Java Speech Markup Language Specification

Version 0.5 — August 28, 1997

The Java™ Speech Markup Language (JSML) is used by applications to annotate text input to Java Speech API speech synthesizers. The JSML elements provide a speech synthesizer with detailed information on how to say the text. JSML includes elements that describe the structure of a document, provide pronunciations of words and phrases, and place markers in the text. JSML also provides prosodic elements that control phrasing, emphasis, pitch, speaking rate, and other important characteristics. Appropriate markup of text improves the quality and naturalness of the synthesized voice. JSML uses the Unicode character set, so JSML can be used to mark up text in most languages of the world.

Sun microsystems

THE NETWORK IS THE COMPUTER™

Contents

-

Notes to Reviewers

This document describes the *Java™ Speech Markup Language* (JSML) and explains how it can be used to annotate text input to Java Speech API speech synthesizers. This specification is an extract from the Java Speech Application Programming Interface (JSAPI) specification that will be released later in 1997. When the full specification is released, the Java Speech Markup Language specification will be included as part of the programming guide.

Review Comments

We are very interested in your input concerning the *Java Speech Markup Language* specification. Send your comments to:

```
javaspeech-comments@sun.com
```

Please be sure to include the version number and date of the document you are reviewing with your comments. We anticipate releasing a small number of updates to our documentation during the review period. These updates will incorporate responses to comments. The earlier we receive your feedback, the more likely it will be taken into consideration for the next update.

Because of the high level of interest in the Java Speech API, the Java Speech Grammar Format and the Java Speech Markup Language, we are unable to respond directly to individual comments or questions, but we will carefully read and evaluate all of the input we receive.

JSML and JSAPI

This specification for the *Java Speech Markup Language* describes a textual representation of input to a speech synthesizer but does not address the issues listed below. These programmatic issues are covered in the documentation for the Java Speech API, which is expected to be released later in 1997.

- Mechanisms for providing marked-up text to a synthesizer.
- Software control of the output of annotated text such as queuing, pause and resume, and variation of pitch and speaking rate.
- Mechanisms for receiving notification of synthesis events including marker events requested in JSML texts.
- Error handling capabilities including incorrect markup.
- Vocabulary management issues such as provision of pronunciations.

Issues for this Release

Many aspects of the *Java Speech Markup Language* are fully specified. However, some areas are still under development. Reviewers are especially encouraged to provide feedback in these areas.

Specification Issues

The following areas in the *Java Speech Markup Language* are not fully defined:

- A formal syntax and Document Type Definition (DTD) for JSML
- An attribute for structural elements for selection of speaking voices

Plans for Future Releases

Sun and its partners are developing new capabilities and features that will appear in a future release of the *Java Speech Markup Language* specification. Features that we are considering for future releases include:

- Intonational phrase patterns for `SENT`, `PARA`, and other elements
- Attribute for specifying the language and region of language for text segments using ISO codes
- Detailed phonetic-prosodic strings

Comments regarding the priority of these features, or other new features that you would like to see, are appreciated.

Web Resources

To obtain information about the Java Speech API, see the web site at:

`http://java.sun.com/products/java-media/speech/`

To obtain information about other Java Media and Communications APIs, see the web site at:

`http://java.sun.com/products/java-media/`

Mailing Lists

Discussion lists have been set up for anyone interested in the Java Speech API, the Java Speech Grammar Format specification, and the Java Speech Markup Language. The `javaspeech-announce` mailing list will carry important announcements about releases and updates. The `javaspeech-interest` mailing list is for open discussion of the Java Speech API and the associated specifications.

To subscribe to the `javaspeech-announce` list or the `javaspeech-interest` list, send email with "`subscribe javaspeech-announce`" or "`subscribe javaspeech-interest`" or both in the message body to:

`javamedia-request@sun.com`

The `javaspeech-announce` mailing list is moderated. It is not possible to send email to that list.

To send messages to the interest list, send email to:

`javaspeech-interest@sun.com`

To unsubscribe from the `javaspeech-announce` or `javaspeech-interest` lists, send email with either "`unsubscribe javaspeech-announce`" or "`unsubscribe javaspeech-interest`" or both in the message body to:

`javamedia-request@sun.com`

Revision History

Version 0.5: First public Beta release.

