## Namespace

Consider an AST with root node named R.

All the nodes in this AST have unique internal identities, which are not suitable for human consumption. We need to have symbolic names as well, not necessarily for all nodes, just for the ones we want to refer to conveniently.

In Freon, having a name means

* The node as a property with key “name” of type “identifier”

The purpose of names is: other nodes (named or unnamed) may refer to a node by way of its name.

There is no other way to refer to nodes (another way might be: navigate, like two steps up, one to the left, then three down)

The names in the AST have to be unique. Otherwise you could not use them for reference unambiguously.

Let is introduce the concept of namespace now.

A namespace is a map (aka dictionary, associative array) where each pair represent a named node (the key is the name and the value is the internal identifier)

The root node R of the AST has a namespace associated with it.

All the names of nodes that are descendants of the root node are in the namespace map, no exceptions.

We can generalize this concept of namespace.

We allow a namespace (a map of node names and node internal identities) to be associated with any node (let’s call this node N)

All the names of nodes that are descendants of N are in the namespace map associated with N.

Moreover the names that are in the namespace map N are removed from the namespace map at the root R

In other words, any name in the AST is included in exactly one namespace map.

## Reference

Let us require that the node N (the one that has a namespace map associated with it) has a name itself, like we already implied in the example above, but now we state explicitly this is a requirement.

Therefore the namespace map at the root has an entry with key N and a value that identifies the node with the namespace N associated.

This requirements allows us to introduce a cascaded reference.

* First we reference the namespace node
* Then we reference a node within the namespace thus referenced.

Example: a reference N.A where the dot means

* Assuming we are now refering to a node with a namespace associated, look up the name-after-the-dot in the associated namespace and refer to the node identified there.

The cascaded reference allows us to refer to any node in the entire AST.

Examples

* Reference “B” for a node that is in the root namespace but not in the N namespace
* Reference “N.A” for a node that is in the N namespace

This way of notating references works for all named nodes in the AST

For convenience (aka syntactical sugar) you may refer to “N.A” as just “A” if the reference is contained in a node that is a descendant of N. This abbreviation is optional.

In fact we have already made use of it here, because strictly speaking (and assuming the root has name R) we should write R.B and R.N.A

To this point we assume that each node is allowed to contain a reference to any other node (including to self). We will discuss mechanism to contrain this later.

In the example above, the root namespace may have a node named D and the inner namespace N may have a node named D. This does not violate the uniqueness requirement. Also both are unambiguously referable, by D and N.D respectively

Some more terminology.

The name of the node (like A or B in the example) is often called as the **‘local name**’ or ‘base name’ or ‘simple name’ or just ‘name’ of the node.

The name of the node that includes all the namespace prefixes up to the root of the AST (like R.B and R.N.A in the example) is often called ‘**fully qualified name**’ or ‘global name’

The name of the node that includes some of the namespace prefixes going up in the AST (like B or N.A – leaving out the root node name) I would call ‘**partially qualified name**’. This is not a common term but I think we need it. (aka ‘regional name’? something between local and global :-)

In some formalisms there is a thing like ‘**unqualified name**’ (XML for example). This means that you have a name that is not part of a namespace (internally this is solved by introducing a “namespace with no name”). OPEN ISSUE does Freon have unqualified names?

## Model Unit / Package

A model unit is an AST with a namespace associated with its root (the root must be named itself). In Freon this is the default, there are further nodes in the AST that have a namespace associated with it, unless you explicitly state this.

A model unit is by default not a namespace in Freon.

A model is a set of model units. The names of the model units must be unique within the model.

OPEN ISSUE do we consider the model to be a namespace? Yes, but it is unnamed. This namespace contains the names of the model units. By convention, we hardly ever put the name of the model in front of the cascaded name, but we might want to if we are into composing models.

In Freon the model is the “universe” (similar to the repository in LionWeb), so we are not into composing models.

The model unit refers to a notion that is often closely linked to namespaces: the package.

Not sure what package has to do with the discussion. Is is assumed that package is a node in the AST ?

But these are very different things.

* A namespace is the unit of naming (unique names of nodes that are all descendants of the node holding the namespace map).
* A package it the unit of conveyance (transmission and storage)

In the java ecosystem, the two concepts are intertwined by multiple conventions

* a package can only hold one namespace.
* The name of the package equals the namespace it holds.
* If you have a reference and you need to load the node it refers to, you find a package based on the namespace.

