EDCS Changes

- Accessing Attribute Data
  - New EDCS structures provide stronger consistency for EDCS Attributes
    - Example: population is a count
    - Example: measurements have units and scale
  - EDCS attributes can have either single values or intervals

<Property Value> in 3.1

- Field elements:

```
SE_Element_Type             meaning;
EDCS_Unit_Code              value_unit;
EDCS_Scale_Code             value_scale;
SE_Property_Data_Value      value;

typedef struct
{
   SE_Element_Code_Type     code_type;
   union
   {
     EDCS_Attribute_Code    attribute;
     SE_Index_Code          index;
     SE_Variable_Code       variable;
   } code;
} SE_Element_Type;
```

<Property Value> in 3.1

```
typedef struct
{
    SE_Property_Data_Value_Type         value_type;
    union
    {
        EDCS_Boolean                    boolean_value;
        SRM_Byte                        byte_value;
        SRM_Byte_Positive               byte_positive_value;
        EDCS_Byte_Unsigned              byte_unsigned_value;
        EDCS_Short_Integer              short_integer_value;
        SRM_Short_Integer_Positive      short_integer_positive_value;
        EDCS_Short_Integer_Unsigned     short_integer_unsigned_value;
        EDCS_Integer                    integer_value;
        EDCS_Integer_Interval           integer_interval_value;
        SRM_Integer_Positive            integer_positive_value;
        EDCS_Integer_Unsigned           integer_unsigned_value;
        EDCS_Integer_Unsigned_Interval  integer_unsigned_interval_value;
        EDCS_Float                      float_value;
        EDCS_Float_Interval             float_interval_value;
        EDCS_Long_Float                 long_float_value;
        EDCS_String                     string_value;
        SRM_Integer_Positive            DT_component_index;
        SRM_Integer_Positive            DT_library_index;
        EDCS_Enumerant_Code             ee_code;
        EDCS_Metadata_Code              em_code;
    } u;
} SE_Property_Data_Value;
```

Intervals in 3.1

```
typedef struct
{
        EDCS_Interval_Type              type;
        EDCS_Float                      upper_bound;
        EDCS_Float                      lower_bound;
} EDCS_Float_Interval;

typedef struct
{
        EDCS_Interval_Type              type;
        EDCS_Integer                    upper_bound;
        EDCS_Integer                    lower_bound;
} EDCS_Integer_Interval;

typedef struct
{
        EDCS_Interval_Type              type;
        EDCS_Integer_Unsigned           upper_bound;
        EDCS_Integer_Unsigned           lower_bound;
} EDCS_Integer_Unsigned_Interval;
```

Intervals in 3.1

Enumerant	Definition
EDCS_INTRVL_TYP_OPEN	The interval is an open interval of the form (lower_bound, upper_bound) such that neither endpoint is in the interval. That is, for each x in the interval, lower_bound < x < upper_bound.
EDCS_INTRVL_TYP_CLOSED	The interval is a closed interval of the form [lower_bound, upper_bound] such that both endpoints are in the interval. That is, for each x in the interval, lower_bound <= x <= upper_bound.
EDCS_INTRVL_TYP_LOWER_CLOSED_UPPER_OPEN	The interval is a semi-open interval of the form [lower_bound, upper_bound) such that of the two endpoints, only the lower endpoint is in the interval. That is, for each x in the interval, lower_bound <= x < upper_bound.
EDCS_INTRVL_TYP_UPPER_CLOSED_LOWER_OPEN	The interval is a semi-open interval of the form (lower_bound, upper_bound] such that of the two endpoints, only the upper endpoint is in the interval. That is, for each x in the interval, lower_bound < x <= upper_bound.

<Property Value> in 4.0

- Field elements:

```
SE_Property_Type        meaning;
EDCS_Attribute_Value    value;


typedef struct
{
  SE_Property_Code      code_type;
  union
  {
   EDCS_Attribute_Code attribute;
   SE_Variable_Code    variable;
  } code;
} SE_Property_Type;
```

<Property Value> in 4.0

```
typedef struct
{
    EDCS_Attribute_Value_Type           attribute_value_type;
    union
    {
        EDCS_Value_Characteristic_Code  characteristic_value;
        EDCS_Real_Value                 real_value;
        EDCS_Integer_Value              integer_value;
        EDCS_Count_Value                count_value;
        EDCS_Count                      index_value;
        EDCS_String                     string_value;
        EDCS_String                     constrained_string_value;
        EDCS_String                     key_value;
        EDCS_Enumerant_Code             enumerant_value;
        EDCS_Boolean                    boolean_value;
        EDCS_Null                       null_value;
    } u;
} EDCS_Attribute_Value;
```

