LS-PrePost Scripting Command Language, Python Language and Data Center

Version	Date	Author	Changes
Rev.1	July 29,2015	Philip Ho	Base Version
Rev.2	Aug 8, 2016	Philip Ho	
Rev.3	Sept 19, 2017	Philip Ho	
Rev.4	Dec 6, 2018	Luo Liangfeng	Component Name List Update
Rev.5	Dec 18, 2018	Philip Ho	Add Example description
Rev.6	June 10, 2019	Luo Liangfeng	Component Name list Update
Rev.7	June 26, 2020	Luo	Add Element Deletion array extraction
Rev.8	August 16, 2020	He Xi	Add description about Python scripting

Introduction

The LS-PrePost Scripting Command Language (SCL) is designed to allow users to write their own script to perform data manipulation for many different purposes. The LS-PrePost SCL is a C like computer language that is executed inside LS-PrePost. The user can execute LS-PrePost commands, retrieve LS-DYNA results, apply LS-DYNA data center extraction functions, and extract results from LS-DYNA d3plot files or model data from the keyword input file. Also, perform additional operations to the extracted data that were not developed within LS-PrePost. The resultant data can be output to the LS-PrePost message file, or user created file, or send it back to LS-PrePost for fringing or plotting. This document describes the application functions interface (API) and how to use SCL along with some example scripts for various operations.

The SCL was first developed with a C-Like computer language. Since LS-PrePost 4.8, Python language was also used in SCL. The user can use Python scripting to do somethings as same as above mentioned. The Python modules in LS-PrePost include DataCenter (provides get_data) and LsPrePost (provides tools of LS-PrePost, like fring_dc_to_model, execute_command, save_dc_to_file, etc.…). The user can take advantage of Python's rich third-party libraries to accomplish their own special tasks.

SCL API Functions

void **ExecuteCommand**(char *cmd)
Purpose: Execute a LS-PrePost command
Input: cmd - a string contains the LS-PrePost command
Return: none
….....

void **Echo**(char *string)
Purpose: Display the text in the LS-PrePost message dialog and in the lspost.msg file
Input: string - text string
Output: none
Return: none
….....

Int **SCLSwitchStateTo**(Int ist)
Purpose: Switch the current state to a specified state
Input: ist - state number, 0 > ist <= largest state
Output: none
Return: flag, 1=success, 0=fail
….....

Int **SCLGetDataCenterInt**(char *parameter_name)
Purpose : Get an integer scalar value from the model. See user parameter name list below for the available scalar values that can be retrieved.
Input: parameter name
Output: none
Return: integer value
….....

Float **SCLGetDataCenterFloat**(char *parameter_name, Int typecode, Int index, Int ipt)
Purpose : Get a floating point value from the model. See Data Center Parameter Name List below for the available floating point value that can be retrieved.
Input: parameter name
Typecode - Element or node type, constants such as "SOLID", "SHELL", "BEAM", "TSHELL", "NODE","SPHNODE" can be used. Not used for the parameter_name, enter as zero.

Index - Index to element.
Ipt - Integration points or layer. Valid for getting component values from element shell, beam and tshell, such as "MID", "INNER", "OUTER", 1, 2, etc, can be used. Also valid for solid fully integrated, base-1,such as "1","2"..."8" can be used.

Output: none

Return: float value

….....

void **SCLGetDataCenterVector**(char *parameter_name, Int externalid, NDCOOR *result)

Purpose: Get a vector from the model data, a vector such as nodal coordinate, displacement vector, velocity vector, acceleration vector, and so on.

```
typedef   ndcoor {
    Float xyz[3];
} NDCOOR;
```

Input: parameter_name
Externalid - Index in the component value array.

Output: result: vector

Return: none

….....

void **SCLGetDataCenterTensor**(char *parameter_name, Int elementType, Int externalid, Int ipt, TENSOR *result)

Purpose: Get stress/strain tensor from the model data.

```
typedef tensor{
    Float xyz[6];
} TENSOR;
```

Input: parameter_name- "global_stress" and "global_strain" can be used.
Typecode - Element type or node type, constant such as "SOLID", "SHELL", "BEAM", "TSHELL", "NODE","SPHNODE" can be used. If it parameter is not needed, please give it zero.
Ipt - Integration points or layer. Valid for getting component values from element shell, beam and tshell, such as "MID", "INNER", "OUTER", 1, 2, etc, can be used. Also valid for solid fully integrated, base-1,such as "1","2"..."8" can be used.

Output: result - tensor

Return: none

….....

Int **SCLGetDataCenterIntArray**(char *parameter_name, Int **results, Int type, Int id)

Purpose: Get an integer array from the model data

Input: parameter_name
type - element type(SHELL, SOLID, TSHELL, etc).
id - element/part/nodeset/elementset id.

Output: results - integer array

Return: array size

….....

Int **SCLGetDataCenterFloatArray**(char *parameter_name, Int typecode, Int ipt, Float **results)

Purpose: Get a floating point array from the model data

Input: parameter_name
Typecode - Element type or node type, constant such as "SOLID", "SHELL", "BEAM", "TSHELL", "NODE","SPHNODE" can be used. Not used for the parameter_name, enter zero.
Ipt - Integration points or layer. Valid for getting component values from element shell, beam and tshell, such as "MID", "INNER", "OUTER", 1, 2, etc, can be used. Also valid for solid fully integrated, base-1,such as "1","2"..."8" can be used.

Output: results - float array

Return: array size

….....

Int **SCLGetDataCenterVectorArray**(char *parameter_name, NDCOOR **results)

Purpose: Get a vector array from the model data

Input: parameter_name

Output: results: vector array

Return: array size

….....

Int **SCLGetDataCenterTensorArray**(char *parameter_name, Int typecode, Int ipt, TENSOR **results)

Purpose: Get a stress or strain tensor array from the model data

Input: parameter_name- "global_stress" and "global_strain" can be used.
Typecode - Element type or node type, constant such as "SOLID", "SHELL", "BEAM", "TSHELL", "NODE","SPHNODE" can be used. If it parameter is not needed, please give it zero.

