ARMY RESEARCH LABORATORY

Migration of a SEDRIS Application for Dismounted Infantry Simulation

by Andrew M. Neiderer

ARL-TR-3009 **July 2003**

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
00 JUL 2003	**N/A**	**-**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Migration of a SEDRIS Application for Dismounted Infantry Simulation	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Computational and Information Sciences Directorate, ARL, Aberdeen Proving Ground, MD 21005-5067**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES
The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**UU**	**48**	

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5067

ARL-TR-3009 **July 2003**

Migration of a SEDRIS Application for Dismounted Infantry Simulation

Andrew M. Neiderer
Computational and Information Sciences Directorate, ARL

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
July 2003	Final	December 2002 – April 2003

4. TITLE AND SUBTITLE
Migration of a SEDRIS Application for Dismounted Infantry Simulation

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Andrew M. Neiderer

5d. PROJECT NUMBER
3TEDNC

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
ATTN: AMSRL-CI-CT
Aberdeen Proving Ground, MD 21005-5067

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-3009

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The U.S. Army Research Laboratory Dismounted Infantry Simulator has been updated for Synthetic Environment Data Representation and Interchange Specification (SEDRIS) 3.1.1 transmittal generation. Specifically, the SEDRIS 3.0.3 write application for a munition/structure interaction, as computed by DISim, is now compliant with the 3.1.1 specification. SEDRIS defines a comprehensive standard for the representation and distribution of environmental data. Our initial effort only involved two of the five components of the SEDRIS technology—the Data Representation Model and the Application Programming Interface. The other three components will be added directly, if needed, since translation would not be necessary. It is our intent to not only present a unique solution but to also identify potential areas of concern for a general 3.0.3 to 3.1.1 migration.

15. SUBJECT TERMS
SEDRIS, Data Representation Model, transmittal generation, distributed simulation programming

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Andrew M. Neiderer
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL	46	19b. TELEPHONE NUMBER (*Include area code*) 410-278-3203

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Tables iv

1. Introduction 1

2. Migration of STF_Fracture 1

3. Conclusion and Future Efforts 4

4. References 5

Appendix A. Tree Representation of STF_Fracture Transmittal 7

Appendix B. 3.0.3 STF_Fracture Class 9

Appendix C. 3.1.1 STF_Fracture Class 25

List of Tables

Table 1. Transmittal root and top level objects of STF_Fracture. 2
Table 2. Changes to model library of STF_Fracture. 3
Table 3. Changes to image library of STF_Fracture 3

1. Introduction

The U.S. Army Research Laboratory (ARL) Dismounted Infantry Simulator (DISim) is a training system for military operations on urban terrain (*1*). One particular application of DISim is the real-time computation and display of an urban structure fractured by a munition (*2*) and subsequent distribution of resultant rubble (*3*). The DISim approximation is empirically based, taking into account only the geometry of the situation; but a physical solution is being considered when real-time display is not compromised. An example of its utility could be when a soldier affixes an explosive on a building and then detonates it for tactical entry to the building interior. Simulation of such a perforation is computed by the DISim Hole Library, which is written in the object-oriented language C++. The resultant geometry and corresponding attribute data for the effected structure(s) are placed in Synthetic Environment Data Representation and Interchange Specification (SEDRIS) transmittal(s); the C++ language bindings of the application programming interface (API) are used for transmittal generation. SEDRIS transmittal formatted (STF) data is then available to compliant read APIs on a network of computers.

ARL recently decided to update DISim for the latest SEDRIS technology (from 3.0.3 to 3.1.1), after several suggestions from the SEDRIS mailing list. Many on that list, which is primarily monitored by SEDRIS associates, believe 3.1.1 is not only more stable but also offers a much more consistent interface. Their web site (*4*) provides both guidance and software to promote migration. This includes a Perl script for global string replacement with exact matching. There are restrictions, however, when running the name_change.pl script as documented by a README file at the same web site. Still our application had to be manually edited for full 3.1.1 compliancy.

It is our hope that this report will assist in 3.0.3 to 3.1.1 migration of SEDRIS applications. We only address the Data Representation Model and the API for interchange of the environmental data. Our application did not use the other components—namely, the Spatial Reference Model, the Environmental Data Coding Specification, and the STF. Those who are just learning SEDRIS may want to go directly to the 3.1.1 specification.

2. Migration of STF_Fracture

Appendix A depicts the format of an STF_Fracture transmittal. The actual C++ class for the 3.0.3 write API (Appendix B) has been modified for 3.1.1 transmittal generation (Appendix C). Both versions are included in the appendices for a line-by-line comparison. The overall design of the class has stayed the same, but the representation (member data) and the interface (member

functions) had to be changed. Data types, class fields, and constants were either renamed or removed as necessary.

The STF_Fracture interface has been divided into three tables. Table 1 consists of methods for instantiating an STF_Fracture object, including the transmittal root and the following top-level objects: description, point of contact, access, data quality, keywords, absolute time point, citation, rocess, and transmittal summary. All text strings, which are simply defined as a sequence of ASCII characters, were updated as shown. Sometimes this was an entire class name (e.g., SE_OBJECT renamed to SE_Object), and other times only the field name had to be changed (e.g., string_value renamed to characters). Table 2 lists string values that were changed when constructing containers of geometric and attribute data. Each geometric primitive, which is a triangle in our simulation environment, has a mapping function pointing to data in the image library. The addImage() method in Table 3 is needed for placing texture data in this library. Note that multiple occurrences of identical replacements are not repeated in successive tables. This was done to minimize entries in these three tables.

Table 1. Transmittal root and top level objects of STF_Fracture.

Method Name	3.0.3 String	3.1.1 String
create()	SE_CREATE SE_SUCCESS	SE_AC_MODE_CREATE SE_STAT_CODE_SUCCESS
createDescription()	SE_OBJECT SE_FIELDS SE_TRANSMITTAL_ROOT_TOKEN string_length string_value SE_SUCCESS SE_DESCRIPTION_TOKEN SE_POINT_OF_CONTACT_TOKEN SE_ACCESS_TOKEN SE_DATA_QUALITY_TOKEN SE_KEYWORDS_TOKEN SE_ABSOLUTE_TIME_POINT_TOKEN SE_CITATION_TOKEN SE_PROCESS_TOKEN SE_TRANSMITTAL_SUMMARY_TOKEN	SE_Object SE_Fields SE_DRM_CLS_TRANSMITTAL_ROOT length characters SE_STAT_CODE_SUCCESS SE_DRM_CLS_DESCRIPTION SE_DRM_CLS_POINT_OF_CONTACT SE_DRM_CLS_ACCESS SE_DRM_CLS_DATA_QUALITY SE_DRM_CLS_KEYWORDS SE_DRM_CLS_ABSOLUTE_TIME_POINT SE_DRM_CLS_CITATION SE_DRM_CLS_PROCESS SE_DRM_CLS_TRANSMITTAL_SUMMARY

The approach taken was to examine the core header file sedris.h for actual changes to the API. Although this may not have been the most efficient approach for simple changes, it did result in a better understanding of the new definition. The alternative is using the previously mentioned script.

The application programmer will probably have little difficulty in strict string replacement for a particular case. However, actual logic changes require a more thorough understanding of SEDRIS technology. In our particular application, there were two main areas of concern. First, it was determined that the SE_AddToTransmittal() function had to be removed in 3.1.1. This

Table 2. Changes to model library of STF_Fracture.

Method Name	3.0.3 String	3.1.1 String
createModelLib()	SE_OBJECT SE_FIELDS SE_MODEL_LIBRARY_TOKEN SE_MODEL_TOKEN string_length string_value SE_GEOMETRY MODEL_TOKEN SE_UNION_OF_PRIMITIVE_GEOMETRY_TOKEN	SE_Object SE_Fields SE_DRM_CLS_MODEL_LIBRARY SE_DRM_CLS_MODEL length characters SE_DRM_CLS_GEOMETRY_MODEL SE_DRM_CLS_UNION_OF_PRIMITIVE_GEOMETRY
addPolygons()	SE_POLYGON_TOKEN SE_IMAGE_MAPPING_FUNCTION_TOKEN SE_REPLACE_IMAGE SE_REPEAT_IMAGE SE_PLANAR_IMAGE_PROJECTION SE_VERTEX_TOKEN SE_LSR_LOCATION_3D_TOKEN LSR_Location_3D.x LSR_Location_3D.y LSR_Location_3D.z SE_RGB_COLOR_TOKEN SE_TEXTURE_COORDINATE_TOKEN	SE_DRM_CLS_POLYGON SE_DRM_CLS_IMAGE_MAPPING_FUNCTION SE_IMG_MAPNG_METH_REPLACE SE_IMG_WRAP_REPEAT SE_IMG_PROJ_TYP_PLANAR SE_DRM_CLS_VERTEX SE_DRM_CLS_LSR_LOCATION_3D LSR_Location_3D.coordinate.x LSR_Location_3D.coordinate.y LSR_Location_3D.coordinate.z SE_DRM_CLS_RGB_COLOUR SE_DRM_CLS_TEXTURE_COORDINATE

Table 3. Changes to image library of STF_Fracture.

Method Name	3.0.3 String	3.1.1 String
createImageLib()	SE_FIELDS SE_IMAGE_LIBRARY_TOKEN	SE_Fields SE_DRM_CLS_IMAGE_LIBRARY
addImage()	SE_OBJECT SE_IMAGE_TOKEN string_length string_value color_model SE_RGB_MODEL mip_fields_array SE_IMAGE_MIP_FIELDS SE_IMAGE_SCAN_RU SE_IMAGE_SCAN_Z_NO_Z SE_UNSIGNED_INTEGER SE_IMAGE_SIGNATURE_LUMINANCE bits_of_first_color_coord bits_of_second_color_coord bits_of_third_color_coord max_value_of_first_color_coord max_value_of_second_color_coord max_value_of_third_color_coord SE_IMAGE_SIGNATURE_123COLOR	SE_Object SE_DRM_CLS_IMAGE length characters colour_model SE_CLR_MDL_RGB mip_extents_array SE_Image_MIP_Extents SE_IMG_SCAN_DIR_RIGHT_UP SE_IMG_SCAN_DIR_Z_NONE SE_IMG_COMP_TYP_UNSIGNED_INTEGER SE_IMG_SIG_LUMINANCE bits_of_colour_coordinate_1 bits_of_colour_coordinate_2 bits_of_colour_coordinate_3 max_value_of_colour_coordinate_1 max_value_of_colour_coordinate_2 max_value_of_colour_coordinate_3 SE_IMG_SIG_123COLOUR

impacted lines 56–59 of the original STF_Fracture.C (Appendix B). The operation is now absorbed when the object is created.

