The EXPRESS Definition Language for IFC Development

EXPRESS is a conceptual schema language which provides for the specification of classes belonging to a defined domain, the information or attributes pertaining to those classes (colour, size, shape etc.) and the constraints on those classes (unique, exclusive etc.). It is also used to define the relations which exist between classes and the numerical constraints applying to such relations.

EXPRESS Layout

EXPRESS uses the semi-colon character ; to terminate an expression. Whitespace and carriage returns are ignored in interpreting or checking the validity of an EXPRESS schema but they can be used to improve the layout of the schema so that it is human readable.

The following examples are exactly equivalent. However, the use of whitespace and carriage returns in the first example makes it much easier to read than the second example.

```
ENTITY IfcClassification;
       ClassificationPublisher   : IfcString;
       ClassificationTable       : IfcString;
       ClassificationNotation    : IfcString;
       ClassificationDescription : IfcString;
END_ENTITY;
```

```
ENTITY IfcClassification; ClassificationPublisher : IfcString; ClassificationTable :
IfcString; ClassificationNotation : IfcString; ClassificationDescription : IfcString;
END_ENTITY;
```

Classes

EXPRESS is used to define classes. Within the class definition, all the attributes and behaviours which characterize it are declared. A class is declared by the keyword ENTITY and terminated by the keyword END_ENTITY.

An entity declaration creates a class and gives it a name. The declaration is terminated by END_ENTITY;

```
ENTITY IfcOwnerID;
       ……
END_ENTITY;
```

Attributes are the characteristics (data or behaviour) which are required to support use and understanding of the class. Attributes may be represented by simple data types (such as real, string, integer), or by other classes. Each attribute has a relationship with the class.

An attribute represented by a simple data type can be shown in EXPRESS by its data type.

```
ENTITY IfcOwnerID;
       Identifier : IfcString;
       OwningApp  : IfcString;
       OwningUser : IfcActor;
END_ENTITY;
```

Simple Data Types

A simple data type represents the atomic unit of data. Allowed simple data types are:-

REAL	Decimal numbers (e.g. 2.56). The decimal point must be present even if a number declared as REAL evaluates to a whole number equivalent to an integer (e.g. 2.0).
INTEGER	A whole number not containing a fraction or decimal element (e.g. 2)
NUMBER	A number that may arbitrarily evaluate to either an integer or a real.
LOGICAL	A value which evaluates to TRUE, FALSE or UNKNOWN.
BOOLEAN	A value which evaluates to TRUE or FALSE only.
BINARY	A sequence of bits, each of which may have the value 0 or 1.
STRING	A sequence of characters. Case of the character is significant.

Attributes

An attribute represented by a relationship to another class is shown by the name of the relationship and the name of the class with which the relationship exists (in the direction of the relationship). Thus for an IfcLayeredElement with a MaterialLayerSet which is declared as a class in its own right, EXPRESS takes the form:-

```
ENTITY IfcLayeredElement
       …
       MaterialLayerSet : IfcMaterialLayerSet;
       …
END_ENTITY;
```

Cardinality Constraints

EXPRESS allows numerical relations to be mandatory or optional. A mandatory attribute which must be asserted is expressed by there being no prefix term before the attribute name as in the example above.

An optional attribute that may be asserted is expressed by the word OPTIONAL appearing as a prefix term before the attribute name

```
ENTITY IfcLayeredElement
       …
       TotalAreaPerSide : OPTIONAL IfcAreaMeasure;
       TotalVolume      : OPTIONAL IfcVolumeMeasure;
       TotalLength      : OPTIONAL IfcLengthMeasure;
       …
END_ENTITY;
```

Aggregation

A number of aggregation relations are supported. These include:

- ARRAY a fixed size collection of things with order represented as A[1:?].
- BAG a collection of things with no order and allowed duplication represented as B[1:?].
- LIST a collection of things with order (sequence) represented as L[1:?].
- SET a collection of things with no order and no duplication represented as S[1:?].

```
ENTITY IfcLayeredElement
	…
	ImplGeoTopStartHeights : LIST [1:?] OF IfcLengthMeasure;
	ImplGeoTopEndHeights   : LIST [1:?] OF IfcLengthMeasure;
	…
END_ENTITY;
```

Within the IFC specification, LIST and SET are the most widely used aggregations.

The above example indicates that there must be at least one start height and one end height for a layered element but that there may be any number (the maximum limit is unspecified as identified by the ? character). If the maximum number of list elements was limited, this would be identified by including the upper bound in place of the unspecified character:-

```
ImplGeoTopStartHeights : LIST [1:7] OF IfcAttLength;
```

Inverse Rule

Attributes explicitly capture a relation between classes and attributes Inverse relations can also be captured between a class and a named attribute of another class or between two classes.

```
ENTITY IfcProjectObject
	…
	ResultOf : SET [0:?] OF IfcProcessObject;
	…
UNIQUE
	UR1: OwnerID;
END_ENTITY;

ENTITY IfcProcessObject;
…
INVERSE
	ResultsIn : SET[0:?] OF IfcProjectObject FOR ResultOf;
END_ENTITY;
```