Contributions

Sun Microsystems, Inc. has worked in partnership with leading speech technology companies to define the specifications for the Java Speech API and the Java Speech Markup Language (JSML). These companies bring decades of research on speech technology and experience in the development and use of speech applications. Sun is grateful for the contributions of:

- Apple Computer, Inc.
- AT&T
- Dragon Systems, Inc.
- IBM Corporation
- Novell, Inc.
- Philips Speech Processing
- Texas Instruments Incorporated

Acknowledgments

JSML has benefited from many previous initiatives to mark up speech output, but two deserve particular mention for using an SGML-like syntax: work at Edinburgh University (Taylor, P. A. and Isard, A., *SSML: A speech synthesis markup language*, Speech Communication 21 (1997) p. 123-133) and the Massachusetts Institute of Technology (Slott, J. M., *A General Platform and Markup Language for Text to Speech Synthesis*, MIT Masters Thesis, 1996).

Java Speech Markup Language Specification

1.0 Introduction

A speech synthesizer provides a computer with the ability to speak. Users and applications provide text to a speech synthesizer, which is then converted to audio.

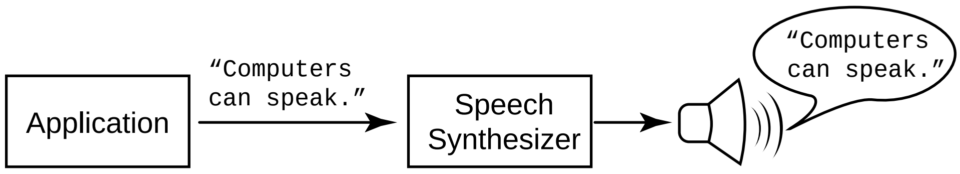

Figure 1 Text from an application is converted to audio output

Speech synthesizers are developed to produce natural-sounding speech output. However, natural human speech is a complex process, and the ability of speech synthesizers to mimic human speech is limited in many ways. For example, speech synthesizers do not "understand" what they say, so they do not always use the right style or phrasing and do not provide the same nuances as people.

The *Java™ Speech Markup Language* (JSML) allows applications to annotate text with additional information that can improve the quality and naturalness of synthesized speech. JSML documents can include *structural* information about paragraphs and sentences. JSML allows control of the *production* of synthesized speech, including the pronunciation of words and phrases, the emphasis of words (stressing or accenting), the placements of boundaries and pauses, and the control of pitch and speaking rate. Finally, JSML allows *markers* to be embedded in text and allows synthesizer-specific controls.

For the example in Figure 1, we might use JSML tags to indicate the start and end of the sentence and to emphasize the word “can”:

```
<SENT>Computers <EMP>can</EMP> speak.</SENT>
```

1.1 Role of JSML

JSML has been developed to support as many types of applications as possible, and to support text markup in many different languages. To make this possible, JSML marks general information about the text and, whenever possible, uses cross-language properties.

Although JSML may be used for text in Japanese, Spanish, Tamil, Thai, English, and nearly all modern languages, a single JSML document should contain text for only a single language. Applications are therefore responsible for management and control of speech synthesizers if output of multiple languages is required.

JSML can be used by a wide range of applications to speak text from equally varied sources, including email, database information, web pages, and word processor documents. Figure 2 illustrates the basic steps in this process.

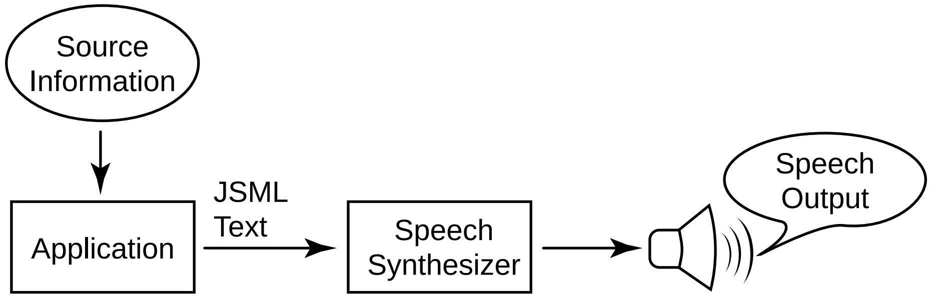

Figure 2 JSML Process

The application is responsible for converting the source information to JSML text using any special knowledge it has about the content and format of the source information. For example, an email application can provide the ability to read email messages aloud by converting messages to JSML. This could involve the conversion of email header information (sender, subject, date, etc.) to a speakable form and might also involve special processing of text in the body of the message (for handling attachments, indented text, special abbreviations, etc.) Here is a sample of an email message converted to JSML:

```
<PARA>Message from <EMP>Alan Schwarz</EMP> about new
synthesis technology.
Arrived at <SAYAS CLASS="time">2pm</SAYAS> today.</PARA>

<PARA>I’ve attached a diagram showing the new way we do
speech synthesis.</PARA>

<PARA>Regards, Alan.</PARA>
```

