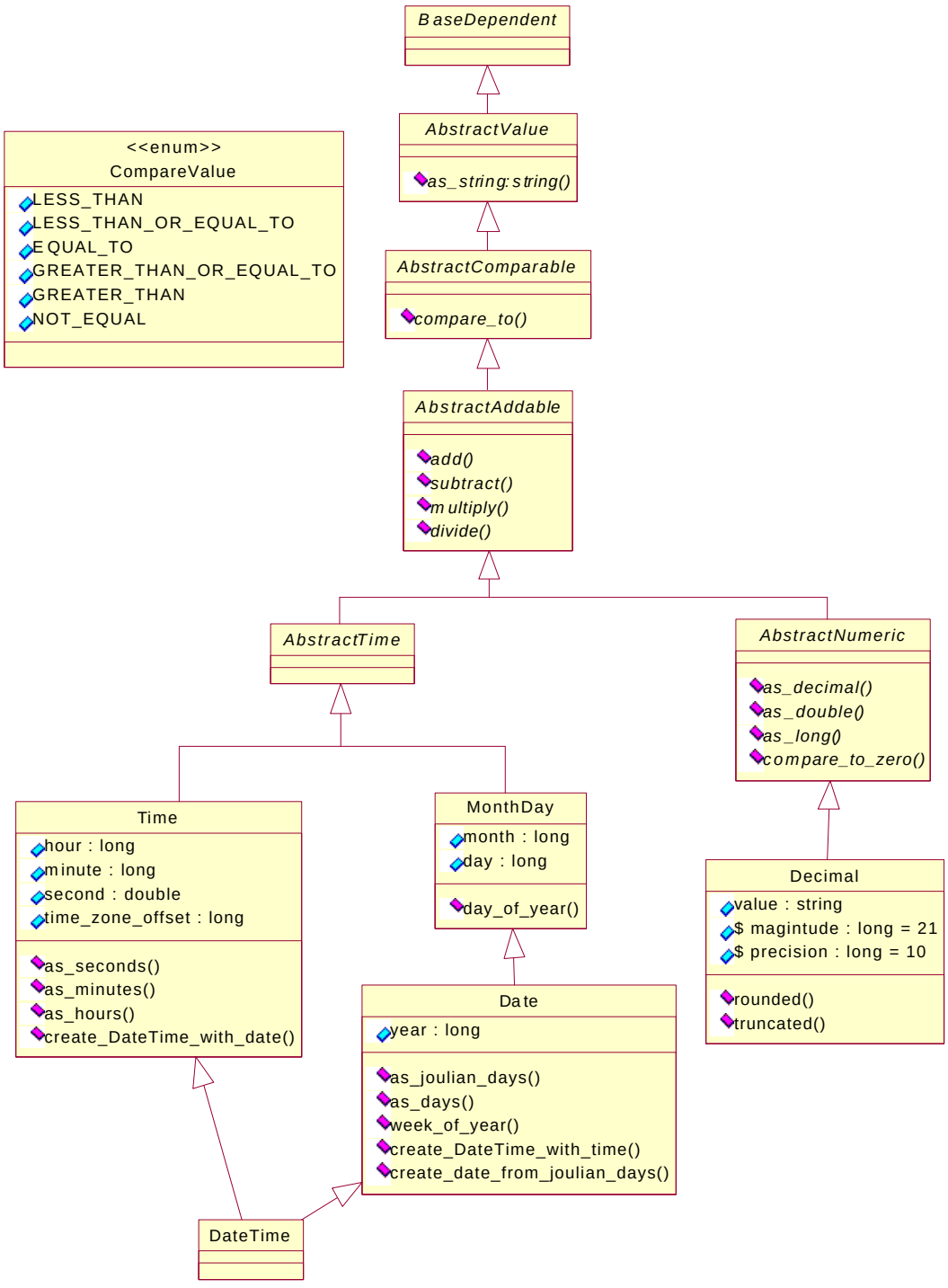
OMG Business Object Domain Task Force

Value Framework
Revision 1.0

Table Of Contents

Value Framework Specification

Value Types



Value types are used in domain specifications to represent common structured information such as numbers and dates.

