2 Fundamentals of XML and BSML

This chapter provides a detailed introduction to the fundamentals of XML. We cover all the essential concepts for understanding XML markup, creating XML elements, and working with XML Namespaces. To make the concepts concrete and focused on bioinformatics, we introduce our first case study and explore the Bioinformatic Sequence Markup Language (BSML). BSML is an open XML standard used to represent biological sequences and sequence annotation data.

The chapter begins with a bare bones BSML document used to represent raw sequence data. As we introduce this first example, we take a bird's-eye view of XML document structure in general, including start tags, end tags, elements, and attributes. We also take a quick tour of the Rescentris Genomic Workspace^{TM}, a freely available software application that visually renders BSML documents.

After our high-level introduction to XML, we turn to a detailed description of the most important XML concepts. The topics include: tag structure, comments, processing instructions, the XML prolog, options for character encoding, XML grammars, and XML Namespaces. We also explore what it means to be "well-formed" and "valid," and how to test for either property.

The chapter concludes with a more detailed overview of the BSML specification. BSML is one of the most mature XML standards in bioinformatics, and has grown to encompass a very large set of bioinformatics sequence data. We do not have space to cover BSML in its entirety, and have therefore chosen to specifically focus on core elements of the BSML specification. We also provide several more BSML examples and explore these further within Genomic Workspace^{TM}.

2.1 Getting Started with BSML

The best way to learn XML is by example. Therefore, before discussing any major concepts we will begin with a sample XML document. This initial example adheres to the Bioinformatic Sequence Markup Language (BSML) [12; 13; 25]. BSML is an open standard for representing and exchanging biological sequence data. This data can include raw sequence data, sequence features, literature references, networks of biological entities, and even graphical display widgets.

BSML is a great place to get started in XML. The first main advantage is that BSML represents one of the very first XML formats specifically created for the life sciences. Second, BSML is comprehensive in scope. Those who have ushered the BSML specification and its continuing evolution have made every effort to ensure that BSML is capable of accurately representing biological reality and all the complexity that this requires. Furthermore, the BSML web site (*http://www.bsml.org*) includes excellent documentation, including tutorial documents, a reference manual, and an FAQ. Finally, Rescentris, Ltd. makes available a free BSML viewer that enables you to visually

Listing 2.1 The SARS virus, encoded in BSML

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- SARS coronavirus Urbani, complete genome. -->
<!-- Accession Number: AY278741  -->
<Bsml>
  <Definitions>
    <Sequences>
      <Sequence id="AY278741" length="29727">
      <Seq-data>
       atattaggtttttacctacccaggaaaagccaaccaacctcgatctcttgtagatctgttct
       ctaaacgaactttaaaatctgtgtagctgtcgctcggctgcatgcctagtgcacctacgcagt
       ataaacaataataaattttactgtcgttgacaagaaacgagtaactcgtccctcttctgcaga
       ctgcttacggtttcgtccgtgttgcagtcgatcatcagcatacctaggtttcgtccgggtgt
       gaccgaaaggtaagatggagagccttgttcttggtgtcaacgagaaaacacacgtccaactca
       gtttgcctgtcc
       [For brevity, sequence is truncated.]
      </Seq-data>
      </Sequence>
    </Sequences>
  </Definitions>
</Bsml>
```

inspect and interact with BSML documents. This makes for much more exciting and interactive examples.

Listing 2.1 shows our first XML example, a bare bones BSML document. The document represents the raw sequence data for the coronavirus responsible for severe acute respiratory syndrome (SARS). The virus sequence is 29,727 base pairs in length, and we have taken the liberty of only displaying the first few hundred base pairs. Let us now examine Listing 2.1, and we will continue with a high-level overview of the document structure.

There is a lot going on in our first example. For now, note the following items of interest:

- Our document begins with the characters "<?xml" This is formally known as the XML prolog and is used to indicate the version of XML and the character encoding.
- The second and third lines of the document are XML comments. Comments begin with the characters "<!--" and end with the characters "-->".
- Every XML document must have a root element. In our case, `Bsml` is the root element, and all other elements are descendants of the root. For example, the `Definitions` element is a child of the root `Bsml` element.
- XML elements are defined with start and end tags. For example, this tag: `<Seq-data>` signals the start of the `Seq-data` element. Likewise, this tag: `</Seq-data>` indicates the end of the element.
- Attributes appear within start element tags, and provide additional information about that element. For example, our document includes two attributes: *id* and *length*. Within BSML, the *id* attribute is used to uniquely identify an element within a document, and the *length* attribute is used to denote the number of base pairs or residues in a sequence.

Every XML document explicitly defines a document structure or element hierarchy. The element hierarchy for our sample document is shown in Figure 2.1. As you can see, `Bsml` is the roots element. The root element contains a `Definitions` element, which in turn contains a [`Sequences`] element. This element then contains a [`Sequence`] element, which in turn contains a [`Seq-data`]

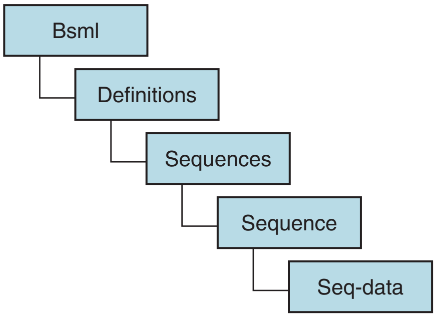
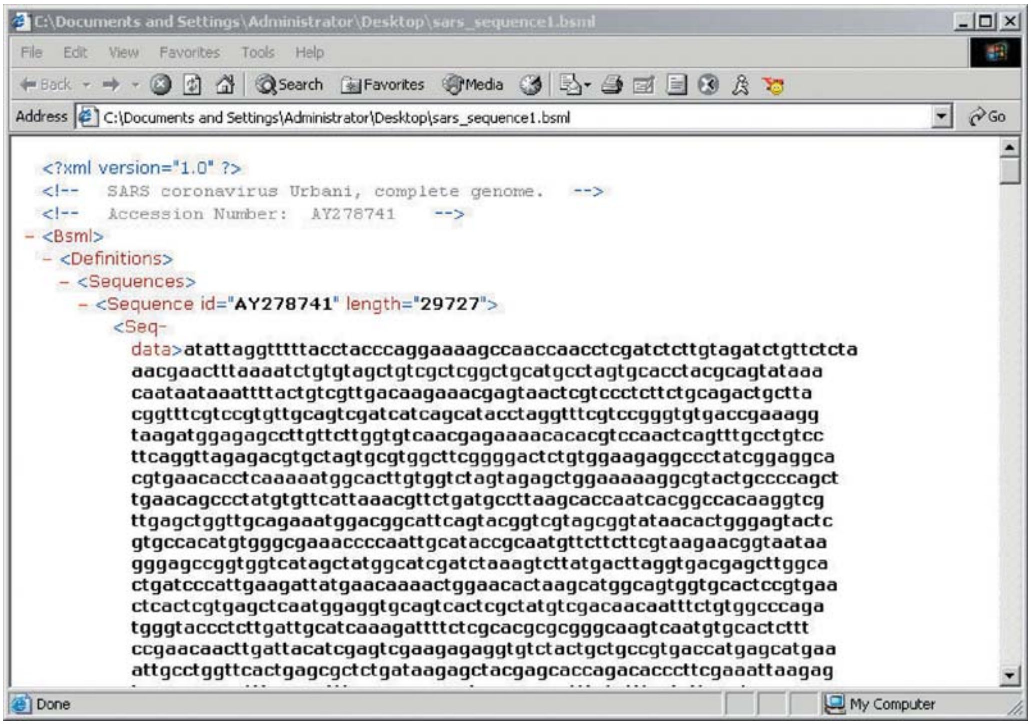


Figure 2.1 Element hierarchy of our first BSML document.


C:\Documents and Settings\Administrator\Desktop\sars_sequence1.bsml
File Edit View Favorites Tools Help
Back Search Favorites Media
Address C:\Documents and Settings\Administrator\Desktop\sars_sequence1.bsml
Go
Done
My Computer

