



Engineering Sketch Pad

UDP/UDF Programming – UDP Basics

For ESP Rev 1.28

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

- Differences between UDPs and UDFs
- How UDP/UDFs are used in ESP
- Anatomy of directory `udpTemplate`
- Steps for generating UDP/UDFs
- Line-by-line description of `template.c`
- Exercise1

- Users can add their own user-defined primitives (UDPs)
 - creates a single* Body
 - do not consume any Bodys from the Stack
 - are written in C, C++, or FORTRAN and are compiled
 - can be written either top-down or bottom-up or both
 - have access to the entire suite of methods provided by EGADS
 - are coupled into ESP dynamically at run time
- Users can add their own user-defined functions (UDFs)
 - are the same as UDPs, except they consume one or more Bodys from the Stack
- Creating UDPs and UDFs involve (almost) the same process

- UDP/UDFs are called with a UDPRIM statement

```
UDPRIM      $primetype $argName1 argValue1 \  
              $argName2 argValue2 \  
              $argName3 argValue3 \  
              $argName4 argValue4
```

- \$primetype must start with a letter
- At most 4 name-value pairs can be specified on the UDPRIM statement
- More name-value pairs can be specified in any number of UDPARG statements that precede the UDPRIM statement

```
UDPARG      $primetype $argName1 argValue1 \  
              $argName2 argValue2 \  
              $argName3 argValue3 \  
              $argName4 argValue4
```

- name-value pairs are processed in order (with possible over-writing)

- For UDP/UDFs that read an external file, one can use << to tell ESP to create a file from the following lines, up to a line that starts with >>
- For example:

```
UDPRIM      editAttr  filename <<  verbose 1
            NODE ADJ2FACE tagType=spar tagIndex=1
            AND   ADJ2FACE tagType=lower
            AND   ADJ2EDGE tagType=root
            SET                                capsConstraint=pointConstraint1
>>
SET          A    10
```

has two Branches (UDPRIM and SET)

- The following generate identical Boxes

```
UDPRIM box dx 1 dy 2 dz 3
```

- and

```
UDPARG box dx 1
```

```
UDPRIM box dy 2 dz 3
```

- and

```
UDPARG box dx 11 dy 22 dz 33
```

```
UDPRIM box dx 1 dy 2 dz 3
```

- and

```
UDPARG box dx 1
```

```
UDPARG box dy 2
```

```
UDPARG box dz 3
```

```
UDPRIM box
```

- Some UDP/UDFs return values to the calling script
- The returned values have names that are prepended by two at-signs (for example: `volume` in the UDP/UDF is available as `@@volume` after the UDPRIM executes)
- These values stay in effect until overwritten by another UDP (or a UDF or a UDC)

- ESP ships with a directory named `$ESP_HOME/contributions/UdpUdf`, which contains UDP/UDFs that are contributed by users
- UdpUdf is pre-populated with several UDP/UDFs
 - `udpTemplate` — a template from which most UDP/UDF development should start
 - the description that follows starts here
 - `udpTrain1` – a full-featured UDP that contains:
 - bottom-up and top-down builds
 - makes a SolidBody, SheetBody, or WireBody
 - has output arguments
 - implements analytic sensitivities
 - `udfTrain2` – a full-featured UDF that contains:
 - inputs from a file
 - manipulation of attributes

- `template.hlp` — file containing the text that will be included in the documentation in ESP's help facility
- `template_1.csm` — file containing the first test case
- `template_1.png` — screen dump when running `template_1.csm` (for help system)
- `Makefile` — file for compiling and linking the `.c` file into the shared library (LINUX and MACOS)
- `NMakefile` — file for compiling and linking the `.c` file into the shared library (Windows)
- `template.c` — file containing the implementation of the UDP/UDF
- `verify_7.8.1` — a directory (folder) that contains data that is to be used during routine testing

```
1 UDPRIM template -
2   purpose:
3       serve as a template for creation of new UDPs
4       as an example, it creates a sphere centered at the origin
5   input Bodys:
6       -none-
7   input arguments (specified as name/value pairs):
8       radius      radius of sphere                      [default 1]
9   output arguments:
10      -none-
11   usage notes:
12       the radius must be positive
13
14       analytic sensitivities are not supported
15   contributed by:
16       John Dannenhoffer john@geocentrictech.com
```

```
1  # template_1
2  # written by John Dannenhoffer
3
4  DESPMTR    RAD        2.0
5
6  UDPRIM     template   radius RAD
7
8  END
```