CDL Specification

```
//-------Abstract value types---------
    dependent AbstractValue;

    exception_kind TypesNotCompatibleException {
        // Can be raised by any value operation taking a abstract types.
        readonly attribute AbstractValue   operation_on;
        readonly attribute AbstractValue   operand;
        readonly attribute string          operation_name;
    };
    dependent AbstractNumeric;
    dependent decimal;

    [FROZEN, is_abstract]
    dependent AbstractValue : BaseDependent {
        [is_query]
        string as_string(); // Conversion locale supplied by local infrastructre.
    };
    enum CompareValue { LESS_THAN, LESS_THAN_OR_EQUAL_TO, EQUAL_TO, GREATER_THAN_OR_EQUAL_TO,
GREATER_THAN, NOT_EQUAL };
    [FROZEN, is_abstract]
    dependent AbstractComparable : AbstractValue {
        [is_query]
        CompareValue compare_to( in AbstractComparable x);
    };
    [FROZEN, is_abstract]
    dependent AbstractAddable : AbstractComparable {
        [is_query]
        AbstractAddable add( in AbstractAddable x);
        [is_query]
        AbstractAddable subtract( in AbstractAddable x);
        [is_query]
        AbstractAddable multiply( in AbstractNumeric x);
        [is_query]
        AbstractAddable divide( in AbstractNumeric x);
    };
    [FROZEN, is_abstract]
    dependent AbstractNumeric : AbstractAddable {
        [is_query]
        Decimal as_decimal();
        [is_query]
        double as_double();
        [is_query]
        long as_long();
        [is_query]
        CompareValue compare_to_zero();

    };
    [FROZEN]
    dependent Decimal : AbstractNumeric {
        [is_read_only, INITIALIZED] attribute string value;
        [is_query]
        Decimal rounded( in long places );
        [is_query]
        Decimal truncated( in long places );
        [type_parameter]
        attribute long magnitude = 21;
        [type_parameter]
        attribute long precision = 10;
        [type_parameter]
        attribute boolean is_signed = TRUE;
    };

    [FROZEN, is_abstract]
    dependent AbstractTime : AbstractAddable {};
    [FROZEN]
    dependent MonthDay : AbstractTime {  // Gregorian day of year
        [is_read_only, INITIALIZED] attribute long month;
        [is_read_only, INITIALIZED] attribute long day;
        [is_query]
        long day_of_year();
    };
    [FROZEN]
    dependent Date : MonthDay {  // Gregorian day since christian year zero
        [is_read_only, INITIALIZED] attribute long year;
        [is_query]
        long as_joulian_days();// Constraint to nasty - do it in code.
        [is_query]
        double as_days() = as_joulian_days();
```

```
        [is_query]
        long week_of_year();
        [MANAGER]
        Date create_date_from_joulian_days( in long days );
        [is_query]
        DateTime create_DateTime_with_time( in Time t );
    };
    [FROZEN]
    dependent Time : AbstractTime {
        [is_read_only, INITIALIZED] attribute long hour;
        [is_read_only, INITIALIZED] attribute long minute;
        [is_read_only, INITIALIZED] attribute double second;
        [is_read_only] attribute long time_zone_offset = 0;  //Supplied by local infrastructure on
create.
        [is_query]
        double as_seconds() = (hour*3600+minute*60+second);
        [is_query]
        double as_minutes() = (hour*60+minute+second/60.0);
        [is_query]
        double as_hours() = (hour+minute/60.00+second/3600.0);
        [is_query]
        DateTime create_DateTime_with_date( in Date d );
    };
    [FROZEN]
    dependent DateTime : Date, Time {
        [is_query]
        double as_seconds() = (as_joulian_days()*86400.0+hour*3600+minute*60+second);
        [is_query]
        double as_minutes() = (as_joulian_days()*1440.0+hour*60+minute+second/60.0);
        [is_query]
        double as_hours() = (as_joulian_days()*24+hour+minute/60.00+second/3600.0);
        [is_query]
        double as_days() = (as_joulian_days()+hour/24.00+minute/1440.00+second/86400.0);
    };
```

Detail

dependent AbstractValue : BaseDependent

Values are immutable objects representing a piece of data that may be simple or structured. Values include numbers, units, dates, time, weights and measures and other units. AbstractValue is the root of the value hierarchy.

