Collection Identifiers for RUcore May 14, 2007

Collection Identifiers

In the May 10, 2007 Software Architecture Working Group, it was decided to use the Noid convention for collection identifiers in RUcore. The noid process allows for a very general and customizable format for generating identifiers. The format consists of a constant prefix and a mask that specifies a format. For all RUcore collections going forward, as of R4.2, we will use a collection ID of the form **rucore.zdddddd**. The numeric part of the identifier will start with the digit string 000100 and will increment sequentially. Thus the collection id's will appear as follows: rucore000100, rucore000101, rucore000102, etc. It should be noted that the persistent ID includes the collection ID as a subfield, so a typical PID would look like the following: 1782.1/rucore000100.collection.[integer assigned by WMS]. Since resources will be cited using this handle, it is also a good marketing mechanism for RUcore. We also agreed to not actually use the noid code but will re-program according to the noid spec. This is simple to do and will give us more standard code. A further explanation of the Noid utility is included below.

The Noid Utility

The **noid** utility creates minters (identifier generators) and accepts commands that operate them. Once created, a minter can be used to produce persistent, globally unique names for documents, databases, images, vocabulary terms, etc.

A **noid** minter is a lightweight database designed for efficiently generating, tracking, and binding unique identifiers, which are produced without replacement in random or sequential order, and with or without a check character that can be used for detecting transcription errors.

An identifier generated by a **noid** minter is also known generically as a "noid" (standing for nice opaque identifier and rhyming with void). While a minter can record and bind any identifiers that you bring to its attention, often it is used to generate, bringing to your attention, identifier strings that carry no widely recognizable meaning. This semantic opaqueness reduces their vulnerability to era- and language-specific change, and helps persistence by making for identifiers that can age and travel well.

A Template is a coded string of the form Prefix.Mask that is given to the noid **dbcreate** command to govern how identifiers will be minted. The Prefix, which may be empty, specifies an initial constant string. For example, upon database creation, in the Template

tb7r.zdd

the Prefix says that every minted identifier will begin with the literal string **tb7r.** Each identifier will end in at least two digits (dd), and because of the z they will be sequentially generated without limit. Beyond the first 100 mint operations, more digits will be added as needed. The minted noids will be, in order,

tb7r00, tb7r01, ..., tb7r100, tb7r101, ..., tb7r1000, ...

The period (".") in the Template does not appear in the identifiers but serves to separate the constant first part (Prefix) from the variable second part (Mask). In the Mask, the first letter determines either random or sequential ordering and the remaining letters each match up with characters in a generated identifier. Perhaps the best way to introduce templates is with a series of increasingly complex examples.

.rddd
    to mint random 3-digit numbers, stopping after 1000th

.sdddddd
    to mint sequential 6-digit numbers, stopping after millionth

.zd
    sequential numbers without limit, adding new digits as needed

bc.rdddd
    random 4-digit numbers with constant prefix bc