An extension declared through the <extension> syntax MUST NOT, in and of itself, cause any change to the semantics of a BPEL process. Rather, the extension declaration defines if the extensions identified by the denoted namespace must be supported or can safely be ignored.

An extension syntax token, in the form of an element or attribute that is used outside of an <extension> element, MUST be included in the BPEL process's definition in order to actually apply an extension semantics to a BPEL process. It is this extension syntax token, rather than the extension declaration, that indicates that new/changed semantics apply.

An extension syntax token can only affect BPEL constructs within the syntax sub-tree of the parent element of the token. In other words, extension syntax token MUST NOT "leak" their semantics outside the subtree. Here are two examples to illustrate the above concept further:

```
<process>
   ...
   <scope>
      <sequence>
         <invoke operation="operation1"
                 foo:invokeProperty="someNature" ... />
         <invoke operation="operation2" ... />
         <invoke operation="operation3"
                 foo:invokeProperty="someNature2" ... />
      </sequence>
   </scope>
</process>
```

The "foo:invokeProperty" extension attribute are applied to <invoke> activities for "operation1" and "operation3". The <invoke> activity for "operation2" must not be affected.

```
<process>
   ...
   <scope>
       <foo:invokeProperty>
          SomeNature
       </foo:invokeProperty>
       <sequence>
           <invoke operation="operation1" ... />
           <invoke operation="operation2" ... />
           <invoke operation="operation3" ... />
       </sequence>
   </scope>
</process>
```

On the other hand, the "foo:invokeProperty" extension element can be potentially applied to all <invoke> activities within the <scope> activity where the extension element are attached to.

It is recommended that the following design guidelines be followed when creating extensions:

- Each extension syntax token should only specify a relatively limited set of semantics that fits into a modular boundary. For example, a single extension syntax token should not specify changes to transaction semantics, security semantics, message delivery, QOS requirements, etc. Each of these changes should be broken out into separate tokens.

- Each extension syntax token should have a set of well-defined attachment points, that is, it should apply to a relatively small number of BPEL constructs. In the "invokeProperty" example above, the set of attachment points are restricted to <invoke> and <scope> constructs rather than to everything in BPEL. Some extensions may require a process level scope or need to apply to a large number of BPEL constructs but that should be the exceptions.