Also, the 3.1.1 specification states that the implementation identifier argument of SE_CreateObject() needs to be replaced by the SE_Transmittal pointer. Secondly, two fields of the image mapping function were removed—the id and presentation_domain fields. SEDRIS 3.1.1 guides state that this can be done safely.

3. Conclusion and Future Efforts

This report presented details for translating the STF_Fracture class of DISim from SEDRIS 3.0.3 to a 3.1.1 implementation. The exact string replacements were given in the three tables. In addition, logic changes were identified, and appropriate methods were modified.

We are now in the process of validating the generated transmittals for full compliancy. "Side-by-Side" (SbS) by Acusoft, Inc., will be used for graphical display of resultant transmittals (*5*). However, the latest version (SbS 2.5) must be used to view 3.1.1 transmittals.

Future research will include the evaluation of existing technologies for synthetic environment representation in a Web environment. Possibilities are extensible markup language applications, such as extensible 3D (X3D), and the extensible modeling and simulation framework (XMSF), which allows for easy interaction over a highly distributed network. SEDRIS provides a thorough description of synthetic environments, but simulators on a network must have a read API to understand these transmittals. While XMSF may not be a replacement for SEDRIS, it could provide an extension for easier interoperation of simulation applications.

4. References

1. Thomas, M. A. *Dismounted Infantry Visualization Research: The Dismounted Infantry Simulation (DISim)*; ARL-TN-193; U.S. Army Research Laboratory: Aberdeen Proving Ground, MD, 2002.

2. Neiderer, A. M.; Thomas, M. A.; Pearson, R. *A Fracturing of Polygonal Objects*; ARL-TR-1649; U.S. Army Research Laboratory: Aberdeen Proving Ground, MD, 1998.

3. Neiderer, A. M.; Thomas, M. A.; Hansen, C. E. *Distribution of Fragments Resulting From Polygonal Object Fracture*; ARL-TN-182; U.S. Army Research Laboratory: Aberdeen Proving Ground, MD, 2001.

4. SEDRIS Home Page. http://www.sedris.org (accessed May 2003)

5. Neiderer, A. M. *SEDRIS Transmittal Generation for a Dismounted Infantry Simulator*; ARL-TR-2801; U.S. Army Research Laboratory: Aberdeen Proving Ground, MD, 2002.

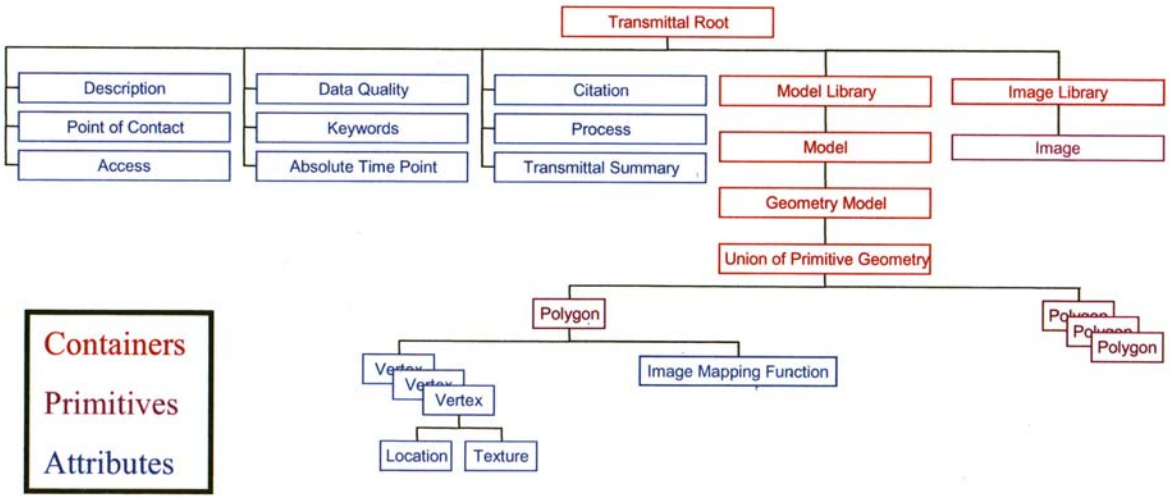
6. Carswell, J.; Campos, J.; Cox, R. *The SEDRIS Transmittal Format*; Science Applications International Corporation: Orlando, FL, 1999.

INTENTIONALLY LEFT BLANK.

Appendix A. Tree Representation of STF_Fracture Transmittal

The following diagram organizes a Synthetic Environment Data Representation and Interchange Specification (SEDRIS) transmittal in hierarchical format. A SEDRIS transmittal is generated for each munition/structure fracture in the Dismounted Infantry Simulator.

STF_Fracture Transmittal



Appendix B. 3.0.3 STF_Fracture Class

The C++ class STF_Fracture encapsulates both data and functions for a Synthetic Environment Data Representation and Interchange Specification 3.0.3 transmittal of a Dismounted Infantry Simulator fracture. The member data and method interfaces are given in the header file STF_Fracture.H; methods are subsequently defined in the source file STF_Fracture.C. Note that a default constructor and destructor are also defined in STF_Fracture.H.