- Make a new directory
 - for example, create `udpExercise1`
- Copy all the files in `udpTemplate` into your new folder
- Rename all files to your new UDP/UDF name
 - for example
 - rename `template_1.csm` to `exercisel_1.csm`
 - rename `template.c` to `exercisel.c`
 - rename `template.hlp` to `exercisel.hlp`
- Edit `Makefile` by changing all occurrences of `template` to the name of your new UDP/UDF
- Edit `NMakefile` by changing all occurrences of `template` to the name of your new UDP/UDF
- Remove the files from `verify_7.8.1`
 - these will be automatically created below

- Modify the file that will provide help to the user
 - for example, modify `exercisel.hlp`
- Modify the `.csm` file(s) that will be used to verify that the UDP/UDF works as intended
 - for example, modify `exercisel_1.csm`
 - add other test files, naming them `exercisel_2.csm`, ...
- Modify the `.c` file to implement your new UDP/UDF
 - for example, modify `exercisel.c`

- Test that your UDP/UDF works as intended
 - for example:
 - run `serveESP exercisel_1` to verify that you get intended results
 - run `sensCSM -geom exercisel_1` to verify that you get correct geometric sensitivities (which are returned as $1.0e-20$ if finite-difference sensitivities are used)
 - run `sensCSM -tess exercisel_1` to verify that you get correct tessellation sensitivities (which are returned as $1.0e-20$ if finite-difference sensitivities are used)
- Build the verification data (to be used for automatic testing)
 - for example:
 - run `serveESP -addVerify exercisel_1`
 - run `sensCSM -geom -addVerify exercisel_1`
 - run `sensCSM -tess -addVerify exercisel_1`
 - this adds files to `verify_7.8.1`

Listing of template.c (1)

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

- General notes:
 - the header files `udpUtilities.h` and `udpUtilities.c` contain lots of code to hide the complexities of UDP/UDFs from the developer
 - the order of the statements is generally important
 - macros defined in these files are written in UPPERCASE
- Lines 1–11: identifying comment
- Lines 13–30: copyright comment

```
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 0
45
46  /* the number of arguments (specified below) */
47  #define NUMUDPARGS 1
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 RADIUS( IUDP)  ((double *) (udps[IUDP].arg[0].val))[0]
54  #define RADIUS_SIZ(IUDP)      udps[IUDP].arg[0].size
55
```

- Line 33: uncomment the define to print DEBUG information
- Line 44: define the number of input Bodys
 - set to 0 for a user-defined primitive (UDP)
- Line 47: number of input and output arguments
- Line 50: this include file sets up transfers to and from ESP
 - this line **MUST** be in your UDP/UDF at this location
- Lines 53–54: define macros to access input and output arguments
 - these lines should be duplicated for every input/output argument
 - the index in `arg` should be incremented for each additional input/output argument

Listing of template.c (3)

```

56  /* data about possible arguments
57      argNames: argument name (must be all lower case)
58      argTypes: argument type: +ATTRINT      integer input
59                               -ATTRINT      integer output
60                               +ATTRREAL      double input  (no sensitivities)
61                               -ATTRREAL      double output (no sensitivities)
62                               +ATTRREALSEN    double input  (with sensitivities)
63                               -ATTRREALSEN    double output (with sensitivities)
64                               +ATTRSTRING     string input
65                               -ATTRSTRING     *** cannot be used ***
66                               +ATTRFILE       input file
67                               -ATTRFILE       *** cannot be used ***
68                               +ATTRREBUILD    forces rebuild if any variable in
69                                               semi-colon-separated list has been changed
70                               -ATTRREBUILD    *** cannot be used ***
71                               +ATTRRECYCLE    forces rebuild (always) by blocking recycling
72                               -ATTRRECYCLE    *** cannot be used ***
73      argIdefs: default value for ATTRINT
74      argDdefs: default value for ATTRREAL or ATTRREALSEN */
75  static char  *argNames[NUMUDPARGS] = {"radius", };
76  static int    argTypes[NUMUDPARGS] = {ATTRREAL, };
77  static int    argIdefs[NUMUDPARGS] = {0,      };
78  static double argDdefs[NUMUDPARGS] = {1.,      };
79
80  /* get utility routines: udpErrorStr, udpInitialize, udpReset, udpSet,
81                          udpGet, udpVel, udpClean, udpMesh */
82  #include "udpUtilities.c"

