or \
               comp + " tri3 Fluid tet4" in osiNames:
                ikey    = dict2Val(
                            parDic['ACS_COMPONENTS'][comp]['wtcomp'])
                if (ikey.find( 'WHEEL' )!= -1 \
                 or ikey.find( 'BODY' ) != -1 \
                 or ikey.find( 'flex' ) != -1 ):
                    bdyStr  = "OSF: %s" % str(comp)
                    bdyType.append( bdyStr )
                if ikey.find( 'WT_BOTTOM' ) != -1:
                    botStr  = "SBC: %s" % str(comp)
                    botType.append( botStr )
                if ikey.find( 'WT_INLET' ) != -1:
                    inStr   = "SBC: %s" % str(comp)
                    inType.append( inStr )
                if ikey.find('POROUS') != -1:
                    prsStr  = "OSF: %s tri3 Fluid tet4" % str(comp)
                    prsType.append( prsStr )
                    bdyType.append( prsStr )
            else:
                if 'surface_output' in parDic['ACS_COMPONENTS'][comp]:
                    srfout = dict2Val(
                        parDic['ACS_COMPONENTS'][comp]['surface_output'])
                    if not srfout:
                        print "%s surface output is not active" % comp
                    else:
                        print "%s surface output does not exist" % comp

    botTup = tuple( botType )
    bdyTup = tuple( bdyType )
    inTup  = tuple( inType )
    prsTup = tuple( prsType )
```

```
    ###create fsi view factors
    fsiView     = {}
    fsiSideL    = {}
    fsiSideR    = {}
    fsiTop      = {}
    for bc in fsiType:
        fsiView[bc]                     = {}
        ( fx1, fx2, fy1, fy2, fz1, fz2 )= fsiBbox[bc]

        c1, c2, c3  = (fx1+fx2)/2, (fy1+fy2)/2, (fz1+fz2)/2
        center      = "%6.3f %6.3f %6.3f" % ( c1, c2, c3 )
        zoomArr     = [ fx2-fx1, fy2-fy1, fz2-fz1 ]
        zoomArr.sort()
        zoomFsi     = zoomArr[2]/zoomArr[0]
        zoomFsi     = zoomFsi * 10

        if tDir == 2:
            fTopY   = c2
            fTopZ   = fz2 + zoomArr[2]
            fLeftY  = fy1 - zoomArr[2]
            fLeftZ  = c3
            fRightY = fy2 + zoomArr[2]
            fRightZ = c3
            ## check if flexible component is
            ## left or right of the body center
            if (Bya - fy1 ) >= 0:
                lf1, lf2, lf3   = \
                    c1 - zoomArr[2], c2 - zoomArr[2], c3 + zoomArr[2] / 2
            else:
                lf1, lf2, lf3   = \
                    c1 - zoomArr[2], c2 + zoomArr[2], c3 + zoomArr[2] / 2
        else:
            fTopY   = fy2 + zoomArr[2]
            fTopZ   = c3
            fLeftY  = c2
            fLeftZ  = fz1 - zoomArr[2]
            fRightY = c2
            fRightZ = fz2 + zoomArr[2]
            if (Bza - fz2) >= 0:
                lf1, lf2, lf3   = \
                    c1 - zoomArr[2], c2 + zoomArr[2], c3 - zoomArr[2] / 2
            else:
                lf1, lf2, lf3   = \
                    c1 - zoomArr[2], c2 + zoomArr[2], c3 + zoomArr[2] / 2

        fsiView[bc]['axis_from']            = ( lf1, lf2, lf3 )
        fsiView[bc]['axis_at']              = ( c1, c2, c3 )
        fsiView[bc]['axis_up']              = axis_up
        fsiView[bc]['center']               = center
        fsiView[bc]['zoom']                 = zoomFsi
        fsiView[bc]['outline']              = "off"

        fsiSideL[bc]                        = copy.copy( fsiView[bc] )
        fsiSideL[bc]['axis_from']           = ( c1, fLeftY, fLeftZ   )

        fsiSideR[bc]                        = copy.copy( fsiView[bc] )
        fsiSideR[bc]['axis_from']           = ( c1, fRightY, fRightZ )

        fsiTop[bc]                          = copy.copy( fsiView[bc] )
        fsiTop[bc]['axis_from']             = ( c1, fTopY, fTopZ     )