The following pages in this appendix appear in their original form, without editorial change.

```
/* ============================================================================================== */
/*                                                                                                */
/*   File name:  STF_Fracture.H                                                                   */
/*                                                                                                */
/*   Purpose:    class to encapsulate (1) functions and (2) data for                              */
/*                SEDRIS transmittal creation of polygons.                                        */
/*                                                                                                */
/*   By:         Andrew M. Neiderer  26 March, '02.                                               */
/* ============================================================================================== */

#include <stdio.h>

#include <Performer/pf.h>                          // for pfVec3

#include "se_read0.h"                              // for opening and retrieving data from SEDRIS transmittals
#include "se_write0.h"                             // for creating and modifying SEDRIS transmittals

class STF_Fracture {
  public:                                          // interface
                                                   //   (1) manager functions
    STF_Fracture()                                 //     (a) default constructor function
    {
#ifdef Debug
fprintf(stdout, "    STF_Fracture()\n\n");
#endif

      _impl_id = "stf";
      _xmittal = NULL;
      _xmittal_name = NULL;
      _root = _model_lib = _union = _image_lib = _image = NULL;
      _model_id = 1;
      _image_id = 1;
    }

    ~STF_Fracture()                                //     (b) default destructor function
    {
#ifdef Debug
fprintf(stdout, "    ~STF_Fracture()\n\n");
#endif

      SE_FreeObject(_union);
      SE_FreeObject(_model_lib);
      SE_FreeObject(_image);
      SE_FreeObject(_image_lib);
      SE_FreeObject(_root);

      if ( SE_CloseTransmittal(_xmittal) != SE_SUCCESS )
        printf("Error: failed to close transmittal\n");
    }

                                                   //   (2) implementor functions
```

```
    int createDescription();                        //      (a) create top level objects
                                                    //
    int createImageLib();                           //      (b) create Image Library
                                                    //
    int addImage(unsigned int *, int, int, int);    //      (c) add an Image to Image Library
                                                    //
    int createModelLib();                           //      (d) create Model Library and add a Model
                                                    //
    int addPolygons(pfVec3 *,                       //      (e) add Polygons to Model's union of geometry
                    pfVec2 *, pfTexture *,          //
                    int);                           //
                                                    //
    int create(char *,                              //      (f) open SEDRIS transmittal, then
               pfVec3 *,                            //           create description, Image Library, and Model Library.
               pfVec2 *, pfTexture *,               //
               int);                                //
                                                    //
  private:                                          //
                                                    //   (3) helping functions
    int createObj(SE_TOKEN_ENUM,                   //     (a) create an object
                  SE_OBJECT *,                      //
                  SE_FIELDS * = NULL);              //
    int addObj(SE_OBJECT,                           //     (b) add an object
               SE_OBJECT,                           //
               SE_FIELDS * = NULL,                  //
               SE_OBJECT = NULL,                    //
               SE_FIELDS * = NULL);                 //
    int freeObj(SE_OBJECT);                         //     (c) release an object
                                                    //
  protected:                                        // representation
    char           *_impl_id;                       //   implementation identifier
    SE_TRANSMITTAL _xmittal;                        //   transmittal ptr
    char           *_xmittal_name;                  //   transmittal name
    SE_OBJECT      _root,                           //   Transmittal Root
                   _model_lib,                      //   Model Library
                   _union,                          //   union of Primitive Geometry
                   _image_lib,                      //   Image Library
                   _image;                          //   Image for texture
    int            _model_id;                       //   model identifier
    int            _image_id;                       //   image identifier
};
```

```
 1: /* ========================================================================================== */
 2: /*                                                                                            */
 3: /*   File name: STF_Fracture.C                                                                */
 4: /*                                                                                            */
 5: /*   Purpose:   implementation of member functions, ie methods, of STF_Fracture class         */
 6: /*                                                                                            */
 7: /*   Functions, parameters:                                                                   */
 8: /*               (1) createObj - create an object                                             */
 9: /*               (2) addObj - add an object                                                   */
10: /*               (3) freeObj - release an object                                              */
11: /*               (4) createDescription - create top level objects                             */
12: /*               (5) createImageLib - create Image Library                                    */
13: /*               (6) addImage - add an Image to Image Library                                 */
14: /*               (7) createModelLib - create Model Library and add a Model                    */
15: /*               (8) addPolygons - add polygons to Model's union of geometry                  */
16: /*               (9) create - open SEDRIS transmittal, then create description, Image         */
17: /*                            Library, and Model Library.                                     */
18: /*                                                                                            */
19: /*   By:        Andrew M. Neiderer 26 March '02                                              */
20: /* ========================================================================================== */
21:
22: #include <stdio.h>
23: #include <iostream.h>
24: #include <string.h>                      // for strlen()
25: #include <malloc.h>                      // for malloc()
26:
27: #include <Performer/pf.h>                // for pfVec3
28: #include <Performer/pr/pfTexture.h>      // for pfTexture::getImage()
29:
30: #include "STF_Fracture.H"                // for STF_Fracture class definition
31:
32: /* ************************************************************ */
33: /* createObj (type, obj, obj_f)                                 */
34: /*   SE_TOKEN_ENUM type      (in) -                             */
35: /*   SE_OBJECT *obj          (in) - object                      */
36: /*   SE_FIELDS *obj_f        (in) - object fields               */
37: /*                                                              */
38: /* Create an object.                                            */
39: /*                                                              */
40: /* Return true (non-zero) or false (zero).                      */
41: /* ************************************************************ */
42:
43: int STF_Fracture::createObj(SE_TOKEN_ENUM type,
44:                             SE_OBJECT *obj,
45:                             SE_FIELDS *obj_f)
46: {
47:
48: #ifdef Debug
49: fprintf(stdout, "    STF_Fracture::createObj()\n\n");
50: #endif
51:
```

```
52:   if ( SE_CreateObject(_impl_id,type,obj) != SE_SUCCESS ) {
53:     printf("Error: Unable to create object\n");
54:     return 0;
55:   }
56:   else if ( SE_AddToTransmittal(*obj,_xmittal) != SE_SUCCESS ) {
57:     printf("Error: Unable to add object to transmittal\n");
58:     return 0;
59:   }
60:   else if ( obj_f  &&  SE_SetFieldsToDefault(type,obj_f) != SE_SUCCESS ) {
61:     printf("Error: Unable to set default fields\n");
62:     return 0;
63:   }
64:
65:   return 1;
66: }
67:
68: /* ************************************************************** */
69: /* addObj (aggr, comp, comp_f, link, link_f)                      */
70: /*   SE_OBJECT aggr          (in) - aggregate                     */
71: /*   SE_OBJECT comp          (in) - component                     */
72: /*   SE_FIELDS *comp_f       (in) - component fields              */
73: /*   SE_OBJECT link          (in) - link                          */
74: /*   SE_FIELDS *link_f       (in) - link fields                   */
75: /*                                                                */
76: /* Add an object.                                                 */
77: /*                                                                */
78: /* Return true (non-zero) or false (zero).                        */
79: /* ************************************************************** */
80:
81: int STF_Fracture::addObj(SE_OBJECT aggr,
82:                          SE_OBJECT comp,
83:                          SE_FIELDS *comp_f,
84:                          SE_OBJECT link,
85:                          SE_FIELDS *link_f)
86: {
87:
88: #ifdef Debug
89: fprintf(stdout, "    STF_Fracture::addObj()\n\n");
90: #endif
91:
92:   if ( (comp_f  &&  SE_PutFields(comp,comp_f) != SE_SUCCESS)  ||
93:        (link  &&  link_f  &&  SE_PutFields(link,link_f) != SE_SUCCESS)  ||
94:        SE_AddComponentRelationship(aggr,comp,link) != SE_SUCCESS ) {
95:     printf("Error: couldn't complete addObj()\n");
96:     return 0;
97:   }
98:
99:   return 1;
100: }
101:
102: /* ************************************************************** */
```

```
103: /* freeObj (obj)                                                    */
104: /*   SE_OBJECT obj          (in) - object                           */
105: /*                                                                  */
106: /* Release an object.                                               */
107: /*                                                                  */
108: /* Return true (non-zero) or false (zero).                          */
109: /* ***************************************************************** */
110:
111: int STF_Fracture::freeObj(SE_OBJECT obj)
112: {
113:
114: #ifdef Debug
115: fprintf(stdout, "    STF_Fracture::freeObj()\n\n");
116: #endif
117:
118:   if ( SE_FreeObject(obj) != SE_SUCCESS ) {
119:     printf("Error: free object failed\n");
120:     return 0;
121:   }
122:
123:   return 1;
124: }
125:
126: /* ***************************************************************** */
127: /* createDescription ()                                             */
128: /*                                                                  */
129: /* Create top level objects - Description, Point of Contact, Access,  */
130: /*  Data Quality, Keywords, Absolute Time Point, Citation, Process,   */
131: /*  Transmittal Summary.                                            */
132: /*                                                                  */
133: /* Return true (non-zero) or false (zero).                          */
134: /* ***************************************************************** */
135:
136: int STF_Fracture::createDescription()
137: {
138:   SE_OBJECT obj;
139:   SE_FIELDS f;
140:   char *s;
141:
142: #ifdef Debug
143: fprintf(stdout, "    STF_Fracture::createDescription()\n\n");
144: #endif
145:
146:   // Transmittal Root
147:
148:   if ( !createObj(SE_TRANSMITTAL_ROOT_TOKEN,&_root,&f) )
149:     return 0;
150:
151:   f.u.Transmittal_Root.name.string_length = strlen(_xmittal_name);
152:   f.u.Transmittal_Root.name.string_value = _xmittal_name;
153:
```

```
154:   if ( SE_PutFields(_root,&f) != SE_SUCCESS  ||
155:        SE_SetRootObject(_xmittal,_root,&obj) != SE_SUCCESS ) {
156:     printf("Error: failed to add Transmittal Root\n");
157:     return 0;
158:   }
159:
160:   // Description
161:
162:   createObj(SE_DESCRIPTION_TOKEN,&obj,&f);
163:   s = "Evaluation";
164:   f.u.Description.abstract.string_value = s;
165:   f.u.Description.abstract.string_length = strlen(s);
166:   addObj(_root,obj,&f);
167:   freeObj(obj);
168:
169:   // Point of Contact
170:
171:   createObj(SE_POINT_OF_CONTACT_TOKEN,&obj,&f);
172:   s = "Army Research Laboratory";
173:   f.u.Point_of_Contact.organization.string_value = s;
174:   f.u.Point_of_Contact.organization.string_length = strlen(s);
175:   s = "SEDRIS customer";
176:   f.u.Point_of_Contact.person_or_position.string_value = s;
177:   f.u.Point_of_Contact.person_or_position.string_length = strlen(s);
178:   s = "APG, MD";
179:   f.u.Point_of_Contact.address.string_value = s;
180:   f.u.Point_of_Contact.address.string_length = strlen(s);
181:   s = "410-278-3203";
182:   f.u.Point_of_Contact.voice_phone.string_value = s;
183:   f.u.Point_of_Contact.voice_phone.string_length = strlen(s);
184:   addObj(_root,obj,&f);
185:   freeObj(obj);
186:
187:   // Access
188:
189:   createObj(SE_ACCESS_TOKEN,&obj,&f);
190:   addObj(_root,obj,&f);
191:   freeObj(obj);
192:
193:   // Data Quality
194:
195:   createObj(SE_DATA_QUALITY_TOKEN,&obj,&f);
196:   f.u.Data_Quality.fictional = SE_TRUE;
197:   addObj(_root,obj,&f);
198:   freeObj(obj);
199:
200:   // Keywords
201:
202:   createObj(SE_KEYWORDS_TOKEN,&obj,&f);
203:   addObj(_root,obj,&f);
204:   freeObj(obj);
```

```
205:
206:    // Absolute Time Point
207:
208:    createObj(SE_ABSOLUTE_TIME_POINT_TOKEN,&obj,&f);
209:    f.u.Absolute_Time_Point.time_significance = SE_CREATION_DATE;
210:    f.u.Absolute_Time_Point.year = 2002;
211:    f.u.Absolute_Time_Point.month = SE_FEBRUARY;
212:    f.u.Absolute_Time_Point.day = 20;
213:    f.u.Absolute_Time_Point.hour = 15;
214:    f.u.Absolute_Time_Point.minutes = 0;
215:    f.u.Absolute_Time_Point.seconds = 0;
216:    addObj(_root,obj,&f);
217:    freeObj(obj);
218:
219:    // Citation
220:
221:    createObj(SE_CITATION_TOKEN,&obj,&f);
222:    s = "Fracture Database";
223:    f.u.Citation.title.string_value = s;
224:    f.u.Citation.title.string_length = strlen(s);
225:    s = "ARL/CISD, APG";
226:    f.u.Citation.originators.string_value = s;
227:    f.u.Citation.originators.string_length = strlen(s);
228:    addObj(_root,obj,&f);
229:    freeObj(obj);
230:
231:    // Process
232:
233:    createObj(SE_PROCESS_TOKEN,&obj,&f);
234:    s = "Pre-process";
235:    f.u.Description.abstract.string_value = s;
