



Engineering Sketch Pad

UDP/UDF Programming – Full-featured UDF

For ESP Rev 1.28

John F. Dannenhoffer, III
john@geocentrictech.com
Geocentric Technologies LLC

- Use the process from the previous session to create `udpTrain2` from `udpTemplate`
- Modify `train2.c` to:
 - get a Body from the stack (that is, write a UDF instead of a UDP)
 - get inputs from a file
 - manipulate attributes
- Exercise3

```
1 UDPRIM train2 Body
2   purpose:
3       serve as a template for creation of new UDFs
4       as an example, modifies attributes on the input Body
5   input Bodys:
6       one SolidBody, SheetBody, or WireBody
7   input arguments (specified as name/value pairs):
8       filename    name of file with directives
9   output arguments:
10      nskip        number of skipped directives
11   usage notes:
12       the directives are in the form:
13           FACE attrName          (deletes    attrName on Faces)
14           FACE attrName attrValue (overwrites attrName on Faces)
15           EDGE attrName          (deletes    attrName on Edges)
16           EDGE attrName attrValue (overwrites attrName on Edges)
17           NODE attrName          (deletes    attrName on Nodes)
18           NODE attrName attrValue (overwrites attrName on Nodes)
19
20       # signifies a comment (can be used anywhere)
21       @ inserts Face, Edge, or Node number into attrValue
22       the directive can either be in UPPERACE or lowercase
23       error messages are printed, but otherwise ignored
24
25       analytic sensitivities are not supported
26   contributed by:
27       John Dannenhoffer john@geocentrictech.com
```

Listing of train_2.csm (1)

```
1 # train2_1
2 # written by John Dannenhoffer
3
4 # make a box
5 BOX      0  0  0  4  3  2
6
7 # add attributes
8 UDPRIM   train2      filename <<
9 #   this   is a comment
10 Face    myFace      @           # error - ignored
11 face    myFace      @           # real attribute
12
13 EDGE    myEdge      edge_@      # string attribute
14 edge    myEdge2     deleteMe    # string attribute
15
16 node    myNode      @           # real attribute
17 node    myNode      @_node      # overwrites with string
18 NODE    myNode      @           # error - no attrName
19
20 edge    myEdge2     @           # deletes attribute
21 >>
22
23 # verify that there were 2 skipped directives (ie, errors)
24 ASSERT   @@nskip 2
```

Listing of `train_2.csm` (2)

```
25 # verify that attributes have been added to the box
26 PATBEG   iface @nface
27     SELECT    FACE  $myFace   iface
28     ASSERT    @sellist[1]    iface
29 PATEND
30
31 PATBEG   iedge @nedge
32     SELECT    EDGE  $myEdge   $edge_+iedge
33     ASSERT    @sellist[1]    iedge
34 PATEND
35
36 SELECT    EDGE  $myEdge2  $deleteMe
37 ASSERT    0  1              # should not execute
38 CATBEG    $edge_not_found
39 CATEND
40
41 PATBEG   inode @nnode
42     SELECT    NODE  $myNode   val2str(inode,0)+$_node
43     ASSERT    @sellist[1]    inode
44 PATEND
45
46 END
```

- Attributes can be associated with any EGADS object (ego)
 - Curve, PCurve, Surface
 - Node, Edge, Loop, Face, Shell, Body
- Attributes are specified as a name/value pair
- Attribute names are arbitrary strings of letters, digits, or underscores
- Attribute values can be:
 - ATTRINT – one or more integers
 - ATTRREAL – one or more doubles
 - ATTRSTRING – a character string
 - ATTRCSYS – doubles that are transformed with `EG_copyObject`

- Operations that apply to attributes include:
 - `EG_attributeNum` – return the number of attributes associated with an ego
 - `EG_attributeAdd` – add an attribute and its value to an ego
 - `EG_attributeDel` – delete the named attribute from an ego
 - `EG_attributeDup` – duplicate the attributes from one ego to another ego
 - `EG_attributeGet` – get the attribute name and value given its index from an ego
 - `EG_attributeRet` – return the attribute value given its name from an ego