Ipt - Integration points or layer. Valid for getting component values from element shell, beam and tshell, such as "MID", "INNER", "OUTER", 1, 2, etc, can be used. Also valid for solid fully integrated, base-1,such as "1","2"..."8" can be used.

Output: results: tensor array

Return: array size

….....

void **SCLFringeDCToModel**(Int typecode, Int avg_opt, Int num, Float* data, Int ist, char *label)

Purpose: Fringe the data center array to current model.

Input: typecode - Node or element type, constant such as "NODE", "0","SOLID","BEAM","SHELL","TSHELL","SPHNODE" can be used.

avg_opt – nodal averaging option, 0=none, 1=nodal

num - array size

data - Float array

ist - State number which data will be assigned

Label - Name of the fringing data to be shown on fringe plot

Output: none

Return: none

….....

void **SCLSaveDCToFile**(char *filename, Int num, Float *data)

Purpose: Save the data center array to file.

Input: filename - name of output file.

Num - array size

Data - Float array

Output: none

Return: none

….....

Int **SCLCmdResultGetValueCount**(void)

Purpose: Get number of results from a LS-PrePost command.

Input: none

Output: none

Return: Number of command results.

….....

Int **SCLCmdResultGetValue**(Int i, Int *type, Int *iv, Float *v)

Purpose: Get value of command results.
Input: i - Index of command results. (starting from 0)
Output: type - Type of command results. 0=integer, 1=float
depends on the data type, one of the following will be used
iv - the integer result
v - the floating point result in double word.
Return: status flag, 1=success, 0=fail
…….

Int **SCLGetUserId**(Int iid, Int dtype)
Purpose: Get the user defined id given an internal id.
Input: iid - Internal id, starting with 0
Dtype - Data type, constant such as NODE, PART, SHELL, SOLID, TSHELL, BEAM, DISCRETE, SEATBELT, SPHNODE can be used.
Output: none
Return: user id
…….

Int **SCLGetInternalID**(Int uid, Int dtype)
purpose: Get internal id given an user id.
Input: uid - User id.
Dtype - Data type, constant such as NODE, PART, SHELL, SOLID, TSHELL, BEAM, DISCRETE, SEATBELT, SPHNODE can be used.
Output: none
Return: Internal id, starting with 0
…….

Int **SCLGetUserPartIDFromUserElementID**(Int uid, Int dtype)

purpose: Get user part id given an user element id.
Input: uid - User id.
Dtype - Data type, constant such as SHELL, SOLID, TSHELL, BEAM and SPHNODE can be used.
Output: none
Return: User part id.
…….

Int **SCLCheckIfPartIsActiveU**(Int uid)
Purpose: Check if a part is active (visible) given a user defined part id
Input: uid - User id.
Output: none
Return: visibility flag, 1 is visible, 0 is invisible.
…….....

Int **SCLCheckIfPartIsActiveI**(Int iid)
Purpose: Check if a part is active (visible) given an internal part id.
Input: iid - internal id.
Output: none
Return: visibility flag, 1 is visible, 0 is invisible.
…….....

Int **SCLCheckIfElementIsActiveU**(Int uid, Int dtype)
Purpose: Check if an element part is active (visible) given an users element id.
Input: uid - User id.
Dtype - Data type, constant such as SHELL, SOLID, TSHELL, BEAM, SPHNODE can be used.
Output: none
Return: visibility flag, 1 is visible, 0 is invisible
…….....

Int **SCLCheckIfElementisActiveI**(Int iid, Int dtype)
Purpose: Check if an element is active (visible) given an internal id.
Input: iid - Internal id.
Dtype - Data type, constant such as SHELL, SOLID, TSHELL, BEAM, SPHNODE can be used.
Output: none
Return: visibility flag, 1 is visible, 0 is invisible.
…….....

Int **SCLInquiryPartTypeU**(Int uid)
Purpose: Check the type of element for a given user defined part id.
Input: uid – user defined external ID for part
Output: none
Return: element type code:
1 – BEAM, 2 – SHELL, 3 – SOLID, 4 – TSHELL
5 – NRBODY, 6 – MASS, 7 – DISCRETE, 8 – SEATBELT
9 – INERTIA, 10 – RGSURF, 11 – SPHNODE,

12 – FLUID, 13 – NURBSPATCH, 14 - PARTICLE

…….

Int **SCLInquiryPartTypeI**(Int iid)
Purpose: Check the type of element for a given internal part id.
Input: iid – internal ID for part
Output: none
Return: element type code:
1 – BEAM, 2 – SHELL, 3 – SOLID, 4 – TSHELL
5 – NRBODY, 6 – MASS, 7 – DISCRETE, 8 – SEATBELT
9 – INERTIA, 10 – RGSURF, 11 – SPHNODE,
12 – FLUID, 13 – NURBSPATCH, 14 - PARTICLE
…….

void **SCLGetModelDirectory**(char *dir)
Purpose: Get model directory.
Input: dir– the directory where model is put
Output: none
…….

Int **SCLGetDataCenterString**(char *parameter_name, Int iid, char *result)
Purpose: Get string from model data.
Input: parameter_name(like “part_name”, “time”).
iid - internal id(zero-based), not used if the parameter_name is time.
Output: result - the string to be obtained.
Return: 1 is valid, 0 invalid.
…….

Geometry object type

The geometry object type is used to define the type of the geometry entities, the symbols are defined here in the following table, use the exact capital letter as shown in the table to define the type in the API functions

OBJ_SOLID
OBJ_SHELL

OBJ_FACE
OBJ_WIRE
OBJ_EDGE
OBJ_VERTEX

Geometry related Functions

Int **SCLCalcGeomArea**(Int id, Int obj_type, Float *result)
Purpose: Calculate the area of a geometry entity.
Input: id - geometry entity id.
obj_type - Geometry object type, see geometry object type table above for definition
Output: result - area value
Return: status of calculation, 1=success, 0=fail
…….....