```
 <?xml version="1.0" ?>
 <!--  SARS coronavirus Urbani, complete genome.  -->
 <!--  Accession Number:  AY278741   -->
- <Bsml>
  - <Definitions>
    - <Sequences>
      - <Sequence id="AY278741" length="29727">
          <Seq-
            data>atattaggtttttacctacccaggaaaagccaaccaacctcgatctcttgtagatctgttctcta
            aacgaactttaaaatctgtgtagctgtcgctcggctgcatgcctagtgcacctacgcagtataaa
            caataataaattttactgtcgttgacaagaaacgagtaactcgtccctcttctgcagactgctta
            cggtttcgtccgtgttgcagtcgatcatcagcatacctaggtttcgtccgggtgtgaccgaaagg
            taagatggagagccttgttcttggtgtcaacgagaaaacacacgtccaactcagtttgcctgtcc
            ttcaggttagagacgtgctagtgcgtggcttcggggactctgtggaagaggccctatcggaggca
            cgtgaacacctcaaaaatggcacttgtggtctagtagagctggaaaaaggcgtactgccccagct
            tgaacagccctatgtgttcattaaacgttctgatgccttaagcaccaatcacggccacaaggtcg
            ttgagctggttgcagaaatggacggcattcagtacggtcgtagcggtataacactgggagtactc
            gtgccacatgtgggcgaaaccccaattgcataccgcaatgttcttcttcgtaagaacggtaataa
            gggagccggtggtcatagctatggcatcgatctaaagtcttatgacttaggtgacgagcttggca
            ctgatcccattgaagattatgaacaaaactggaacactaagcatggcagtggtgcactccgtgaa
            ctcactcgtgagctcaatggaggtgcagtcactcgctatgtcgacaacaatttctgtggcccaga
            tgggtaccctcttgattgcatcaaagattttctcgcacgcgcgggcaagtcaatgtgcactcttt
            ccgaacaacttgattacatcgagtcgaagagaggtgtctactgctgccgtgaccatgagcatgaa
            attgcctggttcactgagcgctctgataagagctacgagcaccagacacccttcgaaattaagag
```

Figure 2.2 A sample screenshot of Internet Explorer. The first sample BSML document is shown. If the XML document does not reference a specific style sheet for transforming to HTML, Internet Explorer will apply a default style sheet. This default style sheet enables users to point and click their way through the element hierarchy. Clicking the + sign expands the element, revealing its direct descendents. Clicking the – sign collapses the element, hiding all its descendants.

element. Note that many BSML documents will have this same structure, and we will explore this structure in detail at the end of the chapter.

Many tools, including XML parsers and web browsers, provide complete access to the XML element hierarchy. For example, Internet Explorer provides an interactive display for browsing an XML document's structure. You can easily open and close nodes, and thereby show or hide specific branches of the element hierarchy. A sample screenshot is shown in Figure 2.2.

2.1.1 Using Genomic Workspace™

Viewing BSML documents within Internet Explorer certainly helps you understand and navigate the document structure, but it's hardly exciting. To appreciate the full power of BSML, it helps to have a BSML-aware browser. Rescentris, Ltd. provides such a browser in its Genomic Workspace™ software application. Genomic Workspace™ enables you to visually browse and interact with BSML documents. Visualization is provided by a number of specialized viewers, such as a hierarchical tree viewer, sequence viewer, sequence editor, and a multiple alignment viewer. In addition to these features, Genomic Workspace™ includes a data conversion and import utility. This enables you to import data in existing data formats, such as GenBank, Swiss-Prot, and EMBL file formats, and convert these records to BSML. This is a particularly useful feature for learning the full BSML specification.

Genomic Workspace™ is written in Java, and runs on most platforms, including Windows, Linux, and Mac OS. You can download a free copy from the Rescentris web site at: *http://www.rescentris.com*.

To explore BSML further, start Genomic Workspace™, and select File → Open, and select the example from Listing 2.1. You should now see a screen like the one shown in Figure 2.3. As you can see, the screen is divided into a number of sections. The main visual window in the center shows a snapshot of the sequence. If our sequence included annotations, such as the location of protein-coding regions, you would see these here too. However, since our example includes only

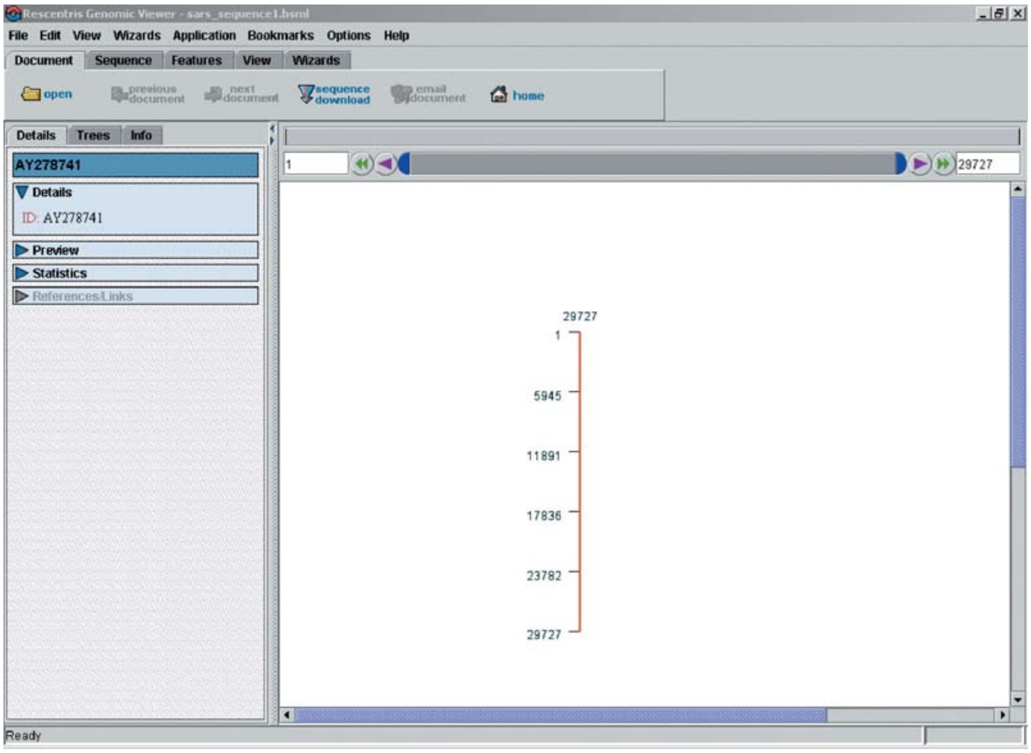

Rescentris Genomic Viewer - sars_sequence1.bsml
File Edit View Wizards Application Bookmarks Options Help
Document Sequence Features View Wizards
open previous document next document sequence download email document home
Details Trees Info
AY278741
Details
ID: AY278741
Preview
Statistics
References/Links
1
29727
29727
1
5945
11891
17836
23782
29727
Ready

Figure 2.3 Screenshot of the Rescentris Genomic Workspace™. First sample BSML document is shown.

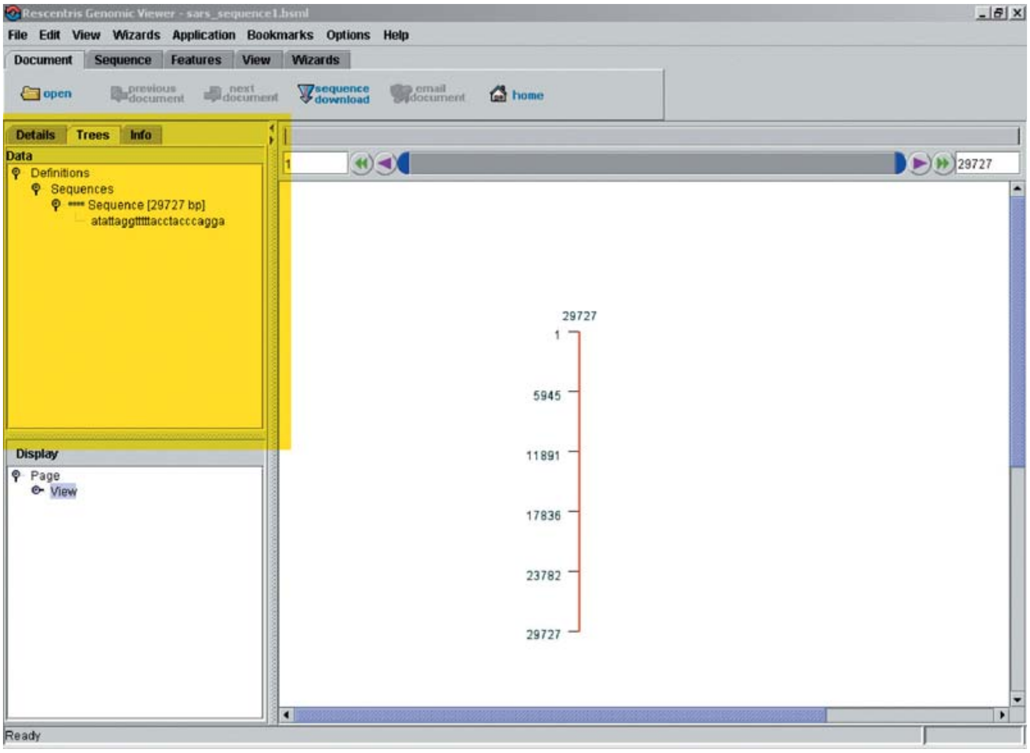


Figure 2.4 Screenshot of the Genomic Workspace™ Tree Viewer (shown highlighted). First sample BSML document is shown.

raw sequence data, we simply see a sequence widget with no annotation. The sequence widget begins at base pair 1 and ends at base pair 29,727. The navigation elements at the top of the main visual window, including the left and right arrows, enable you to zoom in and scroll through the sequence.

To the left of the main visual window, you will see three tabbed windows. By default, the "Details" tab is shown. We have not provided much information in our sample document, but you can see that the sequence ID and the sequence length are displayed. If you click on the Tree tab, you will see an interactive tree showing the complete element hierarchy. See Figure 2.4. Not surprisingly, the tree shown here matches the tree structure we saw earlier in Internet Explorer.

Next, let's explore the BSML Sequence Viewer. To access this, select View → Sequence → Sequence Viewer. Then, click the icon for "Zoom to Base Pair Level." You should now see a screen similar to that shown in Figure 2.5. As you can see, the 5′ to 3′ strand is shown, along with its complement. Below and above the strands are translation frames for amino acids. Just as in the main visual window, the sequence viewer includes navigation buttons for zooming in and scrolling through the entire sequence. Again, if this example included annotations, you would see these here. When we get to annotations at the end of the chapter, we will return to this view.

Hopefully, this gives you a taste of both BSML and the Genomic Workspace™. Once we explore the fundamentals of XML, we will return to both topics and explore them in more detail.

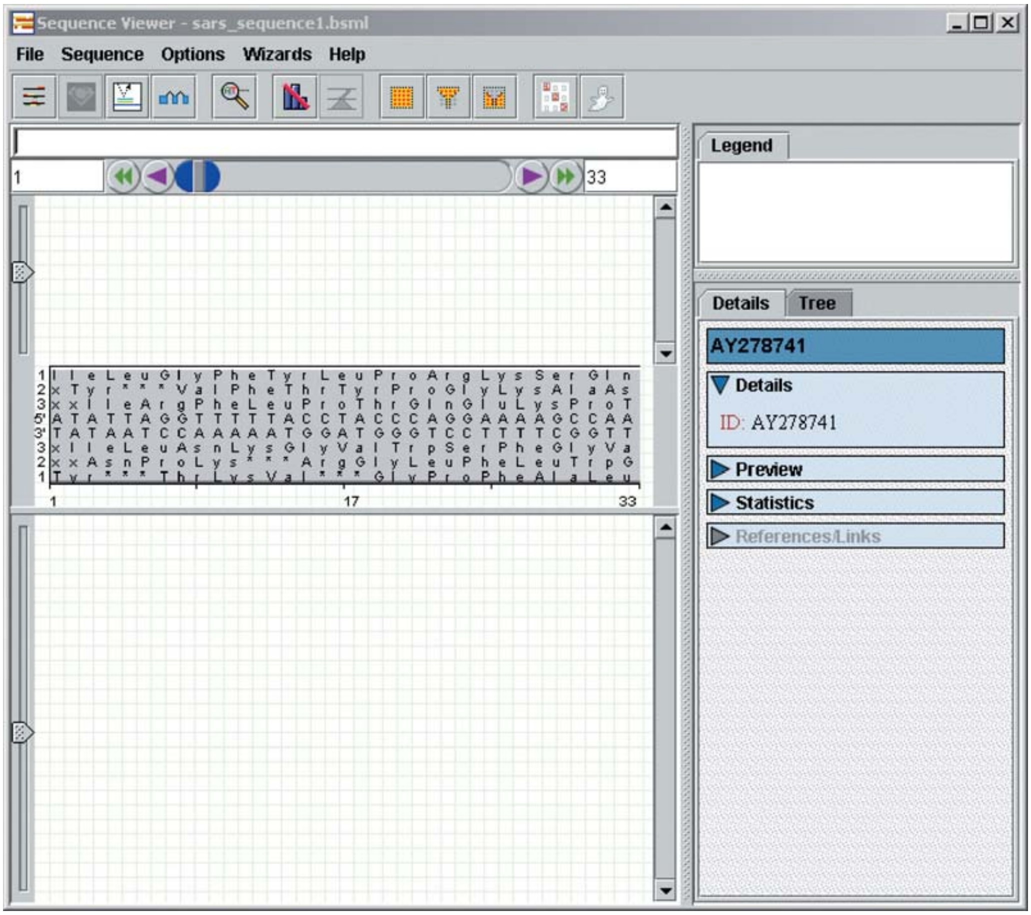


Figure 2.5 The Genomic Workspace™ sequence viewer. First sample BSML document is shown.

2.2 Fundamentals of XML

We now turn to the fundamental rules and concepts of XML. These rules apply regardless of XML application. For example, we could be dealing with e-commerce data, real estate listings, or genomics data. Some have argued that XML has grown in complexity, and that we are now inundated with too many XML specifications and XML protocols. This is certainly true, but if we stick to the core XML 1.0 specification, you may be surprised that there are only a handful of major concepts. Furthermore, the main rules for constructing XML documents are quite straightforward. We have made every effort to distill these concepts and rules into bite-size sections below.

2.2.1 Working with Elements

In its most basic form, an XML document consists of a set of elements. An element represents a discrete unit of data, such as a product listing, news headline, or biological sequence. With

XML, you can create elements for anything you want, and you are not restricted to a predefined list of elements. Furthermore, you can nest elements inside one another, and create any element hierarchy you like. For example, a product listing can include a description and a price, a news headline can include a title and a news category, and a sequence can contain references to scientific papers.

An XML element is formally defined with a start tag and a corresponding end tag. Start tags always take the form: `<ELEMENT_NAME>` , whereas end tags always take the form: `</ELEMENT_NAME>` . For example, the `Seq-data` element is defined with a start `<Seq-data>` tag and an end `</Seq-data>` tag. The complete element therefore looks like this:

```
<Seq-data>gcaggcgcagtgtgagcggcaacatggcgtccaggtc</Seq-data>
```

XML requires that every start tag must have a matching end tag. This is true even for empty XML elements. An empty element is one that does not contain any textual data or subelements, but may contain attributes. For example, the following empty element includes a cross-reference to the EMBL database:

```
<cross_reference database="EMBL" id="M29855"></cross_reference>
```

As a shortcut, you can specify empty elements with the more concise syntax: `<ELEMENT_NAME />` . For example:

```
<cross_reference database="EMBL" id="M29855"/>
```

XML has specific rules on naming XML elements. Specifically, element names must begin with a letter, an underscore character ("_"), or a colon character (":"). Names can then continue with letters, digits, hyphens, underscore, or colons. Names cannot begin with the letters "xml" or any case combination of "xml," as these are specifically reserved for use by the specification.

XML is also case sensitive. This is particularly important to remember when matching start tags with end tags. For example, the following example will result in an error:

```
<Seq-data>gcaggcgcagtgtgagcggcaacatggcgtccaggtc</SEQ-DATA>
```

In this example, `Seq-data` is not equal to `SEQ-DATA` , and the XML parser will report that the start tag is missing a matching end tag.

Every XML document must contain exactly one root element. This root element represents the entry point for traversing the entire element hierarchy. It is not legal to have more than one root element.

2.2.2 Working with Attributes

Attributes are used to provide additional information about a specific element. For example, you can specify *width* and *height* attributes for an HTML `img` element or you can add a *length* attribute to a BSML `Sequence` element. You can specify as many attributes for an element as you need, and they need not be placed in any specific order. Attributes are always placed within the start tag and never within the end tag.

XML requires that attribute values appear within quotes. You can use single quotes `(')` , or double quotes `(")` . For example, the following excerpt specifies an *id* attribute:

```
<Sequence id="AY064249">
    ...
</Sequence>
```

2.2.3 The XML Prolog

XML documents should (but are not actually required to) begin with an XML prolog. The XML prolog includes an XML declaration and an optional reference to a Document Type Declaration (DTD). The XML declaration specifies the XML version number and optional character encoding information. The declaration must begin with the characters: `<?xml` and end with the characters `?>` .

The current version of XML is 1.1, but many individuals (and all the examples in this book) continue to use XML 1.0. XML 1.1 does not represent a significant break from XML 1.0, and primarily focuses on character encoding issues. For example, it includes revised rules on including Unicode characters and expanding the set of end-of-line characters.

As a quick example, the following XML prolog specifies XML version 1.0:

```
<?xml version="1.0"?>
```

Details on character encoding will be covered later in this section.

2.2.4 Comments

XML comments begin with the characters: `<!--` and end with the characters: `-->` . Here is an example comment:

```
<!-- SARS coronavirus Urbani, complete genome. -->
```

Comments can span multiple lines, if needed. To maintain SGML compatibility, the character sequence “- -” is not permitted within comments.

2.2.5 Processing Instructions

Processing instructions are special XML directives, used to forward information to software applications. In certain scenarios, a single XML document may be processed by one or more software applications. This XML document may include processing instructions specifically directed at these applications, possibly providing important application parameters or other hints for processing.

Processing instructions must begin with the characters `<?` , and must end with the characters `?>` . Within these tags, a processing instruction consists of two parts:

- The first part is the software target. This indicates the target of the directive, usually specifying a specific software application or a specific type of software application.
- The second part is a list of one or more processing instructions. This can be any arbitrary text, but usually takes the form of name/value pairs, called pseudo-attributes.

Processing instructions are frequently used in XSL Transformations (XSLT). With XSLT, you can transform an XML document into another XML format or to an HTML document. XML documents use processing instructions to pass application parameters to XSLT parsers. Here is an example XSLT processing instruction:

```
<?xml-stylesheet type="text/xsl" href="bsml_to_html.xsl"?>
```

In the line above, we have specified a target value of "xml-stylesheet." We have also specified two name/value pairs. The first specifies the MIME type of the transformation document, and the second specifies the name of the specific XSLT template to use. In this case, we are using the BSML to HTML XSLT style sheet.

2.2.6 Character Encoding

As stated above, the XML declaration can include optional information about character encoding. The XML specification requires that all XML parsers support Unicode. Unicode is a character-encoding standard that provides support for most languages on Earth. This is in sharp contrast to ASCII (American Standard Code for Information Interchange), which supports only English or Latin characters. Unicode is made available by the Unicode Consortium, but is also officially endorsed by the ISO (International Organization for Standardization). The terms Unicode and ISO-10646 refer to the same standard.

XML parsers are required to support two specific encodings of Unicode/ISO-10646: UTF-16 and UTF-8. UTF-16 encodes Unicode characters using 16-bit characters. For text documents which primarily consist of ASCII characters, UTF-16 can result in inefficient storage and unnecessarily large documents. For these documents, it is more efficient to use the UTF-8 encoding schema. UTF-8 uses a few tricks to more compactly store Unicode characters. Specifically, ASCII characters are stored within one byte, and other characters are stored as multibyte sequences.

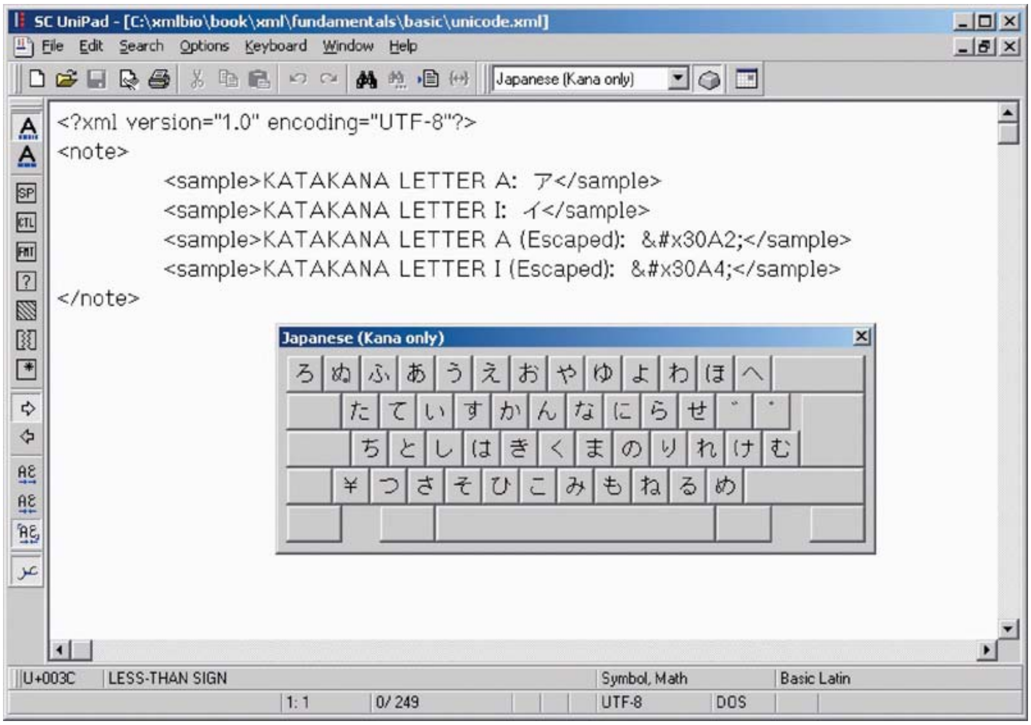
Within the XML declaration, you can use the *encoding* declaration to specify the character encoding of your XML document. For example, the following XML declaration specifies XML version 1.0 and UTF-8 character encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Besides UTF-8 and UTF-16, you can also specify other character encodings, such as one of the ISO-8859 family of character encodings. This includes Latin 1 (ISO 8859-1), which contains characters for English and most Western European languages; Latin 2 (ISO 8859-2), which contains character for most Eastern European languages; or Cyrillic (ISO 8859-5), which contains characters for Russian and Russian-influenced languages, such as Bulgarian and Macedonian.

If you plan to create XML documents, which make extensive use of Unicode characters, it helps to use a Unicode enabled editor. For example, for Windows platforms, you might consider using the excellent UniPad editor (*http://www.unipad.org*). UniPad comes with its own set of fonts, meaning that it works out of the box without having to install separate Windows system fonts. It also provides several easy options for inputting Unicode characters, including keyboard shortcuts and virtual keyboards. A screenshot of UniPad with a sample XML document (and the Japanese virtual keyboard) is shown in Figure 2.6.

If your document consists of just ASCII characters, and you want to include an occasional non-ASCII character, you can do so with a character escape sequence. Character sequences begin with: &#[followed by a decimal value] or &#x[followed by a hexadecimal value]. Escape sequences


SC UniPad - [C:\xmlbio\book\xml\fundamentals\basic\unicode.xml]
File Edit Search Options Keyboard Window Help
Japanese (Kana only)
U+003C LESS-THAN SIGN
Symbol, Math
Basic Latin
1: 1
0/ 249
UTF-8
DOS

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
          <sample>KATAKANA LETTER A:  ア</sample>
          <sample>KATAKANA LETTER I:  イ</sample>
          <sample>KATAKANA LETTER A (Escaped):  &#x30A2;</sample>
          <sample>KATAKANA LETTER I (Escaped):  &#x30A4;</sample>
</note>
```

Figure 2.6 UniPad in action. UniPad provides several options for inputting Unicode characters, including virtual keyboards. The Japanese (Katana only) keyboard is shown.

must end with a semicolon (";"). For example, the following escape code references the Japanese Kana letter A:

```
<sample>KATAKANA LETTER A (Escaped): &#x30A2;</sample>
```

Certain characters in XML have special importance, because they are used to denote XML markup. For example, the less than sign (<) is used as the first character for XML tags. If you want to use one of these reserved characters within element text, you must use its corresponding character escape sequence. There are only five reserved characters in XML, and each of these has a corresponding character escape sequence. These are defined as follows:

- &amp;   ampersand sign (&)
- &lt;   less than sign (<)
- &gt;   greater than sign (>)
- &apos;   apostrophe (')
- &quote; quote (")

2.2.7 CDATA Sections

Occasionally, you may want to escape an entire section of text. Text that is stored within a CDATA section is preserved exactly as it is. Reserved characters, such as the less than sign (<), which would normally be interpreted as markup characters, are no longer interpreted as such. This can be useful if you want to include sample XML or HTML markup examples within your XML document. CDATA sections must begin with the characters `<![CDATA[` , and must end with the characters `]]>` .

Here is a sample XML document with a CDATA section:

```
<note>
    <section>
    <![CDATA[
        In XML, start tags always take the  form: <ELEMENT_NAME>.
    ]]>
    </section>
</note>
```

Without the CDATA section, this XML document would result in an error; specifically, an XML parser would complain that the `<ELEMENT_NAME>` element was missing a corresponding end tag. However, because this text is actually contained within a CDATA section, the parser knows to ignore all the markup characters and preserve the text as it is.

2.2.8 Creating Well-Formed XML Documents

XML has very strict requirements on what constitutes a legal XML document. If an XML document meets these specific requirements, it is said to be *well-formed*.

To be well-formed, an XML document must meet the following requirements:

- Every start tag must have a corresponding end tag. The only exception to this rule is the empty element tag syntax, e.g., `<ELEMENT_NAME/>` .
- Elements must be properly nested. In other words, a subelement must have its start and end tags defined within the scope of the parent element. For example, this example is nested properly and therefore well-formed:

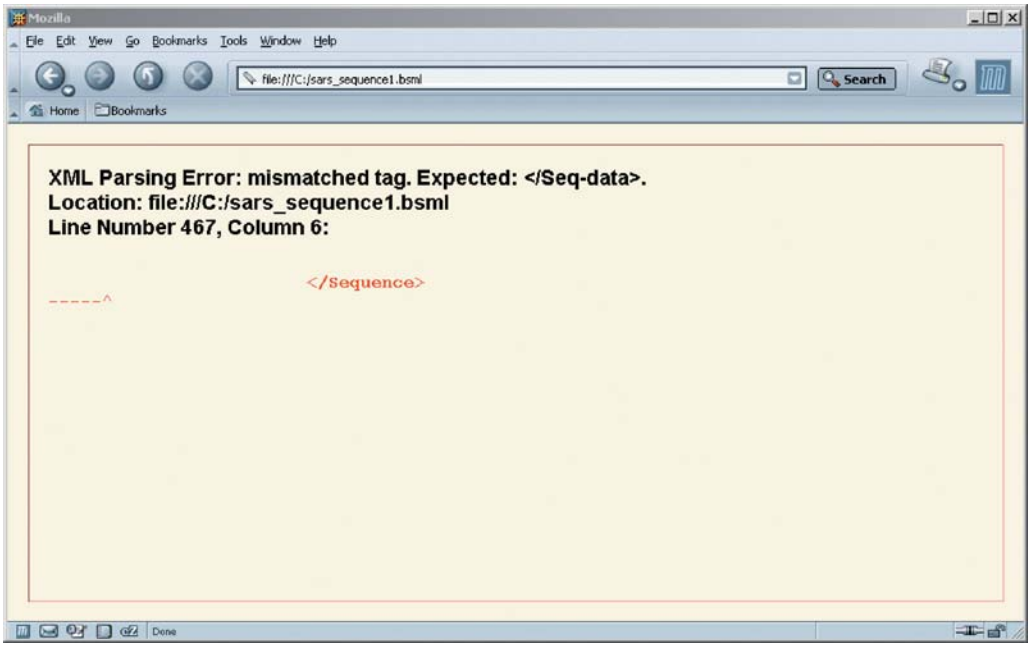
  ```
  <Sequence>
      <Seq-data>atggcgtccaggtctaagcggcgtgccgtg</Seq-data>
  </Sequence>
  ```

  However, this example is not property nested, and therefore not well-formed:

  ```
  <Sequence>
      <Seq-data>atggcgtccaggtctaagcggcgtgccgtg
  </Sequence>
  </Seq-data>
  ```

- All attribute values must appear within quotes.
- Every XML document must have exactly one root element.
- Reserved characters, such as the less than sign, are always treated as markup. If they appear on their own, they must be specified with character escape sequences, or placed within a CDATA section.