236:    f.u.Description.abstract.string_length = strlen(s);
237:    addObj(_root,obj,&f);
238:    freeObj(obj);
239:
240:    // Transmittal Summary
241:
242:    createObj(SE_TRANSMITTAL_SUMMARY_TOKEN,&obj,&f);
243:    f.u.Transmittal_Summary.geometry_present = SE_PRESENT_IN_MODELS;
244:    f.u.Transmittal_Summary.models_present = SE_TRUE;
245:    addObj(_root,obj,&f);
246:    freeObj(obj);
247:
248:    return 1;
249: }
250:
251: /* ************************************************************** */
252: /* createImageLib ()                                              */
253: /*                                                                */
254: /* Create Image Library.                                          */
255: /*                                                                */
```

```
256: /* Return true (non-zero) or false (zero).                          */
257: /* ****************************************************************** */
258:
259: int STF_Fracture::createImageLib()
260: {
261:   SE_FIELDS f;
262:
263: #ifdef Debug
264: fprintf(stdout, "    STF_Fracture::createImageLib()\n\n");
265: #endif
266:
267:   createObj(SE_IMAGE_LIBRARY_TOKEN,&_image_lib,&f);
268:   addObj(_root,_image_lib,&f);
269:
270:   return 1;
271: }
272:
273: /* ****************************************************************** */
274: /* addImage (data, xSize, ySize, components)                          */
275: /*   unsigned int *data             (in) - image data                 */
276: /*   int xSize                      (in) - horizontal size            */
277: /*   int ySize                      (in) - vertical size              */
278: /*   int components                 (in) - no. of 8-bit components    */
279: /*                                         per pixel.                 */
280: /*                                                                    */
281: /* Add Image to Image Library.                                        */
282: /*                                                                    */
283: /* Return true (non-zero) or false (zero).                            */
284: /* ****************************************************************** */
285:
286: int STF_Fracture::addImage(unsigned int *data,
287:                            int xSize, int ySize, int components)
288: {
289:   SE_FIELDS f;
290:   char *s;
291:
292: #ifdef Debug2
293: fprintf(stdout, "    STF_Fracture::addImage()\n\n");
294: #endif
295:
296:   if ( _image != (SE_OBJECT) NULL )
297:     freeObj(_image);
298:   createObj(SE_IMAGE_TOKEN,&_image,&f);
299:
300:   f.u.Image.ID = _image_id;
301:   s = "Texture";
302:   f.u.Image.name.string_value = s;
303:   f.u.Image.name.string_length = strlen(s);
304:   f.u.Image.color_model = SE_RGB_MODEL;
305:   f.u.Image.level_count = 1;
306:   f.u.Image.mip_fields_array = (SE_IMAGE_MIP_FIELDS *) calloc(1, sizeof(SE_IMAGE_MIP_FIELDS));
```

```
307:   f.u.Image.mip_fields_array->size_horizontal = xSize;
308:   f.u.Image.mip_fields_array->size_vertical = ySize;
309:   f.u.Image.mip_fields_array->size_z = 1;
310:   f.u.Image.scan_direction = SE_IMAGE_SCAN_RU;
311:   f.u.Image.scan_direction_z = SE_IMAGE_SCAN_Z_NO_Z;
312:   f.u.Image.component_data_type = SE_UNSIGNED_INTEGER;
313: //  f.u.Image.data_is_little_endian = SE_TRUE;
314:
315:   switch (components) {
316:     case 1:
317:       f.u.Image.data_is_3D = SE_FALSE;
318:       f.u.Image.image_signature = SE_IMAGE_SIGNATURE_LUMINANCE;
319:       f.u.Image.bits_of_alpha = 0;
320:       f.u.Image.bits_of_luminance = 8;
321:       f.u.Image.bits_of_first_color_coord = 0;
322:       f.u.Image.bits_of_second_color_coord = 0;
323:       f.u.Image.bits_of_third_color_coord = 0;
324:       f.u.Image.max_value_of_alpha = 0;
325:       f.u.Image.max_value_of_luminance = 255;
326:       f.u.Image.max_value_of_first_color_coord = 0;
327:       f.u.Image.max_value_of_second_color_coord = 0;
328:       f.u.Image.max_value_of_third_color_coord = 0;
329:
330:       break;
331:     case 2:
332:       f.u.Image.data_is_3D = SE_FALSE;
333:       f.u.Image.image_signature = SE_IMAGE_SIGNATURE_123COLOR;
334:       f.u.Image.bits_of_alpha = 0;
335:       f.u.Image.bits_of_luminance = 0;
336:       f.u.Image.bits_of_first_color_coord = 8;
337:       f.u.Image.bits_of_second_color_coord = 8;
338:       f.u.Image.bits_of_third_color_coord = 0;
339:       f.u.Image.max_value_of_alpha = 0;
340:       f.u.Image.max_value_of_luminance = 0;
341:       f.u.Image.max_value_of_first_color_coord = 255;
342:       f.u.Image.max_value_of_second_color_coord = 255;
343:       f.u.Image.max_value_of_third_color_coord = 0;
344:
345:       break;
346:     case 3:
347:       f.u.Image.data_is_3D = SE_TRUE;
348:       f.u.Image.image_signature = SE_IMAGE_SIGNATURE_123COLOR;
349:       f.u.Image.bits_of_alpha = 0;
350:       f.u.Image.bits_of_luminance = 0;
351:       f.u.Image.bits_of_first_color_coord = 8;
352:       f.u.Image.bits_of_second_color_coord = 8;
353:       f.u.Image.bits_of_third_color_coord = 8;
354:       f.u.Image.max_value_of_alpha = 0;
355:       f.u.Image.max_value_of_luminance = 0;
356:       f.u.Image.max_value_of_first_color_coord = 255;
357:       f.u.Image.max_value_of_second_color_coord = 255;
```

```
358:       f.u.Image.max_value_of_third_color_coord = 255;
359:
360:       break;
361:     case 4:
362:       f.u.Image.data_is_3D = SE_FALSE;
363:       f.u.Image.image_signature = SE_IMAGE_SIGNATURE_123COLOR;
364:       f.u.Image.bits_of_alpha = 8;
365:       f.u.Image.bits_of_luminance = 0;
366:       f.u.Image.bits_of_first_color_coord = 8;
367:       f.u.Image.bits_of_second_color_coord = 8;
368:       f.u.Image.bits_of_third_color_coord = 8;
369:       f.u.Image.max_value_of_alpha = 8;
370:       f.u.Image.max_value_of_luminance = 0;
371:       f.u.Image.max_value_of_first_color_coord = 255;
372:       f.u.Image.max_value_of_second_color_coord = 255;
373:       f.u.Image.max_value_of_third_color_coord = 255;
374:
375:       break;
376:     default:
377:       f.u.Image.data_is_3D = SE_TRUE;
378:       f.u.Image.image_signature = SE_IMAGE_SIGNATURE_123COLOR;
379:       f.u.Image.bits_of_alpha = 0;
380:       f.u.Image.bits_of_luminance = 0;
381:       f.u.Image.bits_of_first_color_coord = 8;
382:       f.u.Image.bits_of_second_color_coord = 8;
383:       f.u.Image.bits_of_third_color_coord = 8;
384:       f.u.Image.max_value_of_alpha = 0;
385:       f.u.Image.max_value_of_luminance = 0;
386:       f.u.Image.max_value_of_first_color_coord = 255;
387:       f.u.Image.max_value_of_second_color_coord = 255;
388:       f.u.Image.max_value_of_third_color_coord = 255;
389:
390:       break;
391:   }
392:   addObj(_image_lib,_image,&f);
393:
394:   SE_PutImageData(_image,0,0,0,xSize - 1,ySize - 1,0,0,xSize * ySize * components,(unsigned char *) data);
395:
396:   return 1;
397: }
398:
399: /* ************************************************************** */
400: /* createModelLib ()                                              */
401: /*                                                                */
402: /* Create Model Library and add a Model.                          */
403: /*                                                                */
404: /* Return true (non-zero) or false (zero).                        */
405: /* ************************************************************** */
406:
407: int STF_Fracture::createModelLib()
408: {
```

```
409:   SE_OBJECT obj, obj2;
410:   SE_FIELDS f;
411:   char *s;
412:
413: #ifdef Debug
414: fprintf(stdout, "    STF_Fracture::createModelLib()\n\n");
415: #endif
416:
417:   createObj(SE_MODEL_LIBRARY_TOKEN,&_model_lib,&f);
418:   addObj(_root,_model_lib,&f);
419:
420:   createObj(SE_MODEL_TOKEN,&obj,&f);
421:   f.u.Model.ID = _model_id;
422:   s = "Fracture";
423:   f.u.Model.name.string_value = s;
424:   f.u.Model.name.string_length = strlen(s);
425:   f.u.Model.model_reference_type = SE_ROOT_MODEL;
426:   f.u.Model.srf_params.is_2D = SE_FALSE;
427:   f.u.Model.srf_params.u.params_3d.spatial_reference_frame = SRM_LSR_3D_SRF;
428:   f.u.Model.srf_params.u.params_3d.u.lsr_parameters.up_direction =
429:     SE_DIRECTION_OF_UP_POSITIVE_TERTIARY_AXIS;
430:   f.u.Model.srf_params.u.params_3d.u.lsr_parameters.forward_direction =
431:     SE_DIRECTION_OF_FORWARD_POSITIVE_SECONDARY_AXIS;
432:   addObj(_model_lib,obj,&f);
433:
434:   createObj(SE_GEOMETRY_MODEL_TOKEN,&obj2,&f);
435:   addObj(obj,obj2,&f);
436:
437:   freeObj(obj);
438:
439:   createObj(SE_UNION_OF_PRIMITIVE_GEOMETRY_TOKEN,&_union,&f);
440:   addObj(obj2,_union,&f);
441:
442:   freeObj(obj2);
443:
444:   return 1;
445: }
446:
447: /* ************************************************************** */
448: /* addPolygons (verts, tcoords, texture, nverts)                  */
449: /*   pfVec3 *verts                (in) - vertices                 */
450: /*   pfVec2 *tcoords              (in) - texture coords at vertices */
451: /*   pfTexture *texture           (in) - Performer texture        */
452: /*   int nverts                   (in) - no. of vertices          */
453: /*                                                                */
454: /* Add polygons to Model's union of geometry.                     */
455: /*                                                                */
456: /* Return true (non-zero) or false (zero).                        */
457: /* ************************************************************** */
458:
459: int STF_Fracture::addPolygons(pfVec3 *verts,
```

```
460:                                pfVec2 *tcoords, pfTexture *texture,
461:                                int nverts)
462: {
463:   // SEDRIS API
464:
465:   unsigned char *data;                    // image
466:   SE_OBJECT poly,
467:             vert,
468:             loc, tex, clr,
469:             IMF;
470:   SE_FIELDS f;
471:
472:   // Performer API
473:
474:   unsigned int *image;                    // array of texels
475:   int components;                         // no. of components per pixel
476:   int ns,                                 // no. of texels in s dimension of image
477:       nt,                                 // no. of texels in t dimension of image
478:       nr;                                 // no. of texels in r dimension of image
479:
480:   int i, j;                               // counters
481:
482: #ifdef Debug
483: fprintf(stdout, "    STF_Fracture::addPolygons()\n\n");
484: #endif
485:
486:   // Performer texture
487:
488:   if ( texture != NULL ) {
489:     texture->getImage(&image,&components,&ns,&nt,&nr);
490:
491:     // add to Image Library
492:     addImage(image,ns,nt,components);
493:
494: #ifdef Debug
495:     cerr << "      addPolygons : addImage complete " << texture->getName() << endl;
496: #endif
497:   }
498:   else {
499: #ifdef Debug
500:     cerr << "      addPolygons : no texture for this polygon" << endl;
501: #endif
502:   }
503:
504:   // each polygon has 3 vertices, ie triangle
505:
506:   for ( i = 0; i < nverts; i += 3 ) {
507: #ifdef Debug
508: fprintf(stdout, "      polygon %d\n\n",i / 3);
509: #endif
510:
```

```
511:     createObj(SE_POLYGON_TOKEN,&poly,&f);
512:     addObj(_union,poly,&f);
513:
514:     // Image Mapping Function
515:
516:     if ( texture != NULL ) {
517:       createObj(SE_IMAGE_MAPPING_FUNCTION_TOKEN,&IMF,&f);
518:       f.u.Image_Mapping_Function.image_id = _image_id;
519:       f.u.Image_Mapping_Function.image_mapping_method = SE_REPLACE_IMAGE;
520:       f.u.Image_Mapping_Function.image_wrap_s = SE_REPEAT_IMAGE;
521:       f.u.Image_Mapping_Function.image_wrap_t = SE_REPEAT_IMAGE;
522:       f.u.Image_Mapping_Function.image_projection_type = SE_PLANAR_IMAGE_PROJECTION;
523:       f.u.Image_Mapping_Function.intensity_level = 1;
524:       f.u.Image_Mapping_Function.gain = 0;
525:       f.u.Image_Mapping_Function.image_detail_mapping = SE_FALSE;
526:       f.u.Image_Mapping_Function.presentation_domain = SE_OTW;
527: #ifdef Debug
528:       fprintf(stdout,"      Image_Mapping_Function.image_id = %d\n\n",_image_id);
529: #endif
530:       addObj(poly,IMF,&f);
531:       SE_AddAssociateRelationship(IMF,_image,NULL,SE_FALSE);
532:       freeObj(IMF);
533:     }
534:
535:     // at each vertex, Cartesian coords and texture coords
536:
537:     for ( int k = 0; k < 3; k++ ) {
538:       createObj(SE_VERTEX_TOKEN,&vert,&f);
539:       addObj(poly,vert,&f);
540:       createObj(SE_LSR_LOCATION_3D_TOKEN,&loc,&f);
541:       f.u.LSR_Location_3D.x = verts[i + k].vec[0];
542:       f.u.LSR_Location_3D.y = verts[i + k].vec[1];
543:       f.u.LSR_Location_3D.z = verts[i + k].vec[2];
544:       addObj(vert,loc,&f);
545:
546:       createObj(SE_RGB_COLOR_TOKEN,&clr,&f);
547:       f.u.RGB_Color.rgb_data.red = 1.0;
548:       f.u.RGB_Color.rgb_data.green = 1.0;
549:       f.u.RGB_Color.rgb_data.blue = 1.0;
550:       addObj(vert,clr,&f);
551:       freeObj(clr);
552:
553:       if ( texture != NULL ){
554:         createObj(SE_TEXTURE_COORDINATE_TOKEN,&tex,&f);
555:         f.u.Texture_Coordinate.s = tcoords[i + k].vec[0];
556:         f.u.Texture_Coordinate.t = tcoords[i + k].vec[1];
557:         addObj(vert,tex,&f);
558:         freeObj(tex);
559:
560:       }
561:
```

```
562:       freeObj(loc);
563:       freeObj(vert);
564:     }
565:
566:     freeObj(poly);
567:   }
568:
569:   if ( texture != NULL )
570:     _image_id++;
571:
572:   return 1;
573: }
574:
575: /* ****************************************************************** */
576: /* create (fileName, verts, tcoords, texture, nverts)                 */
577: /*   char *fileName                (in) - SEDRIS transmittal          */
578: /*   pfVec3 *verts                 (in) - vertices                    */
579: /*   pfVec2 *tcoords               (in) - texture coords at vertices  */
580: /*   pfTexture *texture            (in) - Performer texture           */
581: /*   int nverts                    (in) - no. of vertices             */
582: /*                                                                    */
583: /* Open SEDRIS transmittal.  Add description, Image Library, and      */
584: /*   Model Library.                                                   */
585: /*                                                                    */
586: /* Return true (non-zero) or false (zero).                            */
587: /* ****************************************************************** */
588:
589: int STF_Fracture::create(char *fileName,
590:                          pfVec3 *verts,
591:                          pfVec2 *tcoords, pfTexture *texture,
592:                          int nverts)
593: {
594:   char buff[1000];
595:
596: #ifdef Debug
597: fprintf(stdout, "    STF_Fracture::create()\n\n");
598: #endif
599:
600:   if ( SE_OpenTransmittalByFile(fileName,_impl_id,SE_CREATE,&_xmittal) != SE_SUCCESS ) {
601:     printf("Error: can't create \"%s\"\n",fileName);
602:     return 0;
603:   }
604:
605:   sprintf(buff,"urn:x-sedris:%s:1",fileName);
606:
607:   if ( SE_SetTransmittalName(_xmittal,buff) != SE_SUCCESS ) {
608:     printf("Error: failed to set transmittal name\n");
609:     return 0;
610:   }
611:
612:   _xmittal_name = fileName;
```

```
613:
614:   if ( !createDescription()  ||  !createImageLib()  ||
615:        !createModelLib()  ||  !addPolygons(verts,tcoords,texture,nverts) )
616:     return 0;
617:
618:   return 1;
619: }
620:
```