Int **SCLCalcGeomVolume**(Int id, Int obj_type, Float *result)
Purpose: Calculate the volume of a geometry entity.
Input: id - geometry element entity id.
obj_type - Geometry object type, see geometry object type table above for definition
Output: result - volume value.
Return: status of calculation, 1=success, 0=fail
…….....

Int **SCLCalcGeomLength**(Int id, Int obj_type, Float *result)
Purpose: Calculate the length of geometry entity.
Input: id - Geometry entity id.
obj_type - Geometry object type, see geometry object type table above for definition
Output: result - length value.
Return: status of calculation, 1=success, 0=fail
…….....

Float **SCLCalcShapeBoundBox**(Int id, Int obj_type, Float *minPnt, Float* maxPnt)
Purpose: Calculate the bounding box of geometry entity.
Input: id – Geometry entity id.
obj_type – Geometry object type, see geometry object type table
Output: minPnt – min point of the shape’s bounding box

maxPnt – max point of the shape's bounding box
Return: status of calculation, 1=success, 0=fail

…….

Float **SCLCalcShapesBoundBox**(Int* ids, Int* obj_types, Int n, Float *minPnt, Float* maxPnt)
Purpose: Calculate the bounding box of geometry entities.
Input: ids – Geometry entities' id.
obj_types – Geometry entities' type.
n – Number of geometry entities.

Output: minPnt – min point of the shapes' bounding box
maxPnt – max point of the shapes' bounding box
Return: status of calculation, 1=success, 0=fail

…….

Int **SCLGeomGetAllShapeIDs**(Int **ids, Int obj_type, Int bLocal)
Purpose: Get ids of all geometry faces.
Input: ids – ids array.
obj_type – Geometry object type, see geometry object type table above for definition
bLocal – if bLocal=1, get all shape including local shape; if bLocal = 0, just get independent shape (non-local) only. A local shape is a shape from a parent object
Output: none
Return: Number of faces.

…….

Int **SCLMeasureGeomShellSolid**(Int id, Int obj_type, Int* numOfFaces, Float* area, Float* volume, Int* bClosed, Float* cgPnt, Float* halfInertiaMatrix, Float* principalVector)
Purpose: Measure shell or solid shape's geometry information
Input: id – the shell or solid shape's ID.
type – the shape's type.
Output: numOfFaces – number faces of the shell or solid shape.
area – area of the shell or solid shape.
volume – volume of the shell or solid shape.
bClosed – close tag of the shell or solid shape.
cgPnt – centre mass point.
halfInertiaMatrix – half inertia matrix. The array is Ixx, Ixy, Ixz,

Iyy, Iyz, Izz.
principalVector – principal vector.
Return: 1=success, 0=fail

….

void **SCLGeomMeasureToText**(Int id, Int obj_type)
Purpose: Issues a geometry measurement for a specified geometry shape
Input: id - the geometry entity id
obj_type - Geometry object type, see geometry object type table above for definition
Output: none
Return: none

….

void **SCLGeomMeasureToText2**(Int id1, Int obj_type1, Int id2, Int obj_type2)
Purpose: Issues a geometry measurement for two specified geometry shapes
Input: id1 - the geometry entity id for shape 1
obj_type1 - Geometry object type for shape 1, see geometry object type table above for definition
id2 - the geometry entity id for shape 2
obj_type2 – Geometry object type for shape 2
Output: none
Return: none

….

Int **SCLGeomMeasureGetValueCount()**
Purpose: Get number of return values from Geometry measurement operation.
Input: none
Output: none
Return: Number of return values.

….

void **SCLGeomMeasureGetValue**(Int i, char *p1, char *p2)
Purpose: Get the geometry measurement value for the ith entry
Input: i - the ith entry
Output: p1 - string contains the name of the measurement
P2 - string contains the values of the measurement, can be multiple floating point numbers, user needs to decode it according to the name
Return: none

…….

Int **SCLGetParentEntityID**(Int childID, Int obj_type1, Int obj_type2)
Purpose: Get the parent entity ID given the child entity ID.
Input: childID – the child entity ID
obj_type1 – object type of the child
obj_type2 – object type of parent
Output: none
Return: parent entity ID
…….

Int **SCLGetParentEntityIDs**(Int childID, Int obj_type1, Int obj_type2, Int **ids)
Purpose: Get the parent entity's ids given the child entity ID.
Input: childID – the child entity ID
obj_type1 – object type of the child
obj_type2 – object type of parent
Output: ids – array contains parent ids. (a child can has multiple parents, e.g. an edge line will have multiple faces)
Return: number of parent entities

…….

Int **SCLGetEntityMaxID**(Int type)
Purpose: Get the max ID of the specified entity type
Input: type – entity type
Return: max ID of the specified entity type
…….

Int **SCLGetSubEntityIDs**(Int parentID, Int obj_type1, Int obj_type2, Int **ids)
Purpose: Get the children entity ids given the parent entity ID
Input: parentID – the parent entity ID
obj_type1 – object type of parent entity
obj_type2 – object type of the sub-entity
Output: ids – array contains children ids
Return: number of children

…….

Int **SCLIsSonParentRelationship**(Int childID,Int obj_type1, Int parentID, Int obj_type2)

Purpose: Check if the child entity belongs to a parent entity
Input: childID – the child entity ID
obj_type1 – object type of the child entity
parentID – the parent entity ID
obj_type2 – object type of the parent entity
Output: none
Return: True or False

…….....

Int **SCLMidPlaneSearchFacePairs**(Int ID,Int obj_type, Int bStrict, Int **ids)
Purpose: Find the mid-plane faces pairs for a given solid entity
Input: ID – the solid entity ID
obj_type – object type, must be OBJ_SOLID
bStrict – key for setting the searching tolerance, 0=relax, 1=strict
Output: face ids array, 2 ids make up a pair
Return: number of pairs

…….....