```

- Line 75: this lists the name of all input/output arguments
 - only lowercase characters and digits can be used
- Line 76: the type of each argument
 - see lines 58–72 for available types
 - in general, positive values used for input and negative values used for output
- Line 77: the default values for all integer arguments
- Line 78: the default values for all real (double) arguments
- Line 82: definitions of the routines needed to interact with ESP
 - this line **MUST** be in your UDP/UDF at this location

```

83  /*
84  ****
85  *
86  *   udpExecute - execute the primitive
87  *
88  ****
89  */
90
91  int
92  udpExecute(ego   context,          /* (in)  EGADS context */
93             ego   *ebody,          /* (out) Body (or model) pointer */
94             int    *nMesh,          /* (out) number of associated meshes */
95             char   *string[])       /* (out) error message */
96  {
97      int      status = EGADS_SUCCESS;
98
99      double   data[18];
100      char     *message=NULL;
101      udp_T    *udps = *Udps;
102
103      ROUTINE(udpExecute);
104
105      /* ----- */
106
107  #ifdef DEBUG
108      /* debug printing of the input arguments */
109      printf("udpExecute(context=%llx)\n", (long long)context);
110      printf("radius(0) = %f\n", RADIUS(0));
111  #endif

```

- Line 91: required so that ESP can load this UDP/UDF
- Line 92: required name of the function that ESP will call to execute this UDP/UDF
 - if a UDP, the first argument (`context`) is needed for many of the EGADS routines
 - if a UDF, the first argument will be `emodel`, which is a EGADS MODEL that contains the input Bodys
- Line 93: a pointer to the Body (or MODEL) that is returned from this UDP/UDF to ESP
- Line 94: `nMesh` is currently not used
- Line 95: the error message returned from this UDP/UDF (might be blank)

- Line 97: the default return status
- Line 99: data that is specific to this UDP/UDF
 - in general, there will be many more variables defined here
- Line 100: the declaration character string into which message will be written
 - see below for macros used to allocate and free this
- Line 101: a required line to have the UDP/UDF properly interact with ESP
- Line 103: a required macro definition for reporting errors
 - this should specify the name of the current function
- Line 107–111: prints (if `DEBUG` was defined in line 33) the input arguments that were automatically set up
 - the argument 0 refers to the current call

Listing of template.c (5)

```
112  /* default return values */
113  *ebody = NULL;
114  *nMesh = 0;
115  *string = NULL;
116
117  /* the place where messages to the user are placed */
118  MALLOC(message, char, 100);
119  message[0] = '\0';
120
121  /* check arguments */
122  if (RADIUS_SIZ(0) > 1) {
123      snprintf(message, 100, "\"radius\" should be a scalar");
124      status = OCSM_ILLEGAL_VALUE;
125      goto cleanup;
126
127  } else if (RADIUS(0) <= 0) {
128      snprintf(message, 100, "\"radius\" should be a positive");
129      status = OCSM_ILLEGAL_VALUE;
130      goto cleanup;
131
132  }
133
134  /* cache copy of arguments for future use */
135  status = cacheUdp(NULL);
136  CHECK_STATUS(cacheUdp);
137
138  #ifdef DEBUG
139      /* debug printing of cached input arguments */
140      printf("radius[%d] = %f\n", numUdp, RADIUS(numUdp));
141  #endif
```

- Lines 113–115: default return values (if something goes wrong below)
- Line 118: invocation of the `MALLOC` macro to allocate memory (in this case 100 characters)
 - ensure that on line 100 the pointer was initialized to `NULL`
- Line 119: initialize `message` to a zero-length string
- Lines 122–132: check the validity of the various input arguments
 - error messages are written in `message`
 - `status` is set to an appropriate error number (which should be negative)
 - `egadsErrors.h` defines standard error numbers for EGADS
 - `OpenCSM.h` defined standard error numbers for OpenCSM
 - control is transferred to `cleanup` so that memory and other cleanup functions are executed
 - do not use `return`

- Line 135: this call is required to cache the inputs so that the sensitivity routine can be applied correctly
- Line 136: a macro to check the return status
 - if `status` is negative, the error is reported and necessary cleanups are performed
- Lines 138–141: more debug prints to show that the input arguments were cached properly
 - an argument of 0 refers to the instance being created
 - positive arguments indicate the instance number

```
142     /* make SolidBody */
143     data[0] = 0;
144     data[1] = 0;
145     data[2] = 0;
146     data[3] = RADIUS(0);
147
148     status = EG_makeSolidBody(context, SPHERE, data, ebody);
149     CHECK_STATUS(EG_makeSolidBody);
150
151     SPLINT_CHECK_FOR_NULL(*ebody);
152
153     /* set the output value(s) */
154
155     /* remember this model (Body) */
156     udps[numUdp].ebody = *ebody;
157
158 }
```