Appendix C. 3.1.1 STF_Fracture Class

The C++ class for Synthetic Environment Data Representation and Interchange Specification 3.1.1 transmittal generation of urban structure fracture is now given. This class is similiar to that in Appendix B but differs due to 3.1.1 requirements.

The following pages in this appendix appear in their original form, without editorial change.

```
/* ============================================================================================== */
/*                                                                                                */
/*   File name:  STF_Fracture.H                                                                   */
/*                                                                                                */
/*   Purpose:    class to encapsulate (1) functions and (2) data for                              */
/*                SEDRIS transmittal creation of polygons.                                        */
/*                                                                                                */
/*   By:         Andrew M. Neiderer  26 March, '02.                                               */
/* ============================================================================================== */

#include <stdio.h>

#include <Performer/pf.h>                       // for pfVec3

#include "se_read0.h"                           // for opening and retrieving data from SEDRIS transmittals
#include "se_write0.h"                          // for creating and modifying SEDRIS transmittals

class STF_Fracture {
  public:                                       // interface
                                                //   (1) manager functions
    STF_Fracture()                              //     (a) default constructor function
    {
#ifdef Debug
fprintf(stdout, "    STF_Fracture()\n\n");
#endif

      _impl_id = "stf";
      _xmittal = NULL;
      _xmittal_name = NULL;
      _root = _model_lib = _union = _image_lib = _image = NULL;
      _model_id = 1;
      _image_id = 1;
    }

    ~STF_Fracture()                             //     (b) default destructor function
    {
#ifdef Debug
fprintf(stdout, "    ~STF_Fracture()\n\n");
#endif

      SE_FreeObject(_union);
      SE_FreeObject(_model_lib);
      SE_FreeObject(_image);
      SE_FreeObject(_image_lib);
      SE_FreeObject(_root);

      if ( SE_CloseTransmittal(_xmittal) != SE_STAT_CODE_SUCCESS )
        printf("Error: failed to close transmittal\n");
    }

                                                //   (2) implementor functions
```

```
      int createDescription();                          //     (a) create top level objects
                                                        //
      int createImageLib();                             //     (b) create Image Library
                                                        //
      int addImage(unsigned int *, int, int, int);      //     (c) add an Image to Image Library
                                                        //
      int createModelLib();                             //     (d) create Model Library and add a Model
                                                        //
      int addPolygons(pfVec3 *,                         //     (e) add Polygons to Model's union of geometry
                      pfVec2 *, pfTexture *,            //
                      int);                             //
                                                        //
      int create(char *,                                //     (f) open SEDRIS transmittal, then
                 pfVec3 *,                              //          create description, Image Library, and Model Library.
                 pfVec2 *, pfTexture *,                 //
                 int);                                  //
                                                        //
    private:                                            //
                                                        //   (3) helping functions
      int createObj(SE_DRM_Class,                       //     (a) create an object
                    SE_Object *,                        //
                    SE_Fields * = NULL);                //
      int addObj(SE_Object,                             //     (b) add an object
                 SE_Object,                             //
                 SE_Fields * = NULL,                    //
                 SE_Object = NULL,                      //
                 SE_Fields * = NULL);                   //
      int freeObj(SE_Object);                           //     (c) release an object
                                                        //
    protected:                                          // representation
      char           *_impl_id;                         //   implementation identifier
      SE_Transmittal _xmittal;                          //   transmittal ptr
      char           *_xmittal_name;                    //   transmittal name
      SE_Object      _root,                             //   Transmittal Root
                     _model_lib,                        //   Model Library
                     _union,                            //   union of Primitive Geometry
                     _image_lib,                        //   Image Library
                     _image;                            //   Image for texture
      int            _model_id;                         //   model identifier
      int            _image_id;                         //   image identifier
};
```

```
 1: /* ========================================================================================== */
 2: /*                                                                                            */
 3: /*   File name: STF_Fracture.C                                                                */
 4: /*                                                                                            */
 5: /*   Purpose:   implementation of member functions, ie methods, of STF_Fracture class         */
 6: /*                                                                                            */
 7: /*   Functions, parameters:                                                                   */
 8: /*               (1) createObj - create an object                                             */
 9: /*               (2) addObj - add an object                                                   */
10: /*               (3) freeObj - release an object                                              */
11: /*               (4) createDescription - create top level objects                             */
12: /*               (5) createImageLib - create Image Library                                    */
13: /*               (6) addImage - add an Image to Image Library                                 */
14: /*               (7) createModelLib - create Model Library and add a Model                    */
15: /*               (8) addPolygons - add polygons to Model's union of geometry                  */
16: /*               (9) create - open SEDRIS transmittal, then create description, Image         */
17: /*                            Library, and Model Library.                                     */
18: /*                                                                                            */
19: /*   By:        Andrew M. Neiderer 20 January '03                                            */
20: /* ========================================================================================== */
21:
22: #include <stdio.h>
23: #include <iostream.h>
24: #include <string.h>                        // for strlen()
25: #include <malloc.h>                        // for malloc()
26:
27: #include <Performer/pf.h>                  // for pfVec3
28: #include <Performer/pr/pfTexture.h>        // for pfTexture::getImage()
29:
30: #include "STF_Fracture.H"                  // for STF_Fracture class definition
31:
32: /* ************************************************************ */
33: /* createObj (type, obj, obj_f)                                 */
34: /*   SE_DRM_Class type      (in) -                              */
35: /*   SE_Object *obj          (in) - object                      */
36: /*   SE_Fields *obj_f        (in) - object fields               */
37: /*                                                              */
38: /* Create an object.                                            */
39: /*                                                              */
40: /* Return true (non-zero) or false (zero).                      */
41: /* ************************************************************ */
42:
43: int STF_Fracture::createObj(SE_DRM_Class type,
44:                             SE_Object *obj,
45:                             SE_Fields *obj_f)
46: {
47:
48: #ifdef Debug
49: fprintf(stdout, "    STF_Fracture::createObj()\n\n");
50: #endif
51:
```

```
52:   if ( SE_CreateObject(_xmittal,type,obj) != SE_STAT_CODE_SUCCESS ) {
53:     printf("Error: Unable to create object\n");
54:     return 0;
55:   }
56:   else if ( obj_f  &&  SE_SetFieldsToDefault(type,obj_f) != SE_DRM_STAT_CODE_SUCCESS ) {
57:     printf("Error: Unable to set default fields\n");
58:     return 0;
59:   }
60:
61:   return 1;
62: }
63:
64: /* ************************************************************** */
65: /* addObj (aggr, comp, comp_f, link, link_f)                      */
66: /*   SE_Object aggr          (in) - aggregate                     */
67: /*   SE_Object comp          (in) - component                     */
68: /*   SE_Fields *comp_f       (in) - component fields              */
69: /*   SE_Object link          (in) - link                          */
70: /*   SE_Fields *link_f       (in) - link fields                   */
71: /*                                                                */
72: /* Add an object.                                                 */
73: /*                                                                */
74: /* Return true (non-zero) or false (zero).                        */
75: /* ************************************************************** */
76:
77: int STF_Fracture::addObj(SE_Object aggr,
78:                          SE_Object comp,
79:                          SE_Fields *comp_f,
80:                          SE_Object link,
81:                          SE_Fields *link_f)
82: {
83:
84: #ifdef Debug
85: fprintf(stdout, "    STF_Fracture::addObj()\n\n");
86: #endif
87:
88:   if ( (comp_f  &&  SE_PutFields(comp,comp_f) != SE_STAT_CODE_SUCCESS)  ||