Note the use of the INVERSE keyword to define the inverse relation

Unique Rule

EXPRESS allows for the uniqueness of attributes to be defined by a 'unique rule'. This specifies that the value of an attribute which is declared to be UNIQUE is associated with only one instance of that class (object). Where more than one attribute is described as unique, each must be included in the UNIQUE declaration.

```
ENTITY IfcProjectObject
…
UNIQUE
	UR1: OwnerID;
END_ENTITY;
```

Derive Rule

In some cases, it is appropriate to include an attribute which can be computed directly from other attributes. This can be achieved through use of derived attributes which are declared following the DERIVE keyword. The following example also introduces the use of function IfcTotalWidth to calculate the derived value:-

```
        ENTITY IfcLayeredElement
        …
           DERIVE
               ImplGeoTotalWidth : IfcLengthMeasure := IfcTotalWidth
(SELF.MaterialLayerSet);
        END_ENTITY;
```

Domain Rule

This is used to provide constraints on the values which attributes may have and is defined following the WHERE keyword. In the example below, the class can only exist if all three lists have the same number of members. This is used to ensure that those lists actually correspond, i.e. that for each thickness also an offset and a material is given.

```
        ENTITY IfcMaterialLayerSet;
               SetName         : OPTIONAL IfcString;
               Offsets         : LIST [1:?] OF IfcLengthMeasure;
               Thicknesses     : LIST [1:?] OF IfcLengthMeasure;
               Materials       : LIST [1:?] OF IfcMaterial;
           WHERE
               WR1 : (HIINDEX(SELF.Offsets) = HIINDEX(SELF.Thicknesses)) AND
                     (HIINDEX(SELF.Thicknesses) = HIINDEX(SELF.Materials));
END_ENTITY;
```

Arithmetical statements that are available within EXPRESS may be used in conjunction with the domain rule to provide constraints on attribute values. For instance, if the perimeter length of a window was constrained to be less than or equal to 4 metres, the rule would take the form:-

```
        ENTITY Window
               WINDOW_LENGTH  : REAL;
               WINDOW_HEIGHT  : REAL;
        WHERE
               perimeter : (WINDOW_LENGTH * 2 + WINDOW_HEIGHT * 2) <= 4.0;
        END_ENTITY;
```

Subtypes

EXPRESS allows for the classification of a class into subtypes. This defines a parent - child relation in which each subclass (referred to as subtype) contains more specific detail than its parent superclass (referred to as supertype).

```
        ENTITY IfcLayeredElement
               ABSTRACT SUPERTYPE OF (ONEOF
                      (IfcFloor,
                       IfcRoofSlab,
                       IfcWall));
               …
        END_ENTITY;

        ENTITY IfcWall;
               SUBTYPE OF (IfcLayeredElement);
               …
        END_ENTITY;
```

Note that the supertype declares the Wall as being 'ONEOF'. This indicates that the layered element is exclusively either a Wall or a Floor or a RoofSlab; it cannot be two or more at the same time. Alternative constraints on subtypes exist, most notably the ANDOR constraint that would allow the layered element to be either a Wall or a Floor or a RoofSlab or any combination of the three subtypes at the same time.

The current use of EXPRESS within IFC excludes all non ONEOF supertype constraints. Therefore no ANDOR or AND clauses will be found within IFC. EXPRESS supports both, single inheritance and multiple inheritance. Having more than one class within the SUBTYPE clause specifies multiple inheritance. IFC development does not use multiple inheritance.

A supertype may be included for the purposes of classification only, such situations occurring where it may be appropriate to include a supertype within the EXPRESS model for clarity. A supertype included for this purpose is never instanced; only its subtypes are used. In this case, it is known as an abstract supertype. The IfcLayeredElement is an example of an abstract supertype.

Declared Data Types

The type of a data item being used may be declared using a TYPE clause. For certain types of data item, this is necessary whilst for simple data types, it is optional. For instance, consider a space type that may be selected from an enumerated list of occupied, technical or circulation. The enumeration is declared as a data type as below:-

```
TYPE IfcSpaceTypeEnum = ENUMERATION OF
       (Occupied, Technical, Circulation);
END_TYPE;
```

Note that the use of the TYPE clause causes the declaration of the attribute within the class to be written in the same manner as if the relationship was with another class.

A SELECT data type defines a named collection of other data types. These may be other entities, a list of string values, a list of real values etc. As with enumeration's, only one item from a SELECT list is used by an instance of the class which uses the TYPE.

```
TYPE IfcBuildingSelect = SELECT
       (IfcBuilding, IfcBuildingStorey);
END_TYPE;
```

Access to Other Schema

Classes that are defined in schema other than that which is current may be accessed using an EXPRESS interface. This allows schema to be partitioned into manageable parts and enables reuse of schema which may have been previously defined or defined elsewhere. There are two interface possibilities.

The REFERENCE specification allows declarations made in other schema (usually classes) to be referenced but does not make them part of the current schema i.e. the declarations remain remote.

The following indicates the referencing of entities defined in the IFC geometry schema.

```
REFERENCE FROM IfcGeometryResource
       (IfcCartesianPoint,   IfcBounded_curve,
        IfcPolyline,         IfcTrimmed_curve,
        IfcCompositeCurve,   IfcPlacement,
        IfcLine,             IfcConic,
        IfcCircle,           IfcEllipse));
```