There are lots of XML editors and command line tools, which can test your XML documents for well-formedness. We explore some of these tools in the next chapter. However, you may have a convenient XML tool already loaded on your machine. In fact, if you have one of the more current


Mozilla
File Edit View Go Bookmarks Tools Window Help
file:///C:/sars_sequence1.bsml
Search
Home
Bookmarks
Done

```
XML Parsing Error: mismatched tag. Expected: </Seq-data>.
Location: file:///C:/sars_sequence1.bsml
Line Number 467, Column 6:

                        </Sequence>
-----^
```

Figure 2.7 Mozilla Web Browser in action. Upon loading an XML document, the built-in parser will automatically check the document for well-formedness and immediately report any errors.

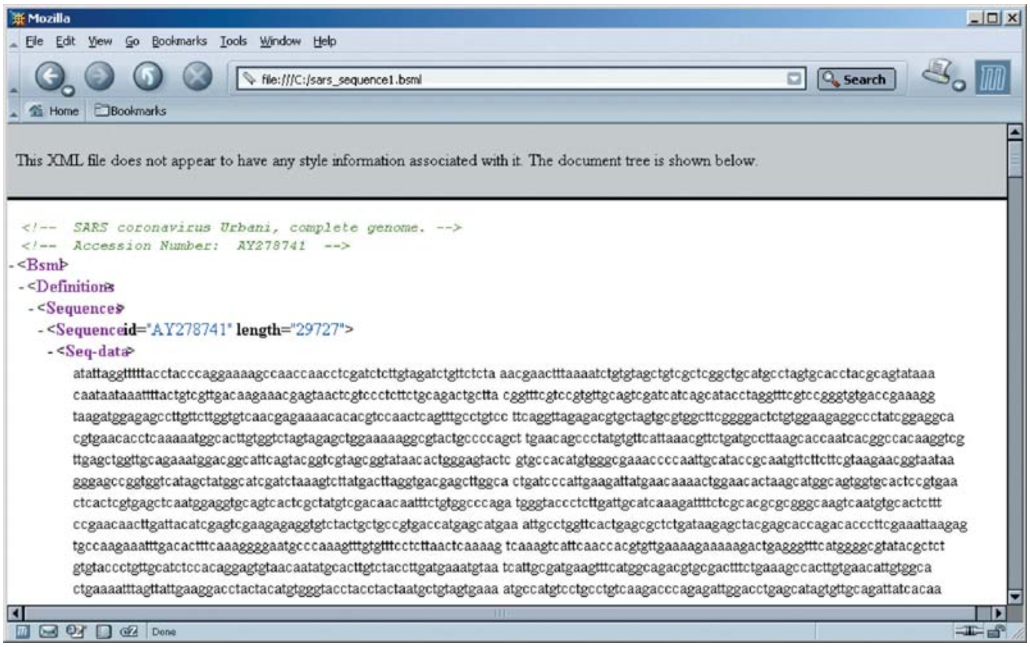
web browsers, such as Internet Explorer 6.0 or Mozilla 1.4 or later, both of these now include lots of built-in XML features and a built-in XML parser. When these browsers load an XML document, the internal XML parser will automatically check for well-formedness and report any errors. For example, Figure 2.7 shows a screenshot of the Mozilla browser. We have just opened a sample XML document, and it immediately reports a missing end tag:

```
XML Parsing Error: mismatched tag. Expected: </Seq-data>.
```

It also reports the exact location of the error. If no errors are encountered, Mozilla uses a default style sheet to render the XML document with a simple tree view. See Figure 2.8.

2.2.9 Creating Valid XML Documents

An XML grammar defines rules for creating XML documents. For example, the BSML specification is actually a grammar, and this grammar spells out specific rules for creating BSML documents. For example, it defines that a `Sequences` element can contain one or more `Sequence` elements, and that the `Sequence` element contains a *length* attribute. If we know that a document adheres to a specific grammar, we already know what type of data we are dealing with, and we can accurately predict the extract structure of the document. We can therefore build applications that are specifically designed to consume specific types of XML documents. Furthermore, if multiple documents adhere to the same grammar, we can process all these documents using the same software application. We can even create new software applications, which aggregate data from multiple disparate sources.


Mozilla
File Edit View Go Bookmarks Tools Window Help
file:///C:/sars_sequence1.bsml
Search
Home
Bookmarks
Done

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<!--  SARS coronavirus Urbani, complete genome. -->
<!--  Accession Number:  AY278741  -->
- <Bsml>
 - <Definitions>
  - <Sequences>
   - <Sequence id="AY278741" length="29727">
    - <Seq-data>
        atattaggtttttacctacccaggaaaagccaaccaacctcgatctcttgtagatctgttctcta aacgaactttaaaatctgtgtagctgtcgctcggctgcatgcctagtgcacctacgcagtataaa
        caataataaattttactgtcgttgacaagaaacgagtaactcgtccctcttctgcagactgctta cggtttcgtccgtgttgcagtcgatcatcagcatacctaggtttcgtccgggtgtgaccgaaagg
        taagatggagagccttgttcttggtgtcaacgagaaaacacacgtccaactcagtttgcctgtcc ttcaggttagagacgtgctagtgcgtggcttcggggactctgtggaagaggccctatcggaggca
        cgtgaacacctcaaaaatggcacttgtggtctagtagagctggaaaaaggcgtactgccccagct tgaacagccctatgtgttcattaaacgttctgatgccttaagcaccaatcacggccacaaggtcg
        ttgagctggttgcagaaatggacggcattcagtacggtcgtagcggtataacactgggagtactc gtgccacatgtgggcgaaaccccaattgcataccgcaatgttcttcttcgtaagaacggtaataa
        gggagccggtggtcatagctatggcatcgatctaaagtcttatgacttaggtgacgagcttggca ctgatcccattgaagattatgaacaaaactggaacactaagcatggcagtggtgcactccgtgaa
        ctcactcgtgagctcaatggaggtgcagtcactcgctatgtcgacaacaatttctgtggcccaga tgggtaccctcttgattgcatcaaagattttctcgcacgcgcgggcaagtcaatgtgcactcttt
        ccgaacaacttgattacatcgagtcgaagagaggtgtctactgctgccgtgaccatgagcatgaa attgcctggttcactgagcgctctgataagagctacgagcaccagacacccttcgaaattaagag
        tgccaagaaatttgacactttcaaaggggaatgcccaaagtttgtgtttcctcttaactcaaaag tcaaagtcattcaaccacgtgttgaaaagaaaaagactgagggtttcatggggcgtatacgctct
        gtgtaccctgttgcatctccacaggagtgtaacaatatgcacttgtctaccttgatgaaatgtaa tcattgcgatgaagtttcatggcagacgtgcgactttctgaaagccacttgtgaacattgtggca
        ctgaaaatttagttattgaaggacctactacatgtgggtacctacctactaatgctgtagtgaaa atgccatgtcctgcctgtcaagacccagagattggacctgagcatagtgttgcagattatcacaa
```

Figure 2.8 Mozilla Web Browser in action (continued). If Mozilla does not find any errors in well-formedness, it will use a default style sheet to display your XML document. In other words, if you see this view, you can be certain that your XML document is in fact well-formed.

There are two main types of XML grammars: Document Type Definitions (DTDs) and XML Schemas. DTDs have been around since the very beginning of XML and are formally specified within the W3C XML 1.0 specification. XML Schemas are a newer specification and provide considerably more features than DTDs. For example, XML Schema supports data typing and enables you to specify that certain elements can only contain integer or float values. With XML Schema, you can also specify regular expression patterns and require that certain elements match those patterns.

We discuss both DTDs and XML Schemas in detail in the next two chapters. For now, we only want to cover one essential point: document *validity*. A document that adheres to all the critical rules in XML is said to be *well-formed*. A document that adheres to all the rules of a specific grammar is said to be *valid*. This is a critical distinction, and one that we will explore many times in the next few chapters.

For now, a very simple example should make the distinctions very clear. First, consider this sample document:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Bsml PUBLIC "-//Labbook, Inc. BSML DTD//EN"
"http://www.labbook.com/dtd/bsml3_1.dtd">
<Bsml>
   <Definitions>
      <Sequences>
          <Sequence id="AY064249" length="1245" molecule="rna">
               <Seq-data>gcaggcgcagtgtgagcggcaacatggcg....
```

```
            </Sequence>
        </Sequences>
    </Definitions>
</Bsml>
```

See any problems here? The end `</Seq-data>` tag is missing. The document is therefore not well-formed. To fix this problem, we simply add the end tag, like this:

```
<Seq-data>gcaggcgcagtgtgagcggcaacatggcg....</Seq-data>
```