89:        (link  &&  link_f  &&  SE_PutFields(link,link_f) != SE_STAT_CODE_SUCCESS)  ||
90:        SE_AddComponentRelationship(aggr,comp,link) != SE_STAT_CODE_SUCCESS ) {
91:     printf("Error: couldn't complete addObj()\n");
92:     return 0;
93:   }
94:
95:   return 1;
96: }
97:
98: /* ************************************************************** */
99: /* freeObj (obj)                                                  */
100: /*   SE_Object obj           (in) - object                        */
101: /*                                                                */
102: /* Release an object.                                             */
```

```
103: /*                                                                  */
104: /* Return true (non-zero) or false (zero).                          */
105: /* ***************************************************************** */
106:
107: int STF_Fracture::freeObj(SE_Object obj)
108: {
109:
110: #ifdef Debug
111: fprintf(stdout, "    STF_Fracture::freeObj()\n\n");
112: #endif
113:
114:   if ( SE_FreeObject(obj) != SE_STAT_CODE_SUCCESS ) {
115:     printf("Error: free object failed\n");
116:     return 0;
117:   }
118:
119:   return 1;
120: }
121:
122: /* ***************************************************************** */
123: /* createDescription ()                                              */
124: /*                                                                   */
125: /* Create top level objects - Description, Point of Contact, Access, */
126: /*  Data Quality, Keywords, Absolute Time Point, Citation, Process,  */
127: /*  Transmittal Summary.                                             */
128: /*                                                                   */
129: /* Return true (non-zero) or false (zero).                           */
130: /* ***************************************************************** */
131:
132: int STF_Fracture::createDescription()
133: {
134:   SE_Object obj;
135:   SE_Fields f;
136:   char *s;
137:
138: #ifdef Debug
139: fprintf(stdout, "    STF_Fracture::createDescription()\n\n");
140: #endif
141:
142:   // Transmittal Root
143:
144:   if ( !createObj(SE_DRM_CLS_TRANSMITTAL_ROOT,&_root,&f) )
145:     return 0;
146:
147:   f.u.Transmittal_Root.name.length = strlen(_xmittal_name);
148:   f.u.Transmittal_Root.name.characters = _xmittal_name;
149:
150:   if ( SE_PutFields(_root,&f) != SE_STAT_CODE_SUCCESS  ||
151:        SE_SetRootObject(_xmittal,_root,&obj) != SE_STAT_CODE_SUCCESS ) {
152:     printf("Error: failed to add Transmittal Root\n");
153:     return 0;
```

```
154:   }
155: #ifdef Debug
156: fprintf(stdout, "      SE_DRM_CLS_TRANSMITTAL_ROOT\n");
157: #endif
158:
159:   // Description
160:
161:   createObj(SE_DRM_CLS_DESCRIPTION,&obj,&f);
162:   s = "Evaluation";
163:   f.u.Description.abstract.length = strlen(s);
164:   f.u.Description.abstract.characters = s;
165:   addObj(_root,obj,&f);
166:   freeObj(obj);
167: #ifdef Debug
168: fprintf(stdout, "      SE_DRM_CLS_DESCRIPTION\n\n");
169: #endif
170:
171:   // Point of Contact
172:
173:   createObj(SE_DRM_CLS_POINT_OF_CONTACT,&obj,&f);
174:   s = "Army Research Laboratory";
175:   f.u.Point_Of_Contact.organization.length = strlen(s);
176:   f.u.Point_Of_Contact.organization.characters = s;
177:   s = "SEDRIS customer";
178:   f.u.Point_Of_Contact.person_or_position.characters = s;
179:   f.u.Point_Of_Contact.person_or_position.length = strlen(s);
180:   s = "APG, MD";
181:   f.u.Point_Of_Contact.address.length = strlen(s);
182:   f.u.Point_Of_Contact.address.characters = s;
183:   s = "410-278-3203";
184:   f.u.Point_Of_Contact.voice_phone.length = strlen(s);
185:   f.u.Point_Of_Contact.voice_phone.characters = s;
186:   addObj(_root,obj,&f);
187:   freeObj(obj);
188: #ifdef Debug
189: fprintf(stdout, "      SE_DRM_CLS_POINT_OF_CONTACT\n\n");
190: #endif
191:
192:   // Access
193:
194:   createObj(SE_DRM_CLS_ACCESS,&obj,&f);
195:   addObj(_root,obj,&f);
196:   freeObj(obj);
197: #ifdef Debug
198: fprintf(stdout, "      SE_DRM_CLS_ACCESS\n\n");
199: #endif
200:
201:   // Data Quality
202:
203:   createObj(SE_DRM_CLS_DATA_QUALITY,&obj,&f);
204:   f.u.Data_Quality.fictional = SE_TRUE;
```

```
205:   addObj(_root,obj,&f);
206:   freeObj(obj);
207: #ifdef Debug
208: fprintf(stdout, "      SE_DRM_CLS_DATA_QUALITY\n\n");
209: #endif
210:
211:   // Keywords
212:
213:   createObj(SE_DRM_CLS_KEYWORDS,&obj,&f);
214:   addObj(_root,obj,&f);
215:   freeObj(obj);
216: #ifdef Debug
217: fprintf(stdout, "      SE_DRM_CLS_KEYWORDS\n\n");
218: #endif
219:
220:   // Absolute Time Point
221:
222:   createObj(SE_DRM_CLS_ABSOLUTE_TIME_POINT,&obj,&f);
223:   f.u.Absolute_Time_Point.time_significance = SE_TIME_SIGNIFICANCE_CREATION_DATE;
224:   f.u.Absolute_Time_Point.year = 2002;
225:   f.u.Absolute_Time_Point.month = SE_MONTH_FEBRUARY;
226:   f.u.Absolute_Time_Point.day = 20;
227:   f.u.Absolute_Time_Point.hour = 15;
228:   f.u.Absolute_Time_Point.minutes = 0;
229:   f.u.Absolute_Time_Point.seconds = 0;
230:   addObj(_root,obj,&f);
231:   freeObj(obj);
232: #ifdef Debug
233: fprintf(stdout, "      SE_DRM_ABSOLUTE_TIME_POINT\n\n");
234: #endif
235:
236:   // Citation
237:
238:   createObj(SE_DRM_CLS_CITATION,&obj,&f);
239:   s = "Fracture Database";
240:   f.u.Citation.title.length = strlen(s);
241:   f.u.Citation.title.characters = s;
242:   s = "ARL/CISD, APG";
243:   f.u.Citation.originators.characters = s;
244:   f.u.Citation.originators.length = strlen(s);
245:   addObj(_root,obj,&f);
246:   freeObj(obj);
247: #ifdef Debug
248: fprintf(stdout, "      SE_DRM_CLS_CITATION\n\n");
249: #endif
250:
251:   // Process
252:
253:   createObj(SE_DRM_CLS_PROCESS,&obj,&f);
254:   s = "Pre-process";
255:   f.u.Description.abstract.length = strlen(s);
```

```
256:   f.u.Description.abstract.characters = s;
257:   addObj(_root,obj,&f);
258:   freeObj(obj);
259: #ifdef Debug
260: fprintf(stdout, "      SE_DRM_CLS_PROCESS\n\n");
261: #endif
262:
263:   // Transmittal Summary
264:
265:   createObj(SE_DRM_CLS_TRANSMITTAL_SUMMARY,&obj,&f);
266:   f.u.Transmittal_Summary.geometry_present = SE_PRESENT_IN_MODELS;
267:   f.u.Transmittal_Summary.models_present = SE_TRUE;
268:   addObj(_root,obj,&f);
269:   freeObj(obj);
270: #ifdef Debug
271: fprintf(stdout, "      SE_DRM_CLS_TRANSMITTAL_SUMMARY\n\n");
272: #endif
273:
274:   return 1;
275: }
276:
277: /* ************************************************************** */
278: /* createImageLib ()                                              */
279: /*                                                                */
280: /* Create Image Library.                                          */
281: /*                                                                */
282: /* Return true (non-zero) or false (zero).                        */
283: /* ************************************************************** */
284:
285: int STF_Fracture::createImageLib()
286: {
287:   SE_Fields f;
288:
289: #ifdef Debug
290: fprintf(stdout, "    STF_Fracture::createImageLib()\n\n");
291: #endif
292:
293:   createObj(SE_DRM_CLS_IMAGE_LIBRARY,&_image_lib,&f);
294:   addObj(_root,_image_lib,&f);
295:
296:   return 1;
297: }
298:
299: /* ************************************************************** */
300: /* addImage (data, xSize, ySize, components)                      */
301: /*   unsigned int *data            (in) - image data              */
302: /*   int xSize                     (in) - horizontal size         */
303: /*   int ySize                     (in) - vertical size           */
304: /*   int components                (in) - no. of 8-bit components */
305: /*                                        per pixel.              */
306: /*                                                                */
```

```
307: /* Add Image to Image Library.                                        */
308: /*                                                                    */
309: /* Return true (non-zero) or false (zero).                            */
310: /* ****************************************************************** */
311:
312: int STF_Fracture::addImage(unsigned int *data,
313:                            int xSize, int ySize, int components)
314: {
315:   SE_Fields f;
316:   char *s;
317:
318: #ifdef Debug2
319: fprintf(stdout, "    STF_Fracture::addImage()\n\n");
320: #endif
321:
322:   if ( _image != (SE_Object) NULL )
323:     freeObj(_image);
324:   createObj(SE_DRM_CLS_IMAGE,&_image,&f);
325:
326:   s = "Texture";
327:   f.u.Image.name.characters = s;
328:   f.u.Image.name.length = strlen(s);
329:   f.u.Image.colour_model = SE_CLR_MDL_RGB;
330:   f.u.Image.level_count = 1;
331:   f.u.Image.mip_extents_array = (SE_Image_MIP_Extents *) calloc(1, sizeof(SE_Image_MIP_Extents));
332:   f.u.Image.mip_extents_array->size_horizontal = xSize;
333:   f.u.Image.mip_extents_array->size_vertical = ySize;
334:   f.u.Image.mip_extents_array->size_z = 1;
335:   f.u.Image.scan_direction = SE_IMG_SCAN_DIR_RIGHT_UP;
336:   f.u.Image.scan_direction_z = SE_IMG_SCAN_DIR_Z_NONE;
337:   f.u.Image.component_data_type = SE_IMG_COMP_TYP_UNSIGNED_INTEGER;
338: //  f.u.Image.data_is_little_endian = SE_TRUE;
339:
340:   switch (components) {
341:     case 1:
342:       f.u.Image.data_is_3D = SE_FALSE;
343:       f.u.Image.image_signature = SE_IMG_SIG_LUMINANCE;
344:       f.u.Image.bits_of_alpha = 0;
345:       f.u.Image.bits_of_luminance = 8;
346:       f.u.Image.bits_of_colour_coordinate_1 = 0;
347:       f.u.Image.bits_of_colour_coordinate_2 = 0;
348:       f.u.Image.bits_of_colour_coordinate_3 = 0;
349:       f.u.Image.max_value_of_alpha = 0;
350:       f.u.Image.max_value_of_luminance = 255;
351:       f.u.Image.max_value_of_colour_coordinate_1 = 0;
352:       f.u.Image.max_value_of_colour_coordinate_2 = 0;
353:       f.u.Image.max_value_of_colour_coordinate_3 = 0;
354:
355:       break;
356:     case 2:
357:       f.u.Image.data_is_3D = SE_FALSE;
```

```
358:       f.u.Image.image_signature = SE_IMG_SIG_123COLOUR;
359:       f.u.Image.bits_of_alpha = 0;
360:       f.u.Image.bits_of_luminance = 0;
361:       f.u.Image.bits_of_colour_coordinate_1 = 8;
362:       f.u.Image.bits_of_colour_coordinate_2 = 8;
363:       f.u.Image.bits_of_colour_coordinate_3 = 0;
364:       f.u.Image.max_value_of_alpha = 0;
365:       f.u.Image.max_value_of_luminance = 0;
366:       f.u.Image.max_value_of_colour_coordinate_1 = 255;
367:       f.u.Image.max_value_of_colour_coordinate_2 = 255;
368:       f.u.Image.max_value_of_colour_coordinate_3 = 0;
369:
370:       break;
371:     case 3:
372:       f.u.Image.data_is_3D = SE_TRUE;