Similarly, a web browser could provide the ability to speak web pages by converting them to JSML. This process would involve conversion from HTML (HyperText Markup Language), the basic format of the web, to JSML. Readers may notice that JSML and HTML have a similar form. This similarity is because the formats share a common ancestor: Standard Generalized Markup Language (SGML). However, their roles are different. HTML is specialized for visual display of information, whereas JSML is for speaking information.

2.0 Markup in JSML

2.1 Basic Markup

The special text in the following example is the *text markup*.

```
<SENT>Computers <EMP>can</EMP> speak.</SENT>
```

This style will be familiar to you if you have used HTML, SGML, or XML. `<SENT>` indicates the start of a sentence element and `</SENT>` ends that sentence. Similarly, `<EMP>` and `</EMP>` mark a region to be *emphasized*.

`SENT` and `EMP` are referred to as *elements*. JSML defines eight elements. The following sections describe elements and other JSML markup in more detail.

2.2 Container Elements

JSML elements are either container elements or empty elements. A *container element* has a balanced start tag and end tag (e.g., `<SENT>` and `</SENT>`). The text appearing between the start and end tags is the *contained text* as shown in Figure 3. An element’s start-tag defines the type of element and may contain one or more attributes. All end-tags have the same name as their matching start-tag.

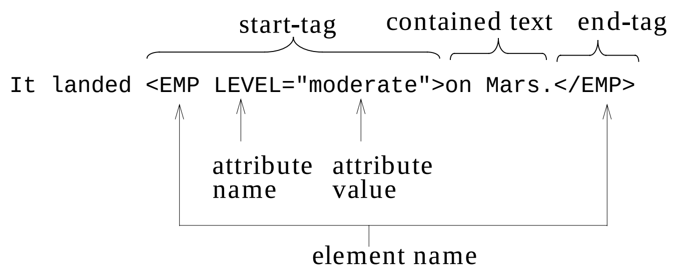

Figure 3 Elements and Attributes

2.2.1 Attributes

Attributes are used to provide additional information about an element. Each JSML element has a set of defined attribute names and, in some cases, the attribute value is restricted to certain strings. For example, an `EMP` element can mark words with a `LEVEL` attribute value of `strong`:

```
Ich bin ein <EMP LEVEL="strong">Berliner!</EMP>
```

2.2.2 Element Nesting

Some JSML elements allow the contained text to contain other elements. This is referred to as *nesting*.

```
<PARA> text with <EMP> more text </EMP> </PARA>
```

Nested elements cannot overlap or intertwine. For example, the following is not legal:

```
<PARA> text with <EMP> more text </PARA> </EMP>
```

2.3 Empty Elements

An empty element has only one tag and does not contain any text. For example, the following results in a large break/pause in the speech at the point that the element occurs:

```
A loud noise was heard, <BREAK SIZE="large"/>and the room
became quiet.
```

Because it doesn’t mark any text, an empty element like `BREAK` doesn’t need an end-tag. Rather, the “/>” marks the end of the start-tag and of the element. Like the container elements, empty elements can include attributes to provide additional information (for example, `SIZE="large"` above).

2.4 Names

All JSML element and attribute names are uppercase. All JSML attribute values are case sensitive. Furthermore, the naming of elements and attributes and the values of attributes are independent. Consequently, it is possible for an element to have an attribute of the same name (though none currently do).

2.5 White Space

Within an element's start- and end-tags, single white space characters can optionally be replaced by multiple white space characters without changing the semantics of the element.

White space contained between an element's start- and end-tags, or not contained by any element, is passed to the speech synthesizer and may affect speech output.

2.6 Undefined Names

Elements or attributes with undefined names are ignored by the speech synthesizer. This feature is useful in automatic generation and processing of JSML. For example, a web browser could generate the following:

```
<URL ORIG="http://acme.com">URL is ACME dot com</URL>
```

In this example, the `ORIG` attribute is used to preserve the original URL. The contained text will be spoken by the speech synthesizer but the URL element tags will be ignored, because they are not defined in JSML and therefore not known to the synthesizer.