Now, consider this sample document:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Bsml PUBLIC "-//Labbook, Inc. BSML DTD//EN"
"http://www.labbook.com/dtd/bsml3_1.dtd">
<Bsml>
   <Definitions>
       <Sequences>
           <Dna id="AY064249" length="1245">
               <Seq-data>gcaggcgcagtgtgagcggcaacatggcg....</Seq-data>
           </Dna>
       </Sequences>
    </Definitions>
</Bsml>
```

See any problems here? All the start tags have matching end tags, everything is nested properly, and all attribute values appear in quotes. It is therefore well-formed. However, you can now see that the `Sequences` element contains a `Dna` element. This may seem just fine, but the BSML grammar does not actually specify a `Dna` element. Therefore, this document does not follow all the rules of the BSML grammar and is considered invalid.

How do we actually know that BSML does not specify a `Dna` element? This is the topic that we explore in great detail in the next two chapters. For now, understand that there is a fundamental difference between well-formedness and validity. To recap, we define these two terms below:

- Well-formed: a document is said to be well-formed if it follows all the main rules defined by the XML specification. For example, every start tag must have a matching end tag, elements must be properly nested, and all attribute values must appear within quotes.
- Valid: a document is said to be valid if it follows all the rules of the referenced XML grammar.

A document can be well-formed, but invalid. This means that the document follows all the main XML rules, but fails to follow the rules of the XML grammar.

2.2.10 Working with XML Parsers

An XML parser (or XML processor) is responsible for parsing an XML document and making its contents available to a calling application. Specific responsibilities include: retrieving XML documents from a local file system or from a network connection, checking to make sure that the document is well-formed, and making the contents of the document available via a standard Application Programming InterFace (API). If you have lots of time to spare, you could, of course, write your own XML parser. However, this may not make the best use of your time! A much more convenient

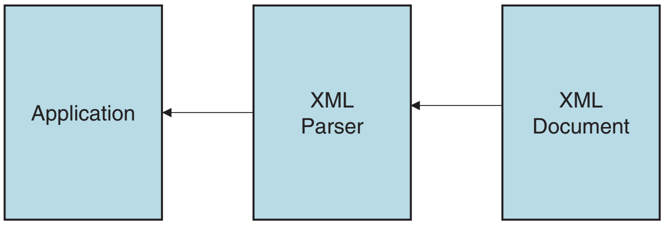


Figure 2.9 A typical XML application consists of three distinct layers.

option is to find an existing XML parser, and plug this into your application. XML parsers are freely available for dozens of programming languages, including C, C++, C#, Java, Perl, and Python.

A typical XML application consists of three distinct layers, see Figure 2.9. Working from right to left, the first layer is an XML document or a set of XML documents. These documents contain useful information, which you want to extract; for example, the documents may contain useful BSML data that you want to analyze further. The second layer is the XML parser. The parser consumes XML documents and makes the content available to the third layer, which is your software application. The XML parser takes care of all XML specific details and enables your application to more easily focus on content and programming logic.

XML parsers are broadly divided into two types:

- validating parser: this parser is capable of validating a document against an XML grammar, such as a DTD or an XML Schema.
- nonvalidating parser: this parser is not capable of validating a document against an XML schema.

As a general rule of thumb, nonvalidating parsers tend to be faster and take up less memory. However, validating parsers tend to be more useful, as you can use them to validate documents, and you don't need to include any validation code within your software application.

We will explore XML parsers in great detail in later sections of this book.

2.3 Fundamentals of XML Namespaces

XML Namespaces were not defined in the original W3C XML 1.0 specification. However, the namespace specification was finalized soon after, and namespaces are now considered a crucial element in the XML family of protocols. They are also a critical building block for other XML specifications, including XML Schemas, XSL Transformations, SOAP, and the Web Service Description Language (WSDL). In this section, we explore why XML Namespaces are important and then describe the mechanics of declaring and using namespaces.

2.3.1 Why We Need XML Namespaces

XML Namespaces are designed to address two very specific issues. First, namespaces prevent name conflicts. If your XML document references a single DTD or XML Schema, this is never an issue. However, if your document references two or more XML grammars, you have the potential for name conflicts. For example, two DTDs might define a `Sequence` element. XML Namespaces lets you attach a namespace to each `Sequence` element, and therefore uniquely identify each element.

Second, namespaces enable you to mark certain elements for processing by a specific software application. From the software module perspective, an XML document consists of actionable elements and nonactionable elements. By filtering for elements from a specific namespace, the software module can determine which elements are actionable and take the appropriate action.

In practice, name conflicts do not actually occur that often. For example, if your document references two grammars, the chance that they both define the same element is small. This is not to minimize name conflicts. It is just to point out that the second scenario of the software module perspective is more common. Hence, let's dig a little deeper into this scenario.

First, consider the following XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<stylesheet version="1.0">
     <template match="/">
         <html>
             <body>
             <h1>BSML Sequence Data:</h1>
             <value-of select="Bsml/Definitions/Sequences/Sequence"/>
             </body>
         </html>
     </template>
</stylesheet>
```

This is an example XSLT document. The document consists of two sets of elements. The first set consists of XSLT specific instructions. For example, `stylesheet` , `template` , and `value-of` are all XSLT instructions. The second set consists of HTML elements. For example, `html`, `body` , and `h1` are all HTML elements. An XSLT application will consume this document and apply the XSLT transformations. In this specific example, the style sheet is responsible for transforming BSML documents into HTML.

From the XSLT software module perspective, it needs an easy way to identify which elements are XSLT and which are not. In other words, it needs an easy way to determine which elements are actionable and which are nonactionable. As the document exists right now, the elements are not clearly partitioned.

Now, consider this XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/
  XSL/Transform">
    <xsl:template match="/">
          <html>
               <body>
               <h1>BSML Sequence Data:</h1>
                     <xsl:value-of select="Bsml/Definitions/Sequences/
                       Sequence"/>
               </body>
           </html>
     </xsl:template>
</xsl:stylesheet>
```

This document now contains an XML namespace declaration for XSLT. Furthermore, all XSLT elements now have an *xsl* prefix. For example, the `template` element is now defined as `xsl:template` . From the software module perspective, it is now a trivial task to determine which elements are XSLT instructions and which are not. It can therefore more easily carry out the XSL transformation.

The "Namespaces in XML" specification [14; 15] is currently available as an official W3C Recommendation. The complete specification is available online at: *http://www.w3.org/TR/REC-xml-names/*.

2.3.2 Declaring and Using XML Namespaces

Now that you understand the rationale for namespaces, let's look into the mechanics of declaring and using XML namespaces.

To use an XML namespace, you must first declare it. XML namespace declarations can occur within any XML element, but in practice, most developers place them at the top of their document usually within the root XML element. A namespace declaration is scoped to the element wherein the declaration occurs and all its subelements.

An XML declaration is a special XML attribute consisting of three parts. The first part is the reserved prefix *xmlns*. The second part is a namespace prefix of your choosing, and the third part is a Uniform Resource Identifier (URI). For example, the following element declares a namespace for XSLT:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
  Transform">
```

The namespace prefix serves as a shortcut to the namespace declaration. You can use whatever namespace prefix you like. However, there are a few common conventions. For instance, the XSLT prefix is usually specified as *xsl* and the XML Schema prefix is usually specified as *xs* or *xsd*.

The URI value serves as a unique identifier and enables you or a software module to unambiguously partition elements into discrete namespaces. Values are most often represented as absolute URLs, e.g., *http://www.w3.org/1999/XSL/Transform*. If you are creating your own namespace, you should have control over the referenced host or URL. Otherwise, you may not be able to ensure absolute uniqueness.

It is important to note that the URI value does not necessarily point to anything meaningful. For example, if you copy and paste a namespace URI value into a web browser, you may or may not find a meaningful resource there. Therefore, the URI value serves as a unique identifier and nothing more.

Having declared a namespace, you later reference the namespace via a *Qualified Name*. A Qualified Name consists of two parts: a namespace prefix and a local element name. The two parts are delimited with a colon character. For example, the following start tag now includes a Qualified Name:

```
<xsl:template match="/">
```

In plain English, this start tag now references the `template` element in the *xsl* namespace. We already know that every start tag must have a matching tag. In this case, the end tag must also include a Qualified Name:

```
</xsl:template>
```

The complete XSLT example above should now make a lot more sense. It is repeated below:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
  Transform">
    <xsl:template match="/">
        <html>
          <body>
          <h1>BSML Sequence Data:</h1>
          <xsl:value-of select="Bsml/Definitions/Sequences/Sequence"/>
          </body>
        </html>
    </xsl:template>
</xsl:stylesheet>
```

We now know that the root element contains a namespace declaration for XSLT. We also know that all XSLT elements are specified with Qualified Names. All other elements, e.g., `html` , `body`, and `h1`, are not namespace qualified and therefore, do not exist within any namespace. Also note that the *select* attribute in the `xsl:value-of` element does not exist in any namespace either. To place an attribute within a namespace, you must explicitly specify it with a Qualified Name. For example:

```
<xsl:value-of xsl:select="Bsml/Definitions/Sequences/Sequence"/>
```

Now, both the element and the attribute share the same namespace.

2.3.3 Declaring a Default Namespace

The XML Namespaces specification supports default namespaces. A default namespace applies to the element where the declaration occurs and all its subelements. All unqualified elements within this scope are assumed to be part of the default namespace.

Default namespaces are specified with the special *xmlns* attribute—this is a special case of the namespace declaration defined above, except that there is no namespace prefix. For example, the following declares a default namespace for the XHTML specification:

```
<html xmlns="http://www.w3.org/1999/xhtml">
```

All unqualified subelements will therefore belong to the XHTML namespace. Here is a slightly longer example:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/
  Transform" xmlns="http://www.w3.org/1999/xhtml">
    <xsl:template match="/">
        <html>
            <body>
            <h1>BSML Sequence Data:</h1>
            <xsl:value-of select="Bsml/Definitions/Sequences/
              Sequence"/>
            </body>
        </html>
    </xsl:template>
</xsl:stylesheet>
```

The root `stylesheet` element now contains two namespace declarations. The first is for XSLT; the second is a default namespace for XHTML. All elements beginning with the *xsl* prefix are explicitly defined to exist within the XSLT namespace. All unqualified elements, e.g., `html` , `body` , and `h1` , now exist within the default XHTML namespace.

Note that default namespaces do not apply to attributes. If you want an attribute to exist within a specific namespace, you must always specify it with a Qualified Name.

2.4 Fundamentals of BSML

As stated at the beginning of the chapter, BSML is an open standard for representing and exchanging bioinformatics sequence data. Since its inception, BSML has grown to accommodate a wide range of bioinformatics data. This now includes:

- raw sequence data, including the ability to reference sequence data stored in other external data files. Sequences can also be represented at several levels, including at the individual sequence record level, chromosome level, and whole genome level.
- sequence annotation, enabling you to attach positional and nonpositional sequence features, such as coding regions, promoter sequences, Single Nucleotide Polymorphisms (SNPs), etc.
- scientific literature references, including the ability to reference full journal citations, along with Pub Med identifiers.
- networks of biological entities, enabling you to encode metabolic and signaling pathways.
- multiple sets of tabular data, enabling you to encode gene expression or Microarray data.
- display widgets, used to store visual representations of sequences. This includes the ability to store image captions, draw sequence features, and reference external GIFs and JPEGs.
- resource information, enabling you to store information about individual investigators, research organizations, and copyright availability.

BSML was originally created by Visual Genomics and was first funded in 1997 by the National Human Genome Research Institute (NHGRI). The goal of the initial NHGRI grant was to develop a standard for representing sequence data in XML, and to release the standard to the public domain. Joseph H. Spitzner, Ph.D. was the primary author of BSML at Visual Genomics. Spitzner continued work on BSML while working at LabBook, Inc., and now works at Rescentris, Ltd. BSML is currently available as a Document Type Definition (DTD), but the data model is at least partially based on preexisting data formats, including the GenBank ASN.1 file format.

The main BSML web site at: *http://www.bsml.org* includes an FAQ, an introductory tutorial, and a complete reference guide.

As this book goes to press, BSML is currently available as version 3.1. Since its original release, a number of organizations have announced support for BSML, including Bristol-Meyers Squibb, IBM, Accelrys, Inc., and the European Bioinformatics Institute (EBI). A number of other organizations have also released BSML conversion programs. For example, the Cold Spring Harbor Laboratory has released a utility for converting GenBank ASN.1 sequence data to BSML. The EBI has also released a utility for converting European Molecular Biology Laboratory (EMBL) documents to BSML.

BSML is not the only XML format for representing sequence data. In fact, there are several alternatives to BSML, including the NCBI DTDs, the Architecture for Genomic Annotation, Visualization and Exchange (AGAVE), Genome Annotation Markup Elements (GAME), and Biopolymer Markup Language (BioML).

As stated in the introduction, the BSML specification is quite large and we do not have the space to explore the specification in full. Instead, we will focus on the core elements, and on the representation of sequences and sequence features.

2.4.1 BSML File Formats

The BSML specification recommends three file extensions for use with BSML. These are defined in Table 2.1.

2.4.2 BSML Document Structure

Every BSML document shares properties and a similar structure. The first property is that every BSML document must begin with an XML prolog and must include a reference to the BSML DTD. Every BSML document will therefore begin like this:

```
<?xml version="1.0"?>
<!DOCTYPE Bsml PUBLIC "-//Labbook,  Inc. BSML DTD//EN"
"http://www.rescentris.com/dtd/bsml3_1.dtd">
```

In the next chapter, we will discuss the exact mechanics of referencing DTDs. For now, note that we are referencing the BSML 3.1 DTD, available on the Rescentris.com web site.

Second, every BSML document must begin with a root `Bsml` element. Following the root element, BSML is divided into three main sections.