Listing of train2.c (1)

```

1  /*
2  ****
3  *                                                                    *
4  *  udfTrain2 -- training UDF                                                                    *
5  *                                                                    *
6  *          this modified attributes on a Body                                                                    *
7  *          sensitivities are not computed                                                                    *
8  *                                                                    *
9  *          Written by John Dannenhoffer @ Geocentric Technologies  *
10 *                                                                    *
11 ****
12 */
13
14 /*
15 * Copyright (C) 2025  John F. Dannenhoffer, III (Geocentric Technologies)
16 *
17 * This library is free software; you can redistribute it and/or
18 *   modify it under the terms of the GNU Lesser General Public
19 *   License as published by the Free Software Foundation; either
20 *   version 2.1 of the License, or (at your option) any later version.
21 *
22 * This library is distributed in the hope that it will be useful,
23 *   but WITHOUT ANY WARRANTY; without even the implied warranty of
24 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
25 *   Lesser General Public License for more details.
26 *
27 * You should have received a copy of the GNU Lesser General Public
28 *   License along with this library; if not, write to the Free Software
29 *   Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
30 *   MA 02110-1301  USA
31 */

```


- General notes:
 - the descriptions given focus on the difference between previous sessions and `train2.c`
 - at times there will be no explanation slide since the code shown is basically a copy-paste-edit of preceeding code
- Lines 4–7: identifying comment (which is different from `template.c`)

Listing of `train2.c` (2)

```
32  /* uncomment the following to get DEBUG printouts */
33  //#define DEBUG 1
34
35  /* the number of "input" Bodys
36
37     this only needs to be specified if this is a UDF (user-defined
38     function) that consumes Bodys from OpenCSM's stack. (the default
39     value is 0).
40
41     if NUMUDPINPUTBODYS>0 then exactly NUMUDPINPUTBODYS are in emodel
42     if NUMUDPINPUTBODYS<0 then up to -NUMUDPINPUTBODYS are in emodel
43 */
44 #define NUMUDPINPUTBODYS 1
45
46 /* the number of arguments (specified below) */
47 #define NUMUDPARGS 2
48
49 /* set up the necessary structures (uses NUMUDPARGS) */
50 #include "udpUtilities.h"
51
52 /* shorthand macros for accessing argument values and velocities */
53 #define FILENAME(IUDP) ((char *) (udps[IUDP].arg[0].val))
54 #define NSKIP( IUDP) ((int *) (udps[IUDP].arg[1].val))[0]
```

- Line 44: one `Body` is removed from the stack, put into an `EGADS MODEL`, and passed to this UDF

Listing of train2.c (3)

```

55  /* data about possible arguments
56      argNames: argument name (must be all lower case)
57      argTypes: argument type: +ATTRINT      integer input
58                               -ATTRINT      integer output
59                               +ATTRREAL      double input  (no sensitivities)
60                               -ATTRREAL      double output (no sensitivities)
61                               +ATTRREALSEN    double input  (with sensitivities)
62                               -ATTRREALSEN    double output (with sensitivities)
63                               +ATTRSTRING     string input
64                               -ATTRSTRING     *** cannot be used ***
65                               +ATTRFILE       input file
66                               -ATTRFILE       *** cannot be used ***
67                               +ATTRREBUILD    forces rebuild if any variable in
68                                               semi-colon-separated list has been changed
69                               -ATTRREBUILD    *** cannot be used ***
70                               +ATTRRECYCLE    forces rebuild (always) by blocking recycling
71                               -ATTRRECYCLE    *** cannot be used ***
72      argIdefs: default value for ATTRINT
73      argDdefs: default value for ATTRREAL or ATTRREALSEN */
74  static char *argNames[NUMUDPARGS] = {"filename", "nskip",  };
75  static int  argTypes[NUMUDPARGS] = {ATTRFILE,  -ATTRINT, };
76  static int  argIdefs[NUMUDPARGS] = {0,          0,          };
77  static double argDdefs[NUMUDPARGS] = {1.,        0.,        };