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

```
    ###  ANIMATION ###
    ##################
    if 'animation' in parDic['ACS_GLOBAL']:
        animation = dict2Val(parDic['ACS_GLOBAL']['animation'])
        digits  = -1
    else:
        animation = False
    if animation: #if animation == 'True':
        print "enter animation"

        try:
            os.mkdir('Animation')
        except:
            pass

        nSteps          = adb.get(  'nSteps'    )
        if 'outputFrequency' in parDic['ACS_GLOBAL']:
            outputFreq  = dict2Val(
                                parDic['ACS_GLOBAL']['outputFrequency'] )
        if 'saved_states' in parDic['ACS_GLOBAL']:
            nSavedStates= dict2Val( parDic['ACS_GLOBAL']['saved_states'])
        startTimeStep   = nSteps - (nSavedStates - 1) * outputFreq

        center              = "%6.3f %6.3f %6.3f" % (Bxa + (Bx2-Bxa)/2,
                                                     Bya, Bz2 + Bz2     )
        sideView                = {}
        sideView['axis_at']     = ( Bxa, Bya, Bza )
        sideView['axis_from']   = ( Bxa, Byf, Bza )
        sideView['center']      = center
        sideView['zoom']        = (Bx2 - Bx1) / (Bz2 - Bz1) * 2

        fvxObj              = AcuFvx(   logFile = "./%s" % logFile,
                                        transient="on",
                                        transStart = startTimeStep,
                                        fvxFile = fvxFile               )

        fvxObj.writeVerbatim(text   = 'fv_script("SIZE 640 360")'   )
        fvxObj.setBackground(color  = "white" )
        fvxObj.addViewTable(name            = "view_table_1",
                            zoom            = sideView['zoom'],
                            angle_axis_at   = sideView['axis_at'],
                            angle_axis_up   = axis_up,
                            angle_axis_from = sideView['axis_from'],
                            axis_markers    = "on",
                            outline         = "off",)
        fvxObj.writeViewTable( "view_table_1" )
        tmpdic ={'center':sideView['center']}
        fvxObj.modifyView(  name = "view_table_1",
                            dTable = {'center':sideView['center']}      )

        fvxObj.addBndSrf(   "bnd_surf_bdy",
                            types       = bdyTup,
                            show_mesh   ="off",
                            scalar_func ="none",
                            transparency= 0.0                           )
        fvxObj.writeBndSrf( "bnd_surf_bdy" )

        fvxObj.addCrdSrf(   name            = "crd_surf_y",
                            axis            = "Y",
                            y_axis_current  = Bya,
                            scalar_func     = "velocity_magnitude"      )
        fvxObj.writeCrdSrf( name = "crd_surf_y" )
```

```
        fvxObj.createDisplacementFormula( name="vwtDisplacement" )

        for bc in fsiType:
            bndName = "bnd_surf_fsi_%s" % str(bc)
            crdName = "crd_surf_fsi_%s" % str(bc)
            fsiTup  = tuple( fsiType[bc] )
            fvxObj.addBndSrf(   bndName,
                                types       = fsiTup,
                                show_mesh   ="off",
                                scalar_func ="vwtDisplacement",
                                transparency= 0.0,
                                visibility  ="off"                      )
            fvxObj.writeBndSrf( bndName )
            fvxObj.addCrdSrf(   name            = crdName,
                                axis            = "Y",
                                y_axis_current  = c2,
                                scalar_func     = "velocity_magnitude",
                                transparency    = 0.2,
                                visibility      = "off"                 )
            fvxObj.writeCrdSrf( crdName )

        ## go through transient loop and create images
        counter = 0
        digits  = len( str(( nSteps - startTimeStep ) / outputFreq + 1 ))
        for ts in range( startTimeStep, nSteps + 1, outputFreq ):
            counter     = counter + 1
            strCounter  = ("%%0%dd" % digits) % counter
            fvxObj.writeVerbatim(
                    text = "set_transient( {time_step = %d},1 )" % ts   )

            ## get handles and turn off all configurations
            fvxObj.updateHdls()
            ##select views
            # side view
            fvxObj.modifyView( name = "view_table_1", dTable = sideView )
            fvxObj.writeVerbatim(
                        'modify(boundary_surface_0,{transparency=0.0})' )
            fvxObj.setVis(  name        = "crd_surf_y",
                            dataType    = 'crdSrf',
                            visibility  = True                          )
            fvxObj.setCrdLeg( name      = "crd_surf_y"                  )
            fvxObj.saveImage(
                fileName  = "Animation/body_section_%s.jpg" % strCounter)

            #fvxObj.updateHdls()
            # fsi views
            fvxObj.writeVerbatim(
                        'modify(boundary_surface_0,{transparency=0.5})' )
            fvxObj.setVis(  name        = "crd_surf_y",
                            dataType    = 'crdSrf',
                            visibility  = False                         )
            for bc in fsiType:
                bndName = "bnd_surf_fsi_%s" % str(bc)
                crdName = "crd_surf_fsi_%s" % str(bc)
                fvxObj.modifyView(  name    = "view_table_1",
                                    dTable  = fsiView[bc]               )
                fvxObj.writeVerbatim( '-- view modified')
                fvxObj.setVis(      name        = bndName,
                                    dataType    = 'bndSrf',
                                    visibility  = True                  )
                fvxObj.writeVerbatim(
                    'modify(%s,{scalar_func="vwtDisplacement"})' \
                    % fvxObj.bndHdl[bndName]                            )
```

```
                fvxObj.setBndLeg(   name    = bndName                   )
                fvxObj.saveImage(
                    fileName = "Animation/fsi_displacement_%s_%s.jpg" % \
                                            ( str(bc), strCounter ) )
                fvxObj.writeVerbatim(
                    'modify(%s,{scalar_func="pressure"})' % \
                    fvxObj.bndHdl[bndName]                              )
                fvxObj.saveImage(
                    fileName = "Animation/fsi_pressure_%s_%s.jpg" % \
                    (str(bc),strCounter)                                )
                fvxObj.writeVerbatim(
                    'modify(%s,{scalar_func="none"})' % \
                    fvxObj.bndHdl[bndName]                              )
                fvxObj.setVis(  name        = crdName,
                                dataType    = 'crdSrf',
                                visibility  = True                      )
                fvxObj.setBndLeg(   name      = bndName, on   =   False )
                fvxObj.setCrdLeg(   name      = crdName )
                fvxObj.saveImage(
                        fileName = "Animation/fsi_section_%s_%s.jpg" % \
                                            ( str(bc), strCounter ) )
                fvxObj.setCrdLeg(   name    = crdName,  on  =   False   )
                fvxObj.setVis(      name        = bndName,
                                    dataType    = 'bndSrf',
                                    visibility  = False                 )
                fvxObj.setVis(      name        = crdName,
                                    dataType    = 'crdSrf',
                                    visibility  = False                 )

    else:
        fvxObj  = AcuFvx(   logFile     = "./%s" % logFile,
                            transient   = "off",
                            fvxFile     = fvxFile                       )
        fvxObj.writeVerbatim(   text    = 'fv_script("SIZE 640 360")'   )
        fvxObj.setBackground(   color   = "white"                       )

    size    = len( fvxObj.bndSrf )
    for i in range( size ):
        fvxObj.writeVerbatim( 'all_handles = get_all_object_handles(1)' )
        fvxObj.writeVerbatim(
                'tmp_hdl = all_handles.boundary_handles[%d]'% (size - i))
        fvxObj.writeVerbatim(           'delete( tmp_hdl )'             )

    tmpList = []
    for item in fvxObj.bndSrf:
        tmpList.append( item )
    for item in tmpList:
        fvxObj.writeDelBndSrf( name = item )

    size    = len( fvxObj.crdSrf )
    for i in range( size ):
        fvxObj.writeVerbatim( 'all_handles = get_all_object_handles(1)' )
        fvxObj.writeVerbatim(
                'tmp_hdl = all_handles.coord_handles[%d]'% (size - i)   )
        fvxObj.writeVerbatim(           'delete(tmp_hdl)'               )

    tmpList = []
    for item in fvxObj.crdSrf:
        tmpList.append( item )
    for item in tmpList:
        fvxObj.writeDelCrdSrf( name = item )

    axis_at     = ( Bxa, Bya, Bza )
```

```
    axis_from   = ( Bxf, Byf, Bzf )
    zoomSort    = [ Tx2-Tx1, Ty2-Ty1, Tz2-Tz1 ]
    zoomSort.sort()

    zoom        = 0.75*zoomSort[2]/zoomSort[1]

    #fvxObj= AcuFvx(    logFile     = "./%s" % logFile,
    #                   transient   = "off",
    #                   fvxFile     = fvxFile                           )
    #fvxObj.writeVerbatim(  text     = 'fv_script("SIZE 640 360")'       )
    #fvxObj.setBackground(  color    = "white"                           )
    fvxObj.addViewTable(name            = "view_table_1",
                        zoom            = zoom,
                        angle_axis_at   = axis_at,
                        angle_axis_up   = axis_up,
                        angle_axis_from = axis_from,
                        axis_markers    = "on",
                        outline         = "on",                         )
    fvxObj.writeViewTable(              "view_table_1"                  )
    center = "%6.3f %6.3f %6.3f" % (    Txa, Tya, Tza                   )
    fvxObj.writeVerbatim(       text='fv_script("CENTER %s")'% center   )

    fvxObj.addBndSrf(   "bnd_surf_bdy",
                        types       = bdyTup,
                        show_mesh   = "off",
                        scalar_func = "none",
                        transparency= 0.0                               )
    fvxObj.writeBndSrf( "bnd_surf_bdy"                                  )
    fvxObj.addBndSrf(   "bnd_surf_bot",
                        types       = botTup,
                        show_mesh   = "off",
                        scalar_func = "none",
                        transparency= 0.5                               )
    fvxObj.writeBndSrf( "bnd_surf_bot"                                  )
    fvxObj.saveImage(   fileName    = "tun_bdy_vew.jpg"                 )

    zoom = 2 * ( Tx2 - Tx1 ) / ( Bx2 - Bx1 )
    fvxObj.writeVerbatim(   "view_table_1.zoom=%f"%zoom)
    fvxObj.writeVerbatim(   text    = 'view_table_1.outline="off"'      )
    fvxObj.writeVerbatim(   text    = 'set_view( view_table_1)'         )
    center = "%6.3f %6.3f %6.3f" % ( Bxa, Bya, Bza )
    fvxObj.writeVerbatim(   text    = 'fv_script("CENTER %s")'% center  )

    fvxObj.writeVerbatim(
                text = 'modify(boundary_surface_1,{visibility="off"})'  )
    fvxObj.writeVerbatim(
                text = 'modify(boundary_surface_0,{scalar_func="none"})')
    fvxObj.saveImage(       fileName = "wnd_tun_log.jpg"                )

    fvxObj.writeVerbatim(   text = \
        'modify(boundary_surface_0,{scalar_func="none",show_mesh="on"})')
    fvxObj.saveImage(       fileName= "bod_msh.jpg"  )

    fvxObj.writeVerbatim(   text = \
        'modify(boundary_surface_0,{scalar_func="pressure",'\
        'show_mesh="off"})'                                             )
    fvxObj.writeVerbatim(
                    text = 'set_legend("1","0.95","0.85","BOUNDARY",1)' )
    fvxObj.saveImage(       fileName = "bnd_press.jpg"                  )

    fvxObj.writeVerbatim(
                    text = 'handle_table = get_all_object_handles(1)'   )
    fvxObj.writeVerbatim(
```

```
    text = 'boundary_handle_0 = handle_table.boundary_handles[1]'   )
fvxObj.writeVerbatim(
    text = 'boundary_handle_1 = handle_table.boundary_handles[2]'   )

if 'inletVelocity' in parDic['ACS_GLOBAL']:
    Vin = dict2Val(     parDic['ACS_GLOBAL']['inletVelocity']       )
fvxObj.createCpFormula( name     = "vwtCp",
                        velocity= Vin,
                        density = rho                               )
fvxObj.addCrdSrf(   "coord_surf_1",
                    show_mesh       = "off",
                    scalar_func     = "vwtCp",
                    axis            = 'X',
                    x_axis_current  = Bx2 + Bxa / 2                 )
fvxObj.writeCrdSrf( "coord_surf_1")
fvxObj.writeVerbatim(
            text = 'modify(coordinate_surface_0,{visibility="off"})')
fvxObj.writeVerbatim(
            text = 'modify(boundary_handle_0,{scalar_func="vwtCp"})')
fvxObj.saveImage(   fileName = "coord_press.jpg"                    )

lookat = "{%6.3f,%6.3f,%6.3f,}" % ( Bxa, Bya, Bza )
if cDir == 1:
    fvxObj.addCrdSrf(   "coord_surf_2",
                        show_mesh       = "on",
                        axis            = 'Y',
                        y_axis_current  = Bya                       )
    lookfrom= "{%6.3f,%6.3f,%6.3f,}" % (    Bxa, Byf, Bza           )
    center  = "%6.3f %6.3f %6.3f" % (       Bxa, Bya, Bz2           )
elif cDir == 2:
    fvxObj.addCrdSrf(   "coord_surf_2",
                        show_mesh       = "on",
                        axis            = 'Z',
                        z_axis_current  = Bza                       )
    lookfrom= "{%6.3f,%6.3f,%6.3f,}" % ( Bxa, Bya, Bzf )
    center  = "%6.3f %6.3f %6.3f" % ( Bxa, By2, Bza )
else:
    fvxObj.addCrdSrf(   "coord_surf_2",
                        show_mesh       = "on",
                        axis            = 'X',
                        x_axis_current  = Bxa                       )
    lookfrom= "{%6.3f,%6.3f,%6.3f,}" % ( Bx2, Bya, Bza )
    center  = "%6.3f %6.3f %6.3f" % ( Bx2, Bya, Bza )

fvxObj.writeVerbatim(   "view_table_1.angle_axis.from=%s" % lookfrom)
fvxObj.writeVerbatim(   "view_table_1.angle_axis.at=%s" % lookat    )
fvxObj.writeVerbatim(   text = 'set_view( view_table_1)' )
fvxObj.writeVerbatim(   text = 'fv_script("CENTER %s")' % center    )

fvxObj.writeCrdSrf(     "coord_surf_2"                              )
fvxObj.writeVerbatim(
            text = 'modify(boundary_handle_0,{scalar_func="none"})' )
fvxObj.saveImage(       fileName = "coord_mesh.jpg"                 )

fvxObj.writeVerbatim(
        text = 'modify(boundary_handle_0,{scalar_func="pressure"})' )
fvxObj.writeVerbatim(
        text = 'modify(coordinate_surface_1,{show_mesh="off"})'     )
fvxObj.writeVerbatim(
    text = 'modify(coordinate_surface_1,{scalar_func="pressure"})'  )
#fvxObj.saveImage(  fileName = "coord_press.jpg"                    )

zoomStr = "%d" % (zoom*0.75)
```

```
    fvxObj.writeVerbatim(   "view_table_1.zoom = %s" % zoomStr          )
    fvxObj.writeVerbatim(   text = 'set_view( view_table_1)'            )
    centerEnd = "%6.3f %6.3f %6.3f" % ( Bx2, Bya, Bza )
    fvxObj.writeVerbatim(   text = 'fv_script("CENTER %s")'% centerEnd  )
    fvxObj.saveImage(   fileName = "coord_press22.jpg"                  )

    fvxObj.writeVerbatim(
                text = 'modify(boundary_handle_0,{scalar_func="none"})' )
    fvxObj.writeVerbatim(   text = \
      'modify(coordinate_surface_1,{scalar_func="velocity_magnitude"})' )
    zoomStr = "%d" % zoom
    fvxObj.writeVerbatim(   "view_table_1.zoom = %s" % zoomStr          )
    fvxObj.writeVerbatim(   text    = 'set_view( view_table_1)'         )
    fvxObj.writeVerbatim(   text    = 'fv_script("CENTER %s")'% center  )
    fvxObj.writeVerbatim(
                      text = 'set_legend("1","0.95","0.85","COORD",2)')
    fvxObj.saveImage(       fileName= "coord_velocity.jpg"              )

    fvxObj.writeVerbatim(
                   text = 'handle_table = get_all_object_handles(1)'    )
    fvxObj.writeVerbatim(
        text = 'boundary_handle_0 = handle_table.boundary_handles[1]'   )
    fvxObj.writeVerbatim(
        text = 'coordinate_handle_1 = handle_table.coord_handles[2]'    )

    fvxObj.writeVerbatim(
                text = 'modify(coordinate_handle_1,{visibility="off"})' )

    if fDir == 0:
        xC  = Bxa
        lookfrom= "{%6.3f,%6.3f,%6.3f,}" % ( Bx2, Bya, Bza )
        lookat  = "{%6.3f,%6.3f,%6.3f,}" % ( Bxa, Bya, Bza )
        fvxObj.writeVerbatim( "view_table_1.angle_axis.at=%s" % lookat  )
        fvxObj.writeVerbatim( "view_table_1.angle_axis.from=%s"%lookfrom)
        fvxObj.writeVerbatim(   text = 'set_view( view_table_1)'        )
        if tDir == 1:
            center = "%6.3f %6.3f %6.3f" % ( Bxa, Bya, Bza )
        if tDir == 2:
            center = "%6.3f %6.3f %6.3f" % ( Bxa, Bya, Bz2 )
        fvxObj.writeVerbatim(   text = 'fv_script("CENTER %s")'% center )
        fvxObj.addCrdSrf(   "coord_surf_3",
                            show_mesh       = "on",
                            axis            = 'X',
                            x_axis_current  = xC                        )
        fvxObj.writeCrdSrf( "coord_surf_3")
        fvxObj.writeVerbatim(
                text = 'modify(boundary_handle_0,{scalar_func="none"})' )
        fvxObj.saveImage(   fileName        = "coord_mesh2.jpg"         )

        fvxObj.writeVerbatim(
            text = 'modify(boundary_handle_0,{scalar_func="pressure"})' )
        fvxObj.writeVerbatim(
            text = 'modify(coordinate_surface_2, {show_mesh="off"})'    )
        fvxObj.writeVerbatim( text = \
            'modify(coordinate_surface_2, {scalar_func="pressure"})'    )
        fvxObj.saveImage(       fileName = "coord_press2.jpg"           )

    fvxObj.writeVerbatim(
        text = 'modify(boundary_handle_0,{scalar_func="none"})'         )
    fvxObj.writeVerbatim(
        text = 'modify(coordinate_surface_2,{visibility="off"})'        )

    lookat  = "{%6.3f,%6.3f,%6.3f,}" % ( Bxa, Bya, Bza )
```

```
    if cDir == 1:
        lookfrom= "{%6.3f,%6.3f,%6.3f,}" % ( Bxa, Bya, Bzf )
        axis_up = "{%d,%d,%d,}" % (0,1,0)
        fvxObj.addCrdSrf(   "coord_surf_4",
                            show_mesh       = "off",
                            axis            = 'Z',
                            z_axis_current  = Bza,
                            scalar_func     = "velocity_magnitude"      )
    if cDir == 2:
        lookfrom= "{%6.3f,%6.3f,%6.3f,}" % ( Bxa, Byf, Bza )
        axis_up = "{%d,%d,%d,}" % (0,0,1)
        fvxObj.addCrdSrf(   "coord_surf_4",
                            show_mesh       = "off",
                            axis            = 'Y',
                            y_axis_current  = Bya,
                            scalar_func     = "velocity_magnitude"      )
    fvxObj.writeVerbatim(   "view_table_1.angle_axis.at=%s" % lookat    )
    fvxObj.writeVerbatim(   "view_table_1.angle_axis.from=%s" % lookfrom)
    fvxObj.writeVerbatim(   "view_table_1.angle_axis.up=%s" % axis_up   )
    fvxObj.writeVerbatim(   text = 'set_view( view_table_1)'            )
    center = "%6.3f %6.3f %6.3f" % ( Bxa, Bya, Bza )
    fvxObj.writeVerbatim(   text = 'fv_script("CENTER %s")'% center )
    fvxObj.writeCrdSrf(     "coord_surf_4" )
    fvxObj.writeVerbatim(
                        text = 'set_legend("1","0.95","0.85","COORD",4)')
    fvxObj.saveImage(       fileName = "coord_velocity2.jpg"            )

    fvxObj.writeVerbatim(
                    text = 'handle_table = get_all_object_handles(1)'   )
    fvxObj.writeVerbatim(
        text = 'boundary_handle_0 = handle_table.boundary_handles[1]'   )
    fvxObj.writeVerbatim(
        text =' coordinate_handle_3 = handle_table.coord_handles[4]'    )

    fvxObj.writeVerbatim(
                text = 'modify(coordinate_handle_3,{visibility="off"})' )
    fvxObj.writeVerbatim(   text = \
     'modify(boundary_handle_0,{scalar_func="none", transparency =0.5})')
    bndIndex                = len( fvxObj.bndHdl )
    for bc in fsiType:
        bndName             = "bnd_surf_fsi_%s" % str(bc)
        fsiTup              = tuple( fsiType[bc] )
        fvxObj.addBndSrf(   bndName,
                            types       = fsiTup,
                            show_mesh   = "off",
                            scalar_func = "surface_y_plus",
                            transparency= 0.0,
                            visibility  = "on"                          )
        fvxObj.writeBndSrf( bndName                                     )
        fsiView[bc]['zoom'] = fsiView[bc]['zoom'] * 2
        fvxObj.modifyView(  name = "view_table_1", dTable = fsiView[bc] )
        fvxObj.setBndLeg(   name = bndName                              )
        fvxObj.saveImage(   fileName = "y_plus_flex_%s.jpg" % str(bc)   )
        '''
        fvxObj.modifyView( name = "view_table_1", dTable = fsiSideR[bc] )
        fvxObj.saveImage( fileName = "y_plus_side_flex_%s.jpg" % str(bc))
        fvxObj.modifyView( name = "view_table_1", dTable = fsiTop[bc]   )
        fvxObj.saveImage( fileName = "y_plus_top_flex_%s.jpg" % str(bc) )
        '''
        fvxObj.setVis(  name        = bndName,
                        dataType    = 'bndSrf',
                        visibility  = False                             )
```

```
    fvxObj.writeVerbatim(
        text = 'handle_table = get_all_object_handles(1)'               )
    fvxObj.writeVerbatim(
        text = 'boundary_handle_0 = handle_table.boundary_handles[1]'   )
    fvxObj.writeVerbatim(
        text = 'coordinate_handle_3 = handle_table.coord_handles[4]'    )
    fvxObj.writeVerbatim(
        text = 'modify(boundary_handle_0,{transparency=0.0})'           )
    fvxObj.writeVerbatim(
        text = 'view_table_1.zoom = %s' % zoomStr                       )

    seedX = []
    seedY = []
    seedZ = []

    if tDir == 1:
        I = 10
        J = 6
    if tDir == 2:
        I = 6
        J = 10

    for i in range(0, I):
        for j in range(0, J):
            seedX.append(Tx1+0.01*(Tx2-Tx1))
            seedY.append(By2-i*(By2-By1)/10)
            seedZ.append(Bz2-j*(Bz2-Bz1)/10)

    seedingX = "%s" % seedX
    seedingY = "%s" % seedY
    seedingZ = "%s" % seedZ

    seedingX = '{'+seedingX.strip('[]')+'}'
    seedingY = '{'+seedingY.strip('[]')+'}'
    seedingZ = '{'+seedingZ.strip('[]')+'}'

    fvxObj.addStrmLn(   "strm_ln_0",
                        calculation_parameters_direction        = "both",
                        calculation_parameters_step             = 5,
                        calculation_parameters_time_limit       = 2,
                        calculation_parameters_release_interval = 3,
                        calculation_parameters_duration         = 1,
                        visibility      = "on",
                        display_seeds   = "off",
                        vector_func     = "velocity",
                        scalar_func     = "velocity_magnitude",
                        mode            = "add",
                        seed_coord      = "XYZ",
                        x               = seedingX,
                        y               = seedingY,
                        z               = seedingZ)
    fvxObj.writeStrmLn( "strm_ln_0" )

    if tDir == 1:
        axis_up = "{%d,%d,%d,}" % (0,1,0)
    if tDir == 2:
        axis_up = "{%d,%d,%d,}" % (0,0,1)
    lookat  = "{%6.3f,%6.3f,%6.3f,}" % ( Bxa, Bya, Bza )
    lookfrom= "{%6.3f,%6.3f,%6.3f,}" % ( Bxf, Byf, Bzf )
    center  = "%6.3f %6.3f %6.3f" % ( Bxa, Bya, Bza )
    fvxObj.writeVerbatim( "view_table_1.angle_axis.at=%s" % lookat      )
    fvxObj.writeVerbatim( "view_table_1.angle_axis.from=%s" % lookfrom  )
    fvxObj.writeVerbatim( "view_table_1.angle_axis.up=%s" % axis_up     )
```

```
    fvxObj.writeVerbatim(   text = 'set_view( view_table_1)'            )
    fvxObj.writeVerbatim(   text = 'fv_script("CENTER %s")' % center    )
    fvxObj.writeVerbatim(
        text = 'set_legend("1","0.95","0.85","STREAM",1)'               )
    fvxObj.saveImage(   fileName = "stream_lines_1.jpg"                 )

    if fDir == 0:
        lookat  = "{%6.3f,%6.3f,%6.3f,}" % ( Bxa, Bya, Bza )
        if tDir == 1:
            axis_up = "{%d,%d,%d,}" % (0,1,0)
            lookfrom= "{%6.3f,%6.3f,%6.3f,}" % ( Bxa, Bya, Bzf )
            center  = "%6.3f %6.3f %6.3f" % ( Bxa, By2, Bza )
        if tDir == 2:
            axis_up = "{%d,%d,%d,}" % (0,0,1)
            lookfrom= "{%6.3f,%6.3f,%6.3f,}" % ( Bxa, Byf, Bza )
            center  = "%6.3f %6.3f %6.3f" % ( Bxa, Bya, Bz2 )
        fvxObj.writeVerbatim( "view_table_1.angle_axis.at=%s" % lookat  )
        fvxObj.writeVerbatim( "view_table_1.angle_axis.from=%s"%lookfrom)
        fvxObj.writeVerbatim( "view_table_1.angle_axis.up=%s" % axis_up )
        fvxObj.writeVerbatim( text  = 'set_view( view_table_1)'         )
        fvxObj.writeVerbatim( text  = 'fv_script("CENTER %s")' % center )
        fvxObj.saveImage(   fileName= "stream_lines_2.jpg"              )
    #fvxObj.writeVerbatim( text= 'modify(strm_ln_0,{visibility="off"})' )

    fvxObj.writeVerbatim(
        text = 'handle_table = get_all_object_handles(1)'               )
    fvxObj.writeVerbatim(
        text = 'boundary_handle_0 = handle_table.boundary_handles[1]'   )
    fvxObj.writeVerbatim(
        text = 'strm_ln_handle_0 = handle_table.streamline_handles[1]'  )
    fvxObj.writeVerbatim(
        text = 'modify(strm_ln_handle_0,{visibility="off"})'            )

    fvxObj.writeVerbatim(
        text = 'modify(boundary_handle_0,{scalar_func="surface_y_plus"})')

    if tDir == 1:
        axis_up = "{%d,%d,%d,}" % (0,1,0)
    if tDir == 2:
        axis_up = "{%d,%d,%d,}" % (0,0,1)
    lookat  = "{%6.3f,%6.3f,%6.3f,}" % ( Bxa, Bya, Bza )
    lookfrom= "{%6.3f,%6.3f,%6.3f,}" % ( Bxf, Byf, Bzf )
    center  = "%6.3f %6.3f %6.3f" % ( Bxa, Bya, Bza )
    fvxObj.writeVerbatim( "view_table_1.angle_axis.at=%s" % lookat      )
    fvxObj.writeVerbatim( "view_table_1.angle_axis.from=%s" % lookfrom  )
    fvxObj.writeVerbatim( "view_table_1.angle_axis.up=%s" % axis_up     )
    fvxObj.writeVerbatim( text  = 'set_view( view_table_1)'             )
    fvxObj.writeVerbatim( text  = 'fv_script("CENTER %s")' % center     )
    fvxObj.saveImage( fileName  = "body_y_plus_front.jpg"               )

    if fDir == 0:
        lookfrom= "{%6.3f,%6.3f,%6.3f,}" % ( Bx2, Bya, Bza )
        lookat  = "{%6.3f,%6.3f,%6.3f,}" % ( Bxa, Bya, Bza )
        fvxObj.writeVerbatim( "view_table_1.angle_axis.at=%s" % lookat  )
        fvxObj.writeVerbatim( "view_table_1.angle_axis.from=%s"%lookfrom)
        fvxObj.writeVerbatim( text = 'set_view( view_table_1)'          )
        if tDir == 1:
            center = "%6.3f %6.3f %6.3f" % ( Bxa, Bya, Bza )
        if tDir == 2:
            center = "%6.3f %6.3f %6.3f" % ( Bxa, Bya, Bza )
        fvxObj.writeVerbatim( text = 'fv_script("CENTER %s")' % center  )
        fvxObj.saveImage( fileName = "body_y_plus_back.jpg"             )
```

```
    if fDir == 0:
        lookat  = "{%6.3f,%6.3f,%6.3f,}" % ( Bxa, Bya, Bza )
        if tDir == 1:
            axis_up = "{%d,%d,%d,}" % (1,0,0)
            lookfrom= "{%6.3f,%6.3f,%6.3f,}" % ( Bxa, By1, Bza )
            center  = "%6.3f %6.3f %6.3f" % ( Bxa, Bya, Bza )
        if tDir == 2:
            axis_up = "{%d,%d,%d,}" % (1,0,0)
            lookfrom= "{%6.3f,%6.3f,%6.3f,}" % ( Bxa, Bya, Bz1 )
            center  = "%6.3f %6.3f %6.3f" % ( Bxa, Bya, Bza )
        fvxObj.writeVerbatim( "view_table_1.angle_axis.at=%s" % lookat  )
        fvxObj.writeVerbatim( "view_table_1.angle_axis.from=%s"%lookfrom)
        fvxObj.writeVerbatim( "view_table_1.angle_axis.up=%s" % axis_up )
        fvxObj.writeVerbatim( text  = 'set_view( view_table_1)'         )
        fvxObj.writeVerbatim( text  = 'fv_script("CENTER %s")' % center )
        fvxObj.saveImage( fileName  = "body_y_plus_bottom.jpg"          )

    fvxObj.writeViewTable(              "view_table_1"                  )
    zoom = 2*(Tx2-Tx1)/(Bx2-Bx1)
    fvxObj.writeVerbatim(           "view_table_1.zoom=%f" % zoom       )
    fvxObj.writeVerbatim(       text = 'view_table_1.outline="off"'     )
    fvxObj.writeVerbatim(       text = 'set_view( view_table_1)'        )
    center  = "%6.3f %6.3f %6.3f" % ( Bxa, Bya, Bza )
    fvxObj.writeVerbatim(       text = 'fv_script("CENTER %s")' % center)

    sVarDic = { 'x_velocity':'x-velocity',
                'y_velocity':'y-velocity',
                'z_velocity':'z-velocity',
                'x_mesh_displacement':'x-mesh_displacement',
                'y_mesh_displacement':'y-mesh_displacement',
                'z_mesh_displacement':'z-mesh_displacement'         }

    allVars = [ 'pressure', 'velocity', 'velocity_magnitude',
                'mesh_displacement', 'eddy_viscosity', 'x_velocity',
                'y_velocity', 'z_velocity', 'x_mesh_displacement',
                'y_mesh_displacement', 'z_mesh_displacement'    ]

    fsiFlag = False
    if 'fsi' in parDic['ACS_GLOBAL']:
        if dict2Val(parDic['ACS_GLOBAL']['fsi']):
            fsiFlag = True

    if 'ACS_BOUNDARY_SURFACES' in parDic:
        for bndSrf in parDic['ACS_BOUNDARY_SURFACES']:
            surfaces    = dict2Val(
                    parDic['ACS_BOUNDARY_SURFACES'][bndSrf]['surfaces'] )
            types       = '{ '
            for srf in surfaces:
                types   = types + '"OSF: %s",' % srf
            types       = types + ' }'
            variables   = dict2Val(
                    parDic['ACS_BOUNDARY_SURFACES'][bndSrf]['variables'])
            fvxObj.writeVerbatim(
                    text = 'handle_table = get_all_object_handles(1)'   )
            fvxObj.writeVerbatim( text = \
              'boundary_handle_last = handle_table.boundary_handles[1]' )
            for var in variables:
                if var not in allVars:
                    continue
                elif var == 'velocity':
                    sVars = ['x_velocity', 'y_velocity', 'z_velocity']
                    sVars = [ x for x in sVars if x not in variables ]
                    sVars = [ sVarDic[x] for x in sVars ]
```

```
                elif var == 'mesh_displacement':
                    if fsiFlag:
                        sVars = [   'x_mesh_displacement',
                                    'y_mesh_displacement',
                                    'z_mesh_displacement']
                        sVars = [ x for x in sVars if x not in variables]
                        sVars = [ sVarDic[x] for x in sVars ]
                    else:
                        sVars = []
                else:
                    if 'mesh_displacement' in var:
                        if fsiFlag:
                            sVars = [ sVarDic[var] ]
                        else:
                            sVars = []
                    else:
                        if var in sVarDic:
                            sVars = [ sVarDic[var] ]
                        else:
                            sVars = [ var ]
                for sVar in sVars:
                    fvxObj.writeVerbatim(text = \
                                    'modify(boundary_handle_last,'\
                                    '{types=%s, scalar_func="%s"})'\
                                    % (types, sVar) )
                    fvxObj.saveImage(fileName="%s_%s.jpg" % (bndSrf,sVar))

    fvxObj.writeVerbatim(
        text = 'handle_table = get_all_object_handles(1)'               )
    fvxObj.writeVerbatim(
        text = 'boundary_handle_0 = handle_table.boundary_handles[1]'   )
    fvxObj.writeVerbatim(
        text = 'modify(boundary_handle_0,{visibility="off"})'           )

    if 'ACS_COORDINATE_SURFACES' in parDic:
        for crdSrf in parDic['ACS_COORDINATE_SURFACES']:
            bbox        = None
            lrange      = None

            axis        = dict2Val(
                parDic['ACS_COORDINATE_SURFACES'][crdSrf]['axis']       )
            point       = dict2Val(
                parDic['ACS_COORDINATE_SURFACES'][crdSrf]['point']      )

            if 'variables' in parDic['ACS_COORDINATE_SURFACES'][crdSrf]:
                variables   = dict2Val(
                  parDic['ACS_COORDINATE_SURFACES'][crdSrf]['variables'])
            else:
                variables   = dict2Val(
                  parDic['ACS_GLOBAL']['coordinate_variables'])

            if 'bounding_box' in parDic['ACS_COORDINATE_SURFACES'][crdSrf]:
                bbox = dict2Val(
                 parDic['ACS_COORDINATE_SURFACES'][crdSrf]['bounding_box'])
            else:
                bbox = dict2Val(
                  parDic['ACS_GLOBAL']['coordinate_bounding_box'])

            if 'legend_range' in parDic['ACS_COORDINATE_SURFACES'][crdSrf]:
                lrange = dict2Val(
                 parDic['ACS_COORDINATE_SURFACES'][crdSrf]['legend_range'])
            else:
                lrange = dict2Val(
```

```
                parDic['ACS_GLOBAL']['coordinate_legend_range'])

            if axis[0] == 1.0:
                Byc = Bya
                Bzc = Bza
                if bbox != None:
                    dr1_axis_min= str(bbox[2])
                    dr1_axis_max= str(bbox[3])
                    dr2_axis_min= str(bbox[4])
                    dr2_axis_max= str(bbox[5])
                    Byc = (bbox[3]+bbox[2])/2
                    Bzc = (bbox[5]+bbox[4])/2
                lookat  = "{%6.3f,%6.3f,%6.3f,}" % ( Bxa, Byc, Bzc )
                lookfrom= "{%6.3f,%6.3f,%6.3f,}" % ( Bx2, Byc, Bzc )
                center  = "%6.3f %6.3f %6.3f" % ( Bxa, Byc, Bzc )
                axis_up = "{%d,%d,%d,}" % (0,0,1)
                axis_new= 'X'
                axis_dr1= 'Y'
                axis_dr2= 'Z'
                axis_cur= point[0]
                sclRat  = max(  (Ty2-Ty1)/(bbox[3]-bbox[2]),
                                (Tz2-Tz1)/(bbox[5]-bbox[4]) )
                imgRat  = max(  (bbox[5]-bbox[4])/(bbox[3]-bbox[2]),
                                (bbox[3]-bbox[2])/(bbox[5]-bbox[4]) )
                zoom    = 2 * sclRat #* imgRat

            if axis[1] == 1.0:
                Bxc = Bxa
                Bzc = Bza
                if bbox != None:
                    dr1_axis_min= str(bbox[0])
                    dr1_axis_max= str(bbox[1])
                    dr2_axis_min= str(bbox[4])
                    dr2_axis_max= str(bbox[5])
                    Bxc = (bbox[1]+bbox[0])/2
                    Bzc = (bbox[5]+bbox[4])/2
                lookat  = "{%6.3f,%6.3f,%6.3f,}" % ( Bxc, Bya, Bzc )
                lookfrom= "{%6.3f,%6.3f,%6.3f,}" % ( Bxc, By1, Bzc )
                center  = "%6.3f %6.3f %6.3f" % ( Bxc, Bya, Bzc )
                axis_up = "{%d,%d,%d,}" % (0,0,1)
                axis_new= 'Y'
                axis_dr1= 'X'
                axis_dr2= 'Z'
                axis_cur= point[1]
                sclRat  = max(  (Tx2-Tx1)/(bbox[1]-bbox[0]),
                                (Tz2-Tz1)/(bbox[5]-bbox[4]) )
                imgRat  = max(  (bbox[5]-bbox[4])/(bbox[1]-bbox[0]),
                                (bbox[1]-bbox[0])/(bbox[5]-bbox[4]) )
                zoom    = 2 * sclRat #* imgRat

            if axis[2] == 1.0:
                Bxc = Bxa
                Byc = Bya
                if bbox != None:
                    dr1_axis_min= str(bbox[0])
                    dr1_axis_max= str(bbox[1])
                    dr2_axis_min= str(bbox[2])
                    dr2_axis_max= str(bbox[3])
                    Bxc = (bbox[1]+bbox[0])/2
                    Byc = (bbox[3]+bbox[2])/2
                lookat  = "{%6.3f,%6.3f,%6.3f,}" % ( Bxc, Byc, Bza )
                lookfrom= "{%6.3f,%6.3f,%6.3f,}" % ( Bxc, Byc, Bz2 )
                center  = "%6.3f %6.3f %6.3f" % ( Bxc, Byc, Bza )
```

```
            axis_up = "{%d,%d,%d,}" % (0,1,0)
            axis_new= 'Z'
            axis_dr1= 'X'
            axis_dr2= 'Y'
            axis_cur= point[2]
            sclRat  = max(  (Tx2-Tx1)/(bbox[1]-bbox[0]),
                            (Ty2-Ty1)/(bbox[3]-bbox[2])   )
            imgRat  = max(  (bbox[3]-bbox[2])/(bbox[1]-bbox[0]),
                            (bbox[1]-bbox[0])/(bbox[3]-bbox[2]) )
            zoom    = 2 * sclRat #* imgRat

        fvxObj.writeVerbatim( "view_table_1.angle_axis.at=%s"%lookat)
        fvxObj.writeVerbatim(
                        "view_table_1.angle_axis.from=%s" % lookfrom)
        fvxObj.writeVerbatim("view_table_1.angle_axis.up=%s"%axis_up)
        fvxObj.writeVerbatim(   "view_table_1.zoom=%f" %    zoom    )
        fvxObj.writeVerbatim(   text = 'set_view( view_table_1)'    )
        fvxObj.writeVerbatim(
               text = 'fv_script("CENTER %s")' % center             )
        fvxObj.writeVerbatim(
               text = 'handle_table = get_all_object_handles(1)'    )
        fvxObj.writeVerbatim(   text = \
            'coordinate_handle_3 = handle_table.coord_handles[4]'   )

        for var in variables:
            try:
                if lrange != []:
                    if isinstance(lrange[0], list):