Int **SCLPropagateFacesByAngle**(Int* IDs, Int nFace, Float angleTol, Int** FaceIDs, Int* FaceNum)
Purpose: Given several seed faces, find the same level faces by angle. This is API is from “Shell by Angle” in “Middle Surface” dialog.
Input: IDs – several seed faces ID
nFace – number of seed faces
angleTol – angle tolerance to propagate
Output:
FaceIDs – several seed faces ID
FaceNum – number of propagated faces
Return: 0: no propagated faces found; 1: find propagated faces
…….....

Int **SCLSearchSimilarShapes**(Int* IDs, Int* Types, Int n, Float relativeRatio, Int bByDisMeasure, Int** similarShapeIDs, Int** similarTypes, Int* nSimilar)
Purpose: Search the looks like shapes with the seed shapes.
Input: IDs – seed shapes ID
Types –seed shapes’ type
n – number of seed shapes
relativeRatio – error ratio, the value is in (0, 1), default value is 0.003

bByDisMeasure – search by distance measure only, default is 0.

Output:

similarShapeIDs – similar shapes' ID after searching
similarTypes – similar shapes' type
nSimilar – number of similar shapes

Return: number of similar shapes

….

void **SCLDeleteModel**()
Purpose: Delete all components

….

void **SCLDeleteAllShape**()
Purpose: Delete all geometry shapes

….

void **SCLDeleteAllFEMPart**()
Purpose: Delete all FEM parts

….

void **SCLDeleteAssembly**(Int assemblyID)
Purpose: Delete assembly by ID
Input: assemblyID – assembly's ID

….

void **SCLDeleteAssemblyShape**(Int assemblyID)
Purpose: Delete all shapes in specified assembly
Input: assemblyID – assembly's ID

….

void **SCLDeleteAssemblyRefGeom**(Int assemblyID)
Purpose: Delete all reference geometry in specified assembly
Input: assemblyID – assembly's ID

….

void **SCLDeleteAssemblyFEMPart**(Int assemblyID)
Purpose: Delete all FEM parts in specified assembly
Input: assemblyID – assembly's ID

….

void **SCLDeleteGPart**(Int gpartID)

Purpose: Delete the specified GPart
Input: gpartID – GPart's ID

….

void **SCLDeleteGPartShape**(Int gpartID)
Purpose: Delete all shapes in specified GPart
Input: gpartID – GPart's ID

….

void **SCLDeleteGPartFEMPart**(Int gpartID)
Purpose: Delete all FEM parts in specified GPart
Input: gpartID – GPart's ID

….

void **SCLDeleteEntity**(Int* IDs, Int* types, Int num)
Purpose: Delete entities (shapes, FEM parts and reference geometry) by ID and Type
Input: IDs – entities' ID
types – entities' type
num – entities' number

….

void **SCLDeleteFEMParts**(Int* IDs, Int num)
Purpose: Delete FEM parts
Input: IDs – parts' ID
num – parts' number

….

void **SCLCopyModel**(Int toAssemblyID, Int toGPartID)
Purpose: Copy all shapes and FEM parts to specified assembly and GPart
Input: toAssemblyID – the specified assembly ID. -1: copy to a new assembly
toGPartID – the specified GPart ID. -1: copy to a new GPart

….

void **SCLCopyAssembly**(Int fromAssemblyID, Int toAssemblyID)
Purpose: Copy all entities(shapes, reference geometry and FEM parts) in some assembly to another assembly

Input: fromAssemblyID – the source assembly ID
toAssemblyID – the destination assembly ID. -1: copy to a new assembly

…….

void **SCLCopyAssemblyShape**(Int fromAssemblyID, Int toAssemblyID)
Purpose: Copy all shapes in some assembly to another assembly
Input: fromAssemblyID – the source assembly ID
toAssemblyID – the destination assembly ID. -1: copy to a new assembly
…….

void **SCLCopyAssemblyRefGeom**(Int fromAssemblyID, Int toAssemblyID)
Purpose: Copy all reference geometry in some assembly to another assembly
Input: fromAssemblyID – the source assembly ID
toAssemblyID – the destination assembly ID. -1: copy to a new assembly

…….
void **SCLCopyAssemblyFEMPart**(Int fromAssemblyID, Int toAssemblyID)
Purpose: Copy all FEM parts in some assembly to another assembly
Input: fromAssemblyID – the source assembly ID
toAssemblyID – the destination assembly ID. -1: copy to a new assembly

…….

void **SCLCopyGPart**(Int fromGpartID, Int toGPartID)
Purpose: Copy all entities in some GPart to another GPart
Input: fromGpartID – the source GPart ID
toGPartID – the destination GPart ID. -1: copy to a new GPart
…….

void **SCLCopyGPartShape**(Int fromGpartID, Int toGPartID)
Purpose: Copy all shapes in some GPart to another GPart
Input: fromGpartID – the source GPart ID
toGPartID – the destination GPart ID. -1: copy to a new GPart
…….

void **SCLCopyGPartFEMPart**(Int fromGpartID, Int toGPartID)
Purpose: Copy all FEM parts in some GPart to another GPart
Input: fromGpartID – the source GPart ID
toGPartID – the destination GPart ID. -1: copy to a new GPart

….

void **SCLCopyEntity**(Int* ids, Int* types, Int num)
Purpose: Copy entities
Input: ids – the source entities' ID
types – the source entities' type
num – number of source entities

….

void **SCLCopyFEMParts**(Int* ids, Int num)
Purpose: Copy FEM parts
Input: ids – the source parts' ID
num – number of source parts

….

void **SCLHoleManage_Analysis**()
Purpose: Start analysis inner hole and outer hole in geometry shapes

….

void **SCLHoleManage_AnalysisShape**(Int ID, Int type)
Purpose: Start analysis inner hole and outer hole in specified face, shell or solid shape

Input: ID – the face, shell or solid's ID
type – the shape's type

….

Int **SCLHoleManage_GetInnerHoleCount**()
Purpose: After analysis, get number of inner holes (inner loop from face)
Return: number of inner hole

….