Note that values are never modified, all calculations return a new instance. Since values are immutable (FROZEN in UML terms) pass by value and pass by reference are semantically equivalent.

string as_string()

Converts the value to a string in a format specific to the value. Since values are implemented within the scope of a single process, any format may be used as long as it is consistent. As_string does no formatting or locale conversions and is therefor not suitable for end-user display. As_string is useful for conversions program-internal string representation and display. Other services provide for formatted conversions.

exception_kind TypesNotCompatibleException

An exception returned by value types when the types of operands is not compatible.

attribute AbstractValue operation_on

The instance the operation is being performed on.

attribute AbstractValue operand

The argument to the operation

attribute string operation_name

The name of the operation.

enum CompareValue

An enumeration of comparison results and queries for use in various operations.

- LESS_THAN
- LESS_THAN_OR_EQUAL_TO
- EQUAL_TO
- GREATER_THAN_OR_EQUAL_TO
- GREATER_THAN
- NOT_EQUAL

dependent AbstractComparable : AbstractValue

Represents all values that can be compared for greater and less than.

CompareValue compare_to(in AbstractComparable x)

Compares the value to another value and returns one of the CompareValues: GREATER_THAN, LESS_THAN or EQUAL_TO.

If types can't be compared, TypesNotCompatible is raised.

dependent AbstractAddable : AbstractComparable

Represents any type that can be added to a unit of the same type.

AbstractAddable add(in AbstractAddable x)

The basic add operation as implemented by each non-abstract type. Returns a new addable instance of the same type as the receiver.

AbstractAddable subtract(in AbstractAddable x)

The basic subtract operation as implemented by each non-abstract type. Returns a new addable instance of the same type as the receiver.

AbstractAddable multiply(in AbstractNumeric x)

The basic multiply operation as implemented by each non-abstract type. Returns a new addable instance of the same type as the receiver.

Note that the argument must be a numeric type. Consider that it makes sense to multiply a time by two, but not to multiply a time by another time

AbstractAddable divide(in AbstractNumeric x)

The basic divide (by x) operation as implemented by each non-abstract type. Returns a new addable instance of the same type as the receiver.

dependent AbstractNumeric : AbstractAddable

Abstract type representing any form of number.

Decimal as_decimal()

Receiver converted to a decimal number.

double as_double()

Receiver converted to a double.

long as_long()

Receiver rounded and converted to an integer.

CompareValue compare_to_zero()

Compares the value to zero and returns one of the CompareValues: GREATER_THAN, LESS_THAN or EQUAL_TO.

dependent Decimal : AbstractNumeric

A number represented as a fixed point decimal.

attribute string value

The decimal number as a string which may also be used to initialize the number. The decimal character is ".". Note that formatting services should be used for localization.

Decimal rounded(in long places)

Returns a new Decimal based on the receiver rounded to the given number of places. If places is negative, rounding is done based on the number of places left of the decimal point.

Decimal truncated(in long places)

Returns a new Decimal based on the receiver truncated to the given number of places. If places is negative, truncation is done based on the number of places left of the decimal point.

attribute long magnitude = 21

A type parameter specifying the maximum number of places to the left of the decimal point. A type parameter may only be changed in a subtype. The default is 21. The default maximum range of a decimal is: +-999,999,999,999,999,999,999.9999999999. This is a very large number.

attribute long precision = 10

A type parameter specifying the maximum number of places to the right of the decimal point. A type parameter may only be changed in a subtype. The default is 10.

attribute boolean is_signed = TRUE

A type parameter indicating if the decimal is signed. . A type parameter may only be changed in a subtype.

dependent AbstractTime : AbstractAddable

An abstract type representing various notions of time.

dependent MonthDay : AbstractTime

A concrete type representing a month and a day without specifying a year. This is used, for example, to specify the date of the financial year in a corporation or a birthday. When an abstract numeric is added or subtracted to a MonthDay it will be computed in units of days.

attribute long month

The month number within a year.

attribute long day

The day within the month. If a day is in excess of the number of days in the month, it will be "rolled over" into the next month.

long day_of_year()

Day of the year based on a non leap year.

dependent Date : MonthDay

A concrete type representing a particular date based on the Gregorian calendar.