                        scalar_min = lrange[variables.index(var)][0]
                        scalar_max = lrange[variables.index(var)][1]
                    else:
                        scalar_min = lrange[0]
                        scalar_max = lrange[1]
                else:
                    scalar_min = 'Auto'
                    scalar_max = 'Auto'
            except:
                scalar_min = 'Auto'
                scalar_max = 'Auto'
            if var not in allVars:
                continue
            elif var == 'velocity':
                sVars = ['x_velocity', 'y_velocity', 'z_velocity']
                sVars = [ x for x in sVars if x not in variables ]
                sVars = [ sVarDic[x] for x in sVars ]
            elif var == 'mesh_displacement':
                if fsiFlag:
                    sVars = [   'x_mesh_displacement',
                                'y_mesh_displacement',
                                'z_mesh_displacement']
                    sVars = [ x for x in sVars if x not in variables]
                    sVars = [ sVarDic[x] for x in sVars ]
                else:
                    sVars = []
            else:
                if 'mesh_displacement' in var:
                    if fsiFlag:
                        sVars = [ sVarDic[var] ]
                    else:
                        sVars = []
                else:
                    if var in sVarDic:
                        sVars = [ sVarDic[var] ]
```

```
                    else:
                        sVars = [ var ]
                for sVar in sVars:
                    if scalar_min != 'Auto' and scalar_max != 'Auto':
                        fvxObj.writeVerbatim(text = \
                                    'modify(coordinate_handle_3,'\
                                    '{axis="%s", %s_axis={current=%f},'\
                                    '%s_axis={min=%s, max=%s},'\
                                    '%s_axis={min=%s, max=%s},'\
                                    'scalar_func="%s",'\
                                    'scalar_range={min=%s, max=%s},'\
                                    'visibility="on"})'\
                                    % (axis_new,axis_new,axis_cur,
                                    axis_dr1,dr1_axis_min,dr1_axis_max,
                                    axis_dr2,dr2_axis_min,dr2_axis_max,
                                    sVar,scalar_min,scalar_max ) )
                    elif scalar_min != 'Auto' and scalar_max == 'Auto':
                        fvxObj.writeVerbatim(text = \
                                    'modify(coordinate_handle_3,'\
                                    '{axis="%s", %s_axis={current=%f},'\
                                    '%s_axis={min=%s, max=%s},'\
                                    '%s_axis={min=%s, max=%s},'\
                                    'scalar_func="%s",'\
                                    'scalar_range={min=%s},'\
                                    'visibility="on"})'\
                                    % (axis_new,axis_new,axis_cur,
                                    axis_dr1,dr1_axis_min,dr1_axis_max,
                                    axis_dr2,dr2_axis_min,dr2_axis_max,
                                    sVar,scalar_min ) )
                    elif scalar_min == 'Auto' and scalar_max != 'Auto':
                        fvxObj.writeVerbatim(text = \
                                    'modify(coordinate_handle_3,'\
                                    '{axis="%s", %s_axis={current=%f},'\
                                    '%s_axis={min=%s, max=%s},'\
                                    '%s_axis={min=%s, max=%s},'\
                                    'scalar_func="%s",'\
                                    'scalar_range={max=%s},'\
                                    'visibility="on"})'\
                                    % (axis_new,axis_new,axis_cur,
                                    axis_dr1,dr1_axis_min,dr1_axis_max,
                                    axis_dr2,dr2_axis_min,dr2_axis_max,
                                    sVar,scalar_max ) )
                    else:
                        fvxObj.writeVerbatim(text = \
                                    'modify(coordinate_handle_3,'\
                                    '{axis="%s", %s_axis={current=%f},'\
                                    '%s_axis={min=%s, max=%s},'\
                                    '%s_axis={min=%s, max=%s},'\
                                    'scalar_func="%s",visibility="on"})'\
                                    % (axis_new,axis_new,axis_cur,
                                    axis_dr1,dr1_axis_min,dr1_axis_max,
                                    axis_dr2,dr2_axis_min,dr2_axis_max,
                                    sVar ) )
                    fvxObj.saveImage(fileName="%s_%s.jpg" % (crdSrf,sVar))

    fvxObj.writeVerbatim(
            text = 'handle_table = get_all_object_handles(1)'           )
    fvxObj.writeVerbatim(
            text = 'coordinate_handle_3 = handle_table.coord_handles[4]')
    fvxObj.writeVerbatim(
            text = 'modify(coordinate_handle_3,{visibility="off"})'     )

    '''
```

```
    acuHome     = os.getenv(    "ACUSIM_HOME"                           )
    acuMach     = os.getenv(    "ACUSIM_MACHINE"                        )
    reportDir   = os.path.join( acuHome, acuMach, 'plugins', 'report'   )
    initFile    = os.path.join( reportDir, 'init.frm' )
    initFile    = fvxObj.getUnixPath(   initFile    )
    fvxObj.readFormula(  fileName = "%s"  % initFile)

    if tDir == 1:
        axis_up = "{%d,%d,%d,}" % (0,1,0)
    if tDir == 2:
        axis_up = "{%d,%d,%d,}" % (0,0,1)
    lookat = "{%6.3f,%6.3f,%6.3f,}" % ( Bxa, Bya, Bza )
    lookfrom = "{%6.3f,%6.3f,%6.3f,}" % ( Bxf, Byf, Bzf )
    center = "%6.3f %6.3f %6.3f" % ( Bxa, Bya, Bza )
    fvxObj.writeVerbatim( "view_table_1.angle_axis.at=%s" % lookat      )
    fvxObj.writeVerbatim( "view_table_1.angle_axis.from=%s" % lookfrom  )
    fvxObj.writeVerbatim( "view_table_1.angle_axis.up=%s" % axis_up     )
    fvxObj.writeVerbatim( text = 'set_view( view_table_1)')
    fvxObj.writeVerbatim( text = 'fv_script("CENTER %s")' % center      )

    fvxObj.addIsoSrf(   "iso_srf_1",
                        show_mesh           = "off" ,
                        scalar_func         = "velocity_magnitude",
                        iso_func            = "Q",
                        iso_value_current   = 1                         )
    fvxObj.writeIsoSrf( "iso_srf_1")
    fvxObj.writeVerbatim(
        text = 'set_legend("1","0.95","0.85","ISO",1)'                  )
    fvxObj.writeVerbatim(
        text = 'handle_table = get_all_object_handles(1)'               )
    fvxObj.writeVerbatim(
        text = 'boundary_handle_0 = handle_table.boundary_handles[1]'   )
    fvxObj.writeVerbatim(
    text='modify(   boundary_handle_0,{visibility="off"})'              )
    fvxObj.saveImage(   fileName = "iso_velocity.jpg"                   )

    fvxObj.writeVerbatim( text = 'delete(strm_ln_0)'                    )
    fvxObj.writeVerbatim( text = 'delete(coordinate_surface_3)'         )
    fvxObj.writeVerbatim( text = 'delete(coordinate_surface_2)'         )
    fvxObj.writeVerbatim( text = 'delete(coordinate_surface_1)'         )
    fvxObj.writeVerbatim( text = 'delete(coordinate_surface_0)'         )
    fvxObj.writeVerbatim( text = 'delete(boundary_handle_1)'            )
    fvxObj.writeVerbatim( text = 'delete(boundary_handle_0)'            )
    '''

    fvxObj.close()
    fvxObj.runAcuFieldView()
    fvxObj.clear()

    if animation: #if animation == 'True':
        if digits != -1:
            createVideo( digits )

#============================================================================
# SetUp
#============================================================================

print  "Set Up"
problem = acuCnf.cnfGetStr( "problem"   )
runId   = acuCnf.cnfGetInt( "run_id"    )
if runId == 0:
    Index   = getMaxFileInd( problem, "Log" )
    if Index == -1:
```

```
        runId   = 1
    else:
        runId   = Index

logFile = "%s.%d.Log" % (   problem , runId )
texFile = problem + ".tex"
parFile = problem + ".vpar"
fvxFile = problem + ".fvx"
print  "Log File = %s " % logFile

#================================================================================
# WT component names
#================================================================================

wtComps                  = {}
wtComps['BODY'     ]     = ( 'solid', [ 1.0, 0.0, 0.0 ], 'on', 'off' )
wtComps['POROUS'   ]     = ( 'solid', [ 1.0, 0.0, 0.0 ], 'on', 'off' )
wtComps['WT_BOTTOM']     = ( 'solid', [ 0.0, 1.0, 1.0 ], 'on', 'off' )
wtComps['WT_INLET' ]     = ( 'solid', [ 0.0, 1.0, 0.0 ], 'on', 'on'  )
wtComps['WT_OUTLET']     = ( 'solid', [ 0.0, 0.0, 1.0 ], 'on', 'on'  )
wtComps['WT_LEFT'  ]     = ( 'solid', [ 1.0, 1.0, 0.0 ], 'on', 'on'  )
wtComps['WT_RIGHT' ]     = ( 'solid', [ 0.0, 1.0, 1.0 ], 'on', 'on'  )
wtComps['WT_TOP'   ]     = ( 'solid', [ 1.0, 0.0, 1.0 ], 'on', 'on'  )
wtComps['WT_SYMMETRY' ] = ( 'solid', [ 1.0, 0.0, 1.0 ], 'on', 'on'  )
for i in range( 24 ):
    ikey            = 'WHEEL_%d' % i
    wtComps[ikey]   = ( 'solid', [ 1.0, 0.0, 1.0 ], 'on', 'off' )

for i in range( 10 ):
    ikey            = 'flex_comp_%d' % (i+1)

compNames   = wtComps.keys()
( compMap, parDic, fsiDic ) = readVars( parFile, compNames )

invCompMap  = {}
for ikey in compMap:
    vals    = compMap[ikey]
    for val in vals:
        invCompMap[val] = ikey

othDir = getMonitorPoints(parDic)
optDir = {}
#================================================================================
# Open a document
#================================================================================

rep = Report(   texFile,
                packages = ("graphicx","hyperref","geometry","movie15") )
rep.modifyPackageOptions(   "hypersetup",
                            optionMap={"pdfborder":"{0 0 0}"}           )

#================================================================================
# Open the model and solution
#================================================================================
adb = Adb( )

#================================================================================
# Front Page
#================================================================================
rho     = 1.225
Aref    = 1
Ainl    = 1
fDir    = 0
```

```
tDir    = 1
cDir    = 2
Vin     = 1.0

(Bx1, Bx2, By1, By2, Bz1, Bz2) = [ 0.0, 1.0, 0.0, 1.0, 0.0, 1.0 ]
(Tx1, Tx2, Ty1, Ty2, Tz1, Tz2) = [ 0.0, 1.0, 0.0, 1.0, 0.0, 1.0 ]

if 'flow_direction' in parDic['ACS_GLOBAL']:
    fDir    = dict2Val(     parDic['ACS_GLOBAL']['flow_direction']      )

if 'top_direction' in parDic['ACS_GLOBAL']:
    tDir    = dict2Val(     parDic['ACS_GLOBAL']['top_direction']       )

if 'cross_direction' in parDic['ACS_GLOBAL']:
    cDir    = dict2Val(     parDic['ACS_GLOBAL']['cross_direction']     )

if 'body_bounding_box' in parDic['ACS_GLOBAL']:
    body_bounding_box = parDic['ACS_GLOBAL']['body_bounding_box']
    (Bx1, Bx2, By1, By2, Bz1, Bz2)  = dict2Val(     body_bounding_box   )

if 'tunnel_bounding_box' in parDic['ACS_GLOBAL']:
    tunnel_bounding_box = parDic['ACS_GLOBAL']['tunnel_bounding_box']
    (Tx1, Tx2, Ty1, Ty2, Tz1, Tz2)  = dict2Val(     tunnel_bounding_box )

Txl = Tx2-Tx1
Tyh = Ty2-Ty1
Tzw = Tz2-Tz1

Bxl = Bx2-Bx1
Byh = By2-By1
Bzw = Bz2-Bz1

if 'reference_area' in parDic['ACS_GLOBAL']:
    Aref    = dict2Val(     parDic['ACS_GLOBAL']['reference_area']      )
else:
    if fDir == 0:
        Aref    = abs(By2-By1)*abs(Bz2-Bz1)
    if fDir == 1:
        Aref    = abs(Bz2-Bz1)*abs(Bx2-Bx1)
    if fDir == 2:
        Aref    = abs(Bx2-Bx1)*abs(By2-By1)

if 'inlet_area' in parDic['ACS_GLOBAL']:
    Ainl    = dict2Val(     parDic['ACS_GLOBAL']['inlet_area']          )
else:
    if fDir == 0:
        Ainl    = abs(Ty2-Ty1)*abs(Tz2-Tz1)
    if fDir == 1:
        Ainl    = abs(Tz2-Tz1)*abs(Tx2-Tx1)
    if fDir == 2:
        Ainl    = abs(Tx2-Tx1)*abs(Ty2-Ty1)

Brat= Aref/Ainl*100

if sys.platform == "win32":
    figExt = "jpg";
else:
    figExt = "jpeg";

if 'ACS_MATERIAL_MODELS' in parDic:
    for matMdl in parDic['ACS_MATERIAL_MODELS']:
        rho = dict2Val(parDic['ACS_MATERIAL_MODELS'][matMdl]['density'])
        break
```

```
else:
    rho = 1.225

#createResidualSolutionRatios( rep, adb )
createResultData(   rep, adb, compMap, parDic, fsiDic,
                    othDir, rho, Aref, fDir, tDir, cDir )
createImages(   rep, compMap, parDic, fsiDic, rho       )

rep.newPage()
rep.addSpace( 2 )

acuHome     = os.getenv( "ACUSIM_HOME" )
acuMach     = os.getenv( "ACUSIM_MACHINE" )
logoFile    = os.path.join( acuHome, acuMach, 'plugins',
                            'report', 'logo_altair.png' )
newFile     = 'logo.png'
if sys.platform == "win32":
    shutil.copy( logoFile, newFile )
    logoFile    = newFile
if os.path.exists( logoFile ):
    rep.addImage( logoFile, scale = 2,)
rep.addSpace( 2 )

imgFct = 640/ ( 6.0 * 72 )

rep.addTitle( "Wind Tunnel Simulation" )

if "engineerName" in parDic['ACS_GLOBAL']:
    engName = "Engineer name: %s" % dict2Val(
                                    parDic['ACS_GLOBAL']['engineerName'])
    rep.addTitle( engName )

rep.fillVSpace( )
rep.addDate( )
rep.addImage( "wnd_tun_log.%s" % figExt, scale = 1.0/imgFct,)

rep.fillVSpace( )
rep.addText( "\\emph{Product of Altair Engineering Inc.}", "center" )
rep.addText( "http://www.altair.com", "center" )

rep.newPage( )

#=============================================================================
# Table of Contents
#=============================================================================
rep.addTableOfContent( )
rep.newPage( )

#=============================================================================
# Introduction
#=============================================================================
rep.addSection( "Summary" )
rep.addText("This report summarizes the results of an external"\
            "aerodynamic CFD analysis performed by "\
            "Altair\\textquoteright s Virtual Wind Tunnel, "\
            "leveraging AcuSolve\\textquoteright s CFD technology."     )
rep.addText(" ")
rep.addText("The first section provides a brief overview of the"\
            " run and its results."                                     )

nElms       = adb.get( "nElms" )
nTotElems   = 0
```

```
for i in range( nElms ):
    nTotElems = nTotElems + adb.get( "nElmElems", i )

cpuTimes    = adb.getCpuTimes()
elapseTimes = adb.getElapseTimes()
cpuHrs      = cpuTimes.sum() / 3600.0
elapseHrs   = elapseTimes.sum() / 3600.0

summTable   = []

runVersion  = adb.get( 'runVersion' )
summTable.append( ( "AcuSolve version", "%s" % runVersion ) )

if 'simulationType' in parDic['ACS_GLOBAL']:
    summTable.append( ( "Simulation type", "%s" % dict2Val(
                        parDic['ACS_GLOBAL']['simulationType'] ) )      )

summTable.append( ( "Element count", "%d" % nTotElems   ) )
summTable.append( ( "Run time ( Elapse time )", "%6.3f h" % elapseHrs ) )
#summTable.append( ( "CPU time", "%6.3f h" % cpuHrs      ) )

if 'inletVelocity' in parDic['ACS_GLOBAL']:
    keyCount = 0
    for comp in parDic['ACS_COMPONENTS']:
        if dict2Val(
           parDic['ACS_COMPONENTS'][comp]['wtcomp'] ) == "WT_INLET":
            keyCount    += 1
            if 'velocity' in parDic['ACS_COMPONENTS'][comp]:
                velocity    = dict2Val(
                              parDic['ACS_COMPONENTS'][comp]['velocity']  )
            else:
                velocity    = ( dict2Val(
                        parDic['ACS_GLOBAL']['inletVelocity'] ),0.0,0.0 )
            ( xVel, yVel, zVel ) = velocity
            summTable.append( ( "%s" % str( comp ).replace( "_", "\_" ),
                          "%.2f %.2f %.2f m/s" % ( xVel, yVel, zVel ) ) )
    if keyCount == 0:
        summTable.append( ( "Inflow velocity", "%s m/s" % dict2Val(
                              parDic['ACS_GLOBAL']['inletVelocity'] ) )   )

summTable.append( ( "Drag coefficient, Cd", "%6.3f" % \
                    dataTable[len(dataTable)-1][1] )                    )
summTable.append( ( "Lift coefficient, Cl", "%6.3f" % \
                    dataTable[len(dataTable)-1][2] )                    )

rep.addTable( summTable, "Problem Information", ref="tab:summ")

#=============================================================================
# Geometry
#=============================================================================
rep.addSection( "Dimensions" )
rep.addText("This sections contains geometric dimensions"\
            "related to the wind tunnel and the body."                  )

dimmTable   = [ ( "Wind tunnel, bounding box",
                "[%6.3f, %6.3f], [%6.3f, %6.3f], [%6.3f, %6.3f]" % \
                (Tx1, Tx2, Ty1, Ty2, Tz1, Tz2)  )                       ]
dimmTable.append( ( "Body, bounding box",
                "[%6.3f, %6.3f], [%6.3f, %6.3f], [%6.3f, %6.3f]" % \
                (Bx1, Bx2, By1, By2, Bz1, Bz2)  )                       )
dimmTable.append( ( "Wind tunnel dimension",
                    "%6.3f m x %6.3f m x %6.3f m." % (Txl, Tyh, Tzw))    )
dimmTable.append( ( "Body dimension",
```

```
                    "%6.3f m x %6.3f m x %6.3f m." % (Bxl, Byh, Bzw))   )
dimmTable.append( ( "Frontal ref. area, Aref", "%s m^{2}" % Aref     )   )
dimmTable.append( ( "Blockage ratio \\%", "%s" % Brat                )   )
dimmTable.append( ( "Distance inflow - body", "%6.3f m" % abs(Bx1-Tx1)) )

rep.addTable( dimmTable, "Geometric Dimensions", ref="tab:dimm")
rep.newPage( )

rep.addFigure( "tun_bdy_vew.%s" % figExt,
    scale=1.0/imgFct, caption = "Virtual Wind Tunnel", ref="fig:tunbdy" )

rep.addSection( "Mesh" )
rep.addText("This section contains mesh statistics and screen"\
            "shots of several cutting planes through the mesh."         )

nNodes      = adb.get("nNodes")

meshTable   = []

meshTable.append( (     "Numb. of nodes", "%d" % nNodes              )   )
meshTable.append( (     "Numb. of elements", "%d" % nTotElems        )   )
if 'nZones' in parDic['ACS_GLOBAL']:
    meshTable.append( ( "Numb. of refinement zones",
                    "%s" % dict2Val(parDic['ACS_GLOBAL']['nZones'] ) )  )

rep.addTable(       meshTable, "Mesh statistics", ref="tab:mesh"        )

meshFigTable = [ ( "Figure \\ref{fig:flowmesh}", "Symmetry plane " )    ]
meshFigTable.append( ( "Figure \\ref{fig:crossmesh}", "Cross section" ) )

rep.addTable(   meshFigTable,
                "Cutting planes through mesh", ref="tab:meshFig"        )
rep.newPage()

rep.addFigure(  "coord_mesh.%s" % figExt, scale = 1.0 / imgFct,
                caption = "Mesh in Flow Direction", ref = "fig:flowmesh")
rep.addFigure(  "coord_mesh2.%s" % figExt, scale = 1.0 / imgFct,
                caption = "Mesh in Cross Direction", ref="fig:crossmesh")

rep.fillVSpace()

#=============================================================================
# Boundary Conditions and Solution Strategy
#=============================================================================
rep.newPage( )

rep.addSection( "Boundary Conditions and Solution Strategy"             )
rep.addText("In this section the boundary conditions and "\
            "the setup for the CFD run are listed."                     )

boundTable  = []
leftCount   = 0
rightCount  = 0

if 'inletVelocity' in parDic['ACS_GLOBAL']:
    keyCount = 0
    for comp in parDic['ACS_COMPONENTS']:
        if dict2Val(
           parDic['ACS_COMPONENTS'][comp]['wtcomp'] ) == "WT_INLET":
            keyCount    += 1
            if 'velocity' in parDic['ACS_COMPONENTS'][comp]:
                velocity    = dict2Val(
                              parDic['ACS_COMPONENTS'][comp]['velocity']  )
```

```
            else:
                velocity    = ( dict2Val(
                        parDic['ACS_GLOBAL']['inletVelocity'] ),0.0,0.0 )
            ( xVel, yVel, zVel ) = velocity
            boundTable.append( ( "%s" % str( comp ).replace( "_", "\_" ),
                               "%.2f %.2f %.2f m/s" % ( xVel, yVel, zVel )))
            if 'Wind_Tunnel_Left' in comp:
                leftCount += 1
            if 'Wind_Tunnel_Right' in comp:
                rightCount += 1
    if keyCount == 0:
        boundTable.append( ( "Inflow velocity",
            "%s m/s" % dict2Val(parDic['ACS_GLOBAL']['inletVelocity'] )))

keyCount = 0
for comp in parDic['ACS_COMPONENTS']:
    if dict2Val(parDic['ACS_COMPONENTS'][comp]['wtcomp']) == "WT_OUTLET":
        keyCount += 1
        boundTable.append(  (   "%s" % str( comp ).replace( "_", "\_" ),
                                "Pressure outlet"                   )   )
        if 'Wind_Tunnel_Left' in comp:
            leftCount += 1
        if 'Wind_Tunnel_Right' in comp:
            rightCount += 1
if keyCount == 0:
    boundTable.append(  (   "Outflow",  "Pressure outlet"   )   )

if leftCount > 0 and rightCount > 0:
    slipFaces = "Top face of wind tunnel"
elif leftCount > 0 and rightCount == 0:
    slipFaces = "Top, right faces of wind tunnel"
elif leftCount == 0 and rightCount > 0:
    slipFaces = "Top, left faces of wind tunnel"
else:
    slipFaces = "Top, right, left faces of wind tunnel"

boundTable.append( ("Slip Walls", "%s" % slipFaces  ) )
boundTable.append( ("No-slip Walls",
                    "wind tunnel ground, body, wheels, heat-exchange" ) )

rep.addTable(   boundTable, "Boundary conditions", ref = "tab:bound"    )

solTable =  []

if 'simulationType' in parDic['ACS_GLOBAL']:
    solTable.append( (  "Simulation type", "%s" % dict2Val(
                        parDic['ACS_GLOBAL']['simulationType'] ) )      )
    if 'nTimeSteps' in parDic['ACS_GLOBAL']:
        try:
            nSteps = adb.get('nSteps')
        except:
            nSteps = dict2Val( parDic['ACS_GLOBAL']['nTimeSteps'] )
        solTable.append( ( "Number of time steps", "%s" % nSteps ) )
    if 'timeIncr' in parDic['ACS_GLOBAL'] and \
       dict2Val( parDic['ACS_GLOBAL']['simulationType'] ) == "transient":
        try:
            nSteps = adb.get('nSteps')
            timeInc= adb.get('timeInc',nSteps-1)
            time   = adb.get('time',nSteps-1)
        except:
            nSteps = dict2Val(parDic['ACS_GLOBAL']['nTimeSteps'])
            timeInc= dict2Val(parDic['ACS_GLOBAL']['timeIncr'])
            time   = nSteps * timeInc
```

```
        solTable.append( ( "Time increment", "%s" % timeInc ) )
        solTable.append( ( "Physical time", "%6.3f" % time ) )

if dict2Val( parDic['ACS_GLOBAL']['simulationType'] ) == "transient":
    solTable.append( ( "Turbulence model", "Detached Eddy Simulation" ) )
else:
    solTable.append( ( "Turbulence model", "Spalart-Allmaras" )         )

if 'movingGround' in parDic['ACS_GLOBAL']:
    if dict2Val( parDic['ACS_GLOBAL']['movingGround'] ) == 1:
        mgVal   = True
    elif dict2Val( parDic['ACS_GLOBAL']['movingGround'] ) == 0:
        mgVal   = False
    else:
        mgVal   = dict2Val( parDic['ACS_GLOBAL']['movingGround'] )
    solTable.append( ( "Moving ground", "%s" % mgVal    ) )
if 'rotatingWheels' in parDic['ACS_GLOBAL']:
    if dict2Val( parDic['ACS_GLOBAL']['rotatingWheels'] ) == 1:
        rwVal   = True
    elif dict2Val( parDic['ACS_GLOBAL']['rotatingWheels'] ) == 0:
        rwVal   = False
    else:
        rwVal   = dict2Val( parDic['ACS_GLOBAL']['rotatingWheels'] )
    solTable.append( ( "Rotating wheels", "%s" % rwVal   ) )

rep.addTable( solTable, "Solution Strategy", ref="tab:sol")

matTable = []

if 'ACS_MATERIAL_MODELS' in parDic:
    for matMdl in parDic['ACS_MATERIAL_MODELS']:
        matDens = dict2Val( parDic['ACS_MATERIAL_MODELS'][matMdl]['density'] )
        matVisc = dict2Val( parDic['ACS_MATERIAL_MODELS'][matMdl]['viscosity'] )
        break
else:
    matDens = 1.225
    matVisc = 1.781e-5

matTable.append( ( "Density", "%s kg/m3" % matDens ) )
matTable.append( ( "Dynamic Viscosity", "%s kg/m-sec" % matVisc ) )

rep.addTable( matTable, "Material Model", ref="tab:mat" )

#=============================================================================
# Results
#=============================================================================
rep.addSection("Results")
rep.addText("In this section the results of the CFD run are reported. "\
            "Table \\ref{tab:res} gives an overview of the different "\
            "result types."                                          )

fsiFlag = False
if 'fsi' in parDic['ACS_GLOBAL']:
    if dict2Val(parDic['ACS_GLOBAL']['fsi']):
        fsiFlag = True

dirEntries      = os.listdir( os.path.join( os.getcwd(), 'Figures' ) )
monitorPoints   = ""
ref             = ""
fileNames       = []

for item in dirEntries:
    if 'pressure' in item:
```

```
        monitorPoints   = monitorPoints + "pressure, "
        ref             = ref + "\\ref{fig:pressCrv}, "
        fileNames.append(item)
    elif 'vel' in item:
        monitorPoints   = monitorPoints + "velocity, "
        ref             = ref + "\\ref{fig:velCrv}, "
        fileNames.append(item)
    elif 'disp' in item:
        monitorPoints   = monitorPoints + "displacement, "
        ref             = ref + "\\ref{fig:dispCrv}, "
        fileNames.append(item)
    elif 'visc' in item:
        monitorPoints   = monitorPoints + "eddy_viscosity, "
        ref             = ref + "\\ref{fig:viscCrv}, "
        fileNames.append(item)

monitorPoints   = monitorPoints.rsplit(',' , 1)[0]
ref             = ref.rsplit(',' , 1)[0]
comps           = ref.split(',')
for ind,comp in enumerate(comps):
    comps[ind]  = comp.split('{')[-1].split('}')[0]

resTable = []
resTable.append( ( "Table \\ref{tab:coeff}",
    "Drag, lift and cross coefficient of "\
    "individual parts and their totals"                             )   )
resTable.append( ( "Table \\ref{tab:areas}",
    "Drag area of individual parts and their totals"                )   )
resTable.append( ( "Figure \\ref{fig:crvList1}",
    "Drag, lift and cross coefficient history"                      )   )
resTable.append( ( "Figure \\ref{fig:bdypres}",
    "Pressure contours on body"                                     )   )
resTable.append( ( "Figure \\ref{fig:mppress}",
    "Pressure coefficient on body surface"                          )   )
resTable.append( (
    "Figure \\ref{fig:bdyplusf}, \\ref{fig:bdyplusr}, \\ref{fig:bdyplusb}",
    "Body Surface y+ contours"                                      )   )
resTable.append( ( "Figure \\ref{fig:strmtp}, \\ref{fig:strmsd}",
    "Streamlines around body"                                       )   )

compfsi = "Figure"
if fsiFlag:
    resTable.append( (  "Table \\ref{tab:flex}",
        "Drag, lift and cross coefficients for the flexible components"))
    resTable.append( (  "Table \\ref{tab:monitor}",
                        "Monitor point information"                 )   )
    for comp in fsiDic:
        compfsi = compfsi + " \\ref{fig:yPlus_%s}" % str(comp)
    resTable.append( (  "%s" % compfsi,
                        "y+ Contour plots for flexible components"  )   )
    compStr = ""
    for file,comp in zip(fileNames,comps):
        compStr = compStr + " \\ref{%s}," % str(comp)
    resTable.append( ( "Figure%s" % compStr,
        "Displacement, pressure and velocity curves for monitor points"))

rep.addTable(       resTable,       "Results",      ref = "tab:res"     )
rep.newPage( )

rep.addTable(       dataTable,      "Coefficients", ref = "tab:coeff"   )
rep.fillVSpace( )
rep.addTable(       areaDataTable,  "Drag Areas",   ref = "tab:areas"   )
rep.fillVSpace( )
```

```
rep.addText("To compute the above aerodynamic coefficients, "\
            "the following equations are used"                          )

rep.beginBullet()
rep.addText( "\\addtolength{\itemindent}{1.5cm}" )
rep.addItem( "\\indent Drag coefficient, "\
        "C_{d}=\\frac{2\\times F_{x}}{rho\\times v^{2}\\times A_{ref}}",
        name = ""                                                       )
rep.endBullet()

rep.addText("")

rep.beginBullet()
rep.addText( "\\addtolength{\itemindent}{1.5cm}" )
rep.addItem( "\\indent Lift coefficient, "\
        "C_{l}=\\frac{2\\times F_{z}}{rho\\times v^{2}\\times A_{ref}}",
        name = ""                                                       )
rep.endBullet()

rep.addText("")

rep.beginBullet()
rep.addText( "\\addtolength{\itemindent}{1.5cm}" )
rep.addItem( "Cross coefficient, "\
        "C_{c}=\\frac{2\\times F_{y}}{rho\\times v^{2}\\times A_{ref}}",
        name = ""                                                       )
rep.endBullet()

rep.addText(" with")

rep.beginBullet()
rep.addText( "\\addtolength{\itemindent}{2cm}" )
rep.addItem( "F_{x}, F_{y}, and F_{z},", name = "" )
rep.endBullet()

rep.addText("forces are acting on the body in x, y and z directions, "\
            "respectively"                                              )

rep.beginBullet()
rep.addText( "\\addtolength{\itemindent}{2cm}" )
if 'ACS_MATERIAL_MODELS' in parDic:
    for matMdl in parDic['ACS_MATERIAL_MODELS']:
        matDens=dict2Val(parDic['ACS_MATERIAL_MODELS'][matMdl]['density'])
        break
else:
    matDens = 1.225
rep.addItem( "rho is density of fluid (%s kg/m^3)" % matDens, name = "" )
rep.endBullet()

rep.addText("")

rep.beginBullet()
rep.addText( "\\addtolength{\itemindent}{2cm}" )
rep.addItem( "v_{}", name = "" )
rep.endBullet()

rep.addText("is free stream velocity")

rep.beginBullet()
rep.addText( "\\addtolength{\itemindent}{2cm}" )
rep.addItem( "A_{ref}", name = "" )
rep.endBullet()
```

```
rep.addText("is frontal projected area of the object")

rep.beginBullet()
rep.addText( "\\addtolength{\itemindent}{1.5cm}" )
rep.addItem( "Drag area, "\
        "C_{d}\\times A_{ref}=\\frac{2\\times F_{x}}{rho\\times v^{2}}",
        name = ""                                                      )
rep.endBullet()

rep.addText("")

rep.beginBullet()
rep.addText( "\\addtolength{\itemindent}{1.5cm}" )
rep.addItem( "Pressure coefficient, "\
        "C_{p}=\\frac{p-p_{infinity}}{0.5\\times rho\\times v^{2}}",
        name = ""                                                      )
rep.endBullet()

rep.fillVSpace()

if sys.platform == "win32":
    rep.addFigure( "Figures/Drag_Plots.png", scale = 1.0 / imgFct,
                   caption = "Coefficients", ref  = "fig:crvList1"     )
else:
    rep.addFigure(  plot1, scale = 1.0 / imgFct,
                   caption = "Coefficients", ref  = "fig:crvList1"     )

if fsiFlag:
    resTable.append( ( "\\ref{tab:flex}",
        "Drag, lift and cross coefficients for the flexible components"))
    rep.addTable( fsiTable, "Flexible Part Coefficients", ref="tab:flex")
    rep.fillVSpace( )

print  "Creating images"

'''
nOutSteps=adb.get('nOutSteps')
outSteps=adb.get('outSteps')
outTimes=adb.get('outTimes')

pressure= adb.getResRatioData( "pressure", type = "final" )
pTim    = pressure[1]
pRes    = pressure[2]

if len(pTim) != len(pRes):
    tranTime = outTimes[len(outTimes)-2]
else:
    tranTime = outTimes[len(outTimes)-1]
'''

if ',' in monitorPoints:
    monitorPoints   = monitorPoints.rsplit(', ' , 1)[0] + ' and ' + \
                      monitorPoints.rsplit(', ' , 1)[1]
    ref             = ref.rsplit(', ', 1)[0] + ' and ' + \
                      ref.rsplit(', ',1)[1]

othTable = [ ("Monitor name", "coordinates") ]
for item in othDir:
    x           = othDir[item]['coords'][0]
    y           = othDir[item]['coords'][1]
    z           = othDir[item]['coords'][2]
    tableItem   = ( str(item), "(%6.3f,%6.3f,%6.3f)" % ( x, y, z ) )
```

```
    othTable.append(tableItem)

if ref:
    resTable.append( ("%s" % ref, "Monitor points") )
    resTable.append( ("\\ref{tab:monitor}", "Monitor point information"))
    rep.addTable(othTable, "Monitor Point information",ref="tab:monitor")
    for file,comp in zip(fileNames,comps):
        caption = ( file.rsplit('.',1)[0] ).replace('_', ' ')
        if sys.platform == "win32":
            rep.addFigure( 'Figures/'+file, scale = 1.0/imgFct,
                           caption = caption, ref = comp )
        else:
            plotfn = os.path.splitext(file)[0]
            if plotfn in optDir.keys():
                plothnd = str(optDir[plotfn])
                rep.addFigure(plothnd, scale = 1.0 / imgFct,
                              caption = caption, ref = comp)

if 'ACS_BOUNDARY_SURFACES' in parDic:
    for bndSrf in parDic['ACS_BOUNDARY_SURFACES']:
        for var in dict2Val(
          parDic['ACS_BOUNDARY_SURFACES'][bndSrf]['variables']  ):
            rep.addFigure( "%s_%s.%s" % (str(bndSrf),str(var),figExt),
            scale   = 1.0/imgFct,
            caption = "user defined boundary surface " + \
                      str(bndSrf) + '_' + str(var),
            ref     = "fig:%s_%s" % (str(bndSrf),str(var))             )

sVarDic = { 'x_velocity':'x-velocity',
            'y_velocity':'y-velocity',
            'z_velocity':'z-velocity',
            'x_mesh_displacement':'x-mesh_displacement',
            'y_mesh_displacement':'y-mesh_displacement',
            'z_mesh_displacement':'z-mesh_displacement' }

if 'ACS_COORDINATE_SURFACES' in parDic:
    for crdSrf in parDic['ACS_COORDINATE_SURFACES']:
        if 'variables' in parDic['ACS_COORDINATE_SURFACES'][crdSrf]:
            variables = dict2Val( parDic['ACS_COORDINATE_SURFACES'][crdSrf]
['variables'] )
        else:
            variables = dict2Val( parDic['ACS_GLOBAL']['coordinate_variables'] )
        for var in variables:
            if var in sVarDic:
                sVar = sVarDic[var]
            else:
                sVar = var
            caption = "%s on %s" % ( str(sVar), str(crdSrf) )
            rep.addFigure( "%s_%s.%s" % (str(crdSrf),str(sVar),figExt),
                       scale   = 1.0/imgFct,
                       caption = caption.replace('_',' '),
                       ref     = "fig:%s_%s" % (str(crdSrf), str(sVar)))

rep.newPage( )

rep.addFigure( "bnd_press.%s" % figExt,     scale = 1.0/imgFct,
        caption = "Body Surface Pressure Contours", ref = "fig:bdypres" )
'''
rep.addFigure( "coord_press22.%s" % figExt, scale = 1.0/imgFct,
        caption = "Mid Plane Pressure Contours", ref = "fig:mppress2"   )
rep.addFigure( "coord_press2.%s" % figExt,  scale = 1.0/imgFct,
        caption = "Cross Plane Pressure Contours", ref = "fig:cppress"  )
'''
```

```
rep.addFigure( "coord_press.%s" % figExt,   scale = 1.0/imgFct,
        caption = "Body Surface Pressure Coefficient", ref="fig:mppress")

rep.newPage( )
'''
rep.addFigure( "coord_velocity.%s" % figExt,    scale = 1.0/imgFct,
            caption = "Mid Plane Velocity Contours", ref = "fig:mpvel"  )
rep.addFigure( "coord_velocity2.%s" % figExt,   scale = 1.0/imgFct,
            caption = "Cross Plane Velocity Contours", ref = "fig:cpvel")

rep.newPage( )
'''
rep.addFigure( "body_y_plus_front.%s" % figExt, scale = 1.0/imgFct,
            caption = "Body Surface y+ Front View", ref = "fig:bdyplusf")
rep.addFigure( "body_y_plus_back.%s" % figExt,  scale = 1.0/imgFct,
            caption = "Body Surface y+ Rear View", ref = "fig:bdyplusr" )
rep.addFigure( "body_y_plus_bottom.%s" % figExt, scale = 1.0/imgFct,
            caption = "Body Surface y+ Bottom View", ref = "fig:bdyplusb")

rep.newPage( )

rep.addFigure( "stream_lines_1.%s" % figExt,    scale = 1.0/imgFct,
               caption = "Stream lines",   ref = "fig:strmtp"          )
rep.addFigure( "stream_lines_2.%s" % figExt,    scale = 1.0/imgFct,
               caption = "Stream lines Side View", ref = "fig:strmsd"  )

#rep.addFigure( "iso_velocity.%s" % figExt, scale=1.0/imgFct,
   #caption = "Iso Surface of Velocity for Q of 1 m/s", ref="fig:isovel")

for comp in fsiDic:
    resTable.append( ( "\\ref{fig:yPlus_%s}" % str(comp),
            "A y+ Contour plot for flexible component %s" % str(comp) ) )
    rep.addFigure( "y_plus_flex_%s.%s" % (str(comp), figExt),
        scale   = 1.0/imgFct,
        caption = "y+ Contour for flexible component " + str(comp),
        ref     = "fig:yPlus_%s" % str(comp)                          )

rep.newPage()
print  "Done."

#=============================================================================
# Summary
#=============================================================================
rep.addSection("References")
rep.beginBullet()
rep.addItem("Altair Engineering Inc, 2015,  Virtual Wind Tunnel, "\
            "Online Documentation", name = ""                          )
rep.addItem("Altair Engineering Inc, 2015,  AcuSolve Command "\
            "Reference Manual.", name = ""                             )
rep.addItem("Hucho, W.-H., 1997, Aerodynamics of Road Vehicles, "\
            "SAE, ISBN 0-7680-0029-7.", name = ""                      )
rep.endBullet()
rep.newPage()
#=============================================================================
# Close the report and make PDF
#=============================================================================

rep.close( )
rep.writePdf( )

#=============================================================================
# Remove temporary files
#=============================================================================
```

```
'''
try:
    dir  = "Figures"
    for file in os.listdir( "Figures" ):
        os.remove( os.path.join( dir, file ) )
    os.rmdir( dir )
except:
    pass
'''
```