```

- **Line 75:** the `filename` argument is of type `ATTRFILE`. Both `ATTRFILE` and `ATTRSTRING` give the UDP/UDF character strings, but `ATTRFILE` handles the filename consistently with ESP

Listing of train2.c (4)

```

78  /* get utility routines: udpErrorStr, udpInitialize, udpReset, udpSet,
79                               udpGet, udpVel, udpClean, udpMesh */
80  #include "udpUtilities.c"
81
82  static int addAttribute(ego eobj, char attrName[], char attrValue[], int index);
83  #ifdef DEBUG
84  static int printAttributes(ego eobj);
85  #endif
86  static int convertAtSign(char instring[], int number, char outstring[];
87
88  ^^L
89  /*
90   *
91   *
92   *   udpExecute - execute the primitive
93   *
94   *
95   */
96
97  int
98  udpExecute(ego   emodel,           /* (in)  Model containing Body */
99             ego   *ebody,          /* (out) Body (or model) pointer */
100            int    *nMesh,           /* (out) number of associated meshes */
101            char   *string[])        /* (out) error message */

```

- Lines 82–88: declare the “helper” routines which are defined at the end of the file
- Line 98: because this is a UDF (and not a UDP), this first argument now has the `MODEL` that contains the input `Bodys` (from the stack). This is different from a UDP, where the first argument is the `context`.

```
102 {
103     int      status = EGADS_SUCCESS;
104
105     char      *message=NULL;
106     udp_T      *udps = *Udps;
107
108     int      oclass, mtype, nchild, *senses, i, nskip;
109     int      eof, inext, nread;
110     int      nnode, inode, nedge, iedge, nface, iface;
111     double    data[18];
112     char      templine[256], command[256], attrName[256], attrValue[256];
113     ego      *ebodys, context, eref, modl;
114     ego      *enodes=NULL, *eedges=NULL, *efaces=NULL;
115     FILE      *fp=NULL;
116
117     ROUTINE (udpExecute);
118
119     /* ----- */
120
121 #ifdef DEBUG
122     /* debug printing of the input arguments */
123     printf("udpExecute(emodel=%llx)\n", (long long)emodel);
124     printf("filename(0) = %s\n", FILENAME(0));
125     printf("nskip( 0) = %d\n", NSKIP( 0));
126 #endif
```


Listing of `train2.c` (6)

```
127  /* default return values */
128  *ebody  = NULL;
129  *nMesh  = 0;
130  *string = NULL;
131
132  /* the place where messages to the user are placed */
133  MALLOC(message, char, 100);
134  message[0] = '\0';
135
136  /* number of directives containing an error */
137  nskip = 0;
138
139  /* check that Model was input that contains one Body */
140  status = EG_getTopology(emodel, &eref, &oclass, &mtype,
141                          data, &nchild, &ebodys, &senses);
142  CHECK_STATUS(EG_getTopology);
143
144  if (oclass != MODEL) {
145      snprintf(message, 100, "expecting a Model, but found oclass=%d", oclass);
146      status = EGADS_NOTMODEL;
147      goto cleanup;
148  } else if (nchild != 1) {
149      snprintf(message, 100, "expecting Model to contain one Body (not %d)", nchild);
150      status = EGADS_NOTBODY;
151      goto cleanup;
152  }
```

- Lines 140–142: extract from `emodel` its class (which should be `MODEL`) and the Bodels that it contains (in `ebodels`)
- Lines 144–147: make sure that `emodel` contains a `MODEL`
- Lines 148–151: make sure that `emodel` only contains one `Body`