This mechanism does allow speech synthesizers to extend the JSML element set by interpreting these additional elements specially. However, application developers should be aware that elements not specified in JSML are not portable across synthesizers and platforms.

2.7 JSML Document Structure

JSML is a subset of XML[1] (Extensible Markup Language), which is a simple dialect of SGML. By being a subset of XML, JSML gains a standardized, extensible syntax that is not tied to the Java Speech API (JSAPI). This means that:

- JSML is readable and editable by both humans and computers.
- General XML editors can be used to simplify writing JSML.
- JSML markup is very regular and easy for a synthesizer to parse.
- Text containing JSML can be prepared by hand using non-JSAPI-specific editors.

Although it is not necessary to know about XML to understand JSML or to use JSML, the following may be of interest. If JSML text starts with:

```
<?XML version="1.0" encoding="UCS-2"?>

<JSML>
```

and ends with:

```
</JSML>
```

then the JSML is a well-formed XML document. This means that a speech synthesizer can use a generic XML parser on JSML text. If a synthesizer supplies the parser with a DTD (Document Type Definition) for JSML, then the synthesizer's work is significantly reduced. These openning and closing elements are optional in JSML documents.

Having a DTD allows the application to use the full power of XML for generating text, for example, entity references that can act as a shorthand for repetitive JSML, and then to use generic text processing tools for generating the JSML.

2.7.1 Splitting JSML Documents

A JSML document must be syntactically complete. Every start tag must be an empty element (no end tag required) or have a matching end tag. If text is split into multiple JSML documents to be spoken in sequence, then the text should be split between paragraphs or perhaps between sentences. This is because each document will be spoken independently and important phrasing and pitch information will be affected by inappropriate boundaries.

1. World Wide Web Consortium Working Draft *Extensible Markup Language Version* 1.0 (August 7, 1997) at http://www.w3.org/TR/WD-xml-lang

2.8 Escaping/Quoting Text

If text to be spoken contains a less-than sign ("<", which is `\u003C`) or an ampersand ("&", which is `\u0026`), then the text needs to be *escaped* or *quoted* to prevent the possibility of some of the text being mistaken for JSML tags. There are several methods available:

- Individual less-than signs may be replaced with one of the following character sequences (without the quotes): "`&lt;`", "`&#60;`", or "`&#x003C;`".
- Individual ampersands may be replaced with one of the following character sequences (without the quotes): "`&amp;`", "`&#38;`", or "`&#x0026;`".
- A `CDATA` section can be placed around the entire text.

A `CDATA` section has the general form of:

```
<![CDATA[the text that is being escaped]]>
```

The text that is being escaped can contain any character sequence that is not the "]]>" sequence.

A `CDATA` section can be used on text that is contained by an element, for example:

```
<EMP>Joe Doe <!CDATA[<joe.doe@acme.com>]]></EMP>
```

and on text that is not contained by an element, for example:

```
<![CDATA[X < Y is a boolean expression.]]>
```

Synthesizers handle `CDATA` sections by stripping away the `<![CDATA[` and `]]>` markup and not parsing the `CDATA` section's contents for JSML.

2.9 Comments

A JSML comment begins with a `<!--` character sequence and ends with a `-->` character sequence and may contain any text except the two-character sequence `--`.

Comments can be placed within text that is to be spoken (the comments will not be spoken).

```
How now brown <!-- This is an example comment --> cow.
```

Comments may not be placed within elements.

3.0 JSML Elements

JSML syntax consists of structural, production, and miscellaneous elements. The following table presents an overview of JSML's elements. These elements are defined in detail in the following sections. The section on structural elements also describes *implicit paragraph marking*, which is an alternative to the `PARA` element.

Element Function	Element Name	Element Type	Element Description
Structure	`PARA`	Container	Specifies that the contained text is a paragraph.
	`SENT`	Container	Specifies that the contained text is a sentence.
Production	`SAYAS`	Container	Specifies how to say the contained text.
	`EMP`	Both	Specifies emphasis for the contained text or immediately following text.
	`BREAK`	Empty	Specifies a break in the speech.
	`PROS`	Container	Specifies a prosodic property, such as baseline pitch, rate, or volume, for the contained text.
Miscellaneous	`MARKER`	Empty	Requests a notification when speech reaches the marker.
	`ENGINE`	Container	Native instructions to a specified speech synthesizer.