Run Time Options for AcuReport

7

Each program requires zero or more command-line options to run.

The supported command-line options for AcuReport are:

-h

Print usage and exit. It is equivalent to help = TRUE [command-line]

- Default value

  The default value of this option is False.
- Example

```
acuReport -h
```

-file<str>

File name of the report generation script ; (_auto, uses `<problem>.rep`)

- Default value

  The default value of this option is "_auto".
- Example

```
acuReport -file pump.rep
```

-pb<str>

Problem name

- Default value

  The default value of this option is "_undefined".
- Example

```
acuReport -pb pump
```

-dir<str>

Working directory which includes all the intermediate and final data generated by ACUSIM modules.

- Default value

  The default value of this option is "ACUSIM.DIR".
- Example

```
acuReport -file pump.rep -dir pump
```

> **Note:** In the above example, pump directory must exist and includes the data generated by ACUSIM modules.

-run<int>

Run number

- Default value

  The default value of this option is 0.
- Example

```
acuReport -file pump.rep -run 1
```

-acs<str>

AcuConsole database name; (_auto, uses <problem>.acs)

- Default value

  The default value of this option is "_auto ".
- Example

```
acuReport -file pump.rep -acs pump.acs
```

-pdf

Create PDF file.

- Default value

  The default value of this option is False.
- Example

```
acuReport -file pump.rep -pdf
```

-rtf

Create `.rtf` file

- Default value

  The default value of this option is False.
- Example

```
acuReport -file pump.rep -rtf
```

-HTML

Create HTML file

- Default value

  The default value of this option is False.

- Example

```
acuReport -file pump.rep -html
```

-usr1<str>

User specific option 1, accessible in the report

- Default value

  The default value of this option is "None".
- Example

```
acuReport -file pump.rep -usr1 case1
```

-usr2<str>

User specific option 2, accessible in the report

- Default value

  The default value of this option is "None".
- Example

```
acuReport -file pump.rep -usr2 case2
```

-v<int>

Verbose level

- Default value

  The default value of this option is 0.
- Example

```
acuReport -file pump.rep -v 1
```

Document Generation

8

This chapter covers the following:

Report()

Create a new LaTex file object and open it.

Usage

```
rep = Report( fileName = None, packages = (),
                      docClass = 'article', docClassOpt = 'psfig,12pt' )
```

Parameters

`fileName` *(string)*
: File name (`.tex` file name.)

`packages` *(list)*
: List of document packages.

`docClass` *(string)*
: document class.

`docClassOpt` *(string)*
: Document class options.

Return Value

`repObj` *(object)*
: Report (document) object.

Description

This routine creates a new LaTex file object and opens it. For example,

```
rep = Report( "pump.tex", packages=("graphicx", "hyperref") )
```

addAuthors()

Add a set of authors to the report.

Usage

addAuthors(*args

Parameters

**args* *(string)*
Name of authors.

Return Value

None

Description

This routine adds a set of authors to the report. The authors name is given by **args*. For example,

```
rep.addAuthors( "Joe Smith" )
```

addBibliography()

Add bibliography to the report.

Usage

`addBibliography( title )`

Parameters

`title` *(string)*
Biblography title which should be added.

Return Value

None

Description

This routine adds a bibliography title to the report. The bibliography title is given by *`title`*. For example,

```
rep.addBibliography( "This document is about AcuReport" )
```

addDate()

Add date to the report.

Usage

addDate(date = None)

Parameters

date (string)
Date (for example, 09/17/2009). If none, the current date will be used.

Return Value

None

Description

This routine adds the date to the report. The date is given by date. For example,

```
rep.addDate( "10/24/09" )
```

addEquation()

Add a formal equation to the report.

Usage

addEquation(equation, ref = None)

Parameters

equation *(string)*
The desired equation in LaTex format.

ref *(string)*
Label of the equation for further reference.

Return Value

None

Description

This routine adds a formal equation to the report. The equation is given by *equation*. The *ref* argument is the label of the equation for further reference. For example,

```
rep.addEquation( r'\rho u_{i,t} + \rho u_j u_{i,j} = -p_{,i} + \tau_{ij,j} + \rho
 b_i', ref = 'eqn:continuity' )
```

addFigure()

Add a figure to the report.

Usage

```
addFigure( fileName, justify='center',
                   caption = None, scale = 1, ref = None )
```

Parameters

`fileName` ***(string)***
: Figure file name. Notice that the full directory of the figure needs to be included.

`justify` ***(string)***
: Figure justification. Valid values are flushleft, flushright and center.

`caption` ***(string)***
: Caption of the figure.

`scale` ***(integer)***
: Image scale.

`ref` ***(string)***
: Label of the figure for further reference.

Return Value

None

Description

This routine adds a figure to the report. The figure fileName, justification, caption and scale are given by *`fileName`*, *`justify`*, *`caption`* and *`scale`*. The *`ref`* argument is the label of the figure for further reference. For example,

```
fname = vis.saveImage( width=600, height=400 )
rep.addFigure( fname, "center", "Geometry of the problem",
               1.0, "fig:geom" )
rep.addText( """The geometry is given in Figure \\ref{fig:geom}."""  )
```

addImage()

Add an image to the report.

Usage

```
addImage( fileName, justify = 'center',
                    scale = 1, hasCaption = False )
```

Parameters

`fileName` *(string)*
: Image file name.

`justify` *(string)*
: Image justification. Valid values are flushleft, flushright and center.

`scale` *(integer)*
: Image scale.

`hasCaption` *(boolean)*
: If True leave the justify block open for adding the caption.

Return Value

None

Description

This routine adds a figure to the report. The figure fileName, justification, caption and scale are given by *`fileName`*, *`justify`*, *`caption`* and *`scale`*. If hasCaption is True, the justify block is left open for adding the caption. For example,

```
fname = vis.saveImage( width=600, height=400 )
rep.addImage( fname, "center", 1.0 )
```

addInlineEquation()

Add an inline equation to the report.

Usage

`addInlineEquation( equation )`

Parameters

`equation` *(string)*
The equation in LaTex format.

Return Value

None

Description

This routine adds an inline equation to the report. The equation is given by *`equation`*.

addItem()

Add an item to a bullet or numbered block in the report.

Usage

`addItem( text, name = None )`

Parameters

`text` *(string)*
: The item body content.

`name` *(string)*
: The item name or title next to bullet or number.

Return Value

None

Description

This routine adds an item to a bullet or numbered block in the report. The item text and name are given by *`text`* and *`name`*. For example,

```
rep.beginBullet( )
rep.addItem( "Sample text", "Item 1" )
```

addSection()

Add a section to the report.

Usage

`addSection( title )`

Parameters

`title` *(string)*
Title of the section.

Return Value

None

Description

This routine adds a section to the report. The section title is given by *`title`*. For example,

```
rep.addSection( "Background" )
```

addSpace()

Add vertical space to the report.

Usage

```
addSpace( spaceSize, unit = "cm")
```

Parameters

`spaceSize` (integer)
: Space Size.

`unit` (string)
: Unit of the size. Valid values are cm, mm and in.

Return Value

None

Description

This routine adds vertical space to the report. The spaceSize and unit are given by *`spaceSize`* and *`unit`*. For example,

```
rep.addSpace( 2, "in" )
```

addSubSection()

Add a sub-section to the report.

Usage

`addSubSection( title )`

Parameters

`title` *(string)*
Title of the sub-section.

Return Value

None

Description

This routine adds a sub-section to the report. The sub-section title is given by *`title`*. For example,

```
rep.addSubSection( "Mesh" )
```

addSubSubSection()

Add a sub-sub-section to the report.

Usage

`addSubSubSection( title )`

Parameters

`title` *(string)*
Title of the sub-section.

Return Value

None

Description

This routine adds a sub-sub-section to the report. The sub-sub section title is given by *`title`*. For example,

```
rep.addSubSubSection( "Mesh values" )
```

addTable()

Add a table to the report.

Usage

```
addTable( table, caption = None, justify = None,
                  ref = None, colsWidths = None, border = True )
```

Parameters

table (list)
A two dimensional list containing the table rows and columns data.

caption (string)
Caption of the table.

justify (string)
Table justification. Valid values are flushleft, flushright and center.

ref (string)
Label of the table for reference.

colsWidths (list)
List of column widths in centimeter.

border (boolean)
If True draw table borders.

Return Value

None

Description

This routine adds a table to the report. The table contents, caption, justification, reference and columns width are given by *table*, *caption*, *justify*, *ref* and *colsWidths*. If *border* is True the table borders will be drawn. For example,

```
presId = -1
massId = -1
for i in range( nOsiVars ):
    name = adb.get( "osiVarName", i )
    if name == "pressure": presId = i
    if name == "mass_flux": massId = i
nOsfs = adb.get( "nOsfs" )
inletId = -1
for i in range(nOsfs):
    name = adb.get( "osfName", i )
    if name == "inflow": inletId = i
pres = adb.get( "osiValues", inletId, presId )
mass = adb.get( "osiValues", inletId, massId )
mpData= [ ("Pressure Drop (Pa)", "Mass Flux (Kg/sec)" ) ]
for i in range(len(pres)):
    mpData.append( ("%.2f" % pres[i], "%.2f" % mass[i]) )
```

```
rep.addTable( mpData, "Fan performance", "center", "tab:mp" )
```

addTableOfContent()

Add a table of contents to the report.

Usage

`addTableOfContent( )`

Parameters

None

Return Value

None

Description

This routine adds a table of contents to the report. For example,

```
rep.addTableOfContent( )
```

addText()

Add text to the report.

Usage

```
addText( text, justify = "flushleft", newLine = True,
                    style = None, size=None )
```

Parameters

text *(string)*
: Text to be added to the document.

justify *(string)*
: Text justification. Valid values are flushleft, flushright and center.

newLine *(boolean)*
: If True, adds a new line after the text.

style *(string)*
: Style of the text. Valid values are emph, textrm, textsf, texttt, textup, textit, textsl, textsc, textbf, textmd, uline, uwave and sout.

size *(string)*
: Text size. Valid values are HUGE, Huge, LARGE, Large, large, small and tiny.

Return Value

None

Description

This routine adds text to the report. The text contents, justification, style and size are given by *text*, *justify*, *style* and *size*. If *newLine* is True a new line will be added after the text. For example,

```
rep.addText( "ACUSIM Software, Inc.", justify="center" )
```

addTitle()

Add a title to the report.

Usage

`addTitle( title )`

Parameters

`title` *(string)*
Title of the document.

Return Value

None

Description

This routine adds a *`title`* to the report. For example,

```
title = adb.get( "title" )
rep.addTitle("Pump Analysis Performed by AcuSolve\\\\" + title )
```

beginBullet()

Start a new bullet block in the report.

Usage

`beginBullet( )`

Parameters

None

Return Value

None

Description

This routine starts a new bullet block in the report. For example,

```
rep.beginBullet( )
```

beginItemize()

Start a new numbered block in the report.

Usage

`beginItemize( indexType = "" )`

Parameters

`indexType` *(string)*
Type of index. Valid values are arabic, alph, Alph, roman, Roman and fnsymbol.

Return Value

None

Description

This routine starts a new numbered block in the report. The index type is given by *`indexType`*. For example,

```
rep.beginItemize( "roman" )
```

close()

Close the report and file.

Usage

`close( )`

Parameters

None

Return Value

None

Description

This routine closes the report and file. For example,

```
rep.close( )
```

convertUnit()

Convert among physical units.

Usage

`newValue = convertUnit( quantity, fromUnit, toUnit )`

Parameters

`quantity` *(real)*
Physical quantity.

`fromUnit` *(string)*
Source unit.

`toUnit` *(string)*
Destination unit.

Return Value

`newValue` *(real)*
The converted value.

Description

This routine converts a physical quantity from *`fromUnit`* to *`toUnit`*. For example,

```
nodeElm = ROOT + RS +'Model'+ RS +'Volumes'+ RS +'impeller' + RS + 'ELEMENT_SET'
rffName = acs.getRef( 'reference_frame', nodeElm )
nodeRff = ROOT + RS + 'REFERENCE_FRAME' + RS + rffName
rot = acs.getArray( 'angular_velocity', nodeRff )
rotz = abs( rot[2] )
rotz = rep.convertUnit( rotz, "rad/sec", "RPM" )
```

endBullet()

Close a numbered block in the report.

Usage

endBullet( )

Parameters

None

Return Value

None

Description

This routine closes a bullet block in the report. For example,

```
rep.endBullet( )
```

endItemize()

Close a numbered block in the report.

Usage

`endItemize( )`

Parameters

None

Return Value

None

Description

This routine closes a numbered block in the report. For example,

```
rep.endItemize( )
```

fillVSpace()

Push text to the end of the page.

Usage

`fillVSpace( )`

Parameters

None

Return Value

None

Description

This routine pushes text to the end of the page. For example,

```
rep.fillVSpace( )
```

modifyPackageOptions()

Modify predefined package options.

Usage

`modifyPackageOptions( package, optionMap )`

Parameters

`package` *(string)*
: Package name (for example, hypersetup).

`optionMap` *(dictionary)*
: Key-value pairs of options.

Return Value

None

Description

This routine modifies predefined package options. For example,

```
rep.modifyPackageOptions( "hypersetup", optionMap={"pdfborder":"{0 0 0}"} )
```

newPage()

Start a new page and clear the page.

Usage

newPage( )

Parameters

None

Return Value

None

Description

This routine starts a new page and clears the page. For example,

```
rep.newPage( )
```

rawLatex()

Adds an unformatted LaTex text to the report.

Usage

`rawLaTeX( text )`

Parameters

`text` *(string)*
The unformatted LaTex text which should be added.

Return Value

None

Description

This routine adds an unformatted LaTex text to the report. This text is given by *`text`*.

writeHtml()

Convert the report to HTML format.

Usage

```
writeHtml( converter = "tth", credit = False )
```

Parameters

converter *(string)*
: Specifies the converter. Valid values are tth and htlatex.

credit *(boolean)*
: Specifies whether tth credit should be exist in HTML output or not.

Return Value

None

Description

This routine converts the report to HTML format. The converter is given by *converter*. For example,

```
rep.writeHtml( )
```

writePdf()

Convert the report to PDF format.

Usage

`writePdf( )`

Parameters

None

Return Value

None

Description

This routine converts the report to PDF format. For example,

```
rep.writePdf( )
```

writeRft()

Convert the report to RTF format.

Usage

`writeRtf( )`

Parameters

None

Return Value

None

Description

This routine converts the report to RTF format. For example,

```
rep.writeRtf( )
```

> **Note:** Latex2rtf does not create the table of contents automatically.

You need to open the outputted `.rtf` file with a word processor such as Office Word and select all of the document (Ctrl+A) and then press F9 (refresh) to populate the table.

2D Plot

9

This chapter covers the following:

Curve()

Creates a curve with the styles provided.

Usage

```
curve = Curve( x, y,
                        name = None, lineType = "solid",
                        lineWidth = 1, symbol = None,
                        symbolSize= 1, color = "blue" )
```

Parameters

x (array)
: The value of the X data of the curve.

y (array)
: The value of the Y data of the curve.

name (string)
: The curve's name.

lineType (string)
: The curve's line types. Valid values are solid(-), dashed(--), dash-dot(-.) and dotted(:).

lineWidth (integer)
: The curve's line width.

symbol (string)
: The curve's symbol. Valid values are points(.), pixels(,), circle(o), triangleUp(^), triangleDown(v), triangleLeft(<), triangleRight(>), square(s), plus(+), cross(x), diamond(D), thinDiamond(d), tripodDown(1), tripodUp(2), tripodLeft(3), tripodRight(4), hexagon(h), rotatedHexagon(H), pentagon(p), verticalLine(|) and horizontalLine(_).

symbolSize (integer)
: The curve's symbol size.

color (string)
: The curve's color string. Valid values are blue(b), green(g), red(r), cyan(c), magenta(m), yellow(y), black(k) and white(w).

Return Value

***curve** (dictionary)*
: The created curve information.

Description

This routine creates a curve with the styles provided. The curve x and y data are given by x and y. The curve name, line type, line width, symbol, symbol size and color are given by *name*, *lineType*, *lineWidth*, *symbol*, *symbolSize* and *color*, respectively. For example,

```
nOsiVars = adb.get( "nOsiVars" )
presId = -1
```

```
massId = -1
for i in range( nOsiVars ):
   name = adb.get( "osiVarName", i )
   if name == "pressure": presId = i
   if name == "mass_flux": massId = i
nOsfs = adb.get( "nOsfs" )
inletId = -1
for i in range(nOsfs):
   name = adb.get( "osfName", i )
   if name == "inflow": inletId = i
pres = adb.get( "osiValues", inletId, presId )
mass = adb.get( "osiValues", inletId, massId )
mpCurve = Curve ( pres, mass, name = "Mass Flux",
                  color = "Red" )
```

Plot2D()

Creates an x-y plot from a set of curves.

Usage

```
fileName = Plot2D( curves, title = "",
                           legend = True, legendPos = "auto",
                           xLabel = "X", xLog = False,
                           xRange = "auto", yLabel = "Y",
                           yLog = False, yRange = "auto",
                           width = 600, height = 400,
                           fileName = None, fileType = "png",
                           dirName = None )
```

Parameters

curves (list)
: List of lists, each child list is a curve.

title (string)
: Plot title.

legend (boolean)
: If True, creates a legend for the lines in the plot.

legendPos (string)
: Legend position in the plot. Valid values are auto, best(0), upper right (1), upper left(2), lower left(3), lower right(4), right(5), center left(6), center right(7), lower center(8), upper center(9), center (10).

xLabel (string)
: The X-axis label.

xLog (boolean)
: The X-axis scale Linear/Log. If True, the scale will be Log.

xRange ((string) or (list))
: The X value range. Valid values are auto, [min,max] where min/max are float numbers.

yLabel (string)
: The Y-axis label.

yLog (boolean)
: The Y-axis Linear/Log. If True, the scale will be Log.

yRange (string or list)
: The Y value range. Valid values are auto, [min,max] where min/max are float numbers.

width (integer)
: Plot width.

height (integer)
: Plot height.

fileName (string)
The name of file to be saved.

fileType (string)
Type of the image file to be saved.

dirName (string)
Directory of output file.

Return Value

***fileName** (string)*
Name of the saved file.

Description

This routine creates an x-y plot from a set of curves. The curves data is given by curves. The plot *title*, *xLabel*, *yLabel*, *width*, *height*, *xRange* and *yRange* are given by *title*, *xLabel*, *yLabel*, *width*, *height*, *xRange* and *yRange*, respectively. If legend is True, a legend will be created for the lines in the plot. The legend position is given by *legendPos*. The plot will be saved in a file. The file name, file type and directory name which the image will be placed in are given by *fileName*, *fileType* and *dirName*. For example,

```
fname = Plot2D( mpCurve, width=600, height=400,
                         xLabel = "Pressure Drop (Pa)",
                         yLabel = "Mass Flux (kg/sec)" )
```

Where *mpCurve* is returned from the Curve function you had as an example in the Curve section.

ReportAcs

10

This module defines methods that get data from the database and adds them to the report (`.tex`) file and are used by the `.rep` file.

This chapter covers the following:

repAcs()

Create a RepAcs instance.

Usage

```
repAcs = ReportAcs( report = None, acs = None)
```

Parameters

report (object)
: Report object.

acs (object)
: Database object.

Return Value

***repAcs** (object)*
: ReportAcs object.

Errors

The report and database objects must be created before.

Description

This routine creates a RepAcs instance. For example,

```
rep = Report( "pump.tex", packages=("graphicx", "hyperref") )
acs = Acs( )
repAcs = ReportAcs( rep, acs )
```

addMaterialModel()

Write out the material model data in the report document.