Listing of train2.c (7)

```
153 /* check arguments */
154 if (STRLEN(FILENAME(0)) <= 0) {
155     snprintf(message, 100, "\"filename\" must be specified");
156     status = OCSM_ILLEGAL_VALUE;
157     goto cleanup;
158 }
159
160 /* cache copy of arguments for future use */
161 status = cacheUdp(NULL);
162 CHECK_STATUS(cacheUdp);
163
164 #ifdef DEBUG
165 /* debug printing of cached input arguments */
166 printf("filename(%d) = %s\n", numUdp, FILENAME(numUdp));
167 printf("nskip(    %d) = %d\n", numUdp, NSKIP(    numUdp));
168 #endif
169
170 /* make a copy of the Body (so that it does not get removed
171    when OpenCSM deletes emodel) */
172 status = EG_copyObject(ebodyys[0], NULL, ebody);
173 CHECK_STATUS(EG_copyObject);
174
175 SPLINT_CHECK_FOR_NULL(*ebody);
176
177 /* get pointer to model */
178 status = EG_getContext(emodel, &context);
179 CHECK_STATUS(EG_getContext);
180
181 status = EG_getUserPointer(context, (void**)(&(modl)));
182 CHECK_STATUS(EG_getUserPointer);
```

- Lines 172–173: make a copy of the input Body (`ebodys[0]`) to be the output Body (`ebody`). This is needed because ESP deletes `emodel` (and its Bodies) when it returns from `udpExecute`. The second argument to `EG_copyObject` is a transformation, which is not needed here.
- Lines 178–179: because a UDF gets a MODEL instead of the context, we need to get the context by calling `EG_getContext` (which can be called on any ego)
- Lines 181–182: get the `OpenCSM Modl` pointer in case we need to access any information from ESP (such as Design Parameters). This is actually not needed for this UDF, but is here in case we change things in the future.

Listing of `train2.c` (8)

```
183  /* get arrays of the Nodes, Edges, and Faces in the Body */
184  status = EG_getBodyTopos(*ebody, NULL, NODE, &nnode, &enodes);
185  CHECK_STATUS(EG_getBodyTopos);
186
187  status = EG_getBodyTopos(*ebody, NULL, EDGE, &nedge, &eedges);
188  CHECK_STATUS(EG_getBodyTopos);
189
190  status = EG_getBodyTopos(*ebody, NULL, FACE, &nface, &efaces);
191  CHECK_STATUS(EG_getBodyTopos);
192
193  /* debug print of attributes before processing */
194  #ifdef DEBUG
195  printf("Attributes before processing\n");
196
197  for (inode = 0; inode < nnode; inode++) {
198      printf("Node %3d\n", inode+1);
199      printAttributes(enodes[inode]);
200  }
201
202  for (iedge = 0; iedge < nedge; iedge++) {
203      printf("Edge %3d\n", iedge+1);
204      printAttributes(eedges[iedge]);
205  }
206
207  for (iface = 0; iface < nface; iface++) {
208      printf("Face %3d\n", iface+1);
209      printAttributes(efaces[iface]);
210  }
211  #endif
```

- Lines 184–185: get the number of Nodes and a list of the Nodes from `ebody`. Examining the documentation for `EG_getBodyTopos` indicates that the last argument is “freeable”, which means that we need to make sure we call `EG_free` (which is done in the cleanup section on line 353). Also note that this list is bias-0 and we typically number the Nodes bias-1.
- Lines 197–200: if `DEBUG` is defined, print the attributes for all the Nodes. Note in line 198 the addition of 1 to `inode` to convert from bias-0 to bias-1.

```
212     /* if FILENAME contains a real filename, open it now and remember
213        if we are at the end of file */
214     if (strncmp(FILENAME(0), "<<\n", 3) != 0) {
215         fp = fopen(FILENAME(0), "rb");
216         if (fp == NULL) {
217             snprintf(message, 100, "could not open file \"%s\"", FILENAME(0));
218             status = EGADS_NOTFOUND;
219             goto cleanup;
220         }
221
222         ieof = feof(fp);
223
224     /* otherwise we have an inline file, so set up the next character to read */
225     } else {
226         inext = 3;
227         ieof = 0;
228     }
```

