Welcome to YAML

YAML is a minimal markup language based on XML and Minimal XML. YAML is based upon the following principles:

- It is primarly for hierarchical textual data representation.
- Compatibility with XML 1.0 is nice, but not a requirement.
- Mathematical / Model integrety is essential
- Verbosity is acceptable, Regularity is preferred
- YAML should complement other notations and data types.
- strong data typing would be ideal
- Direct support for Standard ML data types would be great.

As such, YAML is defined as a labeled tree structure. It differs from XML in many ways:

- Only elements are supported, no PI, Comment, Attribute, etc.
- Element abbreviated syntax, <tag/>, is allowed.
- Elements have a value or children, never both.
- Namespaces are not supported, and the colon is used.
- UTF-16/ISO8859-1 are the only encodings (for now)
- More than one top level element is allowed (for XML compatibility don't do this, but it is necessary for concatination to be a closed operator)
- Whitespace treatment is just like HTML, one or more occurances of tab, enter, etc. are compressed into a single space.

Data Typing

Tag names consist of two parts, a label and a data type. These parts are seperated by a colon. The supported data types are:

Integer	:int	Exact integer (equality operator)
Real	:real	Inexact floating point approximation (no equality operator)
Char	:char	A single character, integers or hex notation may be used.
Boolean	:bool	Either "true" or "false".
String	(default)	A string value, & < and > allowed.
List	:list	This is an ordered list, of items having the same type.
Record	:record	This is an unordered map, each label must be unique.
Tuple		
:tuple	Use _1, _2, etc for the tag names.	

For convienence, the following "derived" types are emitted. They are not Standard ML types, but may be useful nonetheless.

Bag	(default)	An ordered list of items having any type.
IntegerList	:intlist	A list of integers, seperated by a space.
RealList	:reallist	A list of real numbers, seperated by a space.

CharList	:charlist	A list of characters, seperated by a space.
Binary	:base64	A string encoded with Base64, in groups of 8 characters seperate by spaces.

Note: If strict XML+Namespaces compatibility is desired, then the document may only have Bag/String content. Otherwise, the above is XML 1.0 compliant, where the "namespace" experiment is a strict data type!

Tag names

In general, Tag names follow the requirements of XML tag names, although tags with periods have special meaning, and tags beginning with an underscore are reserved. In particular, tags with one or more periods must use a DNS based structure where the right-most parts are a top level domain, like "com", "org", "co.uk". Then, immediately preceding the top level domain, is the registerd part, like "clarkevans". And preceding the registered part, is up to the user. Therefore: "timesheet.clarkevans.com", and "zoom.mytag.domain.co.uk" would both be valid names according to this scheme.

Also note, that the data type can be appended to the end of any of these data types using the :, as described above. Thus, "timesheet.clarevans.com:list" would be a globally qualified list.

Information Model

Theinformation model for YAML is as follows (borrowing heavily from Minimal XML).

Tag	:=	Sequence of one or more characters
Value	:=	Sequence of one or more characters
Node	:=	Tag
	+	Value xor Children
Children	:=	Ordered sequence of one or more Nodes

Also, the Node may have the following computed information.

ExtVal	The extended value of the node. This is the Value if the node has a value, otherwise it is defined as the ExtVal of the first child in the sequence of Children
Type	One of the data types enumerated above.
Label	The Tag without the :type trailer
Segs[]	An array of Label segments between the periods

Example

```
<timesheet.clarkevans.com:record>
  <person:record>
    <id:int>
      293945</id:int>
    <name:record>
      <given>
        Clark</given>

      <family>
        Evans</family></name:record></person:record>
  <journal:list>
    <journal:record>
      <date:record>
        <day:int>
          12</day:int>
        <month:int>
          1</month:int>
        <year:int>
          2001</year:int></date:record>
      <description>
```

```
        On this day, I worked on three topics,
        soon to folow.</description>
      <entry:list>
        <entry:record>
          <duration:int>
             120</duration:int>
          <project>
            Self-Study, ML</project>
          <description>
            Finished Chapter 3 of book.</description>
          <reference:list>
            <reference>
              Elements of ML Programming, by Jeffery
              D. Ullman, ML97 Edition</reference>
                </reference:list></entry:record>
        <entry:record>
          <duration:int>
            90</duration:int>
          <project>
            Double Gemini</project>
          <description>
            Worked on software development schedule, final delivery
            date end of March.</description></entry:record>
              </entry:list></journal:record></journal:list>
              </timesheet.clarkevans.com:record>
```

Some thoughts

1. This is very preliminary thoughts on the subject, feedback is very welcome.
2. With a :record type, /path/expr can be sure that there is one and only one "expr" entry for a given path, etc.
3. A common problem with lists is that list[3] is usually not modeled well with xml/xpath. Most paths have the form /mytype-list/mytype. This can be given a short-hand since the name portion must be the same with this scheme. so /mytype:list/mytype[n] -> /mytype[n]
4. You might say that this is moving some of the schema into the markup, yes. I think this is good, as strict data typing is possible.
5. I'm thinking of making "record" the default, and elminating the bag type...
6. Hmm. I'd like a way to create user defined ADTs. I was thinking that a "c style structure" definition could be used. Hmm.