Int **SCLHoleManage_GetOutHoleCount**()
Purpose: After analysis, get number of out holes (out hole is grouped by multiple faces)

Return: number of out hole

…….

Int **SCLHoleManage_GetInnerHoleInfor**(Int holeID, char** holeName, Int* holeWireID, Int** holeEdgeIDs, Int* holeEdgeCount, Float* size)
Purpose: After analysis, get the inner hole geometry information
Input: holeID – the inner hole's ID
Output: holeName – the inner hole's name
holeWireID – wire ID of the inner hole
holeEdgeIDs – edges' ID of the inner hole
holeEdgeCount – number of edges from the inner hole
size – diagonal length of the inner hole's bounding box
Return: 0: invalid inner hole; 1: success

…….

Int **SCLHoleManage_GetOutHoleInfor**(Int holeID, char** holeName, Int** holeEdgeIDs, Int* holeEdgeCount, Float* size, Int* bFilled)
Purpose: After analysis, get the out hole geometry information
Input: holeID – the out hole's ID
Output: holeName – the out hole's name
holeEdgeIDs – edges' ID of the out hole
holeEdgeCount – number of edges from the out hole
size – diagonal length of the inner hole's bounding box
bFilled – out hole filled tag (some large out hole should not be filled)
Return: 0: invalid out hole; 1: success

…….

Int **SCLHoleManage_FillHole**(Int bInnerHole, Int holeID)
Purpose: After analysis, fill specified inner or out hole
Input: bInnerHole – tag of inner hole or out hole
holeID – the hole's ID
Return: 0: invalid out hole; 1: success

…….

void **SCLHollowManage_Analysis**(Int bSimpleHollowOnly)
Purpose: Start analysis hollow in geometry shapes
Input: bSimpleHollowOnly – a tag to check simple or complex hollow

…….

void **SCLHollowManage_AnalysisShape**(Int id, Int type, Int

bSimpleHollowOnly)
Purpose: Start analysis hollow in geometry shell or solid
Input: id – the shell or solid shape's ID
type – the shape's type
bSimpleHollowOnly – a tag to check simple or complex hollow
…….

Int **SCLHollowManage_GetHollowCount**()
Purpose: After analysis, get number of hollow
Return: number of hollow
…….
Int **SCLHollowManage_GetHollowInfor**(Int hollowID, char** hollowName, Int** hollowFaceIDs, Int* holeFaceCount, Float* size)
Purpose: After analysis, get the hollow geometry information
Input: hollowID – the hollow's ID
Output: hollowName – the hollow's name
hollowFaceIDs – faces' ID of the hollow
holeFaceCount – number of faces from the hollow
size – diagonal length of the hollow's bounding box
Return: 0: invalid hollow; 1: success
…….

Int **SCLHollowManage_FillHollow**(Int hollowID)
Purpose: After analysis, fill one hollow
Input: hollowID – the hollowID's ID

Return: 0: failed; 1: success

…….

Int **SCLHollowManage_FillAll**()
Purpose: After analysis, fill all hollows
Return: 0: failed; 1: success

…….

Int **SCLSetDBEntityColor**(Int ID, Int type, Float* color)
Purpose: Set entity's color
Input: ID – the entity' ID
type – the entity' type

color – the color array (size 4, the last component is transparency)
Return: 0: failed; 1: success

….....

Int **SCLGetDBEntityColor**(Int ID, Int type, Float* color)
Purpose: Get entity's color
Input: ID – the entity' ID
type – the entity' type
Output: color – the color array (size 4, the last component is transparency)
Return: 0: failed; 1: success

LS-PrePost Scripting Command Language specifics and limitations

LSPP-SCL is almost like the C programming language with the following exceptions:

1. Combined assignments such as i++, i--, --i, ++i, i+=, i*=; are not supported, must use i=i+1; i=i-1; i=i+n; i=i*x; i=i/n;
2. For integer data declaration, you must use "Int" not "int".
3. For floating point number declaration, you must use "Float", not "float".
4. Do not type cast data conversion, e.g. Int i; Float x;
   i = x; (correct way), i = (Int)x; (not supported)
5. Switch case: do... while - not supported
6. Conditional operator: (boolean) ? : - not supported

How to use Scripting Command Language in LS-PrePost

There are 2 ways to execute the scripting command language file:
1. Run it with the regular LS-PrePost command, or within the command file, use the runscript command to execute the scl file, parameters can also be passed to the script.

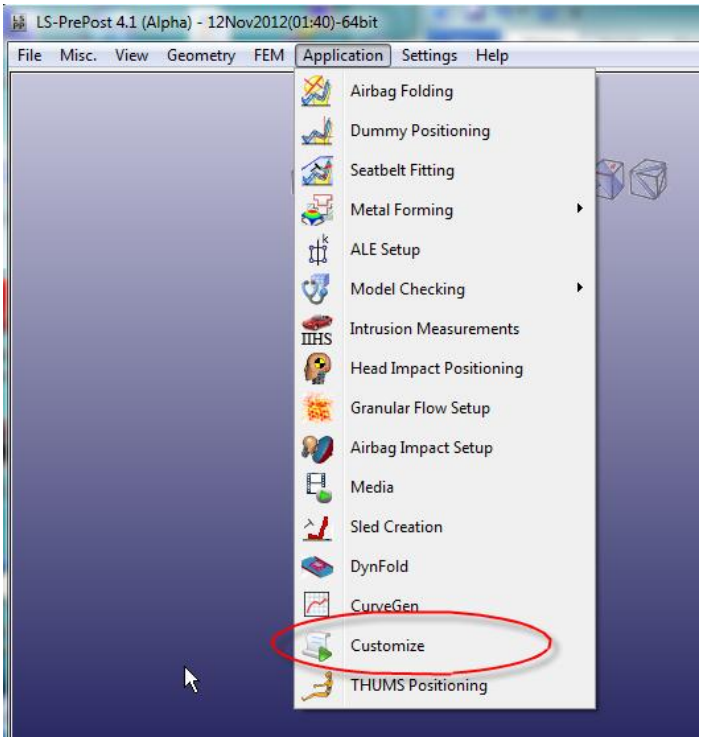
The syntax:
runscript "sclfilename" optional parameters