- Lines 214–222: if the first three characters of `FILENAME (0)` are not “<<\” then this is a real filename that should be opened
- Lines 225–228: if the first characters of `FILENAME (0)` were “<<\n”, then this is actually an inline file, where each lines of the inline file follow. So, we need to start reading at the fourth character. Note that `fp` was initialized to `NULL` above so it does not need to be set here.


```
229  /* read until end of file */
230  while (ieof == 0) {
231
232      /* read the next line */
233      if (fp != NULL) {
234          (void) fgets(templine, 255, fp);
235          if (feof(fp) > 0) break;
236
237          /* overwrite the \n and \r at the end */
238          if (STRLEN(templine) > 0 && templine[STRLEN(templine)-1] == '\n') {
239              templine[STRLEN(templine)-1] = '\0';
240          }
241          if (STRLEN(templine) > 0 && templine[STRLEN(templine)-1] == '\r') {
242              templine[STRLEN(templine)-1] = '\0';
243          }
244      } else {
245          i = 0;
246          for (; inext < STRLEN(FILENAME(0)); inext++) {
247              if (FILENAME(0)[inext] == '\n' || FILENAME(0)[inext] == '\r') break;
248
249              templine[i++] = FILENAME(0)[inext];
250          }
251          templine[i] = '\0';
252          inext++;
253
254          if (inext >= STRLEN(FILENAME(0))) {
255              ieof = 1;
256          }
257      }
```

- Line 230: read until `feof` is not 0
- Lines 233–243: a real file is being read, so get the next line into `templine` and record whether or not we have reached the end of file. Also remove the `\n` or `\r` from the line (if they exist)
- Lines 244–256: an inline file is being read, so copy from `FILENAME(0)` into `templine` until we get an end of line. Also if we have gotten to the end of the inline file, set `feof` to 1 to get out of the loop in line 230.

```
258     printf("processing: %s\n", templine);
259
260     /* remove inline comments */
261     for (i = 0; i < strlen(templine); i++) {
262         if (templine[i] == '#') {
263             templine[i] = '\0';
264             break;
265         }
266     }
267
268     /* read the fields out of templine */
269     attrValue[0] = '\0';
270
271     nread = sscanf(templine, "%s %s %s", command, attrName, attrValue);
272
273     /* whole-line comment or blank line */
274     if (nread < 0) {
275
276         /* line with only command (error) */
277     } else if (nread < 2) {
278         printf("    error encountered when reading: %s\n", templine);
279         nskip++;
280     }
```

- Lines 261–266: remove comments from `templine`
- Lines 269–271: read out of `templine` up to 3 space-delimited fields. Note that the third field is initialized to a blank string in case only 2 fields were read.
- Lines 274–279: take care of blank lines and lines that do not contain at least 2 fields

Listing of `train2.c` (12)

```

280      /* attributes on all Nodes */
281      } else if (strcmp(command, "node") == 0 || strcmp(command, "NODE") == 0) {
282          for (inode = 0; inode < nnode; inode++) {
283              status = addAttribute(enodes[inode], attrName, attrValue, inode+1);
284              CHECK_STATUS(addAttribute);
285          }
286
287      /* attributes on all Edges */
288      } else if (strcmp(command, "edge") == 0 || strcmp(command, "EDGE") == 0) {
289          for (iedge = 0; iedge < nedge; iedge++) {
290              status = addAttribute(eedges[iedge], attrName, attrValue, iedge+1);
291              CHECK_STATUS(addAttribute);
292          }
293
294      /* attributes on all Faces */
295      } else if (strcmp(command, "face") == 0 || strcmp(command, "FACE") == 0) {
296          for (iface = 0; iface < nface; iface++) {
297              status = addAttribute(efaces[iface], attrName, attrValue, iface+1);
298              CHECK_STATUS(addAttribute);
299          }
300
301      /* illegal command */
302      } else {
303          printf("    illegal command (%s) when reading: %s\n", command, templine);
304          nskip++;
305      }
306  }