I think that looking at the package as a (namespace) node in the AST, there is nothing special about it.

The last point may lead to complications.

When looking at a reference in java like java.nio.file.Paths, there is no way to decide

* Is there a package “java.nio” that contains the namespace “file” which contains the class Paths (class happens to be a namespace too)
* Is there a package “java.nio.file” which contains the class Paths

This is a problem for the part of the system that is in charge of finding packages that contain a particular reference (aka classloader). The problem is caused by relying on convention (deriving the package name from the reference) instead of configuration (keeping a registry of which packages contain which referable nodes)

OPEN ISSUE how do we avoid such issues for the Freon “model unit loader”? model unit also has the dual purpose of being a namespace and a package

## Scope

Scoping rules are rules to constrain which references are allowed to originate from a particular node in the AST.

Scopes and namespaces are different things

* A namespace is a mechanism for achieve unique names and unambiguous references without a global authority.
* A scope is a mechanism for constraining references based on the node where the reference originates (is contained).

One common scope rule is “up up but not down” : hierarchical rule

From a node Z in the AST you are allowed to reference names contained in

* the namespace N that Z is in
* the namespace M that N is in
* the namespace L that M is in

in other words: all the way up to the root, but never down.

*TODO discuss shadowing and its interaction with partially qualified names*

One other common scope rule is

* import a namespace

import can be interpreted in two ways

when importing a namespace Q in a namespace P

1. all the names present in Q are considered also present in P. This may lead to name clashes
2. all the names present in Q can be used as a reference in the form of Q.name, this does not lead to clashes.

*OPEN ISSUE what does importing a namespace in Freon do? I understand it is option 1)*

Importing a namespace with option 2) often comes with syntactic sugar like aliasing.

Example:

* import “java.nio.file” as “file”
* refer to class “java.nio.file.Paths” as “file.Paths”

One might argue that such scope rules should not be hardwired in Freon. Some of them make sensible defaults, others should be easily customizable on level 2.

In Freon only the hierarchical rule is buitin.

## Requirements for MuDForM

These requirements come from conventions (and positive experience) of names and namespaces in the clojure language and the datomic database.

Since clojure is “just another programming language”, Freon should be able to accommodate this.

Not quite: Freon is **not** meant to be used for programming languages, but for domain specific languages, whose users are **not** developers..

**Notation of cascaded references aka qualified names?**

In java the character separating the parts of as FQN is the "."

(like in java.nio.file.Paths)

What we want to have is

* the local name is separated by "/"

(like in java.nio.file/Paths)

In Freom the separator character is “just a projection” and the separator character can be set in the .edit file. The FQN is actually a list of names in the internal Freon code.

**Attribute containers**

Many concepts in MuDForM are attribute containers: entity, activity, …

An attribute container has a unique local name (in some namespace), for example Customer in the namespace Examples.Bank

An attribute may seem to have a unique local name inside the attribute container, for example the attribute name is ID and the regional name is Customer.ID and the global name is Examples.Bank.Customer.ID

But this is what we want to present

Examples.Bank/Customer>ID

Where Examples.Bank is the namespace name and Customer>ID is the name of the attribute that is unique within that namespace

This is important for the way that attributes are conveyed through the system outside their container. Attributes must have a unique name.

*In freon I want to be able to compute the local name of the attribute by concatenating the name of the container (*Customer*)and a locally unique label (*ID*)*

What is described here does not fit in the description in the rest of the document. “Customer” should be a namespace as well, otherwise e.g. its ID would be visible inside “Examples.Bank” as would all ID’s of all Customers.

What I understand that you want is something like the following:

* Customer is a “pseudo” namespace, names inside should be unique.
* All names/attributes inside Customer should be directly visible (pseudo-exported) in the parent namespace Examples.Bank
* The visible name of all attributes inside Customer should be derived from its local name + its pseudo-container name.

The global name of the ID attribute would then consist of three names:

* Example
* Bank
* Customer>ID

**General Observations**

* The use of “,”, “/”, “>” inside a qualified name looks rather unique to me. Never encountered this in any programming language I know. In this sense Clojure is really not just another programming language.
* I (think I) understand the subtle distinction between “.”, “/” and “<\>”. I am not convinced that this distinction is helpful for non-developers, I wouldn’t be surprised if it would make things only jharder for them.