

# Generic parser implementation

Horand Gassmann, Jun Ma, Kipp Martin

November 11, 2009

## Abstract

This documents makes some recommendations concerning elements, rules, names, content and structure of bison rule files for parsing documents used within the OS framework. The emphasis is on uniformity; computational efficiency is secondary; it is expected. The guidelines were developed in parallel with the development of the OSrL parser, but they have application for other parsers.

1. For ease of development, trouble shooting and maintenance it is useful to have treatment of the different elements that is as uniform as possible. Computational efficiency is secondary; it is expected that the compiler will be able to deal with such issues.
2. Every *<element>* is parsed using three production rules: *elementStart*, *elementAttributes* and *elementContent*.
3. *elementStart* matches the opening tag (*<element>*); its code section can be used to verify that the element was indeed expected at this spot, particularly in cases where this is hard to infer from the Schema. There are two instances when such checks need to be made:
  - (a) If the elements do not have to appear in any particular order, it is necessary to verify that there was no prior occurrence of this *<element>* within the scope of its parent.
  - (b) If the element is contained in a repeat group, we must make sure that there are not more occurrences than specified in the *numberOf...* attribute of its parent.

In addition the code can be used to initial the attribute list. The occurrence of attributes is tracked with indicators *xxxAttPresent*, which can be set to **false** in this section. If an element has an optional *numberOf...* attribute, the variable holding the number of these items should be set to zero here to provide a default when the *numberOf...* attribute is missing.

4. *elementAttributes* is included as a separate rule so that checks can be made after the entire list of attributes has been processed. It is necessary to check that all mandatory attributes have indeed been provided, and there may be other checks as needed. The production rule is *elementAttributes: elementAttList*  
where *elementAttList* is a standard list rule, which expands into  
*elementAttList: | elementAttList elementAtt.*
5. *elementAtt* matches any of a list of attributes allowed under the current element, as in  
*elementAtt: elementxxxAtt | elementyyyAtt | elementzzzAtt ...*

6. Each *elementxxxAtt* is used to perform specific data checks, such as membership in an enumerative list, nonnegativity, etc., and to store the attribute value into the internal data structure. Moreover, attribute names, unlike element names, tend to be reused frequently. Thus *elementxxxAtt* may be a generic rule shared among many elements.
7. *elementxxxAtt* is also used to verify that the attribute has not been seen previously within the scope of the current element, to change its status from not present to present, and to assign the attribute value to a temporary variable.
8. If an element allows only a single attribute, the above can be streamlined, the rule *elementxxxAtt* replacing the rule *elementAttributes*.
9. If an element has no attributes, this rule is simply omitted.
10. An element attribute may be used to record the number of child elements that are given in an array list. The parser records the number of child elements actually encountered and compares against the declared number. Any discrepancy is recorded. Such *numberOf...* attributes also allocate the storage space for the child elements and set the counter to 0.
11. *elementContent* can be empty or nonempty. This is normally expressed by the rule  
*elementContent: elementEmpty | elementBody*  
 In some rare cases modifications from this rule are needed in order to avoid reduce/reduce conflicts when an element has several optional children that must occur in a particular sequence, for instance  
*variables: variableValues variableValuesString basisStatus otherVariableResultsArray*  
 where each of the child elements may be omitted — or indeed all of them together.
12. The code section in the *elementContent* rule can be used for consistency checks, storage of information into the data structure and, most importantly, to increment counters.
13. Empty element content is typically either “></element>” or simply “/>”. Code may be needed to detect empty element content and throw an appropriate error.
14. *elementBody* expands into a variety of different patterns, as needed. There could be
  - (a) an array of <child> elements, which is distinguished from an element list by using the rule name *childArray*, which expands into  
*childArray: | childArray childElement*
  - (b) several children in arbitrary order (*childList*) with a similar expansion
  - (c) other constructs as appropriate.
15. Each <childElement> would then be treated again as under point 2.