```

- Lines 281–285: process lines that start with `either node` or `NODE` by calling `addAttribute` (which is defined below) on the Node. Note the switch from `bias-0` (in `enodes`) to `bias-1` by adding 1 to `inode`

```
307     /* debug print of attributes after processing */
308 #ifdef DEBUG
309     printf("Attributes after processing\n");
310
311     for (inode = 0; inode < nnode; inode++) {
312         printf("Node %3d\n", inode+1);
313         printAttributes(enodes[inode]);
314     }
315
316     for (iedge = 0; iedge < nedge; iedge++) {
317         printf("Edge %3d\n", iedge+1);
318         printAttributes(eedges[iedge]);
319     }
320
321     for (iface = 0; iface < nface; iface++) {
322         printf("Face %3d\n", iface+1);
323         printAttributes(efaces[iface]);
324     }
325 #endif
326
327     /* add a special Attribute to the Body to tell OpenCSM that there
328        is no topological change and hence it should not adjust the
329        Attributes on the Body in finishBody() */
330     status = EG_attributeAdd(*ebody, "__noTopoChange__", ATTRSTRING,
331                             0, NULL, NULL, "udfTrain2");
332     CHECK_STATUS(EG_attributeAdd);
```

- Lines 330–332: since the topology of then input body (`ebodys[0]`) and output body (`ebody`) are the same, we can set a special attribute on the Body to tell ESP to simply use the attributes already on (`ebody`), which were put there during the `EG_copyObject` back on line 172.


```
333     /* set the output value(s) */
334     NSKIP(numUdp) = nskip;
335
336     /* remember this model (Body) */
337     udps[numUdp].ebody = *ebody;
338
339 cleanup:
340 #ifdef DEBUG
341     printf("udpExecute -> numUdp=%d, *ebody=%llx\n", numUdp, (long long)(*ebody));
342 #endif
343
344     if (fp != NULL) fclose(fp);
345
346     if (enodes != NULL) EG_free(enodes);
347     if (eedges != NULL) EG_free(eedges);
348     if (efaces != NULL) EG_free(efaces);
349
350     if (strlen(message) > 0) {
351         *string = message;
352         printf("%s\n", message);
353     } else if (status != EGADS_SUCCESS) {
354         FREE(message);
355         *string = udpErrorStr(status);
356     } else {
357         FREE(message);
358     }
359
360     return status;
361 }
```

Listing of train2.c (15)

```

362  /*
363  ****
364  *                                     *
365  *   udpSensitivity - return sensitivity derivatives for the "real" argument *
366  *                                     *
367  * ****
368  */
369
370  int
371  udpSensitivity(ego      ebody,          /* (in)  Body pointer */
372                int      npnt,          /* (in)  number of points */
373                int      entType,       /* (in)  OCSM entity type */
374                int      entIndex,      /* (in)  OCSM entity index (bias-1) */
375                double   uvs[],         /* (in)  parametric coordinates for evaluation */
376                double   vels[])       /* (out) velocities */
377  {
378      int      status = EGADS_SUCCESS;
379
380      int      iudp, judp;
381
382      ROUTINE(udpSensitivity);
383
384      /* ----- */
385
386  #ifdef DEBUG
387      if (uvs != NULL) {
388          printf("udpSensitivity(ebody=%llx, npnt=%d, entType=%d, entIndex=%d, uvs=%f %f)\n",
389              (long long)ebody, npnt, entType, entIndex, uvs[0], uvs[1]);