Usage

repAcs.addMaterialModel(name)

Parameters

name (string)
The material model name. Valid values are Air, Aluminum and Water.

Return Value

None

Errors

The *name* should be valid.

Description

This routine writes out the material model data in the report document. The material model name that its data should be written is given by *name*. For example,

```
nodeElm = ROOT + RS +'Model'+ RS +'Volumes'+ RS +'impeller' + RS + 'ELEMENT_SET'
matName = acs.getRef( 'material_model', nodeElm )
repAcs.addMaterialModel( matName )
```

addSimpleBC()

Write out the simple boundary condition data in the report document.

Usage

`repAcs.addSimpleBC( name )`

Parameters

`name` *(string)*
The simple BC model name.

Return Value

None

Errors

The *`name`* should be valid.

Description

This routine writes out the simple boundary condition data in the report document. The model name that its data should be written is given by *`name`*. For example,

```
repAcs.addMaterialModel( "wall" )
```

getPrbDesc()

Get the problem description and equations.

Usage

```
prbData = repAcs.getPrbDesc( )
```

Parameters

None

Return Value

prbData *(list)*
Problem description info which is (mode, flow, temp, rad, spec, nSpecs, turb, mesh, ext).

Errors

None

Description

This routine get the problem description and equations. For example,

```
(mode,flow,temp,rad,spec,nSpecs,turb,mesh,ext) = repAcs.getPrbDesc( )
```

AcuSolve Database Access (Adb)

11

This chapter covers the following:

Adb

Create an AcuDbAssist instance.

Usage

adb = Adb(problemName = None, dirName = None, runId = None)

Parameters

problemName *(string)*
: Name of the problem.

dirName *(string)*
: Name of the working directory.

runId *(integer)*
: Run number.

Return Value

None

Errors

The working directory must exist and have the required information.

Description

This routine creates an AcuDbAssist instance, which is an interface to the acudb module for opening and extracting the last AcuSolve solution database, and transferring its data to the Report and plotting them. The adb problem is given by *problemName* and the working directory that includes the information of adb is given by *dirName*. For example,

```
adb = Adb( problemName = "pump",
                     dirName = "ACUSIM.DIR",
                     runId = 1 )
```

get ()

Return an Acudb value.

Usage

```
value = adb.get( name, index1 = None,
                           index2 = None, index3 = None )
```

Parameters

name (string)
: Name of the data.

index1 (integer)
: Index 1 of the data.

index2 (integer)
: Index 2 of the data.

index3 (integer)
: Index 3 of the data.

Return Value

***value** (depends on the variable)*
: Acudb value

Errors

Index1, 2 ,3 must have proper values.

Description

This routine returns an acudb value. The adb variable name is given by *name*. For example,

```
OsiVars = adb.get( "nOsiVars" )
for i in range( nOsiVars ):
name = adb.get( "osiVarName", i )
```

getCpuTimes()

Return the "Run Data" CPU Time value.

Usage

```
cputimes = adb.getCpuTimes( )
```

Parameters

None

Return Value

cpuTimes *(array numarray)*
List of CPU Time values

Errors

None

Description

This routine returns the "Run data" CPU Times values. For example,

```
cpuTimes = adb.getCpuTimes( )
```

getElapseTimes()

Return the "Run Data" Elapsed Time value.

Usage

```
elpstimes = adb.getElapseTimes( )
```

Parameters

None

Return Value

elpsTimes *(array numarray)*
List of Elapsed Time values

Errors

None

Description

This routine returns the "Run data" Elapsed Times values. For example,

```
elpsTimes = adb.getElapseTimes( )
```

getLinIterData()

Return the "Linear Iterations" steps, times and values data.

Usage

```
linIterData = adb.getLinIterData( var, index = 0 )
```

Parameters

`var` (string) or (integer)
The "Linear Iterations" variable name/index.

`index` (integer)
The index of nIters.

Return Value

***`linIterData`** (tuple)*
The "Linear Iterations" data including steps, times and values

Errors

`var` must be a valid name or index.

`Index` must have proper value.

Description

This routine returns the "Linear Iterations" steps, times and values data. The variable name or index is given by `var`. For example,

```
linData = adb.getLinIterData( 1,2)
```

getLinIterSteps()

Return the "Linear Iterations" steps value.

Usage

```
linIterSteps = adb.getLinIterSteps( var, index = 0 )
```

Parameters

var (string) or (integer)
The "Linear Iterations" variable name/index.

index (integer)
The "Linear Iterations" index.

Return Value

***linIterSteps** (array numarray)*
The "Linear Iterations" steps values

Errors

var must be a valid name or index.

Index must have proper value.

Description

This routine returns the "Linear Iterations" steps value. The variable name or index is given by *var*. For example,

```
linSteps = adb.getLinIterSteps( 1,2 )
```

getLinIterTimes()

Return the "Linear Iterations" times value.

Usage

```
linIterTimes = adb.getLinIterTimes( var, index = 0 )
```

Parameters

var (string or integer)
The "Linear Iterations" variable name/index.

index (integer)
The "Linear Iterations" index.

Return Value

***linIterTimes** (array numarray)*
The "Linear Iterations" times values.

Errors

var must be a valid name or index.

index must have proper value.

Description

This routine returns the "Linear Iterations" times value. The variable name or index is given by *var*. For example,

```
linTimes = adb.getLinIterTimes( 1,2 )
```

getLinIterValues()

Return the "Linear Iterations" values.

Usage

```
linIterValues = adb.getLinIterValues( var, index = 0 )
```

Parameters

`var` (string) or (integer)
The "Linear Iterations" variable name/index.

`index` (integer)
The "Linear Iterations" index.

Return Value

***`linIterValues`** (array numarray)*
The "Linear Iterations" values

Errors

`var` must be a valid name or index

`Index` must have proper value.

Description

This routine returns the "Linear Iterations" values. The variable name or index is given by *`var`*. For example,

```
linValues = adb.getLinIterValues( 1,2 )
```

getLinIterVarIndx()

Map the "Linear Iterations" variable into an index.

Usage

```
varIndx = adb.getLinIterVarIndx( var )
```

Parameters

`var` (string) or (integer)
The "Linear Iterations" variable name/index.

Return Value

***`varIndx`** (integer)*
The mapped index

Errors

`var` must be a valid name or index

Description

This routine maps the "Linear Iterations" variable into an index. The variable name or index is given by *`var`*. For example,

```
varIndx = adb.getLinIterVarIndx( 1 )
```

getLinIterVarNames()

Returns a list of "Linear Iterations" variable names.

Usage

```
varNames = adb.getLinIterVarNames( )
```

Parameters

None

Return Value

varNames *(list)*
A list of "Linear Iterations" variable names.

Errors

None

Description

This routine returns a list of "Linear Iterations" variable names. For example,

```
varNames = adb.getLinIterVarNames( )
```

getOeiNameIndx()

Map the OEI name into an index.

Usage

`oeiNameIndx = adb.getOeiNameIndx( name )`

Parameters

`name` *(string) or (integer)*
The OEI name/index.

Return Value

`oeiNameIndx` *(integer)*
The mapped index.

Errors

`name` must be a valid name or index.

Description

This routine maps the OEI name into an index. The OEI name or index is given by *`name`*. For example,

```
oeiNameIndx = adb.getOeiNameIndx( 1 )
```

getOeiNames()

Return the list of OEI names.

Usage

```
oeiNames = adb.getOeiNames( )
```

Parameters

None

Return Value

oeiNames *(list)*
List of OEI names.

Description

This routine returns the list of OEI names. For example,

```
oeiNames = adb.getOeiNames( )
```

getOeiSteps()

Return the list of OEI time steps.

Usage

`oeiSteps = adb.getOeiSteps( name )`

Parameters

`name` *(integer) or (string)*
Name or ID of an OEI name.

Return Value

`oeiSteps` *(list)*
List of OEI time steps.

Description

This routine returns the list of OEI time steps. This function needs an index as argument which is given by *`name`*. For example,

```
oeiSteps = adb.getOeiSteps( 1 )
```

getOeiTimes()

Return OEI run times.

Usage

`oeiTimes = adb.getOeiTimes( name )`

Parameters

`name` *(integer) or (string)*
Name or ID of an OEI name.

Return Value

`oeiTimes` *(list)*
List of OEI run times.

Description

This routine returns the list of OEI run times. This function needs an index as argument which is given by *`name`*. For example,

```
oeiTimes = adb.getOeiTimes( 3 )
```

getOeiValues()

Return the list of OEI integrated values.

Usage

```
oeiVals = adb.getOeiValues( name, var, unit = None )
```

Parameters

name (integer) or (string)
Name or ID of an OEI name.

var (integer) or (string)
Name or ID of an OEI variable.

unit (string)
If unit is not None, the value will be converted to the new unit and be returned.

Return Value

***oeiVals** (list)*
List of OEI integrated values.

Description

This routine returns the list of OEI integrated values. This function needs two indices as argument which are given by *name* and *var*. If *unit* is not None, the value will be converted to the new unit and be returned. For example,

```
trac = adb.getOeiValues( "wall", "traction", "N" )
```

getOeiVarNames()

Return the list of OEI variable names.

Usage

`oeiVarNames = adb.getOeiVarNames( )`

Parameters

None

Return Value

`oeiVarNames` *(list)*
List of OEI variable names.

Description

This routine returns the list of OEI variable names. For example,

```
oeiVars = adb.getOeiVarNames( )
```

getOeiVarUnit()

Return the EI unit (default unit of unit category) of adb variable.

Usage

`oeiVarUnit = adb.getOeiVarUnit( name )`

Parameters

`name` *(integer) or (string)*
Name or ID of an OEI name.

Return Value

`oeiVarUnit` *(list) or (string)*
The EI unit (default unit of unit category) of adb variable.

Description

This routine returns the EI unit (default unit of unit category) of adb variable. This function needs an index as argument which is given by *`name`*. For example,

```
oeiVarUnit = adb.getOeiVarUnit( "wall" )
```

getOfcNameIndx()

Map the OFC name into an index.

Usage

`ofcNameIndx = adb.getOfcNameIndx( name )`

Parameters

`name` *(integer) or (string)*
The OFC name/index.

Return Value

`ofcNameIndx` *(integer)*
The mapped index.

Errors

`name` must be a valid name or index.

Description

This routine maps the OFC name into an index. The OFC name or index is given by *`name`*. For example,

```
ofcNameIndx = adb.getOfcNameIndx( 1 )
```

getOfcNames()

Return the list of OFC names.

Usage

```
ofcNames = adb.getOfcNames( )
```

Parameters

None

Return Value

ofcNames *(list)*
List of OFC names.

Description

This routine returns the list of OFC names. For example,

```
ofcNames = adb.getOfcNames( )
```

getOfcSteps()

Return the list of OFC time steps.

Usage

```
ofcSteps = adb.getOfcSteps( name )
```

Parameters

name (integer) or (string)
Name or ID of an OFC name.

Return Value

***ofcSteps** (list)*
List of OFC time steps.

Description

This routine returns the list of OFC time steps. This function needs an index as argument which is given by *name*. For example,

```
ofcSteps = adb.getOfcSteps( 1 )
```

getOfcTimes()

Return OFC run times.

Usage

`ofcTimes = adb.getOfcTimes( name )`

Parameters

`name` *(integer) or (string)*
Name or ID of an OFC name.

Return Value

`ofcTimes` *(list)*
List of OFC run times.

Description

This routine returns the list of OFC run times. This function needs an index as argument which is given by *`name`*. For example,

```
ofcTimes = adb.getOfcTimes( 3 )
```

getOfcValues()

Return the list of OFC integrated values.

Usage

```
ofcVals = adb.getOfcValues( name, var, unit = None )
```

Parameters

name (integer) or (string)
Name or ID of an OFC name.

var (integer) or (string)
Name or ID of an OFC variable.

unit (string)
If unit is not None, the value will be converted to the new unit and be returned.

Return Value

***ofcVals** (list)*
List of OFC integrated values.

Description

This routine returns the list of OFC integrated values. This function needs two indices as argument which are given by *name* and *var*. If *unit* is not None, the value will be converted to the new unit and be returned. For example,

```
trac = adb.getOfcValues( "wall", "traction", "N" )
```

getOfcVarNames()

Return the list of OFC variable names.

Usage

```
ofcVarNames = adb.getOfcVarNames( )
```

Parameters

None

Return Value

ofcVarNames *(list)*
List of OFC variable names.

Description

This routine returns the list of OFC variable names. For example,

```
ofcVars = adb.getOfcVarNames( )
```

getOfcVarUnit()

Return the FC unit (default unit of unit category) of adb variable.

Usage

```
ofcVarUnit = adb.getOfcVarUnit( name )
```

Parameters

name (integer) or (string)
Name or ID of an OFC name.

Return Value

***ofcVarUnit** (list) or (string)*
The FC unit (default unit of unit category) of adb variable.

Description

This routine returns the FC unit (default unit of unit category) of adb variable. This function needs an index as argument which is given by *name*. For example,

```
ofcVarUnit = adb.getOfcVarUnit( "wall" )
```

getOhcNameIndx()

Map the OHC name into an index.

Usage

`ohcNameIndx = adb.getOhcNameIndx( name )`

Parameters

`name` *(integer) or (string)*
The OHC name/index.

Return Value

`ohcNameIndx` *(integer)*
The mapped index.

Errors

`name` must be a valid name or index.

Description

This routine maps the OHC name into an index. The OHC name or index is given by *`name`*. For example,

```
ohcNameIndx = adb.getOhcNameIndx( 1 )
```

getOhcNames ()

Return the list of OHC names.

Usage

```
ohcNames = adb.getOhcNames( )
```

Parameters

None

Return Value

ohcNames *(list)*
List of OHC names.

Description

This routine returns the list of OHC names. For example,

```
ohcNames = adb.getOhcNames( )
```

getOhcSteps()

Return the list of OHC time steps.

Usage

```
ohcSteps = adb.getOhcSteps( name )
```

Parameters

name or id of an OHC name (integer) or (string)
Name or ID of an OHC name.

Return Value

ohcSteps (list)
List of OHC time steps.

Description

This routine returns the list of OHC time steps. This function needs an index as argument which is given by *name*. For example,

```
ohcSteps = adb.getOhcSteps( 1 )
```

getOhcTimes()

Return OHC run times.

Usage

`ohcTimes = adb.getOhcTimes( name )`

Parameters

`name` *(integer) or (string)*
Name or ID of an OHC name.

Return Value

`ohcTimes` *(list)*
List of OHC run times.

Description

This routine returns the list of OHC run times. This function needs an index as argument which is given by *`name`*. For example,

```
ohcTimes = adb.getOhcTimes( 3 )
```

getOhcValues()

Return the list of OHC integrated values.

Usage

`ohcVals = adb.getOhcValues( name, var, unit = None )`

Parameters

`name` ***(integer) or (string)***
Name or ID of an OHC name.

`var` ***(integer) or (string)***
Name or ID of an OHC variable.

`unit` ***(string)***
If unit is not None, the value will be converted to the new unit and be returned.

Return Value

`ohcVals` ***(list)***
List of OHC integrated values.

Description

This routine returns the list of OHC integrated values. This function needs two indices as argument which are given by *`name`* and *`var`*. If *`unit`* is not None, the value will be converted to the new unit and be returned. For example,

```
trac = adb.getOhcValues( "wall", "traction", "N" )
```

getOhcVarNames()

Return the list of OHC variable names.

Usage

```
ohcVarNames = adb.getOhcVarNames( )
```

Parameters

None

Return Value

ohcVarNames *(list)*
List of OHC variable names.

Description

This routine returns the list of OHC variable names. For example,

```
ohcVars = adb.getOhcVarNames( )
```

getOhcVarUnit()

Return the HC unit (default unit of unit category) of adb variable.

Usage

```
ohcVarUnit = adb.getOhcVarUnit( name )
```

Parameters

name (integer) or (string)
Name or ID of an OHC name.

Return Value

***ohcVarUnit** (list) or (string)*
The HC unit (default unit of unit category) of adb variable.

Description

This routine returns the HC unit (default unit of unit category) of adb variable. This function needs an index as argument which is given by *name*. For example,

```
ohcVarUnit = adb.getOhcVarUnit( "wall" )
```

getOqiNameIndx()

Map the OQI name into an index.

Usage

`oqiNameIndx = adb.getOqiNameIndx( name )`

Parameters

`name` *(integer) or (string)*
Name OQI name/index.

Return Value

`oqiNameIndx` *(integer)*
The mapped index.

Errors

`name` must be a valid name or index.

Description

This routine maps the OQI name into an index. The OQI name or index is given by *`name`*. For example,

```
oqiNameIndx = adb.getOqiNameIndx( 1 )
```

getOqiNames()

Return the list of OQI names.

Usage

```
oqiNames = adb.getOqiNames( )
```

Parameters

None

Return Value

oqiNames *(list)*
List of OQI names.

Description

This routine returns the list of OQI names. For example,

```
oqiNames = adb.getOqiNames( )
```

getOqiSteps()

Return the list of OQI time steps.

Usage

`oqiSteps = adb.getOqiSteps( name )`

Parameters

`name` *(integer) or (string)*
Name or ID of an OQI name.

Return Value

`oqiSteps` *(list)*
List of OQI time steps.

Description

This routine returns the list of OQI time steps. This function needs an index as argument which is given by *`name`*. For example,

```
oqiSteps = adb.getOqiSteps( 1 )
```

getOqiTimes()

Return OQI run times.

Usage

`oqiTimes = adb.getOqiTimes( name )`

Parameters

`name` *(integer) or (string)*
Name or ID of an OQI name.

Return Value

`oqiTimes` *(string)*
List of OQI run times.

Description

This routine returns the list of OQI run times. This function needs an index as argument which is given by *`name`*. For example,

```
oqiTimes = adb.getOqiTimes( 3 )
```

getOqiValues()

Return the list of OQI integrated values.

Usage

```
oqiVals = adb.getOqiValues( name, var, unit = None )
```

Parameters

name (integer) or (string)
Name or ID of an OQI name.

var (integer) or (string)
Name or ID of an OQI variable.

unit (string)
If unit is not None, the value will be converted to the new unit and be returned.

Return Value

***oqiVals** (list)*
List of OQI integrated values.

Description

This routine returns the list of OQI integrated values. This function needs two indices as argument which are given by *name* and *var*. If *unit* is not None, the value will be converted to the new unit and be returned. For example,

```
trac = adb.getOqiValues( "wall", "traction", "N" )
```

getOqiVarNames()

Return the list of OQI variable names.

Usage

```
oqiVarNames = adb.getOqiVarNames( )
```

Parameters

None

Return Value

oqiVarNames *(list)*
List of OQI variable names.

Description

This routine returns the list of OQI variable names. For example,

```
oqiVars = adb.getOqiVarNames( )
```

getOqiVarUnit()

Return the QI unit (default unit of unit category) of adb variable.

Usage

```
oqiVarUnit = adb.getOqiVarUnit( name )
```

Parameters

name (integer) or (string)
Name or ID of an OQI name.

Return Value

***oqiVarUnit** (list) or (string)*
The QI unit (default unit of unit category) of adb variable.

Description

This routine returns the QI unit (default unit of unit category) of adb variable. This function needs an index as argument which is given by *name*. For example,

```
oqiVarUnit = adb.getOqiVarUnit( "wall" )
```

getOriNameIndx()

Map the ORI name into an index.

Usage

```
oriNameIndx = adb.getOriNameIndx( name )
```

Parameters

name *(string) or (integer)*
The ORI name/index.

Return Value

oriNameIndx *(integer)*
The mapped index.

Errors

name must be a valid name or index.

Description

This routine maps the ORI name into an index. The ORI name or index is given by *name*. For example,

```
oriNameIndx = adb.getOriNameIndx( 1 )
```

getOriNames()

Return the list of ORI names.

Usage

```
oriNames = adb.getOriNames( )
```

Parameters

None

Return Value

oriNames *(list)*
List of ORI names.

Errors

name must be a valid name or index.

Description

This routine returns the list of ORI names. For example,

```
oriNames = adb.getOriNames( )
```

getOriSteps()

Return the list of ORI time steps.

Usage

`oriSteps = adb.getOriSteps( name )`

Parameters

`name` *(integer) or (string)*
Name or ID of an ORI name.

Return Value

`oriSteps` *(list)*
List of ORI time steps.

Description

This routine returns the list of ORI time steps. This function needs an index as argument which is given by *`name`*. For example,

```
oriSteps = adb.getOriSteps( 1 )
```

getOriTimes()

Return ORI run times.

Usage

`oriTimes = adb.getOriTimes( name )`

Parameters

`name` *(integer) or (string)*
Name or ID of an ORI name.

Return Value

`oriTimes` *(list)*
List of ORI time steps.

Description

This routine returns the list of ORI run times. This function needs an index as argument which is given by *`name`*. For example,

```
oriTimes = adb.getOriTimes( 3 )
```

getOriValues()

Return the list of ORI integrated values.

Usage

```
oriVals = adb.getOriValues( name, var, unit = None )
```

Parameters

name (integer) or (string)
: Name or ID of an ORI name.

var (integer) or (string)
: Name or ID of an ORI variable.

unit (string)
: If unit is not None, the value will be converted to the new unit and be returned.

Return Value

***oriVals** (list)*
: List of ORI integrated values.

Description

This routine returns the list of ORI integrated values. This function needs two indices as argument which are given by *name* and *var*. If *unit* is not None, the value will be converted to the new unit and be returned. For example,

```
trac = adb.getOriValues( "wall", "traction", "N" )
```

getOriVarNames()

Return the list of ORI variable names.

Usage

```
oriVarNames = adb.getOriVarNames( )
```

Parameters

None

Return Value

oriVarNames *(list)*
List of ORI variable names.

Description

This routine returns the list of ORI variable names. For example,

```
oriVars = adb.getOriVarNames( )
```

getOriVarUnit()

Return the RI unit (default unit of unit category) of adb variable.

Usage

```
oriVarUnit = adb.getOriVarUnit( name
```

Parameters

name (integer) or (string)
Name or ID of an ORI name.

Return Value

***oriVarUnit** (list) or (string)*
The RI unit (default unit of unit category) of adb variable.

Description

This routine returns the RI unit (default unit of unit category) of adb variable. This function needs an index as argument which is given by *name*. For example,

```
oriVarUnit = adb.getOriVarUnit( "wall" )
```

getOsiNameIndx()

Map the OSI name into an index.

Usage

`osiNameIndx = adb.getOsiNameIndx( name )`

Parameters

`name` *(string) or (integer)*
The OSI name/index.

Return Value

`osiNameIndx` *(integer)*
The mapped index.

Errors

`name` must be a valid name or index.

Description

This routine maps the OSI name into an index. The OSI name or index is given by *`name`*. For example,

```
osiNameIndx = adb.getOsiNameIndx( 1 )
```

getOsiNames()

Return the list of OSI names.

Usage

```
osiNames = adb.getOsiNames()
```

Parameters

None

Return Value

osiNames *(list)*
List of OSI names.

Description

This routine returns the list of OSI names. For example,

```
osiNames = adb.getOsiNames( )
```

getOsiSteps()

Return the list of OSI time steps.

Usage

```
osiSteps = adb.getOsiSteps( name )
```

Parameters

name (integer) or (string)
Name or ID of an OSI name.

Return Value

***osiSteps** (list)*
List of OSI time steps.

Description

This routine returns the list of OSI time steps. This function needs an index as argument which is given by *name*. For example,

```
osiSteps = adb.getOsiSteps( 1 )
```

getOsiTimes()

Return OSI run times.

Usage

`osiTimes = adb.getOsiTimes( name )`

Parameters

`name` *(integer) or (string)*
Name or ID of an OSI name.

Return Value

`osiTimes` *(list)*
List of OSI run times.

Description

This routine returns the list of OSI run times. This function needs an index as argument which is given by *`name`*. For example,

```
osiTimes = adb.getOsiTimes( 3 )
```

getOsiValues()

Return the list of OSI integrated values.

Usage

```
osiVals = adb.getOsiValues( name, var, unit = None )
```

Parameters

name (integer) or (string)
: Name or ID of an OSI name.

var (integer) or (string)
: Name or ID of an OSI variable.

unit (string)
: If unit is not None, the value will be converted to the new unit and be returned.

Return Value

***osiTimes** (list)*
: List of OSI run times.

Description

This routine returns the list of OSI integrated values. This function needs two indices as argument which are given by *name* and *var*. If *unit* is not None, the value will be converted to the new unit and be returned. For example,

```
trac = adb.getOsiValues( "wall", "traction", "N" )
```

getOsiVarNames()

Return the list of OSI variable names.

Usage

```
osiVarNames = adb.getOsiVarNames( )
```

Parameters

None

Return Value

osiVarNames *(list)*
List of OSI variable names.

Description

This routine returns the list of OSI variable names. For example,

```
osiVars = adb.getOsiVarNames( )
```

getOsiVarUnit()

Return the SI unit (default unit of unit category) of adb variable.

Usage

```
osiVarNames = adb.getOsiVarNames( )
```

Parameters

name (integer) or (string)
Name or ID of an OSI name.

Return Value

***osiVarUnit** (list) or (string)*
The SI unit (default unit of unit category) of adb variable.

Description

This routine returns the SI unit (default unit of unit category) of adb variable. This function needs an index as argument which is given by *name*. For example,

```
osiVarUnit = adb.getOsiVarUnit( "wall" )
```

getOthNameIndx()

Map the OTH name into an index.

Usage

```
othNameIndx = adb.getOthNameIndx( name )
```

Parameters

name *(string) or (integer)*
The OTH name/index.

Return Value

othNameIndx *(integer)*
The mapped index.

Errors

name must be a valid name or index.

Description

This routine maps the OTH name into an index. The OTH name or index is given by name. For example,

```
othNameIndx = adb.getOthNameIndx( 1 )
```

getOthNames()

Return the list of OTH names.

Usage

```
othNames = adb.getOthNames( )
```

Parameters

None

Return Value

othNames *(list)*
List of OTH names.

Description

This routine returns the list of OTH names. For example,

```
othNames = adb.getOthNames( )
```

getOthNodes()

Return a list of OTH nodes.

Usage

```
othNodes = adb.getOthNodes( )
```

Parameters

None

Return Value

othNodes *(list)*
List of OTH nodes.