- Lines 143–146: set up the array needed for `EG_makeSolidBody`
 - see `EGADS.pdf` (page 94) for a description of what there data are
- Line 148: make the SPHERE `SolidBody`
- Line 149: same as line 136
- Line 151: the invocation of a macro to ensure that `EG_makeSolidBody` actually returned a `Body`
- Line 154: this UDP/UDF does not return any values
- Line 156: a required line to remember the `Body` that was made (so that the sensitivity routines work properly)

```
159 cleanup:
160 #ifdef DEBUG
161     printf("udpExecute -> numUdp=%d, *ebody=%llx\n", numUdp, (long long)(*ebody));
162 #endif
163
164     if (strlen(message) > 0) {
165         *string = message;
166         printf("%s\n", message);
167     } else if (status != EGADS_SUCCESS) {
168         FREE(message);
169         *string = udpErrorStr(status);
170     } else {
171         FREE(message);
172     }
173
174     return status;
175 }
```

- Line 159: the `cleanup` label is **REQUIRED** for the macros to work properly
 - this is where execution continues if an error is encountered
- Lines 160–162: more **DEBUG** printing
- Lines 164–172: required code to properly return error messages to ESP
 - lines 168 and 171 show the `FREE` macro, which has been set up to work with the `MALLOC` macro
- Line 174: the **ONLY** return statement from this function, which returns the error status back to ESP

Listing of template.c (8)

```

176  /*
177  *****
178  *                                                                 *
179  *   udpSensitivity - return sensitivity derivatives for the "real" argument *
180  *                                                                 *
181  *   *****
182  */
183
184  int
185  udpSensitivity(ego      ebody,          /* (in)  Body pointer */
186                int      npnt,          /* (in)  number of points */
187                int      entType,       /* (in)  OCSM entity type */
188                int      entIndex,      /* (in)  OCSM entity index (bias-1) */
189                double   uvs[],         /* (in)  parametric coordinates for evaluation */
190                double   vels[])        /* (out) velocities */
191  {
192      int      status = EGADS_SUCCESS;
193
194      int      iudp, judp;
195
196      ROUTINE(udpSensitivity);
197
198      /* ----- */
199
200  #ifdef DEBUG
201      if (uvs != NULL) {
202          printf("udpSensitivity(ebody=%llx, npnt=%d, entType=%d, entIndex=%d, uvs=%f %f)\n",
203                (long long)ebody, npnt, entType, entIndex, uvs[0], uvs[1]);

```

- For now, leave this code exactly as it is in `template.c`

Listing of template.c (9)

```
204     } else {
205         printf("udpSensitivity(ebody=%llx, npnt=%d, entType=%d, entIndex=%d, uvs=NULL)\n",
206             (long long)ebody, npnt, entType, entIndex);
207     }
208 #endif
209
210     /* check that ebody matches one of the ebodys */
211     iudp = 0;
212     for (judp = 1; judp <= numUdp; judp++) {
213         if (ebody == udps[judp].ebody) {
214             iudp = judp;
215             break;
216         }
217     }
218     if (iudp <= 0) {
219         status = EGADS_NOTMODEL;
220         goto cleanup;
221     }
222
223     /* the following line should be included if sensitivities
224        are not computed analytically */
225     status = EGADS_NOLOAD;
226
227 cleanup:
228 #ifdef DEBUG
229     printf("udpSensitivity -> vels=%f %f %f\n", vels[0], vels[1], vels[2]);
230 #endif
231     return status;
232 }
```

- For now, leave this code exactly as it is in `template.c`

- Make a new UDP (`exercisel`) in the directory (folder) `udpExercisel`
 - the purpose is to make a cylinder that is centered at the origin
 - the input parameters are:
 - `length` - the length of the cylinder
 - `diam` - the diameter of the cylinder
 - `dirn` - a string containing the (single) character “x”, “X”, “y”, “Y”, “z”, or “Z” to indicate that the cylinder’s axis should be along the x-, y-, or z- axis
- Make sure that your UDP does appropriate error checking