```

- Since we are not computing sensitivities, this is exactly the same routine as it was in `udpTemplate`

Listing of train2.c (16)

```

390     } else {
391         printf("udpSensitivity(ebody=%llx, npnt=%d, entType=%d, entIndex=%d, uvs=NULL)\n",
392             (long long)ebody, npnt, entType, entIndex);
393     }
394 #endif
395
396     /* check that ebody matches one of the ebodys */
397     iudp = 0;
398     for (judp = 1; judp <= numUdp; judp++) {
399         if (ebody == udps[judp].ebody) {
400             iudp = judp;
401             break;
402         }
403     }
404     if (iudp <= 0) {
405         status = EGADS_NOTMODEL;
406         goto cleanup;
407     }
408
409     /* the following line should be included if sensitivities
410        are not computed analytically */
411     status = EGADS_NOLOAD;
412
413 cleanup:
414 #ifdef DEBUG
415     printf("udpSensitivity -> vels=%f %f %f\n", vels[0], vels[1], vels[2]);
416 #endif
417     return status;
418 }

```

Listing of train2.c (17)

```

419  /*
420  ****
421  *
422  *   addAttribute - add the specified attribute to the ego
423  *
424  ****
425  */
426
427 static int
428 addAttribute(ego      eobj,          /* (in)   ego to get attribute */
429              char      attrName[],   /* (in)   attribute name */
430              char      attrValue[],  /* (in)   attribute value (may contain @ */
431              int        index)       /* (in)   index (such as bias-1 Node number) */
432  {
433      int          status=EGADS_SUCCESS; /* (out)  return status */
434
435      int          attrType, attrLen;
436      CINT         *tempIlist;
437      CDOUBLE      *tempRlist;
438      char         tempValue[256];
439      CCHAR        *tempClist;
440
441      ROUTINE (printAttributes);
442
443      /* ----- */

```

- Lines 428–431: definition of the function to add an attribute named `attrName` with the (possibly modified) value `attrValue` to the Node, Edge, or Face, whose `ego` is `eobj`
- Lines 436–439: the declaration `CINT` is simply a macro that expands to `const int`, `CDOUBLE` expands to `const double`, and `CCHAR` expands to `const char`. These are needed to match the declarations of `EG_attributeRet` and `EG_attributeGet` in `EGADS`

```
444 /* remove the attribute if it exists */
445 status = EG_attributeRet(eobj, attrName, &attrType, &attrLen,
446                        &tempIlist, &tempRlist, &tempClist);
447 if (status == EGADS_SUCCESS) {
448     status = EG_attributeDel(eobj, attrName);
449     CHECK_STATUS(EG_attributeDel);
450 }
451
452 /* if attrValue was a percent-sign, add an integer attribute */
453 if (strcmp(attrValue, "%") == 0) {
454     status = EG_attributeAdd(eobj, attrName, ATTRINT, 1,
455                             &index, NULL, NULL);
456     CHECK_STATUS(EG_attributeAdd);
457
458 /* otherwise add a string attribute */
459 } else if (STRLEN(attrValue) > 0) {
460     status = convertAtSign(attrValue, index, tempValue);
461     CHECK_STATUS(convertAtSign);
462
463     status = EG_attributeAdd(eobj, attrName, ATTRSTRING, 0,
464                             NULL, NULL, tempValue);
465     CHECK_STATUS(EG_attributeAdd);
466 }
467
468 cleanup:
469     return status;
470 }
```

- Lines 445–450: if the `attrName` attributes exists on `eboj`, delete it. The reason we check if it exists first is to avoid an error being generated by `EG_attributeDel`. Notice on line 446 the three places that an attribute could be returned, depending on its type
- Lines 453–456: if the `attrValue` was a single at-sign, we are going to store a single integer for this attribute
- Lines 460–461: convert `attrValue` to `tempValue` by replacing at-signs with the value of `index`. The routine that does this is listed below
- Lines 463–465: add a string attribute to `ego`