- **Definitions**: this section stores biological sequences and sequence annotations. The section can also include tables of associated data and network graphs.
- **Research**: this section stores information about experimental research, such as experimental conditions, program queries, or search parameters. For example, you can store query parameters for a specific BLAST search.
- **Display**: this section stores display widgets and references external image files. This section is primarily used by software applications that are capable of visually rendering sequence data. For example, the Rescentris Genomic Workspace^{TM} application uses this section to store visual representations of sequences and their features.

Table 2.1 BSML file extensions

File Extension	Description
*.bsml	A regular BSML file
.bsmz	This is a gzipped archive of one or more BSML documents, along with related resources, such as external data files and images
.bso	This is a BSML overlay file. An overlay file consists of BSML fragments, which can be overlaid onto an existing BSML document. For example, two researchers can swap overlay files for a base sequence record, and more easily exchange and compare sequence annotations

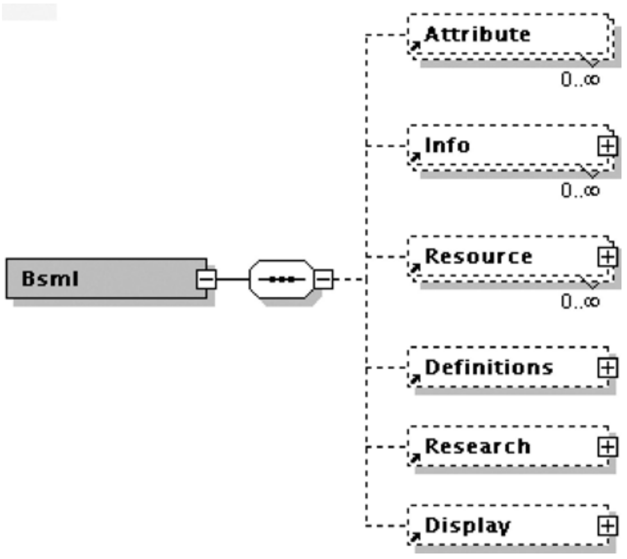


Figure 2.10 A bird's-eye view of the BSML DTD. The first level of elements is shown. (Document was created with XML Spy®.)

For a bird's-eye view of the BSML document structure, refer to Figure 2.10. As you can see, the root of the document structure is specified with a `Bsml` element. Under this, you have three common elements: `Attribute` , `Info` , and `Resource` . The `Attribute` element is used to store arbitrary name/value pairs which do not fit into any other elements. For example:

```
<Attribute name="definition" content="SARS coronavirus Urbani,
  complete genome."/>
```

Most BSML elements can contain 0 or more `Attribute` elements, providing a catch-all category for any data that doesn't fit the existing BSML data model. Furthermore, the BSML `Info` element can be used to store sets of `Attribute` elements.

The BSML `Resource` element is used to store metadata about the BSML document. In general, metadata elements are used to store "data about data." For example, a web page can contain metadata that describes the author, description, keywords, and the date last updated. The BSML metadata elements are based on the Dublin Core Metadata Initiative [19; 22; 24], one of the most popular metadata standards, used primarily to describe web pages and XML documents. Dublin Core is a minimal specification consisting of just 15 basic elements, such as Title, Description, Creator, Rights, and Date. By using `Resource` elements, you can therefore add important metadata to your BSML documents and record authorship, organizational affiliation, and copyright availability. Furthermore, since most BSML elements contain a `Resource` element, you can even add metadata to specific portions of the document. For example, you can record who made a specific sequence annotation and when they made it.

For more information about the Dublin Core Metadata Initiative, see: *http://www.dublincore.org*. The web site includes a number of relevant documents, including a guide to "Using Dublin Core" and a list of "Guidelines for implementing Dublin Core in XML."

Table 2.2 Text/binary formats for storing raw sequence data. For use with the BSML seq-data-import element

Format Name	Description
IUPACna	One-letter IUPAC codes for nucleic acids (see Appendix A for IUPAC codes)
IUPACaa	One-letter IUPAC codes for amino acids (see Appendix A for IUPAC codes)
NCBI2na	NCBI compact binary representation for nucleic acids. Each nucleic acid is represented with just 2 bits. Does not allow for ambiguity or gaps
NCBI4na	NCBI moderately compact binary representation for nucleic acids. Each nucleic acid is represented with 4 bits. Does include provisions for ambiguity and gaps

2.4.3 Representing Sequences

Sequence elements represent the heart of any BSML document. As we have already seen, these elements are used to store raw sequence data and can also be used to store annotations about the raw sequence. For example, we can add positional and nonpositional sequence features. Within this section, we consider the mechanics of representing raw sequences. In the next section, we move on to discuss sequence features.

The first important detail to note is that all sequence data must appear with the BSML `Definitions` section. This section contains a `Sequences` element, which can contain any number of `Sequence-import` or `Sequence` elements. `Sequence-import` elements are used to reference sequence data stored within other BSML files. For example, consider the following document fragment:

```
<Definitions>
   <Sequences>
         <Sequence-import source="sars_sequence1.bsml" id="AY278741"/>
   </Sequences>
</Definitions>
```

This document references sequence id=AY27841 in the sars_sequence1.bsml file.

In contrast to the `Sequence-import` element, the `Sequence` element is used to define sequence data within a BSML document. However, even in this case, the actual raw sequence data can be stored within the BSML document itself or within an external text or binary file. To represent raw sequence data, use the `Seq-data` element. To import data from an external text or binary file, use the `Seq-data-import` element. When importing data, you must specify a *source* attribute specifying the location of the file, and a *format* attribute specifying the text/binary format (see Table 2.2 for details). For example, consider the following document excerpt:

```
<Definitions>
    <Sequences>
            <Sequence id="AY278741" length="29727">
                   <Seq-data-import format="IUPACaa" source="
                     sars_sequence.txt"/>
            </Sequence>
     </Sequences>
</Definitions>
```

Table 2.3 Main attributes of the `Sequence` element

Attribute Name	Description
comment	Usually used to indicate a displayable description of the sequence record. See also the *title* attribute
db-source	Used to identify a public database, such as GenBank, EMBL, or the DNA Database of Japan (DDBJ). See also the *ic-acckey*
ic-acckey	An accession number used to uniquely identify a sequence record within the international consortium of nucleotide sequence databases. The consortium consists of GenBank, the EMBL Nucleotide Sequence Database, and the DNA Database of Japan (DDBJ). This attribute is usually used in conjunction with the *db-source* attribute.
length	Indicates the length of the sequence
local-acckey	An accession number used to uniquely identify a sequence record within a local or private database
molecule	Indicates the type of molecule represented. Options include: "dna," "rna," "aa" (amino acid), "na" (nucleic acid), "other-mol," and "mol-not-set." If you do not specify a molecule attribute, it defaults to "dna"
title	A displayable name for the sequence record. See also the *comment* attribute
topology	Specifies the topology of the sequence. Usually indicated with the values "linear" or "circular"

In this case, we are defining a new sequence for the same SARS virus as Listing 2.1, but specifying that the actual sequence data is stored in an external text file.

When using the `Seq-data` element, you must stick to IUPAC codes for nucleic acids and amino acids (see Appendix A). However, the data can include white space characters and numbers. For example, the following document excerpt is considered valid:

```
<Definitions>
    <Sequences>
            <Sequence id="AY278741" length="29727">
                   <Seq-data>
    1 atattaggtt tttacctacc caggaaaagc caaccaacct cgatctcttg tagatctgtt
    61 ctctaaacga actttaaaat ctgtgtagct gtcgctcggc tgcatgccta gtgcacctac
    121 gcagtataaa caataataaa ttttactgtc gttgacaaga aacgagtaac tcgtccctct
    181 tctgcagact gcttacggtt tcgtccgtgt tgcagtcgat catcagcata cctaggtttc
            [For brevity, sequence is truncated.]
                   </Seq-data>
            </Sequence>
     </Sequences>
     </Definitions>
</Bsml>
```

Each `Sequence` element can include a number of attributes. The main attributes are defined in Table 2.3. A more complete example of the SARS virus, along with more fully detailed attributes, is also provided in Listing 2.2.

2.4.4 Representing Sequence Features

In addition to raw sequence data, BSML can also represent sequence features. A sequence feature is any piece of annotation that provides additional details regarding a specific location or range of sequence data. When we get to Chapter 6, we will spend more time formally defining sequence annotation, and discuss in detail the Distributed Annotation System (DAS). For now, it is simplest to think of sequence annotation as any piece of data that provides additional details regarding a raw sequence record. For example, we can take a raw sequence record, and identify important parts, such as promoter regions, protein-coding regions, and 5′ and 3′ untranslated regions. We can also annotate sequence records with important references to scientific articles. Sequence features

Listing 2.2 The SARS virus, Take 2. This example is identical to Listing 2.1, except that we have now added additional attributes.

```
<?xml version="1.0"?>
<!DOCTYPE Bsml PUBLIC "-//Labbook, Inc. BSML DTD//EN"
"http://www.rescentris.com/dtd/bsml3_1.dtd">
<Bsml>
  <Definitions>
     <Sequences>
       <Sequence id="AY278741" title="AY278741" molecule="rna"
          length="29727" db-source="GenBank" ic-acckey="AY278741"
          topology="linear" strand="ss" representation="raw">
          <Attribute name="definition" content="SARS coronavirus
            Urbani, complete genome."/>
          <Attribute name="submission-date" content="21-APR-2003"/>
          <Attribute name="version" content="AY278741.1 GI:30027617"/>
          <Attribute name="source" content="SARS coronavirus Urbani"/>
          <Seq-data>
          atattaggtttttacctacccaggaaaagccaaccaacctcgatctcttgtagatctgtt
          ctctaaacgaactttaaaatctgtgtagctgtcgctcggctgcatgcctagtgcacctac
          gcagtataaacaataataaattttactgtcgttgacaagaaacgagtaactcgtccctct
          tctgcagactgcttacggtttcgtccgtgttgcagtcgatcatcagcatacctaggtttc
          gtccgggtgtgaccgaaaggtaagatggagagccttgttcttggtgtcaacgagaaaaca
          cacgtccaactcagtttgcctgtcc
          [For brevity, sequence is truncated.]
          </Seq-data>
        </Sequence>
     </Sequences>
   </Definitions>
</Bsml>
```

are an important element in other file formats as well. For example, the GenBank Flat File Format includes extensive support for sequence features and includes a recommended list of feature types.

In BSML, each sequence can contain any number of features. Features are formally nested within a `Feature-tables` element and individual features are defined within a `Feature` element. Two types of features are supported: positional and nonpositional. Positional features are tied to specific sequence locations and can be used to represent a host of sequence annotations, including protein-coding regions, locations of predicted genes, single nucleotide polymorphisms (SNPs), etc. Nonpositional features are not tied to any specific region of sequence, but are instead associated with the sequence record as a whole. For example, you can attach literature references that are associated with the entire sequence record.

Nonpositional features are slightly less complex than positional features. Let's take a look at an example, shown in Listing 2.3. This new example adds a single nonpositional feature detailing the direct submission to GenBank. More specifically, it lists the primary contributors of the work and their affiliation with the Centers for Disease Control and Prevention. As you can see, the `Reference` element contains a list of authors, a title, and the complete journal reference. For references to published material, you can include cross-reference identifiers to MEDLINE and PubMed.

