Basic SemiXML tests

Testing the basic use of SemiXML

Purpose of this report is that it tests the parsing and translating process of the several parts of the language. Here only the simple structures are tested and not the methods to generate new content.

The tests comprises the following chapters

- Generate a simple element
- Add attributes.
- Nested elements.
- Element blocks and their different types.

Generate a simple element

The code below will test a simple top level structure. A one liner**$some-element [ ]**should translate into **<some-element></some-element>**. Because there is no added configuration, it defaults to XML as its input as well as output so it will have an XML prelude. Also, because of the lack of this configuration, the element is interpreted as not self closing.

```
 5. use SemiXML::Sxml;
    my SemiXML::Sxml $sxml .= new;
    isa-ok $sxml, SemiXML::Sxml;                                    ✓

    $sxml.parse(:content('$some-element []'));
10. like ~$sxml, /'<some-element></some-element>'/,                 ✓
    "The generated xml is conforming the standard";
```

Add attributes.

Elements can have attributes. The attributes are, in a way, the same as in XML. The attributes, however, can be more simple when there are no spaces in their values. Quoting can be left out in those cases no matter what punctuation is used. We first test a single attribute, then we add some more and play with quotes and type.

```
    # bracketed value
    $sxml.parse(:content('$some-element attribute=<v a l u e> []'));
15. like ~$sxml, / 'attribute="v a l u e"' /,                       ✓
                 'attribute with bracketed <> value';

    # unquoted value
    $sxml.parse(:content('$some-element attribute=value []'));
20. like ~$sxml, / 'attribute="value"' /,                           ✓
                 'attribute without spaces in value';

    # double quoted value
    $sxml.parse(:content('$some-element attribute="v a l u e" []'));
25. like ~$sxml, / 'attribute="v a l u e"' /,                       ✓
                 'attribute with double quoted value';

    # single quoted value
    $sxml.parse(:content("$some-element attribute='v a l u e' []"));
30. like ~$sxml, / 'attribute="v a l u e"' /,                       ✓
                 'attribute with single quoted value';

    # more than one attribute
    $sxml.parse(:content('$some-element a1=v1 a2=v2 []'));
35. like ~$sxml, / 'a1="v1"' /, 'attribute a1 found';               ✓
    like ~$sxml, / 'a2="v2"' /, 'attribute a2 found';               ✓
```

```
    # boolean attributes
    $sxml.parse(:content('$some-element =a1 =!a2 []'));
40. like ~$sxml, / 'a1="1"' /, 'true boolean attribute a1 found';
    like ~$sxml, / 'a2="0"' /, 'false boolean attribute a2 found';
```

✓
✓

Nested elements.

Elements can be nested to any level. As we see later, there are some ways to control this behavior.

```
    $sxml.parse(:content('$some-element [ $some-other-element ]'));
    like ~$sxml, /:s '<some-element>'
45.                  '<some-other-element></some-other-element>'
                     '</some-element>'
                 /, 'An element within another';
```

✓

Element blocks and their different types.

Elements can have different types of blocks. One can specify it depending on the contents of the block. An element can also have more than one block.

```
    $sxml.parse(:content('$some-element [ block 1 ][ block 2 ]'));
50. like ~$sxml, /:s '<some-element>'
                     'block 1 block 2'
                     '</some-element>'
                 /, 'two blocks on an element';
```

✓

Other types beside '[' are '{' and '«'. The body '[...]' is used to have a normal block with possible nested elements. It is also possible to add comments starting with a '#' character. The use of '\{ ... \}' however, will inhibit the nesting of elements. The body '« ... »' works in the same way as '\{ ... \}' and is added as a convenience to enclose code more easily. If you want to use those markers of the body in text, the characters must be escaped with a '\\' character. For example '\\\{'. For the normal block '[...]' the '$' and '#' must also be escaped if you want the character in the text.

```
55. $sxml.parse(:content('$a1 { $a2 [ ] }'));
    like ~$sxml, /:s '<a1>$a2 [ ]</a1>' /, 'Inner element is not translated';
```

✓