Real Values in 4.0

```
typedef struct
{
    EDCS_Unit_Code              unit;
    EDCS_Scale_Code             scale;
    EDCS_Long_Float_Value       float_value;
} EDCS_Real_Value;

typedef struct
{
    EDCS_Numeric_Value_Type     numeric_value_type;
    union
    {
        EDCS_Long_Float             single_value;
        EDCS_Long_Float_Interval    open_interval;
        EDCS_Long_Float_Interval    ge_lt_interval;
        EDCS_Long_Float_Interval    gt_le_interval;
        EDCS_Long_Float_Interval    closed_interval;
        EDCS_Long_Float             gt_semi_interval;
        EDCS_Long_Float             ge_semi_interval;
        EDCS_Long_Float             lt_semi_interval;
        EDCS_Long_Float             le_semi_interval;
    } u;
} EDCS_Long_Float_Value;
```

Integer Values in 4.0

```
typedef struct
{
    EDCS_Numeric_Value_Type  numeric_value_type;
    union
    {
        EDCS_Integer             single_value;
        EDCS_Integer_Interval    open_interval;
        EDCS_Integer_Interval    ge_lt_interval;
        EDCS_Integer_Interval    gt_le_interval;
        EDCS_Integer_Interval    closed_interval;
        EDCS_Integer             gt_semi_interval;
        EDCS_Integer             ge_semi_interval;
        EDCS_Integer             lt_semi_interval;
        EDCS_Integer             le_semi_interval;
    } u;
} EDCS_Integer_Value;
```

Count Values in 4.0

```
typedef struct
{
    EDCS_Numeric_Value_Type  numeric_value_type;
    union
    {
        EDCS_Count           single_value;
        EDCS_Count_Interval  open_interval;
        EDCS_Count_Interval  ge_lt_interval;
        EDCS_Count_Interval  gt_le_interval;
        EDCS_Count_Interval  closed_interval;
        EDCS_Count           gt_semi_interval;
        EDCS_Count           ge_semi_interval;
        EDCS_Count           lt_semi_interval;
        EDCS_Count           le_semi_interval;
    } u;
} EDCS_Count_Value;
```

Sample Code

```
add_attribution(    SE_Transmittal transmittal,   SE_Object      object)
{
      SE_Object  prop_value;
      SE_Fields  prop_value_fields;

      if (SE_CreateObject( transmittal, SE_DRM_CLS_PROPERTY_VALUE, &prop_value)
         != SE_STAT_CODE_SUCCESS)
      {
          fprintf(stderr, "Failed to create a <Property Value>");
          return;
      }

      if (SE_SetFieldsToDefault(SE_DRM_CLS_PROPERTY_VALUE, &prop_value_fields)
         != SE_DRM_STAT_CODE_SUCCESS)
      {
                fprintf(stderr, "ERROR: Can't set fields to default");
                return;
      }
```

Sample Code

```
    prop_value_fields.u.Property_Value.meaning.code_type = SE_PROP_CODE_TYP_ATTRIBUTE;
    prop_value_fields.u.Property_Value.meaning.code.attribute = EAC_ABSOLUTE_ELEVATION_ACCURACY;
    prop_value_fields.u.Property_Value.value.attribute_value_type = EDCS_AVT_INTEGER;
    prop_value_fields.u.Property_Value.value.u.integer_value.numeric_value_type =
    EDCS_NVT_SINGLE_VALUE;
    prop_value_fields.u.Property_Value.value.u.integer_value.u.single_value = 1;

    if (SE_AddComponentRelationship(object, prop_value, NULL) != SE_STAT_CODE_SUCCESS)
    {
        fprintf(stderr, "Failed to connect <Property Value> component");
        SE_RemoveFromTransmittal(prop_value, transmittal);
        SE_FreeObject(prop_value);
        return;
    }

    if (SE_FreeObject(prop_value) != SE_STAT_CODE_SUCCESS)
    {
            fprintf(stderr, "Failed to free <Property Value> component");
            return;
    }
}
```