Defining a Type

mint_types_tn_1 / version 0.1
25-6-27
Leon Starr

#types
#type_definition
#technote
#mx

Let's study the process of defining some example User Types from system level building blocks.

Defining Shaft ID

We start with Elevator Case Study data type example: Shaft ID.

This is going to be a Domain Specific Type.

A Shaft ID is an "S" followed by a counting number 1, 2, 3 ...
It's purpose is strictly for naming so you can't do any math on the numeric values and you can't compare them in any other sense than equality.

Since all these values are defined prior to runtime, there is no need for any operation
to assign new values.

It makes sense, then to build these up from the mx_string system type.
And since we don't anticipate having that many shafts, we can limit the size to a maximum
of say, three digits. So our desired format can be expressed as a regex:

```
S[1-9][0-9]{0,2}
```

A set of General Types comes standard with MX to support type specification.

We're going to build directly on top of the String general type.

Shaft ID -> String -> mx_string

Shaft ID -> String
  configure regex:
    `S[1-9][0-9]{0,2}`

We will not set a default value, and since the regex limits the length to a maximum of four characters,
we don't need to explicitly provide a max length. In fact, all max lengths are specified via regex anyway.
Since we don't specify a default value, none is applied to this type.

Defining Distance

This is a General Type (see below)

If we wanted to define a Domain Specific Type like Height (for floor positions) we might want to restrict the range to be >= zero and adjust the precision.

Height -> Distance
  range >= 0
  precision 2

Domain Specific Types

Both General and Domain Specific Types are user types as opposed to System Types. But Domain Specific Types are intended to be of use within only one domain within a given system. For example, Pressure or Temperature are examples of domain specific types. True, these types might be used in multiple domains in different Systems. But within the *same* System, they should only appear in a single Domain as a good modeling practice.

General Types

General types are built up from the System Types and may be specified directly by the user, though they are discouraged for the most part as most domains will need to impose further constraints on them.

There isn't anything special about how General Types are defined. By "general" we just mean that these types have applicabilty as building blocks across a wide variety of Modeled Domains.

User types are inherited from System or other User Types.
Inheritance can modify the parent by:
- adding new operations and excluding or overriding inherited operations

- defining or overriding the default value
- defining or overriding symbolic selectors
- adjusting the range for numeric types (built up from a numeric system type)
- adding or excluding components and setting their type specific characteristics

Distance
-> Rational
base units: m
precision: 6
ops:
  to feet -> mx_rational

Natural
-> Integer
range: >= 1
default = 1

Posint
-> integer
range: >= 0

Integer
-> mx_integer
default = 0

Rational
-> mx_rational

String
-> mx_string
configure op:
  set regex: mx_string
  set default value : mx_string

Set
-> mx_set
configure op:
  set symbols : mx_set[mx_string]

System Types

These are platform independent types that serve as an elemental set of building blocks for defining user types.
They cannot be used directly, but can serve as a basis for defining user types. In other words, the lowest level user types
are built directly on System Types.

The numeric types can be constrained with a range.

To support execution, a corresponding implementation type must be selected. Some known implementation types are specified below.

mx_boolean
non-numeric
operations: set, unset, toggle, test
values: true / false

mx_integer
numeric
operations: add, sub, mult, div
selectors: minint, maxint
comparisons: eq, gt, ...
implementations:
    python int
    tclral integer

mx_rational
numeric
operations: add, sub, mult, div
selectors: minint, maxint
comparisons: eq, gt, ...
implementations:
    python float
    tclral double

mx_string
non-numeric
implementations:
    python str
    tclral string

mx_set

non-numeric