Description

This routine returns the list of OTH nodes. For example,

```
othNodes = adb.getOthNodes( )
```

getOthSteps()

Return the list of OTH time steps.

Usage

```
othSteps = adb.getOthSteps( name )
```

Parameters

name (integer) or (string)
Name or ID of an OTH name.

Return Value

***othSteps** (list)*
List of OTH time steps.

Description

This routine returns the list of OTH time steps. This function needs an index as argument which is given by *name*. For example,

```
othSteps = adb.getOthSteps( 1 )
```

getOthTimes()

Return OTH run times.

Usage

```
othTimes = adb.getOthTimes( name )
```

Parameters

name (integer) or (string)
Name or ID of an OTH name.

Return Value

***othTimes** (list)*
List of OTH run times.

Description

This routine returns the list of OTH run times. This function needs an index as argument which is given by *name*. For example,

```
othTimes = adb.getOthTimes( 3 )
```

getOthValues()

Return the list of OTH integrated values.

Usage

```
othVals = adb.getOthValues( name, node, var, unit = None )
```

Parameters

name (integer) or (string)
: Name or ID of an OTH name.

node (integer) or (string)
: Name or index of an OTH node.

var (integer) or (string)
: Name or ID of an OTH variable.

unit (string)
: If unit is not None, the value will be converted to the new unit and be returned.

Return Value

***othVals** (list)*
: List of OTH integrated values.

Description

This routine returns the list of OTH integrated values. This function needs three indices as argument which are given by *name*, *node* and *var*. If *unit* is not None, the value will be converted to the new unit and be returned. For example,

```
othVals = adb.getOthValues( 0, 1, 3 )
```

getOthVarNames()

Return the list of OTH variable names.

Usage

```
othVarNames = adb.getOthVarNames( )
```

Parameters

None

Return Value

`othVarNames` *(list)*
List of OTH variable names.

Description

This routine returns the list of OTH variable names. For example,

```
othVars = adb.getOthVarNames( )
```

getOthVarUnit()

Return the TH unit (default unit of unit category) of adb variable.

Usage

```
othVarUnit = adb.getOthVarUnit( name )
```

Parameters

name (integer) or (string)
Name or ID of an OTH name.

Return Value

***othVarUnit** (list) or (string)*
The TH unit (default unit of unit category) of adb variable.

Description

This routine returns the TH unit (default unit of unit category) of adb variable. This function needs an index as argument which is given by *name*. For example,

```
othVarUnit = adb.getOthVarUnit( "wall" )
```

getResRatioData()

Return the "Residual Ratio" steps, times and values data.

Usage

```
resRatioData = adb.getResRatioData( var, type = 'all' )
```

Parameters

var (integer) or (string)
The "Residual Ratio" variable name/index.

type (string)
The type value; 'all', 'initial' and 'final'.

Return Value

***resRatioData** (tuple of arrays)*
The "Residual Ratio" data.

Errors

var must be a valid name or index.

type must be 'all', 'initial' or 'final'.

Description

This routine returns the "Residual Ratio" steps, times and values data. The "Residual Ratio" name or index is given by *var*. For example,

```
resRatioData = adb.getResRatioData( 'velocity' )
```

getResRatioSteps()

Return the "Residual Ratio" steps value.

Usage

```
resRatioSteps = adb.getResRatioSteps( var, type = 'all' )
```

Parameters

var (integer) or (string)
The "Residual Ratio" variable name/index.

type (string)
The type value; 'all', 'initial' and 'final'.

Return Value

***resRatioSteps** (numarray)*
The "Residual Ratio" step values.

Errors

var must be a valid name or index.

type must be 'all', 'initial' or 'final'.

Description

This routine returns the "Residual Ratio" steps value. The "Residual Ratio" name or index is given by *var*. For example,

```
resRatioSteps = adb.getResRatioSteps( 'velocity' )
```

getResRatioTimes()

Return the "Residual Ratio" times value.

Usage

```
resRatioTimes = adb.getResRatioTimes( var, type = 'all' )
```

Parameters

var (string) or (integer)
The "Residual Ratio" variable name/index.

type (string)
The type value; 'all', 'initial' and 'final'.

Return Value

***resRatioTimes** (numarray)*
The "Residual Ratio" time values.

Errors

var must be a valid name or index.

type must be 'all', 'initial' or 'final'.

Description

This routine returns the "Residual Ratio" times value. The "Residual Ratio" name or index is given by *var*. For example,

```
resRatioTimes = adb.getResRatioTimes( 'velocity' )
```

getResRatioValues()

Return the "Residual Ratio" values.

Usage

```
resRatioValues = adb.getResRatioValues( var, type = 'all', unit = None )
```

Parameters

var (string) or (integer)
The "Residual Ratio" variable name/index.

type (string)
The type value; 'all', 'initial' and 'final'.

unit (string)
The "Residual Ratio" variable unit.

Return Value

***resRatioValues** (numarray)*
The "Residual Ratio" values.

Errors

var must be a valid name or index.

type must be 'all', 'initial' or 'final'.

Description

This routine returns the "Residual Ratio" values. The "Residual Ratio" name or index is given by *var*. For example,

```
resRatioValues = adb.getResRatioValues( 'velocity' )
```

getResRatioVarIndx()

Map the "Residual Ratio" variable into an index.

Usage

`resRatioVarIndx = adb.getResRatioVarIndx( var )`

Parameters

`var` *(string) or (integer)*
The "Residual Ratio" variable name/index.

Return Value

`resRatioVarIndx` *(integer)*
The mapped index.

Errors

`var` must be a valid name or index.

Description

This routine returns "Residual Ratio" variable into an index. The "Residual Ratio" name or index is given by *`var`*. For example,

```
resRatioVarIdx = adb.getResRatioVarIndx( 'velocity' )
```

getResRatioVarNames()

Return a list of "Residual Ratio" variable names.

Usage

```
resRatioVarNames = adb.getResRatioVarNames( )
```

Parameters

None

Return Value

resRatioVarNames *(list)*
A list of "Residual Ratio" variable names.

Description

This routine returns a list of "Residual Ratio" variable names. For example,

```
resRatioVarNames = adb.getResRatioVarNames( )
```

getResRatioVarUnit()

Return the SI unit of the "Residual Ratio" variable.

Usage

```
resRatioVarUnit = adb.getResRatioVarUnit( name )
```

Parameters

name (string)
The "Residual Ratio" variable name.

Return Value

***resRatioVarUnit** (string)*
The SI unit of the variable.

Errors

name must be a valid name.

Description

This routine returns the SI unit of the "Residual Ratio" variable. The "Residual Ratio" name is given by *name*. For example,

```
resRatioVarUnit = adb.getResRatioVarUnit( 'velocity' )
```

getSolRatioData()

Return the "Solution Ratio" steps, times and values data.

Usage

```
solRatioData = adb.getSolRatioData( var, type = 'all' )
```

Parameters

var (string) or (integer)
The "Solution Ratio" variable name/index.

type (string)
The type value; 'all', 'initial' and 'final'.

Return Value

***solRatioData** (tuple of arrays)*
The "Solution Ratio" data.

Errors

var must be a valid name or index.

type must be 'all', 'initial' or 'final'.

Description

This routine returns the "Solution Ratio" steps, times and values data. The "Solution Ratio" name or index is given by *var*. For example,

```
solRatioData = adb.getSolRatioData( 'pressure' )
```

getSolRatioSteps()

Return the "Solution Ratio" steps value.

Usage

```
solRatioSteps = adb.getSolRatioSteps( var, type = 'all' )
```

Parameters

var (string) or (integer)
The "Solution Ratio" variable name/index.

type (string)
The type value; 'all', 'initial' and 'final'.

Return Value

***solRatioSteps** (numarray)*
The "Solution Ratio" step values.

Errors

var must be a valid name or index.

type must be 'all', 'initial' or 'final'.

Description

This routine returns the "Solution Ratio" steps value. The "Solution Ratio" name or index is given by *var*. For example,

```
solRatioSteps = adb.getSolRatioSteps( 'pressure' )
```

getSolRatioTimes()

Return the "Solution Ratio" times value.

Usage

```
solRatioTimes = adb.getSolRatioTimes( var, type = 'all' )
```

Parameters

var (string) or (integer)
The "Solution Ratio" variable name/index.

type (string)
The type value; 'all', 'initial' and 'final'.

Return Value

***solRatioTimes** (numarray)*
The "Solution Ratio" time values.

Errors

var must be a valid name or index.

type must be 'all', 'initial' or 'final'.

Description

This routine returns the "Solution Ratio" times value. The "Solution Ratio" name or index is given by *var*. For example,

```
solRatioTimes = adb.getSolRatioTimes( 'pressure' )
```

getSolRatioValues()

Return the "Solution Ratio" values.

Usage

```
solRatioValues = adb.getSolRatioValues( var, type = 'all', unit = None )
```

Parameters

var (string) or (integer)
The "Solution Ratio" variable name/index.

type (string)
The type value; 'all', 'initial' and 'final'.

unit (string)
The "Solution Ratio" variable unit.

Return Value

***solRatioValues** (numarray)*
The "Solution Ratio" values.

Errors

var must be a valid name or index.

type must be 'all', 'initial' or 'final'.

Description

This routine returns the "Solution Ratio" values. The "Solution Ratio" name or index is given by *var*. For example,

```
solRatioValues = adb.getSolRatioValues( 'pressure' )
```

getSolRatioVarIndx()

Map the "Solution Ratio" variable into an index.

Usage

```
solRatioVarIndx = adb.getSolRatioVarIndx( var )
```

Parameters

var (string) or (integer)
The "Solution Ratio" variable name/index.

Return Value

***solRatioVarIndx** (integer)*
The mapped index.

Errors

var must be a valid name or index.

Description

This routine returns "Solution Ratio" variable into an index. The "Solution Ratio" name or index is given by *var*. For example,

```
solRatioVarIdx = adb.getSolRatioVarIndx( 'pressure' )
```

getSolRatioVarNames()

Return a list of "Solution Ratio" variable names.

Usage

```
solRatioVarNames = adb.getSolRatioVarNames()
```

Parameters

None

Return Value

solRatioVarNames *(list)*
A list of "Solution Ratio" variable names.

Description

This routine returns a list of "Solution Ratio" variable names. For example,

```
solRatioVarNames = adb.getSolRatioVarNames( )
```

getSolRatioVarUnit()

Return the SI unit of the "Solution Ratio" variable.

Usage

```
solRatioVarUnit = adb.getSolRatioVarUnit( name )
```

Parameters

name *(string)*
The "Solution Ratio" variable name.

Return Value

solRatioVarUnit *(string)*
The SI unit of the variable.

Errors

name must be a valid name.

Description

This routine returns the SI unit of the "Solution Ratio" variable. The "Solution Ratio" name is given by *name*. For example,

```
solRatioVarUnit = adb.getSolRatioVarUnit( 'pressure' )
```

getSteps()

Return the "Run Data" steps value.

Usage

```
steps = adb.getSteps( )
```

Parameters

None

Return Value

steps *(numarray)*
The steps values.

Description

This routine returns the "Run Data" steps value. For example,

```
steps = adb.getSteps( )
```

getTimeIncs()

Return the "Time Increment" value.

Usage

```
timeIncs = adb.getTimeIncs( )
```

Parameters

None

Return Value

timeIncs *(numarray)*
The "Time Increment" values.

Description

This routine returns the "Time Increment" values. For example,

```
timeIncs = adb.getTimeIncs( )
```

getTimes()

Return the "Run Data" times value.

Usage

```
times = adb.getTimes( )
```

Parameters

None

Return Value

`times` *(numarray)*
The time values.

Description

This routine returns the times values. For example,

```
times = adb.getTimes( )
```

getVarUnit()

Return the SI unit of the variable according to its adb name.

Usage

`varUnit = adb.getVarUnit( name, adbOut )`

Parameters

`name` *(string)*
The variable name.

`adbOut` *(string)*
Related adb out name.

Return Value

`varUnit` *(string)*
The variable SI unit.

Description

This routine returns the SI unit of the variable according to its adb name. For example,

```
osiUnit = adb.getVarUnit( "pressure", "osi" )
```

Volume Access Functions

12

This chapter covers the following:

getNVols()

Return the number of volumes.

Usage

```
nVols = vis.getNVols( )
```

Parameters

None

Return Value

nVols *(integer)*
Number of volumes.

Errors

None

Description

This routine returns the number of volumes. For example,

```
nVols = vis.getNVols( )
```

getVolName()

Return the name of a volume specified by its ID.

Usage

```
volName = vis.getVolName( volId )
```

Parameters

volId (integer)
Volume ID.

Return Value

***volName** (string)*
Volume name corresponding to the volume ID.

Errors

volId should be valid.

Description

This routine returns the name of a volume specified by its ID which is given by *volId*. For example,

```
volName = vis.getVolName( 1 )
```

getVolActor()

Return volume actor specified by its information.

Usage

```
volActor = vis.getVolActor( volInfo )
```

Parameters

volInfo (integer) or (string)
Name or ID of a volume.

Return Value

***volActor** (object)*
Volume actor corresponding to the *volInfo*.

Errors

volInfo should be valid.

Description

This routine returns the volume actor specified by its name or ID which is given by *volInfo*. For example,

```
imp = vis.getVolActor( "impeller" )
```

Surface Access Functions

13

This chapter covers the following:

getNSrfs()

Return the number of surfaces.

Usage

```
nSrfs = vis.getNSrfs( )
```

Parameters

None

Return Value

nSrfs *(integer)*
Number of surfaces.

Errors

None

Description

This routine returns the number of surfaces. For example,

```
nSrfs = vis.getNSrfs( )
```

getSrfName()

Return the name of a surface specified by its ID.

Usage

```
srfName = vis.getSrfName( srfId )
```

Parameters

`srfId` *(integer)*
Surface ID.

Return Value

`srfName` *(string)*
Surface name corresponding to the surface ID.

Errors

`srfId` should be valid.

Description

This routine returns the name of a surface specified by its ID which is given by *`srfId`*. For example,

```
srfName = vis.getSrfName( 1 )
```

getSrfActor()

Return surface actor specified by its information.

Usage

```
srfActor = vis.getSrfActor( srfInfo )
```

Parameters

`srfInfo` (integer) or (string)
Name or ID of a surface.

Return Value

***`srfActor`** (object)*
Surface actor corresponding to the *`srfInfo`*.

Errors

`srfInfo` should be valid.

Description

This routine returns the surface actor specified by its name or ID which is given by *`srfInfo`*. For example,

```
fan = vis.getSrfActor( "impeller" )
```

Periodics Access Functions

14

This chapter covers the following:

getNPbcs()

Return the number of periodics.

Usage

```
nPbcs = vis.getNPbcs( )
```

Parameters

None

Return Value

nPbcs *(integer)*
Number of periodics.

Errors

None

Description

This routine returns the number of periodics. For example,

```
nPbcs = vis.getNPbcs( )
```

getPbcName()

Return the name of a periodic specified by its ID.

Usage

pbcName = vis.getPbcName(pbcId)

Parameters

pbcId (integer)
Periodic ID.

Return Value

***pbcName** (string)*
Periodic name corresponding to the periodic ID.

Errors

pbcId should be valid.

Description

This routine returns the name of a periodic specified by its ID which is given by *pbcId*. For example,

```
pbcName = vis.getPbcName( 1 )
```

getPbcActor()

Return periodic actor specified by its information.

Usage

pbcActor = vis.getPbcActor(pbcInfo)

Parameters

pbcInfo (integer) or (string)
Name or ID of a periodic.

Return Value

***pbcActor** (object)*
Periodic actor corresponding to the *pbcInfo*.

Errors

pbcInfo should be valid.

Description

This routine returns the periodic actor specified by its name or ID which is given by *pbcInfo*. For example,

```
pbc = vis.getPbcActor( 0 )
```

Node Access Functions

15

This chapter covers the following:

getNNbcs()

Return the number of nodes.

Usage

```
nNbcs = vis.getNNbcs( )
```

Parameters

None

Return Value

nNbcs *(integer)*
Number of nodes.

Errors

None

Description

This routine returns the number of nodes. For example,

```
nNbcs = vis.getNNbcs( )
```

getNbcName()

Return the name of a node specified by its ID.

Usage

nbcName = vis.getNbcName(nbcId)

Parameters

nbcId (integer)
Node ID.

Return Value

***nbcName** (string)*
Node name corresponding to the node ID.

Errors

nbcId should be valid.

Description

This routine returns the name of a node specified by its ID which is given by *nbcId*. For example,

```
nbcName = vis.getNbcName( 1 )
```

getNbcActor()

Return node actor specified by its information.

Usage

nbcActor = vis.getNbcActor(nbcInfo)

Parameters

nbcInfo (integer) or (string)
Name or ID of a node.

Return Value

***nbcActor** (object)*
Node actor corresponding to the *nbcInfo*.

Errors

nbcInfo should be valid.

Description

This routine returns the node actor specified by its name or ID which is given by *nbcInfo*. For example,

```
nbc = vis.getNbcActor( 1 )
```

Scalar and Vector Variables Functions

16

This chapter covers the following:

getNVars()

Return the number of variables (both scalar and vector variables).

Usage

```
nVars = vis.getNVars( )
```

Parameters

None

Return Value

nVars *(integer)*
Number of scalar and vector variables.

Errors

None

Description

This routine returns the number of both scalar and vector variables. For example,

```
nVars = vis.getNVars( )
```

getVarName()

Return the name of a variable corresponds to *varId*.

Usage

varName = vis.getVarName(varId)

Parameters

varId *(integer)*
Variable ID.

Return Value

varName *(string)*
Variable name corresponding to the variable ID.

Errors

varId should be valid.

Description

This routine returns the name of a variable specified by its ID which is given by *varId*. For example,

```
varName = vis.getVarName( 2 )
```

getVarDim()

Return dimension of a variable which corresponds to *varInfo* (variable name or ID).

Usage

```
varDim = vis.getVarDim( varInfo )
```

Parameters

varInfo (integer) or (string)
: Name or ID of a variable.

Return Value

***varDim** (integer)*
: Dimension of the variable corresponding to the *varInfo*.

Errors

varInfo should be valid.

Description

This routine returns the dimension of a variable which corresponds to *varInfo*, which is the name or ID of the variable. For example,

```
varDim = vis.getVarDim( "velocity" )
```

getNSclrVars()

Return the number of scalar variables.

Usage

```
nSVars = vis.getNSclrVars( )
```

Parameters

None

Return Value

nSVars *(integer)*
Number of scalar variables.

Errors

None

Description

This routine returns the number of scalar variables. For example,

```
nSVars = vis.getNSclrVars( )
```

getSclrVarName()

Return a scalar variable name that corresponds to valid ID.

Usage

```
sclrVarName = vis.getSclrVarName( varId )
```

Parameters

varId (integer)
Variable ID.

Return Value

***sclrVarName** (string)*
Name of the scalar variable corresponding to the *varId*.

Errors

varId should be valid.

Description

This routine returns a scalar variable name that corresponds to *varId*. For example,

```
sclrVarName = vis.getSclrVarName( 2 )
```

getNVecVars()

Return the number of vector variables.

Usage

`nVVars = vis.getNVecVars( )`

Parameters

None

Return Value

`nVVars` *(integer)*
Number of vector variables.

Errors

None

Description

This routine returns the number of vector variables. For example,

```
nVVars = vis.getNVecVars( )
```

getVecVarName()

Return a vector variable name that corresponds to the variable ID.

Usage

```
vecVarName = vis.getVecVarName( varId )
```

Parameters

varId (integer)
Variable ID.

Return Value

***vecVarName** (string)*
Name of the vector variable corresponding to the *varId*.

Errors

varId should be valid.

Description

This routine returns a vector variable name that corresponds to *varId*. For example,

```
vecName = vis.getVecVarName( 0 )
```

setSclVar()

Set the current scalar variable to variable information (scalar variable name or ID).

Usage

`vis.setSclrVar( varInfo )`

Parameters

`varInfo` *(integer) or (string)*
Variable ID or variable name.

Return Value

None

Errors

`varInfo` should be valid.

Description

This routine sets the current scalar variable to *`varInfo`*. The *`varInfo`* could be the name or ID of the scalar variable. For example,

```
vis.setSclrVar( "pressure" )
```

setSclrLimits()

Set the minimum and maximum limits of the current scalar field.

Usage

```
vis.setSclrLimits( minVal = None, maxVal = None )
```

Parameters

minVal (integer)
: Minimum value of the scalar variable.

maxVal (integer)
: Maximum value of the scalar variable.

Return Value

None

Errors

minVal and *maxVal* should be integer.

Description

This routine sets the minimum and maximum limits of the current scalar field to *minVal* and *maxVal*. For example,

```
vis.setSclrLimits( 10, 50 )
```

setVecVar()

Set the current vector variable to variable information (vector variable name or ID).

Usage

`vis.setVecVar( varInfo )`

Parameters

`varInfo` *(integer) or (string)*
Variable ID or variable name.

Return Value

None

Errors

`minVal` and *`maxVal`* should be integer.

Description

This routine sets the current vector variable to *`varInfo`*. The *`varInfo`* could be the name or ID of the vector variable. For example,

```
vis.setVecVar( "velocity" )
```

setVecScale()

Set the vector scale of current vector field.

Usage

```
vis.setVecScale( vecScale )
```

Parameters

`vecScale` ***(integer)***
Vector scale.

Return Value

None

Errors

`vecScale` should be integer.

Description

This routine sets the vector scale of the current vector field to *`vecScale`*. For example,

```
vis.setVecScale( 5 )
```

Iso-Surface and Cut-Plane Functions

17

This chapter covers the following:

addIsoSurface()

Create an IsoSurface actor.

Usage

`isoActor = vis.addIsoSurface( isoVec, isoVal, name, vols = None )`

Parameters

`isoVec` *(list) or (string)*
Iso-surface vector.

`isoVal` *(integer)*
Iso-surface value.

`name` *(string)*
Optional name.

`vols` *(list), (string) or (integer)*
Volume(s).

Return Value

`isoActor` *(integer)*
Iso-surface actor.

Errors

`vols` should include valid volumes or be None.

Description

This routine creates an IsoSurface actor from the *`vols`* based on the *`isoVec`* and *`isoVal`*, names it *`name`* and adds its actor to Scene Graph. For example,

```
pres = vis.addIsoSurface( "pressure",
                                    0.1,
                                    "pres-iso" )
```

getNIsos()

Return the number of iso-surface actors.

Usage

```
nIsos = vis.getNIsos( )
```

Parameters

None

Return Value

nIsos *(integer)*
Number of iso-surface actors.

Errors

None

Description

This routine returns the number of iso-surface actors. For example,

```
nIsos = vis.getNIsos( )
```

getIsoName()

Return name of an iso-surface object.

Usage

`isoName = vis.getIsoName( isoId )`

Parameters

`isoId` *(integer)*
Iso-surface ID.

Return Value

`isoName` *(string)*
Iso-surface name.

Errors

`isoId` should be valid.

Description

This routine returns the name of an iso-surface object which is specified by its ID *`isoId`*. For example,

```
nIsos = vis.getNIsos( )
for i in range(nIsos):
     name = vis.getIsoName( i )
```

getIsoActor()

Return an iso-surface actor.

Usage

```
isoActor = vis.getIsoActor( isoInfo )
```

Parameters

`isoInfo` *(integer) or (string)*
: Iso-surface Name or ID.

Return Value

`isoActor` *(object)*
: Iso-surface actor.

Errors

`isoInfo` should be valid.

Description

This routine returns an iso-surface actor which is specified by *`isoInfo`*. The isoInfo could be iso-surface ID or name. For example,

```
actor = vis.getIsoActor( "pres-iso" )
```

delIsoActor()

Remove an iso-surface actor.

Usage

`vis.delIsoActor( isoInfo )`

Parameters

`isoInfo` *(integer) or (string)*
Iso-surface Name or ID.

Return Value

None

Errors

`isoInfo` should be valid.

Description

This routine removes an iso-surface actor which is specified by *`isoInfo`* from scene graph. The *`isoInfo`* could be iso-surface ID or name. For example,

```
vis.delIsoActor( "pres-iso" )
```

addCPlane()

Create a cut-plane actor.

Usage

cplActor = vis.addCPlane(point1, point2, point3, name, vols = None)

Parameters

point1 *(list)*
Point 1 information.

point2 *(list)*
Point 2 information.

point3 *(list)*
Point 3 information.

name *(string)*
optional name.

vols *(list), (string) or (integer)*
volume(s).

Return Value

cplActor *(object)*
Cut-plane actor.

Errors

vols should include valid volumes or be None.

Description

This routine creates a cut-plane actor from the *vols* based on the *point1*, *point2* and *point3*, names it *name* and adds its actor to Scene Graph. For example,

```
imp = vis.getVolActor( "impeller" )
impBbox = imp.bndBox( )
xmin = bndBox[0][0]
xmax = bndBox[0][1]
ymin = bndBox[1][0]
ymax = bndBox[1][1]
zmin = bndBox[1][0]
zmax = bndBox[1][1]
ymid = ( impBbox[1][0] + impBbox[1][1] ) / 2
cpl = vis.addCPlane( [xmin, ymid, zmin],
                     [xmax, ymid, zmin],
                     [xmax, ymid, zmax],
                     "mid-plane" )
```

getNCpls()

Return the number of cut-plane actors.

Usage

```
nCpls = vis.getNCpls( )
```

Parameters

None

Return Value

nCpls *(integer)*
Number of cut-plane actors.

Errors

None.

Description

This routine returns the number of cut-plane actors. For example,

```
nCpls = vis.getNCpls( )
```

getCplName()

Return the name of a cut-plane object.

Usage

`cplName = vis.getCplName( cplId )`

Parameters

`cplId` *(integer)*
Cut-plane ID.

Return Value

`cplName` *(string)*
Cut-plane name.

Errors

`cplId` should be valid.

Description

This routine returns the name of a cut-plane object which is specified by its ID *`cplId`*. For example,

```
nCpls = vis.getNCpls( )
for i in range(nCpls):
    name = vis.getCplName( i )
```

getCplActor()

Return a cut-plane actor.

Usage

```
cplActor = vis.getCplActor( cplInfo )
```

Parameters

`cplInfo` (integer) or (string)
Cut-plane Name or ID.