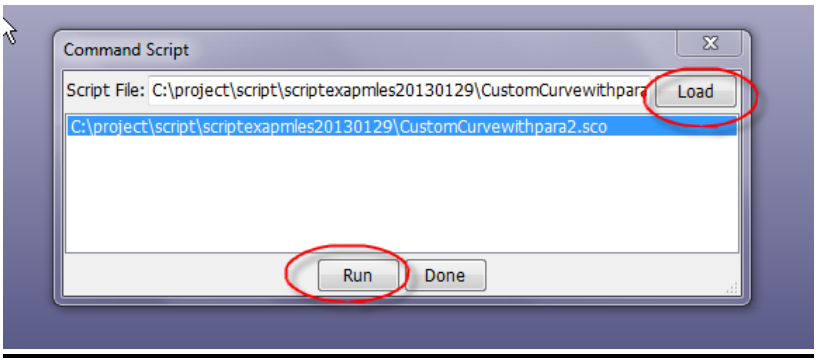
Example: the following command file will execute the script that creates a X-Y

curve with the parameters defined in the command file.

```
parameter  pa 9.0E+07
parameter  pb 7000.0
parameter  pc  4.0E+07
parameter  npt  300
parameter  xmin  0.0
parameter  xmax  0.00126
runscript "customcurve.scl"  &npt &pa &pb &pc &xmin &xmax
```

2. Run it by going to the Application pull down menu, select “Customize” and in the pop up dialog, click “Load” to load the SCL file, then click “Run” to execute. Running script this way parameter cannot be passed to the script file, as shown in the following pictures:





Python API Functions

LsPrePost module:

def **execute_command**(cmd)

    pass

Purpose: Execute a LS-PrePost command
Input: cmd - a string contains the LS-PrePost command
Return: none

….....

def **echo**(string)

    pass

Purpose: Display the text in the LS-PrePost message dialog and in the lspost.msg file
Input: string - text string
Output: none
Return: none

….....

def **switch_state**(ist)

    pass

Purpose: Switch the current state to a specified state
Input: ist - state number, 0 > ist <= largest state
Output: none
Return: flag, 1=success, 0=fail

….....

def **fringe_dc_to_model**(typecode, avg_opt, num, data, ist, label)

    pass

Purpose: Fringe the data center array to current model.
Input: typecode - Node or element type, constant such as "NODE",

"0","SOLID","BEAM","SHELL","TSHELL","SPHNODE" can be used. (**from DataCenter import Type**)
avg_opt – nodal averaging option, 0=none, 1=nodal
num - array size
data - Float array
ist - State number which data will be assigned
Label - Name of the fringing data to be shown on fringe plot

Output: none
Return: none

….....

def **save_dc_to_file**(filename, num, data)

```
pass
```

Purpose: Save the data center array to file.
Input: filename - name of output file.
num - array size
data - Float array
Output: none
Return: none

….....

def **cmd_result_get_value_count**()

```
pass
```

Purpose: Get number of results from a LS-PrePost command.
Input: none
Output: none
Return: Number of command results.

….....

def **cmd_result_get_value**(i)

```
pass
```

Purpose: Get value of command results.
Input: i - Index of command results. (starting from 0)
Output: depends on the data type, one of the following will be used
the integer result
the floating point result in double word.

Return: status flag, 1=success, 0=fail

…….

def **check_if_part_is_active_u**(uid)

pass

Purpose: Check if a part is active (visible) given a user defined part id
Input: uid - User id.
Output: none
Return: visibility flag, 1 is visible, 0 is invisible.

…….

def **check_if_part_is_active_i**(iid)

pass

Purpose: Check if a part is active (visible) given an internal part id.
Input: iid - internal id.
Output: none
Return: visibility flag, 1 is visible, 0 is invisible.

…….

def **check_if_element_is_active_u**(uid, type)

pass

Purpose: Check if an element is active (visible) given an users element id.
Input: uid - User id.
type - Data type, constant such as SHELL, SOLID, TSHELL, BEAM, SPHNODE can be used(**from DataCenter import Type**).
Output: none
Return: visibility flag, 1 is visible, 0 is invisible

…….

def **check_if_element_is_active_i**(iid, dtype)

pass

Purpose: Check if an element is active (visible) given an internal id.
Input: iid - Internal id.
type - Data type, constant such as SHELL, SOLID, TSHELL, BEAM, SPHNODE can be used(**from DataCenter import Type**).

Output: none
Return: visibility flag, 1 is visible, 0 is invisible.

…….

DataCenter module:

def **get_data**(parameter_name, type=-1, id=-1, ipt=-999, ist=-1)
    pass

Purpose: Get data from model.
Input: parameter_name(like “part_name”, “time”).
Keyword arguments(ignore if not necessary):

1. type - Data type, constant such as NODE, PART, SHELL, SOLID, TSHELL, BEAM, SPHNODE can be used(**from DataCenter import Type**).
2. id - internal id(zero-based).
3. ipt - Integration points or layer. Valid for getting component values from element shell, beam and tshell(**from DataCenter import Ipt**), such as "MEAN", "UPPER", "LOWER" can be used. Also valid for solid fully integrated, base-1,such as "1","2"..."8" can be used.
4. ist - State number which data will be extracted (starting from 1)

Return: result

…….

```
class Type(Enum):
    PART = 0
    BEAM = 1
    SHELL = 2
    SOLID = 3
    TSHELL = 4
    SPHNODE = 11
    NODE = 14

class Ipt(Enum):
    MAX = -4
    UPPER = -3
    LOWER = -2
```

```
    MIDDLE = -1


class Vector():
    def x(self):
        pass
    def y(self):
        pass
    def z(self):
        pass


class Tensor():
    def x(self):
        pass
    def y(self):
        pass
    def z(self):
        pass
    def xy(self):
        pass
    def yz(self):
        pass
    def zx(self):
        pass
```