Listing 2.3 The SARS virus, Take 3. The record now includes a single nonpositional feature, describing the direct submission to GenBank.

```
<?xml version="1.0"?>
<!DOCTYPE Bsml PUBLIC "-//Labbook, Inc . BSML DTD//EN"
"http://www.rescentris.com/dtd/bsml3_1.dtd">
<Bsml>
  <Definitions>
     <Sequences>
       <Sequence id="AY278741" title="AY278741" molecule="rna"
        length="29727" db-source="GenBank" ic-acckey="AY278741"
        topology="linear" strand="ss" representation="raw">
         <Attribute name="definition" content="SARS coronavirus
          Urbani, complete genome."/>
         <Attribute name="submission-date" content="21-APR-2003"/>
         <Attribute name="version" content="AY278741.1 GI:30027617"/>
         <Attribute name="source" content="SARS coronavirus Urbani"/>
         <Feature-tables id="AY278741.FTS1">
           <Feature-table id="AY278741.FTS1.FTB1" title="Genbank
             References" class="GB_REFERENCES">
            <Reference id="REF1" title="Direct Submission">
              <RefAuthors>Bellini,W.J., Campagnoli,R.P.,
                 Icenogle,J.P., Monroe,S.S., Nix,W.A., Oberste,M.S.,
                 Pallansch,M.A. and Rota,P.A.
               </RefAuthors>
               <RefTitle>Direct Submission</RefTitle>
               <RefJournal>Submitted (17-APR-2003) Division of Viral
               and Rickettsial Diseases, Centers for Disease Control
               and Prevention, 1600 Clifton RD, NE, Atlanta, GA
               30333, USA</RefJournal>
             </Reference>
           </Feature-table>
        </Feature-tables>
       <Seq-data>
       atattaggtttttacctacccaggaaaagccaaccaacctcgatctcttgtagatctgttctc
       taaacgaactttaaaatctgtgtagctgtcgctcggctgcatgcctagtgcacctacgcagta
       taaacaataataaattttactgtcgttgacaagaaacgagtaactcgtccctcttctgcagac
       tgcttacggtttcgtccgtgttgcagtcgatcatcagcatacctaggtttcgtccgggtgtga
       ccgaaaggtaagatggagagccttgttcttggtgtcaacgagaaaacacacgtccaactcagt
       ttgcctgtcc
        [For brevity, sequence is truncated.]
       </Seq-data>
     </Sequence>
   </Sequences>
 </Definitions>
</Bsml>
```

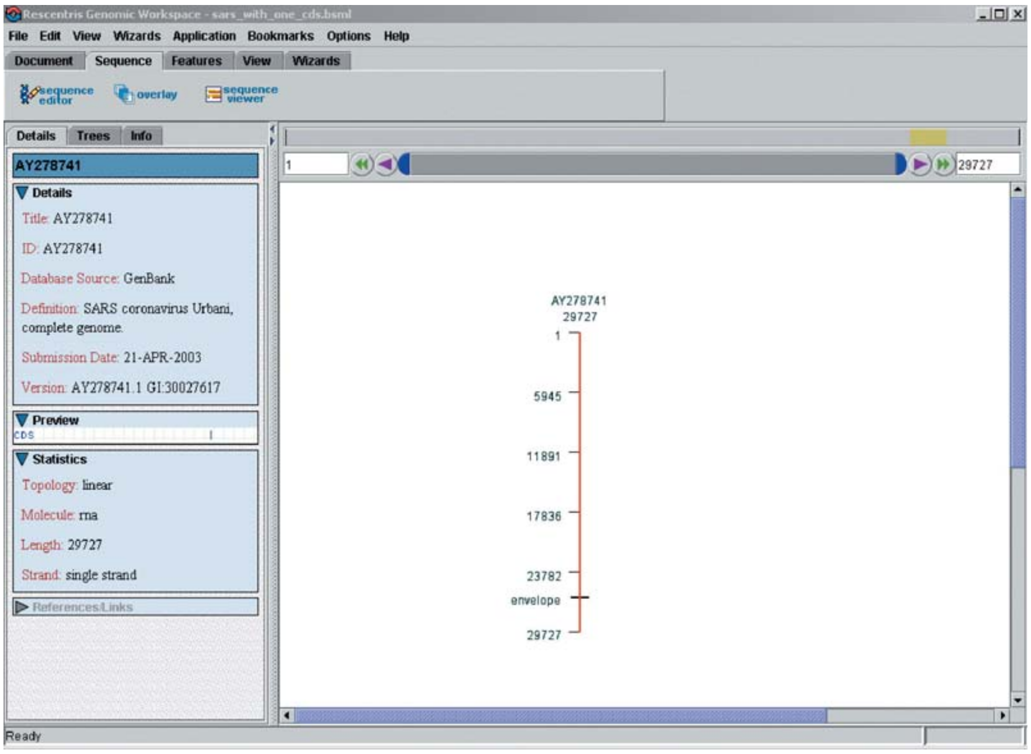
Positional features are just slightly more complicated. Each feature can contain any number of *Qualifier* and location elements. A *Qualifier* element describes a name/value attribute that describes the feature. A location element describes the location of the feature. Two types of locations can be specified: *Site-loc* and *Interval-loc*. A *Site-loc* identifies a single point within a raw sequence; an

Listing 2.4 SARS virus, Take 4. The record now includes a single positional feature.

```
<?xml version="1.0"?>
<!DOCTYPE Bsml PUBLIC "-//Labbook, Inc. BSML DTD//EN"
"http://www.rescentris.com/dtd/bsml3_1.dtd">
<Bsml>
  <Definitions>
    <Sequences>
      <Sequence id="AY278741" title="AY278741" molecule="rna"
         length="29727" db-source="GenBank" ic-acckey="AY278741"
         topology="linear" strand="ss" representation="raw">
         <Attribute name="definition" content="SARS coronavirus
          Urbani, complete genome."/>
         <Attribute name="submission-date" content="21-APR-2003"/>
         <Attribute name="version" content="AY278741.1 GI:30027617"/>
         <Attribute name="source" content="SARS coronavirus Urbani"/>
         <Feature-tables  id="AY278741.FTS1">
         <Feature-table id="AY278741.FTS1.FTB2" title="Genbank
            Features" class="GB_FEATURES">
         <Feature id="AY278741.FTS1.FTB2.FTR9" title="envelope
            protein" class="CDS" comment="envelope protein"
            display-auto="1">
         <Interval-loc startpos="26117" endpos="26347"/>
         <Qualifier value-type="note" value="envelope protein"/>
         <Qualifier value-type="codon_start" value="1"/>
         <Qualifier value-type="product" value="E protein"/>
         <Qualifier value-type="protein_id" value="AAP13443.1"/>
         <Qualifier value-type="db_xref" value="GI:30027622"/>
         <Qualifier value-type="translation" value="MYSFVSEETGTLIVNSVL
           LFLAFVVFLLVTLAILTALRLCAYCCNIVNVSLVKPTVYVYSRVKNLNSSEGV
           PDLLV"/>
          </Feature>
        </Feature-table>
      </Feature-tables>
    <Seq-data>
    atattaggtttttacctacccaggaaaagccaaccaacctcgatctcttgtagatctgttctcta
    aacgaactttaaaatctgtgtagctgtcgctcggctgcatgcctagtgcacctacgcagtataaa
    caataataaatttactgtcgttgacaagaaacgagtaactcgtccctcttctgcagactgctta
    cggtttcgtccgtgttgcagtcgatcatcagcatacctaggtttcgtccgggtgtgaccgaaagg
    taagatggagagccttgttcttggtgtcaacgagaaaacacacgtccaactcagtttgcctgtcc
     [For brevity, sequence is truncated.]
    </Seq-data>
   </Sequence>
  </Sequences>
 </Definitions>
</Bsml>
```

Interval-loc identifies a specific interval or range of raw sequence data. Again, a specific example should clarify the most important points. Take a look at Listing 2.4.

If you download the full SARS virus genome record from GenBank, you will see that it includes dozens of features. However, to keep the example more manageable, we have chosen to just include one positional feature in Listing 2.4. As you can see, this feature identifies a single coding sequence



Title: AY278741
ID: AY278741
Database Source: GenBank
Definition: SARS coronavirus Urbani, complete genome.
Submission Date: 21-APR-2003
Version: AY278741.1 GI:30027617
Topology: linear
Molecule: rna
Length: 29727
Strand: single strand

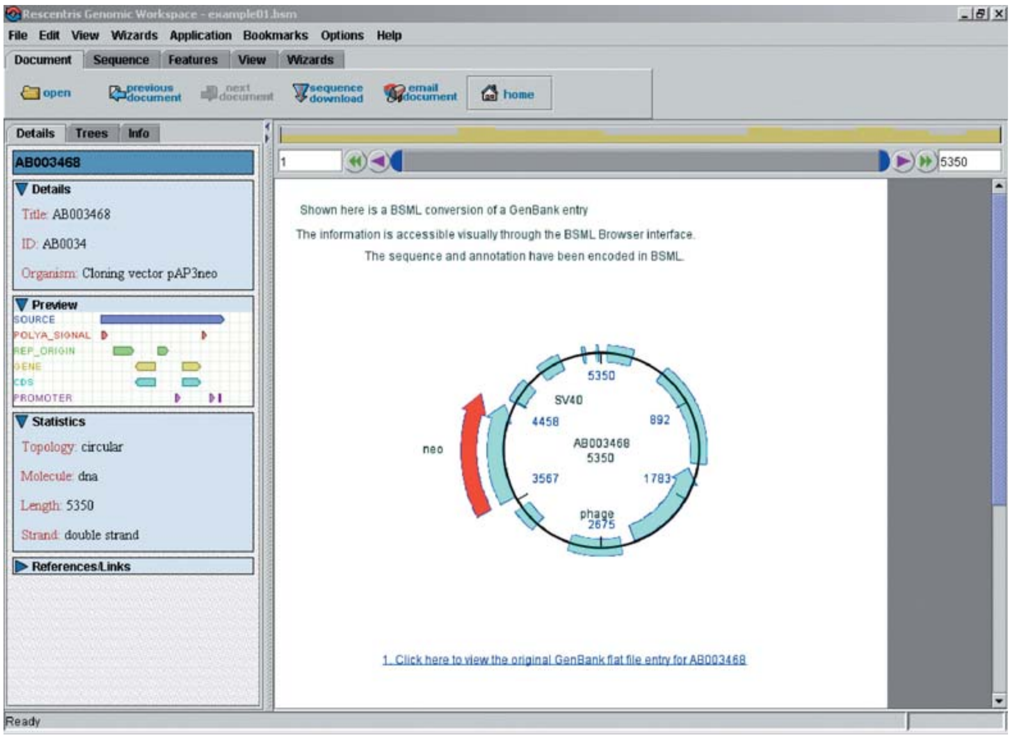
Figure 2.11 Sample screenshot of the Rescentris Genomic Workspace^{TM} application. We have just loaded the SARS example from Listing 2.4. Note that our envelope protein is now included in the main sequence window (it is denoted with a single line between the markers 23,782 and 29,727).

region, identifying the SARS virus envelope protein. The coding region spans a specific interval of sequence data and we therefore use the `Interval-loc` element:

```
<Interval-loc startpos="26117"  endpos="26347"/>
```

As stated above, each feature can include any number of `Qualifier` elements. In this case, we use `Qualifier` elements to denote important attributes. For example, we identify the protein ID, a cross-reference to the protein GI number in GenBank, and the amino acid sequence of the translated region.

The Rescentris Genomic Workspace^{TM} application will automatically draw all sequence features for you. For example, Figure 2.11 shows a screenshot of our revised SARS example. As you can see, our single feature is overlaid onto the main sequence widget in the center of the screen. Of course, this is one of the simplest possible feature examples. If you import a fully annotated sequence with multiple features, Genomic Workspace^{TM} will draw all these features for you as well. You can then interactively select specific features and drill down to an increased level of detail. For example, Figure 2.12 shows a screenshot of one of the sample BSML files that comes bundled with the viewer. All features are displayed around the perimeter of the main circular sequence widget. If you select one of the features in the main window, detailed feature information is immediately displayed in the "Details" panel on the left.



AB003468

Details

Title: AB003468

ID: AB0034

Organism: Cloning vector pAP3neo

Preview

Statistics

Topology: circular

Molecule: dna

Length: 5350

Strand: double strand

References/Links

Shown here is a BSML conversion of a GenBank entry

The information is accessible visually through the BSML Browser interface.

The sequence and annotation have been encoded in BSML.

1. Click here to view the original GenBank flat file entry for AB003468

Figure 2.12 Sample screenshot of the Rescentris Genomic Workspace™ application. A fully annotated BSML sequence is shown.