Return Value

***`cplActor`** (object)*
Cut-plane actor.

Errors

`cplInfo` should be valid.

Description

This routine returns a cut-plane actor which is specified by *`cplInfo`*. The *`cplInfo`* could be cut-plane ID or name. For example,

```
actor = vis.getCplActor( "mid-plane" )
```

delCplActor()

Removes a cut-plane actor.

Usage

`vis.delCplActor( cplInfo )`

Parameters

`cplInfo` *(integer) or (string)*
Cut-plane Name or ID.

Return Value

None

Errors

`cplInfo` should be valid.

Description

This routine removes a cut-plane actor which is specified by *`cplInfo`* from scene graph. The *`cplInfo`* could be cut-plane ID or name. For example,

```
vis.delCplActor( 0 )
```

Iso_Line Functions

18

This chapter covers the following:

addIsoLine()

Create an IsoLine actor.

Usage

isoLnActor = vis.addIsoLine(isoVec, isoVal, name, srfs = None)

Parameters

isoVec (list) or (string)
Ico-line vector.

isoVal (integer)
Iso-line value.

name (string)
Optional name.

srfs (list), (string) or (integer)
Surface(s).

Return Value

***isoLnActor** (object)*
Iso-line actor.

Errors

srfs should include valid surfaces or be None.

Description

This routine creates an IsoLine actor from the *srfs* based on the *isoVec* and *isoVal*, named it *name* and adds its actor to Scene Graph. For example,

```
pMin = vis.sclrValue.min()
pMax = vis.sclrValue.max()
dp = (pMax - pMin) / 11
for i in range(10):
   pVal = i *1 + 4
pres = vis.addIsoLine( "pressure",
                       pVal,
                       "pres-isoLine",
                       'wall' )
```

getNIsoLns()

Return the number of iso-line actors.

Usage

```
nIsoLns = vis.getNIsoLns( )
```

Parameters

None

Return Value

nIsoLns *(integer)*
Number of iso-line actors.

Errors

None

Description

This routine returns the number of iso-line actors. For example,

```
nIsoLns = vis.getNIsoLns( )
```

getIsoLnName()

Return name of an iso-line object.

Usage

isoLnName = vis.getIsoLnName(isoLnId)

Parameters

isoLnId (integer)
Iso-line ID.

Return Value

***isoLnName** (string)*
Iso-line name.

Errors

isoLnId should be valid.

Description

This routine returns the name of an iso-line object which is specified by its ID *isoLnId*. For example,

```
nIsoLns = vis.getNIsoLns( )
for i in range(nIsoLns):
    name = vis.getIsoLnName( i )
```

getIsoLnActor()

Return an iso-line actor.

Usage

```
isoLnActor = vis.getIsoLnActor( isoLnInfo )
```

Parameters

`isoLnInfo` *(integer) or (string)*
Iso-line Name or ID.

Return Value

`isoLnActor` *(object)*
Iso-line actor.

Errors

`isoLnInfo` should be valid.

Description

This routine returns an iso-line actor which is specified by *`isoLnInfo`*. The *`isoLnInfo`* could be iso-line ID or name. For example,

```
actor = vis.getIsoLnActor( "pres-isoLine" )
```

delIsoLnActor()

Remove an iso-line actor.

Usage

vis.delIsoLnActor(isoLnInfo)

Parameters

isoLnInfo (integer) or (string)
Iso-line Name or ID.

Return Value

None

Errors

isoLnInfo should be valid.

Description

This routine removes an iso-line actor which is specified by *isoLnInfo* from scene graph. The *isoLnInfo* could be iso-line ID or name. For example,

```
vis.delIsoLnActor( "pres-isoLine" )
```

Tufts Functions

19

This chapter covers the following:

addTufts()

Create a Tufts actor.

Usage

`tufts = vis.addTufts( points, name, vols = None )`

Parameters

`points` *(list)*
Random points which should be projected to create tufts actor.

`name` *(string)*
Optional name.

`vols` *(list), (string) or (integer)*
Volume(s).

Return Value

`tufts` *(object)*
Tufts actor

Errors

`vols` should include valid volumes or be None.

Description

This routine creates a Tufts actor from the *`vols`*, names it *`name`* and adds its actor to Scene Graph. For example,

```
points = vis.genPoints( type = "2d_grid",
                                point1 = [xmin, ymid, zmin],
                                point2 = [xmax, ymid, zmin],
                                point3 = [xmax, ymid, zmax],
                                nXPoints= 100,
                                nYPoints= 100 )
tufts = vis.addTufts( points, "mid-plane" , 'impeller')
```

delTufts()

Remove a Tufts actor.

Usage

`vis.delTufts( tuftsInfo )`

Parameters

`TuftsInfo` *(integer) or (string)*
Tuft's name or ID.

Return Value

None

Errors

`tuftsInfo` should be valid.

Description

This routine removes a Tufts actor which is specified by *`tuftsInfo`* from the scene graph. The *`TuftsInfo`* could be Tufts ID or name. For example,

```
vis.delTufts( "mid-plane" )
```

Clip Plane Functions

20

This chapter covers the following:

addClipPlane()

Create a Clip Plane for an actor.

Usage

```
clipPlane = vis.addClipPlane( point1, point2, point3,
                                        side = 'up', actor = None )
```

Parameters

point1 *(list)*
: Point1 is used for creating clip plane.

point2 *(list)*
: Point1 is used for creating clip plane.

point3 *(list)*
: Point3 is used for creating clip plane.

side *(string)*
: Side of the clip plane.

actor *(object)*
: The actor which the clipping is applied to it. If actor is None (the default), the clipping is applied to all 3D objects.

Return Value

clipPlane *(object)*
: Clip Plane object

Errors

actor should already be created and be valid.

Description

This routine creates a clip plane for the specified *actor*, and adds it to Scene Graph. If *actor* is None (the default), the clipping is applied to all 3D objects. For example,

```
bndBox = vis.bndBox()
imp = vis.getVolActor( "impeller" )
impBbox = imp.bndBox()
xmin = bndBox[0][0]
xmax = bndBox[0][1]
ymin = bndBox[1][0]
ymax = bndBox[1][1]
zmin = bndBox[1][0]
zmax = bndBox[1][1]
ymid = (impBbox[1][0] + impBbox[1][1] ) / 2
clp = vis.addClipPlane( [xmin, ymid, zmin],
                        [xmax, ymid, zmin],
                        [xmax, ymid, zmax],
                         side = "up" )
```

delClipPlane()

Remove a clip plane from an actor.

Usage

```
vis.delClipPlane( actor = None )
```

Parameters

actor (object)
: The actor with a clip plane that should be removed. If actor is None, all of the 3D objects's clip planes will be removed.

Return Value

None

Errors

actor should be valid.

Description

This routine removes the clip plane from an actor which is specified by *actor*. If *actor* is None, all of the 3D object's clip planes will be removed. For example,

```
vis.delClipPlane( )
```

activeClipPlane()

Activate\deactivate the clip plane of an actor.

Usage

```
vis.activeClipPlane( active = True, actor = None )
```

Parameters

`active` (boolean)
: This flag indicates if the clip plane should be activated or deactivated.

`actor` (object)
: The actor that has a clip plane should be activated\deactivated. If actor is None, all of the 3D object's clip planes will be activated\deactivated.

Return Value

None

Errors

`actor` should be valid.

Description

This routine activates/deactivates the clip plane for an actor which is specified by *`actor`*. If *`actor`* is None, all of the 3D object's clip planes will be activated/deactivated. *`active`* indicated activation \deactivation. For example,

```
vis.activeClipPlane( active = False )
```

Time Step Functions

21

This chapter covers the following:

getNSteps()

Return the number of time steps.

Usage

```
nSteps = vis.getNSteps( )
```

Parameters

None

Return Value

nSteps *(integer)*
Number of time steps.

Errors

None

Description

This routine returns the number of time steps. For example,

```
nSteps = vis.getNSteps( )
```

getSteps()

Return the list of time steps.

Usage

```
steps = vis.getSteps( )
```

Parameters

None

Return Value

steps *(list)*
List of time steps.

Errors

None

Description

This routine returns the list of time steps. For example,

```
steps = vis.getSteps( )
```

getTimes()

Return the list of times.

Usage

```
times = vis.getTimes( )
```

Parameters

None

Return Value

times *(list)*
List of times.

Errors

None

Description

This routine returns the list of times. For example,

```
times = vis.getTimes( )
```

setStep()

Set the current time step.

Usage

`vis.setStep( step )`

Parameters

`step` *(string)*
The time step which should be set.

Return Value

None

Errors

`step` should be valid.

Description

This routine sets the current time step to *`step`*. For example,

```
vis.setStep( "2" )
```

setStepId()

Set the current time step ID.

Usage

`vis.setStepId( stepId )`

Parameters

`stepId` *(integer)*
The time step ID which should be set.

Return Value

None

Errors

`stepId` should be valid.

Description

This routine sets the current time step ID to *`stepId`*. For example,

```
vis.setStepId( 3 )
```

Display Orientation Functions

22

This chapter covers the following:

home()

Set the camera to its home position.

Usage

`vis.home( )`

Parameters

None

Return Value

None

Errors

None

Description

This routine sets the camera to its home position. For example,

```
vis.home( )
```

fit()

Set the camera to fit all the actors in the Scene Graph.

Usage

`vis.fit( )`

Parameters

None

Return Value

None

Errors

None

Description

This routine sets the camera to fit all the actors in the Scene Graph. For example,

```
vis.fit( )
```

snap()

Compute and align the camera to the closest principal axis.

Usage

`vis.snap( )`

Parameters

None

Return Value

None

Errors

None

Description

This routine computes and aligns the camera to the closest principal axis. For example,

```
vis.snap( )
```

snapz()

Compute and align the camera to the closest z plane.

Usage

`vis.snapz( )`

Parameters

None

Return Value

None

Errors

None

Description

This routine computes and aligns the camera to the closest z plane. For example,

```
vis.snapz( )
```

alignDir()

Set the camera to look up in a special direction.

Usage

`vis.alignDir( dir )`

Parameters

`dir` *(string)*

The desired direction.Valid values are x+, +x, xplus, y+, +y, yplus, z+, +z, zplus, x-, -x, xminus, y-, -y, yminus, z-, -z and zminus in case insensitive manor.

Return Value

None

Errors

`dir` should be valid.

Description

This routine sets the camera to look up in a special direction specified by *`dir`*. For example,

```
vis.alignDir( dir = 'y-' )
```

rotate()

Rotate the camera by special degrees in special directions.

Usage

```
vis.rotate( dir, angle = 45 )
```

Parameters

dir (string)
: The desired direction. Valid values are x+, +x, xplus, y+, +y, yplus, z+, +z, zplus, x-, -x, xminus, y-, -y, yminus, z-, -z and zminus in case insensitive manor.

angle (integer)
: Degree of rotation.

Return Value

None

Errors

dir should be valid.

angle should be integer.

Description

This routine rotates the camera by special degrees given by *angle* in special directions given by *dir*. For example,

```
vis.rotate( dir ="x+", angle = -20. )
```

zoom()

Zoom the camera by a special scale.

Usage

```
vis.zoom( scale = 1. )
```

Parameters

`scale` *(integer)*
Zoom scale.

Return Value

None

Errors

`scale` should be integer.

Description

This routine zooms the camera by a special scale given by *`scale`*. For example,

```
vis.zoom( 5 )
```

Add\Remove Actors Functions

23

This chapter covers the following:

addTxtActor()

Add a 2D text actor to the scene graph.

Usage

```
txtActor = vis.addTxtActor( text = "ACUSIM",
                                    position = [0.5,0.5],
                                    style = "Times New Roman:bold",
                                    fontSize = 12,
                                    color = [1,0,0],
                                    horAlignment = "CENTER",
                                    verAlignment = "BOTTOM"' )
```

Parameters

text (string)
: Annotation text.

position (list)
: Annotation position.

style (string)
: Annotation style.

fontSize (integer)
: Annotation font size.

color (list)
: Annotation color.

horAlignment (string)
: Annotation horizontal alignmentValid values are RIGHT, CENTER and LEFT.

verAlignment (string)
: Annotation vertical alignment. Valid values are BOTTOM, TOP and HALF.

Return Value

***txtActor** (object)*
: The 2D text actor.

Errors

None

Description

This routine adds a 2D text actor to the scene graph. For example,

```
txtActor = vis.addTxtActor( text = 'Test',
                                    color = [0,0,1],
                                    fontSize = 20 )
```

remTxtActor()

Remove 2D text from the scene graph.

Usage

```
vis.remTxtActor( txtActor )
```

Parameters

txtActor (object)
The text actor which should be removed.

Return Value

None

Errors

txtActor must exist.

Description

This routine removes 2D text actor from the scene graph. For example,

```
vis.remTxtActor( txtActor )
```

addImgActor()

Add an image actor to the scene graph.

Usage

```
imgActor = vis.addImgActor( filename = None,
                                    image = None,
                                    position = [0.5,0.5],
                                    width = -1,
                                    height = -1,
                                    horAlignment = "CENTER",
                                    verAlignment = "BOTTOM" )
```

Parameters

fileName *(string)*
: Image file name.

image *(string)*
: Image to be added.

position *(list)*
: Image actor position.

width *(integer)*
: Image width.

height *(integer)*
: Image height.

horAlignment *(string)*
: Image horizontal alignment. Valid values are RIGHT, CENTER and LEFT.

verAlignment *(string)*
: Image vertical alignment. Valid values are BOTTOM, TOP and HALF.

Return Value

imgActor *(object)*
: The image actor.

Errors

None

Description

This routine adds an image actor to the scene graph. For example,

```
imgActor = vis.addImgActor( filename = "logo_med.jpg",
                                     position = [0.73,0.78,0.5],
                                     width = 300 ,
                                     height = 60 )
```

remImgActor()

Remove an image actor from the scene graph.

Usage

`vis.remImgActor( imgActor )`

Parameters

`imgActor` *(object)*
The image actor which should be removed.

Return Value

None

Errors

`imgActor` must exist.

Description

This routine removes an image actor from the scene graph. For example,

```
vis.remImgActor( imgActor )
```

addSphereActor()

Add a sphere actor to the scene graph.

Usage

```
sphActor = vis.addSphereActor( center = [0,0,0], radius = None,
                               color = [0,1,0], text = None,
                               pointSize = 2, lineWidth = 2,
                               fontSize = None, vis = True,
                               trans = False, transVal = 0.5 )
```

Parameters

center (array)
: Center of the sphere.

radius (real)
: Sphere radius.

color (list of RGB)
: Sphere color.

text (string)
: Sphere text.

pointSize (integer)
: Sphere point size.

lineWidth (integer)
: Sphere line width.

fontSize (integer)
: Sphere text font size.

vis (boolean)
: Sphere visibility.

trans (boolean)
: Sphere transparency.

transVal (integer)
: Sphere transparency value.

Return Value

***sphereActor** (object)*
: The sphere actor.

Errors

None

Description

This routine adds a sphere actor to the scene graph. For example,

```
sphActor = vis.addSphereActor( radius = 3 )
```

delSphereActor()

Remove sphere actor from the scene graph.

Usage

`vis.delSphereActor( sphActor )`

Parameters

`sphActor` *(object)*
The sphere actor which should be removed.

Return Value

None

Errors

`sphActor` must exist.

Description

This routine removes sphere actor from the scene graph. For example,

```
vis.delSphereActor( sphActor )
```

addCmapLegendActor()

Add 2D color map legend to a specified location of scene graph.

Usage

```
cmapActor = vis.addCmapLegendActor( text = "Color Map Legend",
                                    textFont = "Times-Roman",
                                    minVal = 0,
                                    maxVal = 5,
                                    nVals = 3,
                                    valFont = "Times-Roman",
                                    xpos = 0.05,
                                    ypos = 0.15,
                                    xsize = 2,
                                    ysize = 10 )
```

Parameters

text (string)
: Color map text which is shown on top of the legend.

textFont (string)
: Color map text font.

minVal (integer)
: Color map minimum value.

maxVal (integer)
: Color map maximum value.

nVals (integer)
: Number of values in color map.

valFont (string)
: Color map values font.

xpos (real)
: Color map X position.

ypos (real)
: Color map Y position.

xsize (integer)
: Color map X size.

ysize (integer)
: Color map Y size.

Return Value

***cmapActor** (object)*
: The color map legend actor.

Errors

None.

Description

This routine adds the 2D color map legend to a specified location of scene graph. For example,

```
cmap = vis.addCmapLegendActor( text = "Pressure",
                                       minVal = 0,
                                       maxVal = 4,
                                       nVals = 4,
                                       xpos = 0.08 )
```

delCmapLegendActor()

Remove the color map legend from the scene graph.

Usage

`vis.delCmapLegendActor( cmapActor )`

Parameters

`cmapActor` *(object)*
The cmap actor which should be removed.

Return Value

None

Errors

`cmapActor` must exist.

Description

This routine removes the color map legend actor from the scene graph. For example,

```
vis.delCmapLegendActor( cmapActor )
```

addGeomActor()

Add a geom actor to the scene graph.

Usage

```
geomActor = vis.addGeomActor( type = "cylinder", center = [0,0,0],
                                        normal = [0,1,0], normalX = [1,0,0],
                                        normalY = [0,1,0], normalZ = [0,0,1],
                                        radius = None, height = None,
                                        width = None, depth = None,
                                        xangle = None, yangle = None,
                                        zangle = None, color = [1,0,0],
                                        point = None, pntList = None,
                                        point1 = None, point2 = None,
                                        text = None, pointSize = 2,
                                        lineWidth = 2, fontSize = None,
                                        arrowLength= None, triSize = None,
                                        tetSize = None, vis = True,
                                        trans = False, transVal = 0.5,
                                        points = None )
```

Parameters

type (string)
: Type of the geom actor. It could be cylinder, cone, box, sphere, tet, triangle, circle, brick, text or quade.

center (array)
: Center of the geom actor.

normal (array)
: Geom normal vector.

normalX (array)
: Geom normalX.

normalY (array)
: Geom normalY.

normalZ (array)
: Geom normalZ.

radius (real)
: Geom radius.

height (real)
: Geom height.

width (real)
: Geom width.

depth (real)
: Geom depth.

`xangle` *(real)*
: Geom xangle.

`yangle` *(real)*
: Geom yangle.

`zangle` *(real)*
: Geom zangle.

`color` *(list of RGB)*
: Geom color.

`point` *(array)*
: Geom point.

`pntList` *(list)*
: Geom point list.

`point1` *(array)*
: Geom first point.

`point2` *(array)*
: Geom second point.

`text` *(string)*
: Geom text.

`pointSize` *(integer)*
: Geom point size.

`lineWidth` *(integer)*
: Geom line width.

`fontSize` *(integer)*
: Geom text font size.

`arrowLength` *(integer)*
: Geom arrow length.

`triSize` *(integer)*
: Geom triangle size.

`tetSize` *(integer)*
: Geom tet size.

`vis` *(boolean)*
: Geom visibility.

`trans` *(boolean)*
: Geom transparency.

`transVal` *(integer)*
: Geom transparency value.

`points` *(array)*
: Geom points.

Return Value

geomActor *(object)*
The geom actor.

Errors

None

Description

This routine adds a geom actor to the scene graph. For example,

```
geomActor = vis.addGeomActor( type = 'cylinder',
                                      radius = 3 )
```

delGeomActor()

Remove geom actor from the scene graph.

Usage

`vis.delGeomActor( geomActor )`

Parameters

`geomActor` *(object)*
The geom actor which should be removed.

Return Value

None

Errors

`geomActor` must exist.

Description

This routine removes geom actor from the scene graph. For example,

```
vis.delGeomActor( geomActor )
```

Set\Get Actors Properties

24

Display, transparency, transparency value, color, visibility, line width and point size of the "Surface actors", "Volume Actors", "PBC Actors", "NBC Actors", "Iso-Surface Actors" and "Cut-Plane Actors" can be set through the following functions.

This chapter covers the following:

display()

Set\get type of an actor.

Usage

actor.display(displayType = None)

Parameters

displayType *(string)*
Display type. Valid values are outline, solid, wireframe, contour, velocity_vector and none.

Return Value

String

If *displayType* = None, the current display type will be returned, or else None.

Errors

displayType must be valid.

Description

This routine sets\gets the display type of an actor. If *displayType* is None, the current display type will be returned. For example,

```
fan = vis.getSrfActor( "impeller" )
fan.display( 'solid_wire' )
or:
fanDisType = fan.display( )
```

transparency()

Set\get transparency on\off of an actor.

Usage

`actor.transparency( trans = None )`

Parameters

`trans` *(boolean)*
Transparency on/off flag.

Return Value

Boolean

If *`trans`* = None, the current transparency flag will be returned, or else None.

Errors

`trans` must be Boolean.

Description

This routine sets\gets transparency on\off of an actor. If *`trans`* is None, the current transparency flag will be returned. For example,

```
fan = vis.getSrfActor( "impeller" )
fan.transparency( True )
or:
fanTrans = fan.transparency( )
```

transparencyVal()

Set\get transparency value of an actor.

Usage

actor.transparencyVal(transVal = None)

Parameters

transVal (integer)
Transparency value.

Return Value

Integer

If *transVal* = None, the current transparency value will be returned, or else None.

Errors

transVal must be an integer.

Description

This routine sets\gets transparency value of an actor. If *transVal* is None, the current transparency value will be returned. For example,

```
fan = vis.getSrfActor( "impeller" )
fan.transparencyVal( 2 )
or:
fanTransVal = fan.transparencyVal( )
```

color()

Set\get color of an actor.

Usage

actor.color(red = None, green = None, blue = None)

Parameters

red (real)
Transparency value.

green (real)
Transparency value.

blue (real)
Transparency value.

Return Value

List(of RGB)

If *red* = None and *green* = None and *blue* = None, the current RGB value will be returned, or else None.

Errors

red, *green* and *blue* must be Real values.

Description

This routine sets\gets color value of an actor. If *red*, *green* and *blue* are None, the current red, green and blue values of color will be returned. For example,

```
fan = vis.getSrfActor( "impeller" )
fan.color( 0.5, 1, 0.4 )
or :
fanColor = fan.color( )
```

setVisibility()

Set the visibility of an actor.

Usage

```
actor.setVisibility( visibility = 'on' )
```

Parameters

visibility *(string)*
Visibility value.

Return Value

None

Errors

visibility must be on or off.

Description

This routine sets the visibility value of an actor. For example,

```
fan = vis.getSrfActor( "impeller" )
fan.setVisibility( "off" )
```

lineWidth()

Set\get line width of an actor.

Usage

`actor.lineWidth( width = None )`

Parameters

`width` *(integer)*
Line width of an actor.

Return Value

Integer

If *`width`* = None, the current width value will be returned, or else None.

Errors

`width` must be integer.

Description

This routine sets\gets the line width value of an actor. If *`width`* is None, the current line width value will be returned. For example,

```
fan = vis.getSrfActor( "impeller" )
fan.lineWidth( 3 )
or :
fanLineWdt = fan.lineWidth( )
```

pointSize()

Set\get point size of an actor.

Usage

`actor.pointSize( size = None )`

Parameters

`size` *(integer)*
Point size of an actor.

Return Value

Integer

If *`size`* = None, the current size value will be returned, or else None.

Errors

`size` must be integer.

Description

This routine sets\gets the size value of an actor. If *`size`* is None, the current point size value will be returned. For example,

```
fan = vis.getSrfActor( "impeller" )
fan.pointSize( 2 )
or :
fanPntSize = fan.pointSize( )
```

Scene Graph Functions

25

This chapter covers the following:

saveImage()

Create and save an image of the screen.

Usage

```
savedFileName = vis.saveImage( width = 600, height = 400,
                                        fileName = None, fileType = 'png',
                                        dirName = None )
```

Parameters

width (integer)
Width of image in pixels.

height (integer)
Height of image in pixels.

fileName (string)
Image file name.

- If the extension is included, the argument of *fileType* is ignored and the actual file type is determined by the extension.
- If the extension is not included, the actual file type is taken from the argument *fileType*.
- If *fileName* is None, *fileType* is used to generate the actual file name of `Image_*.fileType`.

fileType (string)
Image file type. Currently only .png format is supported.

dirName (string)
The directory that the image will be saved in. CWD represents the current working directory of AcuSolve. If *dirName* is not specified, the image file will be saved in `CWD/Figures/`. Otherwise the file will be saved in `CWD/Figures/dirName/`.

Return Value

***savedFileName** (string)*
Saved image file name, which contains the full directory of the file `savedFileName = CWD/Figures/dirName/filename`.

Errors

None

Description

It was found that 600 pixels can fit well into the width of a regular letter paper. If you would like to save the image with more pixels, the following way is suggested:

```
scaleFactor=3
imageFileName = vis.saveImage( width=600*scaleFactor, height=400*scaleFactor )
```

```
addFigure(imageFileName, scale=1.0/scaleFactor)
```

Here the `scaleFactor` is an integer. At first an image with pixels of `600*scaleFactor` x `400*scaleFactor` is saved. But this raw image cannot fit into the paper very well. So in `addFigure` the image is scaled smaller with the same `scaleFactor`. In the final .pdf report, the image has more pixels but the same physical size, so a higher resolution is obtained.

The following section of code generates a figure in which the size of the actor fits well:

```
# cpl_y is a cut plane object produced by vis.addCPlane
cutBndBox=cpl_y.sceneGraph.getBoundingBox()
# cutBndBox gives the bounding box of the coordinates of cpl_y, which is a list
 contains [xmin,ymin,zmin,xmax,ymax,zmax]
aspectRatio=(cutBndBox[3]-cutBndBox[0])/(cutBndBox[5]-cutBndBox[2])
# Since the cut plane is normal to Y axis, the camera will be adjusted normal to
 Y axis. The width of the actor is measured along X axis and the height is along Z
 axis. The aspect ratio is obtained by (xmax-xmin)/(zmax-zmin)
vis.zoom( -math.log(aspectRatio)  )
# The command above gives a view in which the actor fits the image best.
imgFct=4
fname = vis.saveImage( width =600*imgFct , height = 600*imgFct/aspectRatio,
 fileName='Mesh_xz.png')
# The size of the image depends on the aspect ratio of the actor
```

setBgColor()

Set background color of the scene graph.

Usage

`vis.setBgColor( color )`

Parameters

`color` (list of RGB)
Background color which should be set.