Listing of train2.c (19)

```

471  /*
472  ****
473  *
474  *   printAttributes - print attributes associated with an ego
475  *
476  ****
477  */
478
479 #ifdef DEBUG
480 static int
481 printAttributes(ego eobj)          /* (in)  ego object */
482 {
483     int      status=EGADS_SUCCESS;    /* (out) return status */
484
485     int      attrType, attrLen, nattr, iattr, i;
486     CINT     *tempIlist;
487     CDOUBLE  *tempRlist;
488     CCHAR    *tempClist, *attrName;
489
490     ROUTINE (printAttributes);
491
492     /* ----- */
493
494     status = EG_attributeNum(eobj, &nattr);
495     CHECK_STATUS (EG_attributeNum);

```

- Lines 494–495: return the number of attributes on `eobj`

```
496     for (iattr = 1; iattr <= nattr; iattr++) {
497         status = EG_attributeGet(eobj, iattr, &attrName, &attrType, &attrLen,
498                                 &tempIlist, &tempRlist, &tempClist);
499         CHECK_STATUS(EG_attributeGet);
500
501         printf("    %-20s ", attrName);
502         if (attrType == ATTRINT) {
503             for (i = 0; i < attrLen; i++) {
504                 printf(" %6d", tempIlist[i]);
505             }
506             printf("\n");
507         } else if (attrType == ATTRREAL) {
508             for (i = 0; i < attrLen; i++) {
509                 printf(" %13.7f", tempRlist[i]);
510             }
511             printf("\n");
512         } else if (attrType == ATTRSTRING) {
513             printf(" %s\n", tempClist);
514         }
515     }
516
517 cleanup:
518     return status;
519 }
520 #endif
```

- Lines 497–499: return the `iattrth` attribute (bias-1) from `eobj`. Notice that we get its type `attrType`, how many there are `attrLen`, and either `tempIlist` (if integers), `tempRlist` (if doubles) or `tempClist` (if a character string)
- Lines 501–515: depending on the type (in `attrType`), we print the values

```
521 /*
522  *****
523  *
524  *   convertAtSign - convert at-signs to number
525  *
526  *****
527  */
528
529 static int
530 convertAtSign(char instring[],      /* (in)  input string */
531               int number,          /* (in)  number to use instead of at-sign */
532               char outstring[])    /* (out) output string */
533 {
534     int      status=EGADS_SUCCESS;  /* (out)  return status */
535
536     int      i, j, natsign=0;
537     char     format[512];
538
539     ROUTINE(convertAtSign);
540
541     /* ----- */
542 }
```

- Lines 530–532: this function take a string (`instring`) and generates a string (`outstring`) in which the at-signs in `instring` are replaced with the integer index

```
542 /* build a format string that replaces @ with %d */
543 j = 0;
544 for (i = 0; i < STRLEN(instring); i++) {
545     if (instring[i] != '@') {
546         format[j++] = instring[i];
547     } else if (natsign >= 3) {
548         format[j++] = instring[i];
549     } else {
550         natsign++;
551         format[j++] = '\\%';
552         format[j++] = 'd';
553     }
554 }
555 format[j++] = '\\0';
556
557 /* if there are no at-signs, we can do a simple copy */
558 if (natsign == 0) {
559     strncpy(outstring, instring, 256);
560
561     /* otherwise print to the outstring */
562 } else {
563     snprintf(outstring, 256, format, number, number, number);
564 }
565
566 //cleanup:
567 return status;
568 }
```

- Make a new UDP (`exercise3`) in the directory (folder) `udpExercise2`
 - the purpose is: copy an Edge attribute with a given name to its Nodes. If the Node already has the attribute, change its value to the string `**conflict**`
 - the input parameter is:
 - `attrname` - name of the attribute to copy
- Make sure that your UDF does appropriate error checking