You may have noticed that the `Feature` element contains a *display-auto* attribute. When set to "1," this provides a hint to the BSML rendering software that you want to automatically display the feature with a separate graphical widget. For example, Genomic Workspace™ uses this information to automatically render and visualize all BSML files.

In BSML 3.1, you can explicitly denote that some features span multiple regions. For example, you can specify all the exons for a protein-coding sequence. To do so, you must specify a *join* attribute, and set it to "1." Following this, the first `Interval-loc` specifies the complete range of the sequence, and each subsequent `Interval-loc` element specifies a specific subrange of data. For example, the following excerpt describes a protein-coding sequence with three exons:

```
<Feature-table>
    <Feature id="sample_protein" class="CDS" display-auto="1"
       join="1">
        <Interval-loc startpos="100" endpos="400"/>
        <Interval-loc startpos="120" endpos="150"/>
        <Interval-loc startpos="190" endpos="210"/>
        <Interval-loc startpos="300" endpos="400"/>
    </Feature>
</Feature-table>
```

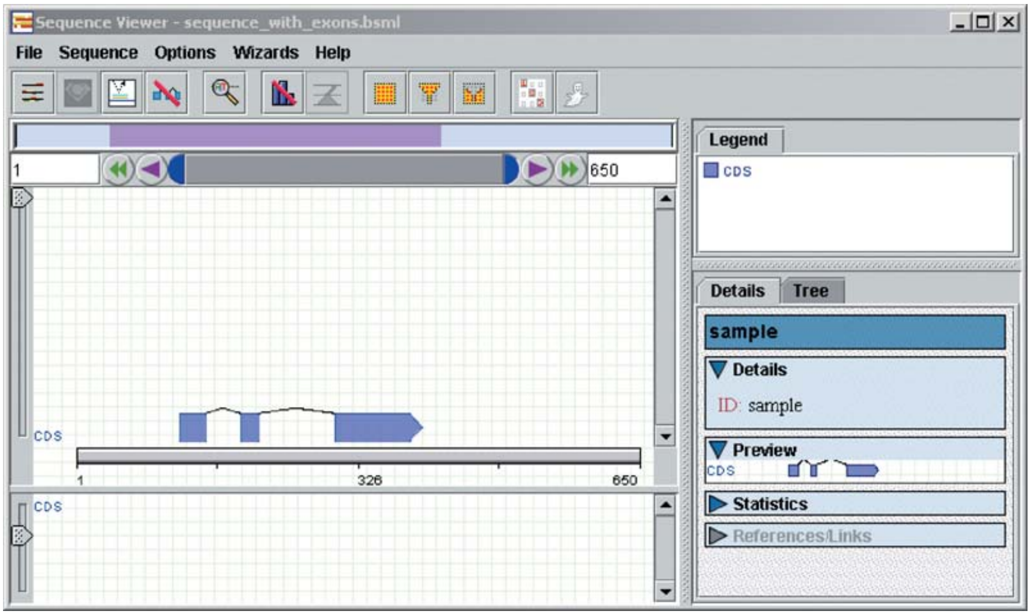


Figure 2.13 Sample screenshot of the Genomic Workspace™ Sequence Viewer. A sample protein-coding sequence with three exons is shown.

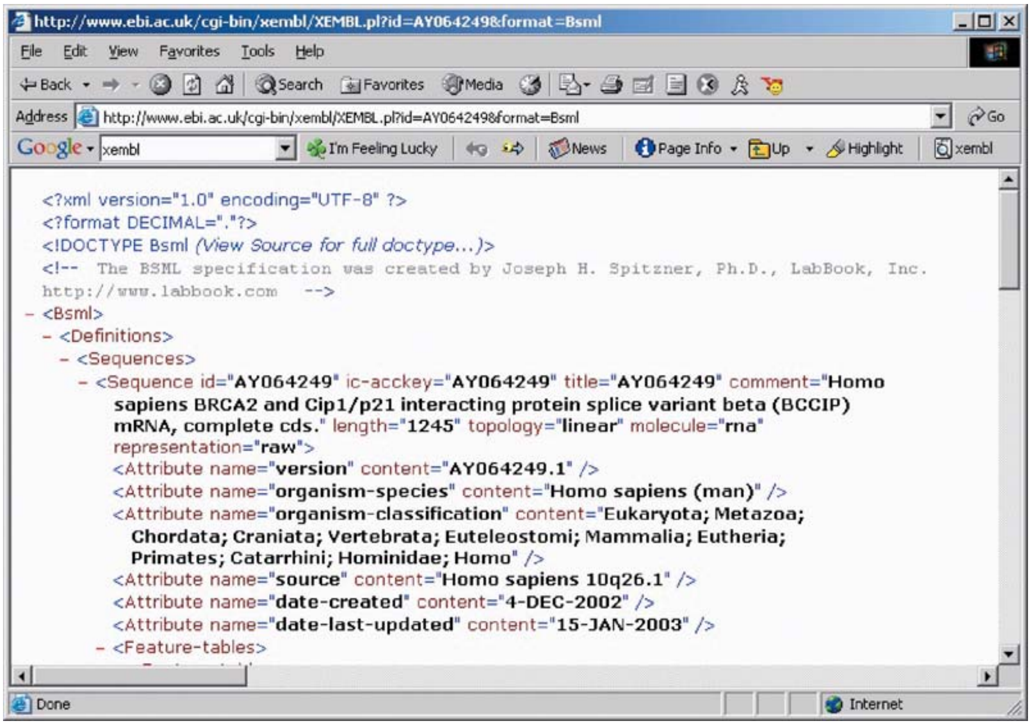
Within Genomic Workspace™ Sequence Viewer you can then choose to explicitly draw all individual exons. A sample screenshot is shown in Figure 2.13.

Genomic Workspace™ includes a number of utilities for importing existing data and converting it to BSML. It also includes functionality for searching and importing data directly from public databases, search as GenBank, Ensembl, Swiss-Prot, and EMBL. To get started, select Wizards → Import and follow the onscreen instructions.

2.4.5 Retrieving Live BSML Data via XEMBL

Before ending our discussion of BSML, we will take a quick tour of the XEMBL service, from the European Bioinformatics Institute. XEMBL provides complete access to the EMBL Nucleotide Sequence Database. This database is produced in collaboration with GenBank and the DNA Database of Japan, and currently provides access to millions of nucleotide sequence records. It also provides access to completed genomes, including the human genome, the fruit fly, and *C. elegans*.

XEMBL [26] is a recently released interface that provides easy XML access to the complete EMBL database. Access is provided via two main methods. The first is a URL interface whereby users specify parameters within a URL and XEMBL returns a complete XML document. The second


http://www.ebi.ac.uk/cgi-bin/xembl/XEMBL.pl?id=AY064249&format=Bsml
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://www.ebi.ac.uk/cgi-bin/xembl/XEMBL.pl?id=AY064249&format=Bsml Go
Google xembl I'm Feeling Lucky News Page Info Up Highlight xembl
Done Internet

```
  <?xml version="1.0" encoding="UTF-8" ?>
  <?format DECIMAL="."?>
  <!DOCTYPE Bsml (View Source for full doctype...)>
  <!-- The BSML specification was created by Joseph H. Spitzner, Ph.D., LabBook, Inc.
  http://www.labbook.com   -->
- <Bsml>
  - <Definitions>
    - <Sequences>
      - <Sequence id="AY064249" ic-acckey="AY064249" title="AY064249" comment="Homo
          sapiens BRCA2 and Cip1/p21 interacting protein splice variant beta (BCCIP)
          mRNA, complete cds." length="1245" topology="linear" molecule="rna"
          representation="raw">
          <Attribute name="version" content="AY064249.1" />
          <Attribute name="organism-species" content="Homo sapiens (man)" />
          <Attribute name="organism-classification" content="Eukaryota; Metazoa;
            Chordata; Craniata; Vertebrata; Euteleostomi; Mammalia; Eutheria;
            Primates; Catarrhini; Hominidae; Homo" />
          <Attribute name="source" content="Homo sapiens 10q26.1" />
          <Attribute name="date-created" content="4-DEC-2002" />
          <Attribute name="date-last-updated" content="15-JAN-2003" />
        - <Feature-tables>
```

Figure 2.14 XEMBL in action.

is a formal web services interface that uses the SOAP protocol and the Web Services Description Language (WSDL). (For details on SOAP, refer to Chapter 9.)

The XEMBL services expect two main parameters: an accession ID and a format. The ID specifies a unique international accession code; for example, SC49845 specifies the AXL2 gene in *Saccharomyces cerevisiae*. The format indicates the XML format of the returned document. Two format options are currently supported: BSML and AGAVE (Architecture for Genomic Annotation, Visualization and Exchange). To retrieve data in BSML format, you must specify format=Bsml; to retrieve data in AGAVE format, you must specify format=sciobj.

The XEMBL home page is available at: *http://www.ebi.ac.uk/xembl*.

You don't need any special tools or toolkits to access the XEMBL URL interface. All you need is a web browser. Simply start your browser, enter the main XEMBL URL and append the ID and format parameters. For example, the URL *http://www.ebi.ac.uk/cgi-bin/xembl/XEMBL.pl?id=AY064249&format=Bsml* retrieves the complete AY064249 record in BSML format. A sample screenshot of the XEMBL response is shown in Figure 2.14.

BSML is currently available as version 3.1. However, as this book goes to press, XEMBL is currently using BSML 2.2.

2.5 Useful Resources

Articles and Tutorials

- *Bioinformatic Sequence Markup Language—BSML 3.1 Tutorials.* LabBook, Inc.

  Provides a general introduction to the Bioinformatic Sequence Markup Language (BSML). Tutorial is available in MS Word and PDF formats at: *http://www.bsml.org/Resources/default.asp.*

- Cibulskis Kristian, "An Introduction to BSML." *XML J.* 4(3).

  Provides a concise introduction to BSML. Article is available online at: *http://www.sys-con.com/xml/archivesa.cfm?volume=04&Issue=03.*

- L. Wang, J. J. Riethoven, and A. Robinson, "XEMBL: distributing EMBL data in XML format." *Bioinformatics* 2002; 18(8): 1147–1148.

  Provides a short description of the XEMBL web service. Includes a discussion of the supported XML formats, and the CORBA back-end. Article can be downloaded from the *Bioinformatics* web site at: *http://bioinformatics.oupjournals.org.*

Web Sites and Web Services

- BSML web site: *http://www.bsml.org*

  Official home of the Bioinformatic Sequence Markup Language. The web site includes a short BSML Overview, an FAQ, a BSML Tutorial, and the official BSML Reference Manual.

- Rescentris, Ltd.: *http://www.rescentris.com*

  Official home of the Rescentris Genomic Workspace^{TM} BSML Viewer.

- Unicode web site: *http://www.unicode.org*

  Official home of the Unicode specification. The site includes an introduction to Unicode, FAQ, and Glossary. Code charts in PDF format are available at: *http://www.unicode.org/charts.*

- XEMBL web service: *http://www.ebi.ac.uk/xembl*

  XEMBL provides XML access to the complete European Molecular Biology Laboratory (EMBL) Nucleotide Sequence Database. BSML and AGAVE formats are currently supported.

- The XML FAQ: *http://www.ucc.ie:8080/cocoon/xmlfaq*

  This is an excellent resource for those new to XML. It includes answers to dozens of questions, including: "What is XML?" "Where can I get an XML Browser?" "Does XML Replace HTML?"

XML Specifications

- XML 1.0 Specification: *http://www.w3.org/TR/REC-xml*

When in doubt, head to the official specification. The specification can be difficult to digest on the first reading. For excellent commentary and jargon-free side notes, check out Tim Bray's Annotated XML Specification at: *http://www.xml.com/axml/testaxml.htm.*

- Namespaces in XML: *http://www.w3.org/TR/REC-xml-names*

  The namespace specification is only 12 pages long and highly readable. The introductory section, "Motivation and Summary," is well worth the read.