From: David McCusker <davidmc@netscape.com>
Subject: [MORK] brief syntax summary related to MDB usage
Date: 1999/03/16
Message-ID: <36EECF06.5C27395@netscape.com>
Content-Transfer-Encoding: 7bit
Content-Type: text/plain; charset=us-ascii
Organization: Ontology Mechanics Guild
Mime-Version: 1.0
Reply-To: davidmc@netscape.com
Newsgroups: netscape.public.mozilla.mail-news

This is a very brief primer on the Mork text format which avoids the use of a grammar as part of the explanation. (For a grammar, and a comparison to XML demonstrating prospects of mechanical translation, see "[MORK]" postings in this group in the middle of December 1998.)

I usually won't say how the MDB usage of Mork is narrower than Mork's definition, and I won't describe how the Mork runtime I'm now writing constrains or organizes Mork in ways not implied by the defintion. And I won't pretend the definition can change to suit my every whim.

Let's suppose you're writing an address book (the usage that motivates this particular posting), and you need to represent collections of address book cards in an address book, where each card is composed of some subset of a miscellaneous, open-ended set of card attributes. In fact, suppose the following LDIF entry is a card you want to represent:

```
dn: cn=John Hackworth,mail=jhackworth@atlantis.com
modifytimestamp: 19981001014531Z
cn: John Hackworth
mail: jhackworth@atlantis.com
xmozillausehtmlmail: FALSE
givenname: John
sn: Hackworth
```

In Mork, each attrribute appears as a cell, composed of a column and a value. One can write "givenname: John" as cell (givenname=John), where both column and value are give explicitly rather than as hex IDs that point to the column and value stored elsewhere. The same cell might be (^8^9), if 8 denoted "givenname" in the column space, and if 9 denoted "John" in the atom space.

Every cell is delimited by an open and close paren. A hex ID is introduced by '^'. A hex ID can be scoped by an explicit space, which is introduced by ':' and followed either by an explicit scope name or else by a hex ID denoting the scope name in the column space. So (^8^9) is semantically identical to (^8:c^9:a), since "c" is the name of the column space, and "a" is the name of the atom space. (You can have other value spaces in Mork, but MDB does not use them now.)

A fully scoped ID is called a Mork oid (object id) because it includes both the hex ID and the name space scope in which the ID is used. So ^8:c and ^9:a are both oids, as are both ^8:^FEED and ^9:^FACE. Since the column and atom scopes are implied, both ^8 and ^9 are also oids.

Because a cell is terminated by ')', this means every ')' in the value
must be escaped. Also, '\' itself must be escaped, as well as '$'
which is used to introduce a byte written as two hex digits. So the
a string written in C syntax as "$cash\\(\015cows)\n" might appear as
a Mork cell like this: (col=\$cash\\($0Dcows\)$0A). Currently I
don't have any Mork syntax to express base 64; I'll add it later.

The entire LDIF record above can be written as a Mork row like this:

```
[1:cards (dn=cn=John Hackworth,mail=jhackworth@atlantis.com)
 (modifytimestamp=19981001014531Z)(cn=John Hackworth)(givenname=John)
 (mail=jhackworth@atlantis.com)(xmozillausehtmlmail=FALSE)(sn=Hackworth)]
```

Every row is delimited by an open and close bracket (except when a row
oid alone is adequate inside a table when a row member is expected).
A row must start with the row oid, which if not explictly scoped will
default to the same row scope as the enclosing table. A row is composed
of cells with unique columns, where each cell is a row attribute. A
row can express meta-attributes as a nested row (with nesed brackets).

The row can also be written with columns replaced with oids like this:

```
< <(atomScope=c)> (80=cards)(81=dn)(82=modifytimestamp)(83=cn)
  (84=givenname)(85=mail)(86=xmozillausehtmlmail)(87=sn) >

[1:^80 (^81=cn=John Hackworth,mail=jhackworth@atlantis.com)
 (^82=19981001014531Z)(^83=John Hackworth)(^84=John)
 (^85=jhackworth@atlantis.com)(^86=FALSE)(^87=Hackworth)]
```

Every dict is enclosed by an open and close angle bracket, and a dict
is composed of aliases (cells with explicit hex ID columns and column
name values) and dict meta-info enclosed by nested angle brackets. For
example, the <(atomScope=c)> above says the dict uses the column scope
as the name space for all the string values defined, so that hex IDs
in this dict live in the column scope name space.

(Note that hex IDs assigned to string tokens usually start with 0x80,
because by convention the hex IDs for 0x0 through 0x7F are associated
with the one byte strings whose single byte is that integer value. So
the integer token value for the column scope 'c' is the same as the
ASCII integer value for character 'c'. I won't bother explaining how
this helps prevent recursion at runtime in bootstrapping name spaces.)

We can also replace all the cell values in that row like this:

```
< <(atomScope=c)> (80=cards)(81=dn)(82=modifytimestamp)(83=cn)
  (84=givenname)(85=mail)(86=xmozillausehtmlmail)(87=sn)>

<(90=cn=John Hackworth,mail=jhackworth@atlantis.com)(91=19981001014531Z)
 (92=John Hackworth)(93=John)(94=jhackworth@atlantis.com)(95=FALSE)
 (96=Hackworth)>

[1:^80 (^81^90)(^82^91)(^83^92)(^84^93)(^85^94)(^86^95)(^87^96)]
```

Collections of rows are called tables, and the are enclosed in braces,
with table meta-info expressed as cells inside nested braces (just as
meta-info elsewhere is expressed by cells in nested delimitors). A

row in a table can either be a row oid, or a row definition using the square bracket notation shown above. There is no such thing as an undefined row oid, since the appearance of a naked oid in Mork is considered the definition of an empy row with no cells specified.

If we assume another card with oid 2:cards for John Galt, then we can make a table containing both John Hackworth and John Galt like this:

```
{ 1:cards {(rowScope=cards)(tableKind=Johns)} 1 2 }
```

Note that table IDs in MDB are allocated in the row scope name space. This table expression is equivalent to the following two alternatives:

```
{ 1:^80 {(rowScope^80:c)(tableKind=Johns)} 1:^80 2:^80 } // with oids

{ 1:^80 {(rowScope^80:c)(tableKind=Johns)} // with explicit row cells
  [1:^80 (^81^90)(^82^91)(^83^92)(^84^93)(^85^94)(^86^95)(^87^96)]
  [2 (mail=galtj@atlantis.com)(cn=John Galt)]
}
```

Okay, that's a good first taste of the Mork text format, but now I have to get some coding work done. Maybe later I'll say more about how one might organize tables and dicts for specific purposes.

David McCusker, speaking only for myself, mozilla mail/news client eng
Values have meaning only against the context of a set of relationships.