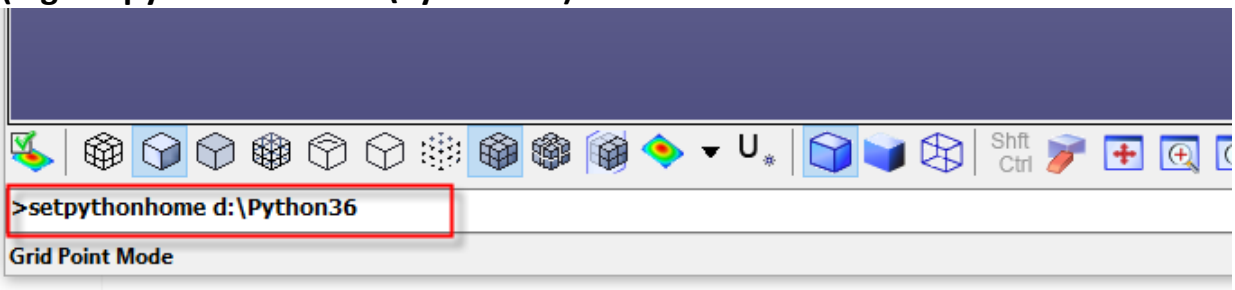
……

How to use Python Scripting in LS-PrePost

Firstly the user should download python 3.6 from
https://www.python.org/ftp/python/3.6.5/python-3.6.5-amd64.exe
then install python3.6 by double clicking the downloaded file (for now LS-

PrePost only supports Python3.6)
When running the Python script the first time, one should issues the following command in LS-PrePost (in the command input area)

setpythonhome “python_home_directory”
(e.g. setpythonhome “D:\Python36”)



The Python home path will be automatically saved to the config file.

And then run Python scripting with the regular LS-PrePost command, or within the command file, use the runpython command to execute the Python scripting, parameters can also be passed to the script.

The syntax:

runpython “pythonscriptingname” optional parameters

Example: the following command file will execute the script that creates a X-Y curve with the parameters defined in the command file.

```
parameter  pa 9.0E+07
parameter  pb 7000.0
parameter  pc  4.0E+07
parameter  npt  300
parameter  xmin  0.0
parameter  xmax  0.00126
runpython "customcurve.py"  &npt &pa &pb &pc &xmin &xmax
```

Data Center Parameter Name list

Name	Type	Status

num_states	Int	Available
num_parts	Int	Available
num_nodes	Int	Available
num_elements	Int	Available
num_materials	Int	Available
largest_node_id	Int	Available
largest_element_id	Int	Available
num_shell_elements	Int	Available
num_beam_elements	Int	Available
num_solid_elements	Int	Available
num_tshell_elements	Int	Available
num_sph_elements	Int	Available
num_discrete_elements	Int	Available
num_seatbelt_elements	Int	Available
num_mass_elements	Int	Available
num_inertia_elements	Int	Available
num_validparts	Int	Available
num_beam_intp	Int	Available
num_active_elements	Int	Available
largest_point_id	Int	Available
largest_vertex_id	Int	Available
largest_edge_id	Int	Available
largest_surface_id	Int	Available

partofelem_id	Int	Available
current_state	Int	Available
num_selection	Int	Available
is_full_integrated	Int	Available
xyplot_numpopupwin	Int	Available
xyplot_numcurves	Int	Available
node_ids	Int array	Available
ids_inset	Int array	Available
element_ids	Int array	Available
elemofpart_ids	Int array	Available
validpart_ids	Int array	Available
selection_ids	Int array	Available
selection_types	Int array	Available
partofmat_ids	Int array	Available
element_connectivity	Int array	Available
active_elements_ids	Int array	Available
largest_time	Float	Available
largest_disp_magnitude	Float	Available
max_stress_x	Float	Available
max_stress_y	Float	Available
max_stress_z	Float	Available
max_stress_xy	Float	Available
max_stress_yz	Float	Available

max_stress_zx	Float	Available
max_strain_x	Float	Available
max_strain_y	Float	Available
max_strain_z	Float	Available
max_strain_xy	Float	Available
max_strain_yz	Float	Available
max_strain_zx	Float	Available
xyplot_maxvalue	Float	Available
xyplot_minvalue	Float	Available
xyplot_lastvalue	Float array	Available
state_times	Float array	Available
nodal_temperatures	Float array	Available
node_x	Float or float array	Available
node_y	Float or float array	Available
node_z	Float or float array	Available
disp_x	Float or float array	Available
disp_y	Float or float array	Available
disp_z	Float or float array	Available
disp_magnitude	Float or float array	Available
state_node_x	Float or float array	Available
state_node_y	Float or float array	Available
state_node_z	Float or float array	Available
velo_x	Float or float array	Available

velo_y	Float or float array	Available
velo_z	Float or float array	Available
velo_magnitude	Float or float array	Available
accel_x	Float or float array	Available
accel_y	Float or float array	Available
accel_z	Float or float array	Available
accel_magnitude	Float or float array	Available
stress_x	Float or float array	Available
stress_y	Float or float array	Available
stress_z	Float or float array	Available
stress_xy	Float or float array	Available
stress_yz	Float or float array	Available
stress_zx	Float or float array	Available
effective_plastic_strain	Float or float array	Available
historyvar	Float array	Available
stress_1stprincipal	Float or float array	Available
stress_2ndprincipal	Float or float array	Available
stress_3rdprincipal	Float or float array	Available
strain_x	Float or float array	Available
strain_y	Float or float array	Available
strain_z	Float or float array	Available
strain_xy	Float or float array	Available
strain_yz	Float or float array	Available

strain_zx	Float or float array	Available
strain_1stprincipal_infin	Float or float array	Available
strain_2ndprincipal_infin	Float or float array	Available
strain_3rdprincipal_infin	Float or float array	Available
lower_eps1	Float array	Available
upper_eps1	Float array	Available
mean_eps1	Float array	Available
lower_eps2	Float array	Available
upper_eps2	Float array	Available
mean_eps2	Float array	Available
sigma1	Float array	Available
sigma2	Float array	Available
mx	Float or float array	Available
my	Float or float array	Available
mxy	Float or float array	Available
qx	Float or float array	Available
qy	Float or float array	Available
nx	Float or float array	Available
ny	Float or float array	Available
nxy	Float or float array	Available
Nx/t-6*Mx/(t*t)	Float or float array	Available
Nx/t+6*Mx/(t*t)	Float or float array	Available
Ny/t-6*My/(t*t)	Float or float array	Available