373:       f.u.Image.image_signature = SE_IMG_SIG_123COLOUR;
374:       f.u.Image.bits_of_alpha = 0;
375:       f.u.Image.bits_of_luminance = 0;
376:       f.u.Image.bits_of_colour_coordinate_1 = 8;
377:       f.u.Image.bits_of_colour_coordinate_2 = 8;
378:       f.u.Image.bits_of_colour_coordinate_3 = 8;
379:       f.u.Image.max_value_of_alpha = 0;
380:       f.u.Image.max_value_of_luminance = 0;
381:       f.u.Image.max_value_of_colour_coordinate_1 = 255;
382:       f.u.Image.max_value_of_colour_coordinate_2 = 255;
383:       f.u.Image.max_value_of_colour_coordinate_3 = 255;
384:
385:       break;
386:     case 4:
387:       f.u.Image.data_is_3D = SE_FALSE;
388:       f.u.Image.image_signature = SE_IMG_SIG_123COLOUR;
389:       f.u.Image.bits_of_alpha = 8;
390:       f.u.Image.bits_of_luminance = 0;
391:       f.u.Image.bits_of_colour_coordinate_1 = 8;
392:       f.u.Image.bits_of_colour_coordinate_2 = 8;
393:       f.u.Image.bits_of_colour_coordinate_3 = 8;
394:       f.u.Image.max_value_of_alpha = 8;
395:       f.u.Image.max_value_of_luminance = 0;
396:       f.u.Image.max_value_of_colour_coordinate_1 = 255;
397:       f.u.Image.max_value_of_colour_coordinate_2 = 255;
398:       f.u.Image.max_value_of_colour_coordinate_3 = 255;
399:
400:       break;
401:     default:
402:       f.u.Image.data_is_3D = SE_TRUE;
403:       f.u.Image.image_signature = SE_IMG_SIG_123COLOUR;
404:       f.u.Image.bits_of_alpha = 0;
405:       f.u.Image.bits_of_luminance = 0;
406:       f.u.Image.bits_of_colour_coordinate_1 = 8;
407:       f.u.Image.bits_of_colour_coordinate_2 = 8;
408:       f.u.Image.bits_of_colour_coordinate_3 = 8;
```

```
409:       f.u.Image.max_value_of_alpha = 0;
410:       f.u.Image.max_value_of_luminance = 0;
411:       f.u.Image.max_value_of_colour_coordinate_1 = 255;
412:       f.u.Image.max_value_of_colour_coordinate_2 = 255;
413:       f.u.Image.max_value_of_colour_coordinate_3 = 255;
414:
415:       break;
416:   }
417:   addObj(_image_lib,_image,&f);
418:
419:   SE_PutImageData(_image,0,0,0,xSize - 1,ySize - 1,0,0,xSize * ySize * components,(unsigned char *) data);
420:
421:   return 1;
422: }
423:
424: /* ************************************************************** */
425: /* createModelLib ()                                              */
426: /*                                                                */
427: /* Create Model Library and add a Model.                          */
428: /*                                                                */
429: /* Return true (non-zero) or false (zero).                        */
430: /* ************************************************************** */
431:
432: int STF_Fracture::createModelLib()
433: {
434:   SE_Object obj, obj2;
435:   SE_Fields f;
436:   char *s;
437:
438: #ifdef Debug
439: fprintf(stdout, "    STF_Fracture::createModelLib()\n\n");
440: #endif
441:
442:   createObj(SE_DRM_CLS_MODEL_LIBRARY,&_model_lib,&f);
443:   addObj(_root,_model_lib,&f);
444:
445:   createObj(SE_DRM_CLS_MODEL,&obj,&f);
446:   s = "Fracture";
447:   f.u.Model.name.characters = s;
448:   f.u.Model.name.length = strlen(s);
449:   f.u.Model.model_reference_type = SE_MDL_REF_TYP_ROOT;
450:   f.u.Model.srf_parameters.parameters.three_d.spatial_reference_frame = SRM_SRF_3D_LSR;
451:   f.u.Model.srf_parameters.parameters.three_d.parameters.lsr.up_direction =
452:     SRM_UP_DIR_POSITIVE_TERTIARY_AXIS;
453:   f.u.Model.srf_parameters.parameters.three_d.parameters.lsr.forward_direction =
454:     SRM_FRWD_DIR_POSITIVE_SECONDARY_AXIS;
455:   addObj(_model_lib,obj,&f);
456:
457:   createObj(SE_DRM_CLS_GEOMETRY_MODEL,&obj2,&f);
458:   addObj(obj,obj2,&f);
459:
```

```
460:   freeObj(obj);
461:
462:   createObj(SE_DRM_CLS_UNION_OF_PRIMITIVE_GEOMETRY,&_union,&f);
463:   addObj(obj2,_union,&f);
464:
465:   freeObj(obj2);
466:
467:   return 1;
468: }
469:
470: /* ******************************************************************** */
471: /* addPolygons (verts, tcoords, texture, nverts)                        */
472: /*   pfVec3 *verts                  (in) - vertices                     */
473: /*   pfVec2 *tcoords                (in) - texture coords at vertices   */
474: /*   pfTexture *texture             (in) - Performer texture            */
475: /*   int nverts                     (in) - no. of vertices              */
476: /*                                                                      */
477: /* Add polygons to Model's union of geometry.                           */
478: /*                                                                      */
479: /* Return true (non-zero) or false (zero).                              */
480: /* ******************************************************************** */
481:
482: int STF_Fracture::addPolygons(pfVec3 *verts,
483:                               pfVec2 *tcoords, pfTexture *texture,
484:                               int nverts)
485: {
486:   // SEDRIS API
487:
488:   unsigned char *data;                    // image
489:   SE_Object poly,
490:             vert,
491:             loc, tex, clr,
492:             IMF;
493:   SE_Fields f;
494:
495:   // Performer API
496:
497:   unsigned int *image;                    // array of texels
498:   int components;                         // no. of components per pixel
499:   int ns,                                 // no. of texels in s dimension of image
500:       nt,                                 // no. of texels in t dimension of image
501:       nr;                                 // no. of texels in r dimension of image
502:
503:   int i, j;                               // counters
504:
505: #ifdef Debug
506: fprintf(stdout, "    STF_Fracture::addPolygons()\n\n");
507: #endif
508:
509:   // Performer texture
510:
```

```
511:   if ( texture != NULL ) {
512:     texture->getImage(&image,&components,&ns,&nt,&nr);
513:
514:     // add to Image Library
515:     addImage(image,ns,nt,components);
516:
517: #ifdef Debug
518:     cerr << "      addPolygons : addImage complete " << texture->getName() << endl;
519: #endif
520:   }
521:   else {
522: #ifdef Debug
523:     cerr << "      addPolygons : no texture for this polygon" << endl;
524: #endif
525:   }
526:
527:   // each polygon has 3 vertices, ie triangle
528:
529:   for ( i = 0; i < nverts; i += 3 ) {
530: #ifdef Debug
531: fprintf(stdout, "      polygon %d\n\n",i / 3);
532: #endif
533:
534:     createObj(SE_DRM_CLS_POLYGON,&poly,&f);
535:     addObj(_union,poly,&f);
536:
537:     // Image Mapping Function
538:
539:     if ( texture != NULL ) {
540:       createObj(SE_DRM_CLS_IMAGE_MAPPING_FUNCTION,&IMF,&f);
541:
542:       f.u.Image_Mapping_Function.image_mapping_method = SE_IMG_MAPNG_METH_REPLACE;
543:       f.u.Image_Mapping_Function.image_wrap_s = SE_IMG_WRAP_REPEAT;
544:       f.u.Image_Mapping_Function.image_wrap_t = SE_IMG_WRAP_REPEAT;
545:       f.u.Image_Mapping_Function.image_projection_type = SE_IMG_PROJ_TYP_PLANAR;
546:       f.u.Image_Mapping_Function.intensity_level = 1;
547:       f.u.Image_Mapping_Function.gain = 0;
548:       f.u.Image_Mapping_Function.image_detail_mapping = SE_FALSE;
549: #ifdef Debug
550:       fprintf(stdout,"      Image_Mapping_Function.image_id = %d\n\n",_image_id);
551: #endif
552:       addObj(poly,IMF,&f);
553:       SE_AddAssociateRelationship(IMF,_image,NULL,SE_FALSE);
554:       freeObj(IMF);
555:     }
556:
557:     // at each vertex, Cartesian coords and texture coords
558:
559:     for ( int k = 0; k < 3; k++ ) {
560:       createObj(SE_DRM_CLS_VERTEX,&vert,&f);
561:       addObj(poly,vert,&f);
```

```
562:       createObj(SE_DRM_CLS_LSR_LOCATION_3D,&loc,&f);
563:       f.u.LSR_Location_3D.coordinate.x = verts[i + k].vec[0];
564:       f.u.LSR_Location_3D.coordinate.y = verts[i + k].vec[1];
565:       f.u.LSR_Location_3D.coordinate.z = verts[i + k].vec[2];
566:
567:       addObj(vert,loc,&f);
568:
569:       createObj(SE_DRM_CLS_RGB_COLOUR,&clr,&f);
570: //      f.u.RGB_Color.rgb_data.red = 1.0;
571: //      f.u.RGB_Color.rgb_data.green = 1.0;
572: //      f.u.RGB_Color.rgb_data.blue = 1.0;
573:       addObj(vert,clr,&f);
574:       freeObj(clr);
575:
576:       if ( texture != NULL ){
577:         createObj(SE_DRM_CLS_TEXTURE_COORDINATE,&tex,&f);
578:         f.u.Texture_Coordinate.s = tcoords[i + k].vec[0];
579:         f.u.Texture_Coordinate.t = tcoords[i + k].vec[1];
580:         addObj(vert,tex,&f);
581:         freeObj(tex);
582:       }
583:
584:       freeObj(loc);
585:       freeObj(vert);
586:     }
587:
588:     freeObj(poly);
589:   }
590:
591:   if ( texture != NULL )
592:     _image_id++;
593:
594:   return 1;
595: }
596:
597: /* ****************************************************************** */
598: /* create (fileName, verts, tcoords, texture, nverts)                 */
599: /*   char *fileName                (in) - SEDRIS transmittal          */
600: /*   pfVec3 *verts                 (in) - vertices                    */
601: /*   pfVec2 *tcoords               (in) - texture coords at vertices  */
602: /*   pfTexture *texture            (in) - Performer texture           */
603: /*   int nverts                    (in) - no. of vertices             */
604: /*                                                                    */
605: /* Open SEDRIS transmittal.  Add description, Image Library, and      */
606: /*   Model Library.                                                   */
607: /*                                                                    */
608: /* Return true (non-zero) or false (zero).                            */
609: /* ****************************************************************** */
610:
611: int STF_Fracture::create(char *fileName,
612:                          pfVec3 *verts,
```

```
613:                            pfVec2 *tcoords, pfTexture *texture,
614:                            int nverts)
615: {
616:   char buff[1000];
617:
618: #ifdef Debug
619: fprintf(stdout, "    STF_Fracture::create()\n\n");
620: #endif
621:
622:   if ( SE_OpenTransmittalByFile(fileName,_impl_id,SE_AC_MODE_CREATE,&_xmittal) != SE_STAT_CODE_SUCCESS ) {
623:     printf("Error: can't create \"%s\"\n",fileName);
624:     return 0;
625:   }
626:
627:   sprintf(buff,"urn:x-sedris:%s:1",fileName);
628:
629:   if ( SE_SetTransmittalName(_xmittal,buff) != SE_STAT_CODE_SUCCESS ) {
630:     printf("Error: failed to set transmittal name\n");
631:     return 0;
632:   }
633:
634:   _xmittal_name = fileName;
635:
636:   if ( !createDescription()  ||  !createImageLib()  ||
637:        !createModelLib()  ||  !addPolygons(verts,tcoords,texture,nverts) )
638:     return 0;
639:
640:   return 1;
641: }
642:
```