Days and months are limited to the correct number for the given date, values that exceed the legal value will be "rolled over" to the next higher unit. When an abstract numeric is added or subtracted it will be in units of days.

attribute long year

The year number.

long as_joulian_days

The date converted to the number of days since 1/1/0 taking into account leap years and adjustments. Returned as an integer.

double as_days()

The date converted to the number of days since 1/1/0 taking into account leap years and adjustments. Returned as a double. In Date, as_days and as_joulian_days is equivalent. In DateTime the time is represented as a fraction of the day.

long week_of_year

The week number within the year.

Date create_date_from_joulian_days(in long days)

A MANAGER scope operation returning a date based on Julian days.

DateTime create_DateTime_with_time(in Time t)

Creates a DateTime by using the receiver date and the supplied time.

dependent Time : AbstractTime

A concrete type representing time as hours / minuets / seconds. An abstract numeric may not be added to Time as the unit of time is indeterminate.

attribute long hour

The hour of the time. Relative to midnight in a date.

Seconds and minutes are limited to being less than 60. Values equal to or in excess of 60 will be rolled over to the next higher unit.

attribute long minute

The minute of the hour.

attribute double second

The second within the minute. Seconds are floating point double numbers.

attribute long time_zone_offset

The time zone offset from GMT in which the time was created. By default, the time_zone_offset is set by the infrastructure when the time is created.

double as_seconds()

The time as a number of seconds. May contain fractions.

double as_minutes()

The time as a number of minutes. May contain fractions.

double as_hours()

The time as a number of hours. May contain fractions.

DateTime create_DateTime_with_date(in Date d)

A date crated based on the receiver time and a given date.

dependent DateTime : Date, Time

A date and time combined into a single instance representing a particular time of a particular day. When an abstract numeric is added or subtracted it will be in units of days.

IDL

```
value AbstractAddable  :
      AbstractComparable{
      AbstractAddable add(
         in AbstractAddable x)raises (Boca::BocaException);
      AbstractAddable subtract(
         in AbstractAddable x)raises (Boca::BocaException);
      AbstractAddable multiply(
         in AbstractNumeric x)raises (Boca::BocaException);
      AbstractAddable divide(
         in AbstractNumeric x)raises (Boca::BocaException);
   };
   value TypesNotCompatibleException  :
      BaseException{
      AbstractValue operation_on() raises(Boca::BocaException);
      AbstractValue operand() raises(Boca::BocaException);
      string operation_name() raises(Boca::BocaException);
   };
   value AbstractNumeric  :
      AbstractAddable{
      decimal as_decimal()raises (Boca::BocaException);
      double as_double()raises (Boca::BocaException);
      long as_long()raises (Boca::BocaException);
      CompareValue compare_to_zero()raises (Boca::BocaException);
   };
   value decimal  :
      AbstractNumeric{
      string value() raises(Boca::BocaException);
      decimal rounded(
         in long places)raises (Boca::BocaException);
      decimal truncated(
         in long places)raises (Boca::BocaException);
   };
   value AbstractTime  :
      AbstractAddable{
   };
   value Time  :
      AbstractTime{
      long hour() raises(Boca::BocaException);
      long minute() raises(Boca::BocaException);
      double second() raises(Boca::BocaException);
      long time_zone_offset() raises(Boca::BocaException);
      double as_seconds()raises (Boca::BocaException);
      double as_minutes()raises (Boca::BocaException);
      double as_hours()raises (Boca::BocaException);
      DateTime create_DateTime_with_date(
         in Date d)raises (Boca::BocaException);
   };
   value MonthDay  :
      AbstractTime{
      long month() raises(Boca::BocaException);
      long day() raises(Boca::BocaException);
      long day_of_year()raises (Boca::BocaException);
   };
   value Date  :
      MonthDay{
      long year() raises(Boca::BocaException);
      long as_joulian_days()raises (Boca::BocaException);
      double as_days()raises (Boca::BocaException);
      long week_of_year()raises (Boca::BocaException);
      DateTime create_DateTime_with_time(
         in Time t)raises (Boca::BocaException);
   };
   value DateTime  :
      Date,
      Time{
   };
};
#endif /* _BOCA_IDL_ */
```

Index of definitions