Ny/t+6*My/(t*t)	Float or float array	Available
Nxy/t-6*Mxy/(t*t)	Float or float array	Available
Nxy/t+6*Mxy/(t*t)	Float or float array	Available
strain_energy_density	Float or float array	Available
Internal_energy_density	float array	Available
kinetic_energy	Float or float array	Available
internal_energy	Float or float array	Available
total_energy	Float or float array	Available
rigidbody_dispx	Float or float array	Available
rigidbody_dispy	Float or float array	Available
rigidbody_dispz	Float or float array	Available
result_rigidbody_disp	Float or float array	Available
rigidbody_velx	Float or float array	Available
rigidbody_vely	Float or float array	Available
rigidbody_velz	Float or float array	Available
result_rigidbody_vel	Float or float array	Available
rigidbody_accelx	Float or float array	Available
rigidbody_accely	Float or float array	Available
rigidbody_accelz	Float or float array	Available
result_rigidbody_accel	Float or float array	Available
von_mises	Float or float array	Available
thickness	Float or float array	Available
area	Float or float array	Available

volume	Float or float array	Available
shell_normal	Vector	Available
displacement	Vector or vector array	Available
velocity	Vector or vector array	Available
acceleration	Vector or vector array	Available
global_stress	Tensor or tensor array	Available
global_strain	Tensor or tensor array	Available
axial_force	Float or Float array	Available
s_shear_resultant	Float or Float array	Available
t_shear_resultant	Float or Float array	Available
s_bending_moment	Float or Float array	Available
t_bending_moment	Float or Float array	Available
torsional_resultant	Float or Float array	Available
axial_stress	Float or Float array	Available
rs_shear_stress	Float or Float array	Available
tr_shear_stress	Float or Float array	Available
plastic_strain	Float or Float array	Available
axial_strain	Float or Float array	Available
strain_maxprincipal	Float or Float array	Available
strain_2ndprincipal	Float or Float array	Available
strain_minprincipal	Float or Float array	Available
x_heatflux	Float or Float array	Available
y_heatflux	Float or Float array	Available

z_heatflux	Float or Float array	Available
heatflux_magnitude	Float or Float array	Available
internal_energy_density	Float or Float array	Available
material_internal_energy	Float or Float array	Available
material_rigidbody_velx	Float or Float array	Available
material_rigidbogy_vely	Float or Float array	Available
material_rigidbogy_velz	Float or Float array	Available
material_result_rigidbody_vel	Float or Float array	Available
part_name	String	Available
time	String	Available
Elementdeletion	Float	Available

Examples:

The examples can be download from the LSTC ftp site:

ftp://ftp.lstc.com/outgoing/lsprepost/SCLexamples/SCL_Examples.zip

As of June, 2020, there are 11 example scripts:

1. Example 1:
Script to get no. of parts in the model. Get all the part IDs, then draw each part by itself and auto center it for each part, capture a picture in png format, and save it to a file which has the part id as the file name.

2. Example 2:
Script to create a plate with 25 shell elements, then extract the following:
1. number of nodes/elements in the model,
2. largest node/element ids,

3. the array of the node ids.
4. get the element connectivity for the last element

3. Example 3:
Script to create a load curve based on a given equation and some parameters, the script will be called by a command file example3.cfile which passes the parameters to the script, the created curve will be written to a file called curve.txt, then load the file back to display in the xy-plot plot interface as a XY graph

4. Example 4:
Script to measure the mass, mass center of gravity and volume of all solid parts in the model, the measured information will be written to file exam4.txt.

Example 4a:
Script to measure the angular velocity, of all solid parts in the model, the measured information will be written to file exam4a.txt.

5. Example 5:
Script to extract the x, y, and z components of the displacement array, then compute the resultant displacement for all the nodes and then fringe the computed result, also write the computed result to a file for each state. The file written can be loaded back into LSPP as User defined fringe data.

6. Example 6:
Script to get x, y, z, global stress components and compute the average stress, then fringe the computed result and write it out to a file.

7. Example 7:
Script to extract the z component of the nodal displacement array, then differentiate it with respect to time, which should give the z velocity array, then extract the z component of the velocity array that was stored in d3plot file and then compute the difference between differentiated result with stored result and then fringe it.

8. Example 8:
Script to extract MX, MY, MXY, QX, QY, NX, NY, NXY resultant forces at the last state for Shell elements from a set of d3plot files, and write the extracted data out to a file

9. Example 9:

Script to look up number of parts in a model, get the part IDs, for each active part, measure the volume of the part by issuing a "Measure Vol part" command, and obtain the results from the command, then write out the return values from the measure command to a file call postdata.txt.

10. Example 10:
Script to read a file that contains nodal coordinates which define the outline of a region on the XY plane. Then create a geometry surface from the outline, and then mesh it with shell elements, drag the shell elements in the Z direction to form a solid block. Delete the shell element part, and keep the solid part, write the solid part to a file. This example uses a LS-PrePost command file (example10.cfile) to call the SCL file, with the input file, output file and a few parameters defined in the command file and pass them to the script.

11. Example 11:
Script to write the nodal coordinate from the selection buffer to a file. The file name is passed through the runscript/runpython command line. This example demonstrates how to pass a string from command line to the script. Also demonstrates how to get items in the selection buffer.