Return Value

None

Errors

None

Description

This routine sets the background color of the scene graph to *`color`*. For example,

```
vis.setBgColor( color = (1,1,1) )
```

setAxis()

Set axis size of the scene graph.

Usage

```
vis.setAxis( size = None )
```

Parameters

size (integer)
Axis size which should be set.

Return Value

None

Errors

None

Description

This routine sets the axis size of the scene graph to *size*. For example,

```
vis.setAxis( 3 )
```

setLineWidth()

Set the line width used in various actors of scene graph.

Usage

`vis.setLineWidth( width = None )`

Parameters

`width` *(integer)*
The line width which should be set.

Return Value

None

Errors

None

Description

This routine sets the line width used in various actors of scene graph to *`width`*. For example,

```
vis.setLineWidth( 2 )
```

setShading()

Set the shading style used in various actors of scene graph.

Usage

`vis.setShading( shading = None )`

Parameters

`shading` *(string)*
The shading style which should be set. Valid values are Flat, Gouraud and Phong.

Return Value

None

Errors

None

Description

This routine sets the shading style used in various actors of scene graph to *`shading`*. For example,

```
vis.setShading( 'Gouraud' )
```

setPointSize()

Set the point size used in various actors of scene graph.

Usage

```
vis.setPointSize( size = None )
```

Parameters

`size` *(integer)*
The point size which should be set.

Return Value

None

Errors

None

Description

This routine sets the point size used in various actors of scene graph to *`size`*. For example,

```
vis.setPointSize( 3 )
```

setTransType()

Set the transparency type of scene graph.

Usage

`vis.setTransType( type )`

Parameters

`type` *(string)*
: Transparency type which should be set. Valid values are SCREEN_DOOR, ADD, DELAYED_ADD, BLEND, SORTED_OBJECT_ADD, DELAYED_BLEND and SORTED_OBJECT_BLEND.

Return Value

None

Error

`type` should be valid.

Description

This routine sets the transparency type of scene graph to *`type`*. For example,

```
vis.setTransType( "BLEND" )
```

bndBox()

Return the bounding box of the scene graph.

Usage

bndBox = vis.bndBox( )

Parameters

None

Return Value

bndBox *(array)*
Bounding box of the scene graph.

Errors

None

Description

This routine returns the bounding box of the scene graph. For example,

```
boundBox = vis.bndBox( )
```

Miscellaneous Functions

26

This chapter covers the following:

setDeform()

Turn on/off deform geometry.

Usage

```
vis.setDeform( deform = True )
```

Parameters

deform (boolean)
Deform geometry.

Return Value

None

Errors

None

Description

This routine turns on/off the deform geometry to *deform*. For example,

```
vis.setDeform( True )
```

toggleLogo()

Toggle the display of the Acusim logo.

Usage

`vis.toggleLogo( )`

Parameters

None

Return Value

None

Errors

None

Description

This routine toggles the display of the Acusim logo. For example,

```
vis.toggleLogo( )
```

setCmap()

Set color map.

Usage

```
vis.setCmap( cmap = 'default' )
```

Parameters

cmap (numarray or name of it as string)
The color map that is set.

Return Value

None

Errors

None

Description

This routine sets the color map for color contour. For example,

```
vis.setCmap( )
```

Basic LaTex Tags

Below are some useful tags in the LaTex formatting system.

\centering	Push content to the center of the page.
\vfill	Add vertical space up to the end of the page.
\hfill	Add horizontal space up to the end of the line.
\vspace	Add vertical space.
\hspace	Add horizontal space.
\vline	Add vertical line.
\hline	Add horizontal line.
\newpage	Start a new page.
\clearpage	Clear page content.
\dotfill	Fill with dots.
\section{text}	Add text to a new section.
\subsection{text}	Add text to a new subsection.
\subsubsection{text}	Add text to a new sub sub section.
\ref{text}	Put a reference in the paragraph.
\label{text}	Add label to image, table, or equation for further reference.
\emph{text}	Write the text in emphasis shape.
\textrm{text}	Write the text in Roman font family.
\textsf{text}	Write the text in sans serif font family.
\texttt{text}	Write the text in teletype font family.
\textup {text}	Write the text in upright shape.
\textit{text}	Write the text in italic shape.
\textsl{text}	Write the text in slanted shape.
\textbf{text}	Write the text in bold shape.
\textsc{text}	Write the text in small capitals.

\textmd{text}	Write the text in medium weight.
\uline{text}	Write the text in with underline.
\uwave{text}	Write the text in with wavy underline.
\sout{text}	Write the text in with strike-out line.
\cite	Write the text in cited shape.
\tiny{text}	Write the text in 6 pt.
\scriptsize {text}	Write the text in 8 pt.
\footnotesize{text}	Write the text in 10 pt.
\small {text}	Write the text in 11 pt.
\normalsize{text}	Write the text in 12 pt.
\large{text}	Write the text in 14 pt.
\Large{text}	Write the text in 17 pt.
\LARGE {text}	Write the text in 20 pt.
\huge{text}	Write the text in 25 pt.
\HUGE{text}	Write the text in 25 pt.
eqn	Write an equation.
\backslash	Backslash.
^	Superscript, such as AcuSolve^{TM} = AcuSolveTM.
_	Subscript, such as x_i = xi.
****	Start new line.
\<space>	Force ordinary space.
\@	Following period ends sentence.
\,	Thin space.
\;	Thick space, math mode.
\:	Medium space, math mode.
\!	Negative thin space, math mode.
\>	Tab.

\<	Back tab.
|	Double vertical lines, math mode.

Python interprets "\" as an escape character. If you simply want to pass this to a function, such as required for the above LaTex controls, you must either prefix it with another "\", such as "\\newpage", or use raw Python string, such as r"\newpage".

The following characters have special meaning in text: # $ % & ~ _ ^ \ { }. To write back slash (\ ), use "\\textbackslash" and to write tilde (~), use "$\\sim$". Add a "\\" before any other character, if you simply want the character to be written:

#	The number (pound) sign is used to define use of arguments, for example, in the \newcommand command.
$	The dollar sign is used to delineate math and displaymath Environments.
%	The percent sign is used to insert Comments in the input file, and to allow line breaks without generating a space.
&	The ampersand is used to separate items in the array and tabular Environments.
~	The tilde generates a non breaking space. To create a tilde in the output, use \verb or the verbatim environment, or cheat by using \~{}, that is, placing a tilde accent over a blank letter.
_	The underscore is used to create subscripts.
^	The carat (circumflex) symbol generates superscripts. To create a carat in the output use \verb or the verbatim environment.
\{ }	The backslash and braces are used in command definitions, for enclosing command arguments, and for delimiting scopes of declarations.

Acupu Functions

28

This chapter covers the following:

appendCrds()

Append a new coordinate/usrIds to the end of an old set. Also, return an inverse map from the new usrIds to the extended vector.

Parameters

`newUsrIds` *(array)*
: An array of new user indices.

`newCrd` *(array)*
: An array of new coordinates.

`oldUsrIds` *(array)*
: An array of old user indices.

`oldCrd` *(array)*
: An array of old coordinates.

Return Value

`usrIds` *(array)*
: New user indice.

`crd` *(array)*
: New coordinates.

`newInvMap` *(array)*
: Inverse map from new usrIds to new 0 based index.

array2Str()

Convert an array to string for printing.

Parameters

`src` (array)
Source array.

Return Value

`str` (string)
String representation.

cksumArray()

Get an array cksum.

Parameters

`src` (array)
Source array.

Return Value

`cksum` (integer)
Array ckecksum.

cksumFile()

Get the file's cksum value.

Parameters

`fileName` *(string)*
The file name.

Return Value

`cksum` *(integer)*
The file's cksum.

crdOrg()

Get the center of a set of nodes.

Parameters

src (array)
Coordinates.

cnn (array)
List of connectivity nodes (optional).

Return Value

***org** (array)*
Center.

cs2Str()

Convert a clean string to a string.

Parameters

`src` (string)
Source string.

Return Value

`dst` (string)
Converted string.

decryptStr()

Decrypt a string.

Parameters

src (string)
Encrypted string.

Return Value

***dst** (string)*
Destination string.

dupNodeMap()

Return a nodal map for duplicate nodes.

Parameters

`crd` *(array)*
: An array of nodal coordinates.

`tol` *(integer)*
: Distance tolerance to consider duplicate nodes.

Return Value

map *(array)*
: Map to duplicate nodes.

elmGradField()

Get the average grad-field within each element.

Parameters

`crd` *(array)*
An array of nodal coordinates.

`cnn` *(array)*
An array of surface connectivity.

`vec` *(array)*
An array of nodal solution.

Return Value

gradVec *(array)*
An array of element gradients.

elmVolume()

Get the volume of each element.

Parameters

`crd` *(array)*
: An array of nodal coordinates.

`cnn` *(array)*
: An array of surface connectivity.

Return Value

`volume` *(array)*
: Ann array of element volumes.

encryptStr()

Encrypt a string.

Parameters

`src` (string)
Source string.

Return Value

`dst` (string)
Encrypted string.

getCnnNodes()

Get a unique list of nodes in a connectivity array.

Parameters

cnn (array)
: An array of connectivity.

Return Value

***nbc** (array)*
: Unique set of nodes in *cnn*.

getFileCnts()

Get file counts.

Parameters

`fileName` *(string)*
The file name.

Return Value

`nLines` *(integer)*
Number of lines (rows) in the file.

`minCols` *(integer)*
Min number of columns of the lines.

`maxCols` *(integer)*
Max number of columns of the lines.

`minWidth` *(integer)*
Min number of chars of the lines.

`maxWidth` *(integer)*
Max number of chars of the lines.

`size` *(integer)*
Total size (in number of chars).

getInvMap()

Build and return an inverse map from user number to 0 based index.

Parameters

`ursIds` *(array)*
A 1D array of user numbers.

Return Value

invMap *(array)*
A 2xn array of inverse map.

getMemoryUsage()

Get memory usage (from first import Acupu).

Parameters

None

Return Value

usage *(integer)*
Heap memory usage in chars.

getProIds()

Get all process IDs on the system.

Parameters

None

Return Value

procIds *(array)*
An array of pid (process ID) and ppid (parent pid).

getSipVoidPtrInt()

Get an integer from a voidPtr.

Parameters

obj (object)
Sip.voidPtr object.

fmt (string)
Format (combination of "vdi" for void/double/int).

Return Value

***val** (integer)*
Return value.

getSrfEdge()

Extract the external/feature edges of a surface set.

Parameters

`crd` *(array)*
: Coordinates.

`angle` *(real)*
: Feature angle.

`cnns` *(array)*
: List of surface connectivity.

Return Value

edges *(array)*
: Surface edges.

getSrfSplit()

Split a surface based on angle.

Parameters

`crd` (array)
Coordinates.

`cnns` (array)
Surface connectivity.

`angle` (real)
Feature angle.

Return Value

`setIds` (array)
Array of set IDs.

getVolSrf()

Extract the external surfaces (quad and triangular faces) of an element set.

Parameters

cnns (array)
List of connectivities.

Return Value

***tris** (array)*
Triangular face connectivity.

***quads** (array)*
Quad face connectivity.

invMap()

Map an integer or an array of integers from user to 0 based index.

Parameters

`invMap` *(array)*
The array returned by getInvMap(usrIds).

`source` *(array) or (integer)*
An int or 1D/2D array of integers.

Return Value

`destination` *(array)*
A mapped version of src.

licIsAltair()

Is this an Altair license?

Parameters

None.

Return Value

altair *(boolean)*
True if GWU licensing. False otherwise.

mapPbcFaces()

Map the faces based on periodicity type.

Parameters

PbcType (string)
Pbc type ("TRANSLATION", "ROTATION").

nSrfs (integer)
Number of faces in each group to be mapped.

areas1 (array)
A 1D array of source face areas.

areas2 (array)
A 1D array of destination face areas.

ctrs1 (array)
A 1D array of source face centers.

ctrs2 (array)
A 1D array of destination face centers.

trans (array)
A 1D array of translation vector (size 3).

axis_pt1 (array)
A 1D array of the first point on the axis of rotation.

axis_pt2 (array)
A 1D array of the second point on the axis of rotation.

angle (real)
Rotation angle (radians) to go from source to destination face.

tolerance (real)
Allowed tolerance while mapping the face, default = 1e.

Return Value

***pbcMap** (array)*
PBC map.

mergeCrds()

Merge a new coordinate/usrIds with an old set. Also, return an inverse map from the new usrIds to the extended vector. Merging is performed purely on usrIds. Any node in the new set with the same usrIds in the old array is mapped to the old set.

Parameters

`newUsrIds` *(array)*
: An array of new user indices.

`newCrd` *(array)*
: An array of new coordinates.

`oldUsrIds` *(array)*
: An array of old user indices.

`oldCrd` *(array)*
: An array of old coordinates.

Return Value

`usrIds` *(array)*
: New user indices.

`crd` *(array)*
: New coordinates.

nodalVolume()

Get the volume associated with each node.

Parameters

`crd` (array)
An array of nodal coordinates.

`cnn` (array)
An array of surface connectivity.

Return Value

***`volume`** (array)*
An array of nodal volumes.

orientSrf()

Reorient the surface connectivity according to the interior element.

Parameters

`elmIds` *(array)*
: An array of surface to element numbers.

`cnn` *(array)*
: An array of surface connectivity.

`pElemIds` *(array)*
: Ann array of parent element set element numbers.

`pCnn` *(array)*
: An array of parent element set connectivity.

`pInvMap` *(array)*
: InvMap of pElemIds (optional).

Return Value

`newCnn` *(array)*
: Reoriented surface connectivity.

pyt2Str()

Convert a clean string to a string.

Parameters

`src` (string)
Source string.

Return Value

`dst` (string)
Converted string.

readArrays()

Read a file into a set of arrays defined by the format.

Parameters

`fileName` ***(string)***
The file name.

`format` ***(tuple)***
A tuple that tells the array's information. For example, (('i',1),('d','*')) or ('d',1).

Return Value

`arrays` ***(tuple)***
A tuple of arrays.

readNastran()

Read a Nastran file.

Parameters

`fileName` *(string)*
: Nastran file name.

Return Value

`usrIds` *(array)*
: User IDs.

`crd` *(array)*
: Coordinates.

`srfs` *(list)*
: List of surfaces (ID, srfIds, cnn).

`cnns` *(list)*
: List of elements (ID, elemIds, cnn).

`lines` *(array)*
: Left-over lines.

readStl()

Read an `.stl` file.

Parameters

`fileName` ***(string)***
`.stl` file name.

`tol` ***(integer)***
Search tolerance.

Return Value

`usrIds` ***(array)***
User IDs.

`crd` ***(array)***
Coordinates.

`srfs` ***(list)***
List of surfaces (cnn, name).

setProgName()

Set program name for internal messages.

Parameters

`name` (string)
Program name.

Return Value

None

srf2Tri()

Convert a surface to triangle.

Parameters

`crd` *(array)*
: An array of coordinates.

`cnn` *(array)*
: An array of surface connectivity.

Return Value

newCnn *(array)*
: An array of tri3 surface connectivity.

srfLayOut()

Get a set of surface layout information.

Parameters

crd (array)
An array of nodal coordinates.

cnn (array)
An array of surface connectivity.

Return Value

***area** (integer)*
Total area.

***center** (array)*
Center of the surface.

***normDir** (array)*
Normal direction.

***dir1** (array)*
Inplane direction 1.

***dir2** (array)*
Inplane direction 2.

***bndBox** (array)*
Bounding box.

srfNodalNorm()

Compute the normal directions for the nodes of surface sets.

Parameters

`crd` *(array)*
: An array of nodal coordinates.

`cnns` *(array)*
: One or more surface connectivity sets.

Return Value

nodalNormals *(array)*
: Nodal normals.

str2Array()

Convert a string to array.

Parameters

`src` (array)
Source string (created by array2Str).

Return Value

`ary` (array)
Return array.

str2Cs()

Convert a string to a clean string.

Parameters

`src` (string)
Source string.

Return Value

`dst` (string)
Clean string.

str2Pyt()

Convert a string to a clean string.

Parameters

`src` (string)
Source string.

Return Value

`dst` (string)
Clean string.

usrMap()

Map an integer or an array of integers from 0 based index to user IDs.

Parameters

`usrIds` *(array)*
: An array of user indices.

`source` *(array) or (integer)*
: An int or 1D/2D array of integers.

Return Value

`destination` *(array)*
: A mapped version of src.

volLayOut()

Get a set of volume layout information.

Parameters

`crd` *(array)*
: An array of nodal coordinates.

`cnn` *(array)*
: An array of surface connectivity.

Return Value

volume *(integer)*
: Total volume.

center *(array)*
: Center of the surface.

bndBox *(array)*
: Bounding box.

writeArrays()

Write arrays in a file.

Parameters

`fileName` ***(string)***
: The file name.

`arrays` ***(array)***
: A list of arrays.

`delimiter` ***(string)***
: Column separator, (default=space) (optional).

`mode` ***(string)***
: w for write (default) or a for append.

Return Value

None

writeEnsightArray()

Write arrays into EnSight array format.

Parameters

`fileName` (string)
: EnSight data file format.

`title` (string)
: Title of the data set.

`dataname` (string)
: Name of the data set.

`vec` (array)
: Vector to output.

`usrIds` (array)
: User IDs (opt).

`binary` (boolean)
: Binary flag.

Return Value

None

writeStl()

Write an `.stl` file.

Parameters

`fileName` (string)
`.stl` file name.

`crd` (array)
Coordinates.

`srfs` (list)
List of surfaces (cnn, name).

Return Value

None

Intellectual Property Rights Notice

This Intellectual Property Rights Notice is exemplary, and therefore not exhaustive, of the intellectual property rights held by Altair Engineering Inc. or its affiliates. Software, other products, and materials of Altair Engineering Inc. or its affiliates are protected under laws of the United States and laws of other jurisdictions.

In addition to intellectual property rights indicated herein, such software, other products, and materials of Altair Engineering Inc. or its affiliates may be further protected by patents, additional copyrights, additional trademarks, trade secrets, and additional other intellectual property rights. For avoidance of doubt, copyright notice does not imply publication. Copyrights in the below are held by Altair Engineering Inc. or its affiliates. Additionally, all non-Altair marks are the property of their respective owners. If you have any questions regarding trademarks or registrations, please contact marketing and legal.

This Intellectual Property Rights Notice does not give you any right to any product, such as software, or underlying intellectual property rights of Altair Engineering Inc. or its affiliates. Usage, for example, of software of Altair Engineering Inc. or its affiliates is governed by and dependent on a valid license agreement.

Altair® HyperWorks®, a Design & Simulation Platform

Altair® AcuSolve® ©1997-2024

Altair® Activate® ©1989-2024

Altair® Automated Reporting Director™ ©2008-2022

Altair® Battery Damage Identifier™ ©2019-2024

Altair® CFD™ ©1990-2024

Altair Compose® ©2007-2024

Altair® ConnectMe™ ©2014-2024

Altair® DesignAI™ ©2022-2024

Altair® DSim™ ©2024

Altair® EDEM™ ©2005-2024

Altair® EEvision™ ©2018-2024

Altair® ElectroFlo™ ©1992-2024

Altair Embed® ©1989-2024

Altair Embed® SE ©1989-2024

Altair Embed®/Digital Power Designer ©2012-2024

Altair Embed®/eDrives ©2012-2024

Altair Embed® Viewer©1996-2024

Altair® e-Motor Director™©2019-2024

Altair® ESAComp® ©1992-2024

Altair® expertAI™ ©2020-2024

Altair® Feko® ©1999-2024

Altair® FlightStream® ©2017-2024

Altair® Flow Simulator™ ©2016-2024

Altair® Flux® ©1983-2024

Altair® FluxMotor® ©2017-2024

Altair® GateVision PRO™ ©2002-2024

Altair® Geomechanics Director™ ©2011-2022

Altair® HyperCrash® ©2001-2023

Altair® HyperGraph® ©1995-2024

Altair® HyperLife® ©1990-2024

Altair® HyperMesh® ©1990-2024

Altair® HyperMesh® CFD ©1990-2024

Altair® HyperMesh ® NVH ©1990-2024

Altair® HyperSpice™ ©2017-2024

Altair® HyperStudy® ©1999-2024

Altair® HyperView® ©1999-2024

Altair® HyperView Player® ©2022-2024

Altair® HyperWorks® ©1990-2024

Altair® HyperWorks® Design Explorer ©1990-2024

Altair® HyperXtrude® ©1999-2024

Altair® Impact Simulation Director™ ©2010-2022

Altair® Inspire™ ©2009-2024

Altair® Inspire™ Cast ©2011-2024

Altair® Inspire™ Extrude Metal ©1996-2024

Altair® Inspire™ Extrude Polymer ©1996-2024

Altair® Inspire™ Form ©1998-2024

Altair® Inspire™ Mold ©2009-2024

Altair® Inspire™ PolyFoam ©2009-2024

Altair® Inspire™ Print3D ©2021-2024

Altair® Inspire™ Render ©1993-2024

Altair® Inspire™ Studio ©1993-2024

Altair® Material Data Center™ ©2019-2024

Altair® Material Modeler™ ©2019-2024

Altair® Model Mesher Director™ ©2010-2024

Altair® MotionSolve® ©2002-2024

Altair® MotionView® ©1993-2024

Altair® Multi-Disciplinary Optimization Director™ ©2012-2024

Altair® Multiscale Designer® ©2011-2024

Altair® newFASANT™ ©2010-2020

Altair® nanoFluidX® ©2013-2024

Altair® NLVIEW® ©2018-2024

Altair® NVH Director™ ©2010-2024

Altair® NVH Full Vehicle™ ©2022-2024

Altair® NVH Standard™ ©2022-2024

Altair® OmniV™ ©2015-2024

Altair® OptiStruct® ©1996-2024

Altair® physicsAI™ ©2021-2024

Altair® PollEx™ ©2003-2024

Altair® PollEx™ for ECAD ©2003-2024

Altair® PSIM™ ©1994-2024

Altair® Pulse™ ©2020-2024

Altair® Radioss® ©1986-2024

Altair® romAI™ ©2022-2024

Altair® RTLvision PRO™ ©2002-2024

Altair® S-CALC™ ©1995-2024

Altair® S-CONCRETE™ ©1995-2024

Altair® S-FRAME® ©1995-2024

Altair® S-FOUNDATION™ ©1995-2024

Altair® S-LINE™ ©1995-2024

Altair® S-PAD™ ©1995-2024

Altair® S-STEEL™ ©1995-2024

Altair® S-TIMBER™ ©1995-2024

Altair® S-VIEW™ ©1995-2024

Altair® SEAM® ©1985-2024

Altair® shapeAI™ ©2021-2024

Altair® signalAI™ ©2020-2024

Altair® Silicon Debug Tools™ ©2018-2024

Altair® SimLab® ©2004-2024

Altair® SimLab® ST ©2019-2024

Altair® SimSolid® ©2015-2024

Altair® SpiceVision PRO™ ©2002-2024

Altair® Squeak and Rattle Director™ ©2012-2024

Altair® StarVision PRO™ ©2002-2024

Altair® Structural Office™ ©2022-2024

Altair® Sulis™ ©2018-2024

Altair® Twin Activate® ©1989-2024

Altair® UDE™ ©2015-2024

Altair® ultraFluidX® ©2010-2024

Altair® Virtual Gauge Director™ ©2012-2024

Altair® Virtual Wind Tunnel™ ©2012-2024

Altair® Weight Analytics™ ©2013-2022

Altair® Weld Certification Director™ ©2014-2024

Altair® WinProp™ ©2000-2024

Altair® WRAP™ ©1998-2024

Altair® HPCWorks®, a HPC & Cloud Platform

Altair® Allocator™ ©1995-2024

Altair® Access™ ©2008-2024

Altair® Accelerator™ ©1995-2024

Altair® Accelerator™ Plus ©1995-2024

Altair® Breeze™ ©2022-2024

Altair® Cassini™ ©2015-2024

Altair® Control™ ©2008-2024

Altair® Desktop Software Usage Analytics™ (DSUA) ©2022-2024

Altair® FlowTracer™ ©1995-2024

Altair® Grid Engine® ©2001, 2011-2024

Altair® InsightPro™ ©2023-2024

Altair® Hero™ ©1995-2024

Altair® Liquid Scheduling™ ©2023-2024

Altair® Mistral™ ©2022-2024

Altair® Monitor™ ©1995-2024

Altair® NavOps® ©2022-2024

Altair® PBS Professional® ©1994-2024

Altair® PBS Works™ ©2022-2024

Altair® Simulation Cloud Suite (SCS) ©2024

Altair® Software Asset Optimization (SAO) ©2007-2024

Altair® Unlimited™ ©2022-2024

Altair® Unlimited Data Analytics Appliance™ ©2022-2024

Altair® Unlimited Virtual Appliance™ ©2022-2024

Altair® RapidMiner®, a Data Analytics & AI Platform

Altair® AI Hub ©2023-2024

Altair® AI Edge™ ©2023-2024

Altair® AI Cloud ©2022-2024

Altair® AI Studio ©2023-2024

Altair® Analytics Workbench™ ©2002-2024

Altair® Graph Lakehouse™ ©2013-2024

Altair® Graph Studio™ ©2007-2024

Altair® Knowledge Hub™ ©2017-2024

Altair® Knowledge Studio® ©1994-2024

Altair® Knowledge Studio® for Apache Spark ©1994-2024

Altair® Knowledge Seeker™ ©1994-2024

Altair® IoT Studio™ ©2002-2024

Altair® Monarch® ©1996-2024

Altair® Monarch® Classic ©1996-2024

Altair® Monarch® Complete™ ©1996-2024

Altair® Monarch® Data Prep Studio ©2015-2024

Altair® Monarch Server™ ©1996-2024

Altair® Panopticon™ ©2004-2024

Altair® Panopticon™ BI ©2011-2024

Altair® SLC™ ©2002-2024

Altair® SLC Hub™ ©2002-2024

October 7, 2024

Technical Support

Altair's support resources include engaging learning materials, vibrant community forums, intuitive online help resources, how-to guides, and a user-friendly support portal.

Visit Customer Support to learn more about our support offerings.

Index

Numerics

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

Z