NO. OF COPIES	ORGANIZATION
2	DEFENSE TECHNICAL INFORMATION CENTER DTIC OCA 8725 JOHN J KINGMAN RD STE 0944 FT BELVOIR VA 22060-6218
1	COMMANDING GENERAL US ARMY MATERIEL CMD AMCRDA TF 5001 EISENHOWER AVE ALEXANDRIA VA 22333-0001
1	INST FOR ADVNCD TCHNLGY THE UNIV OF TEXAS AT AUSTIN 3925 W BRAKER LN STE 400 AUSTIN TX 78759-5316
1	US MILITARY ACADEMY MATH SCI CTR EXCELLENCE MADN MATH THAYER HALL WEST POINT NY 10996-1786
1	DIRECTOR US ARMY RESEARCH LAB AMSRL D DR D SMITH 2800 POWDER MILL RD ADELPHI MD 20783-1197
1	DIRECTOR US ARMY RESEARCH LAB AMSRL CS IS R 2800 POWDER MILL RD ADELPHI MD 20783-1197
3	DIRECTOR US ARMY RESEARCH LAB AMSRL CI OK TL 2800 POWDER MILL RD ADELPHI MD 20783-1197
3	DIRECTOR US ARMY RESEARCH LAB AMSRL CS IS T 2800 POWDER MILL RD ADELPHI MD 20783-1197

NO. OF COPIES	ORGANIZATION
	ABERDEEN PROVING GROUND
2	DIR USARL AMSRL CI LP (BLDG 305) AMSRL CI OK TP (BLDG 4600)

NO. OF
COPIES ORGANIZATION

ABERDEEN PROVING GROUND

6 DIR USARL
AMSRL CI CT
P JONES
M THOMAS
A NEIDERER (4 CPS)