BML: The Smallest Markup Language

There's been a trend to simplifying markup languages, initiated by a not necessarily perception that SGML was too complex, and by the questionable claim that no users want to look at the raw markup. XML was the first step in this trend. JSON and MicroXML are further attempts to simplify things.

A new markup language called BML (Basic Markup Language) has been created strictly as an aid in developing the Bobbee programming language, and which is minimalistic in all respects.

BML can be considered an example of how simple a markup language can become, while remaining useful.

The BML package, implemented using the Bobbee language, supports both rule-based and procedural processing of BML data, and supports both serial and tree-based rule processing.

- BML has only one kind of node that serves as either markup or data, determined by the user.
- There is one rule type: “bmlItem” for serial processing.
- There is one rule type: “bmlNode” for tree processing.

- Entering BML markup by hand can be inconvenient, requiring escaping of all spaces and parentheses that are data, amongst other things.

These limitations are appropriate for machine-to-machine communication, with only the occasional intervention by human and similar-type beings.

An Example of Using BML

```
(p (text: (Each\ node:)))
(list
  (item (text: (is\ surrounded\ in\ parentheses,)))
  (item (text: (consists\ of\ a\ text\ sequence,\ and)))
  (item (text: (has\ zero\ or\ more\ nested\ nodes.)))
)
```

The designer of a particular markup notation determines which nodes are markup labels and which are data, in the same way that they determine the import of any element or data in XML. In this example, the subcomponents of the "text:" tag are text, and everything else is markup.

Although BML input superficially looks like Lisp s-expressions, it is textual markup and data. Don't be fooled by the parentheses. On the other hand, yes, BML was inspired, in part, by Lisp, which is itself a markup language.

A Bobbee Program That Processes That Data

```
program bmltree;

choose bmlNode ("p") {
  print ("<P>"); processChildren; println ("</P>");
}
choose bmlNode ("list") {
  print ("<UL>"); processChildren; println ("</UL>");
}
choose bmlNode ("item") {
  print ("<LI><P>"); processChildren; println ("</P></LI>");
}
choose bmlNode ("text:") {
  print (* [0].textValue);
}
```

The Bobbee Program's Output

The program translates BML input into a simple XML form:

```
<P>Each node:</P>
<UL><LI>is surrounded in parentheses,</LI>
<LI>consists of a text sequence, and</LI>
<LI>has zero or more nested nodes.</LI>
<UL>
```

About The Bobbee BML Processing Program

The example BML program processes BML input as follows:

- A program is prefixed with a declaration of the optional features it uses. In this case, **program bmltree;** says use the "BML tree" package.
- Markup language processing rules consist of the keyword **choose**, followed by the name of type chosen by the rule (in the example it's "bmlNode") followed by an optional condition and the processing done when the rule is selected.
- **processChildren** calls the rules needed to process the components of the recognized markup node.
- The "*****" identifies the currently selected markup node, which when doing "tree" processing can be indexed to select its children and their properties without having to have rules to do the job.

How this comes about is defined entirely using the Bobbee language itself. The language mechanisms are defined below and can be used to define any markup language, and any markup language processing a user wishes. At present, there are packages for SGML, XML, JSON, MicroXML and BML processing available, but others can be added.

How BML Tree Processing Is Defined

This declaration defines the meaning of a "bmlNode" rule:

```
def choose bmlNode (BmlNode) choose (*.textValue)
  default {print ("(" + *.textValue); processChildren;
           print (")");}
```

- **choose bmlNode** is selected when a "BmlNode" node type is encountered.
- If one or more names are specified in a **choose** rule then they are compared to the "textValue" of the "BmlNode".
- The **default** part defines the default behaviour if no rule is specified. In this case it just outputs the node with its origin markup.

How Bobbee Is Told What To Do With BML Input

This is the bmltree.bjp file, which defines BML tree processing. It declares that BML parsing is done by the bj.bml.* package with help from the bj.patternmatching.* package.

```
# Display a message when compiling:
"Use BML tree profile.";

# Import what's needed by these declarations or the user:
import bj.bml.*, bj.patternmatching.*;

# Define the "choose" rules, what's used as the selecting
# "name", and the rule's default behaviour, if any:
def choose bmlNode (BmlNode) choose (*.textValue)
  default {print ("(" + *.textValue); processChildren;
           print (")");}

# Define the different ways that markup processing can be
# initiated, especially what kinds of inputs are supported:
def parseBml (in : CharSequence) : void :-
  bmlNode (BmlTreeParser.parse (in));

def parseBml (in : InputStream) : void :-
  bmlNode (BmlTreeParser.parse (in));

def parseBml (source : MatchableInput) : void :-
  bmlNode (BmlTreeParser.parse (source));

def parseBml (in : Readable) : void :-
  bmlNode (BmlTreeParser.parse (in));

def parseBml (data : String) : void :-
  bmlNode (BmlTreeParser.parse (data));

def parseBmlFile (fileName : String) : void :-
  bmlNode (BmlTreeParser.parseFile (fileName));
```