4.0 Structural Elements

4.1 PARA

PARA	Container element that marks the contained text as a paragraph.
`MARK`	Optional attribute that requests a notification when the synthesizer's production of audio reaches this element's contained text. Its value is the text to be made available when the notification occurs.

The `PARA` element declares a range of text to be a paragraph. For example:

```
<PARA>This a short paragraph.</PARA><PARA>The subject has
changed, so this is a new paragraph.</PARA>
```

`PARA` elements do not contain other `PARA` elements; that is, `PARA` elements do not embed or nest. For example, the following is not legal:

```
<PARA>The raven spoke.

<PARA>I’ve come from Norway at the command of the king.
He sues for peace.</PARA>

</PARA>
```

4.2 Implicit Paragraph Marking

In JSML, a blank line (that is, a line that contains only whitespace characters) that separates one block of text from another is treated the same as explicitly marking the block as a paragraph. Strictly speaking, a blank line is not an element, however, it does serve the same function as the `PARA` element.

The following fragments result in the same speech:

```
She went to school and passed the tests.

When she returned to her bicycle, the sun had set.
```

and

```
<PARA>She went to school and passed the tests.</PARA>
<PARA>When she returned home, the sun had set.</PARA>
```

and

```
<PARA>She went to school and passed the tests.</PARA>

<PARA>When she returned home, the sun had set.</PARA>
```

and

```
<PARA>She went to school and passed the tests.

When she returned home, the sun had set.</PARA>
```

A blank line can be created by any of the following or by inserting white space (that is, any combination of spaces, `\u0020`, horizontal tabulations, `\u0009`, and ideographic spaces, `\u3000`) in any of the following:

- Consecutive carriage return and line feed pairs (that is, `\u000D \u000A \u000D \u000A`)
- Consecutive line feeds/newlines (that is, `\u000A \u000A`)
- Consecutive Unicode line separators (that is, `\u2028 \u2028`)
- A single Unicode paragraph separator (that is, `\u2029`)

4.3 SENT

SENT	Container element that marks the contained text as a sentence.
MARK	Optional attribute that requests a notification when the synthesizer's production of audio reaches this element's contained text. Its value is the text to be made available when the notification occurs.

The SENT element declares a range of text to be a sentence. For example:

```
<SENT>C'est la vie.</SENT>
```

SENT elements do not contain other SENT elements, that is, SENT elements do not embed or nest. For example, the following is not legal:

```
<SENT>He said, <SENT>"I leave tomorrow."</SENT></SENT>
```

5.0 Production Elements

5.1 SAYAS

SAYAS	Container element that says how to pronounce a word or short phrase. One of the SUB, CLASS, or PHON attributes is required.
SUB	Optional attribute having a value of the text that is to be spoken as a substitute for the contained text.
CLASS	Optional attribute indicating how to pronounce the contained text. Values: date, digits, literal, number, time.
PHON	Optional attribute having a value of a string of IPA (International Phonetic Alphabet) characters or the Java \uXXXX representation of the Unicode IPA characters that are to be spoken instead of the contained text.
MARK	Optional attribute that requests a notification when the synthesizer's production of audio reaches this element's contained text. Its value is the text to be made available when the notification occurs.

It is frequently difficult for a synthesizer to determine how to pronounce abbreviations, acronyms, proper names (particularly those originating in a language that is different from that of the synthesizer), domain-specific jargon, and homographs[2]. If an application has information that resolves a difficulty, it can provide that information to the synthesizer by using the SAYAS element.

5.1.1 SUB (Substitute)

The `SUB` attribute defines substitute text to be spoken instead of the contained text. For example:

```
<SAYAS SUB="I triple E">IEEE</SAYAS>
```

5.1.2 CLASS

When the `CLASS` attribute value is `date`, the contained text should be pronounced as a date. For example:

```
<SAYAS CLASS="date">Jan. 1952</SAYAS>
<!--spoken as January nineteen fifty-two -->
```

Note that simply stating that something is a date does not always yield the desired pronunciation. A `SUB` attribute may be required. For example, 4/3/97 is ambiguous in:

```
<SAYAS CLASS="date">4/3/97</SAYAS>
```

It might be spoken as “April third nineteen ninety-seven” or as “March fourth nineteen ninety-seven.” It is unambiguous if a `SUB` attribute is used:

```
<SAYAS SUB="March fourth nineteen ninety-seven">4/3/97
</SAYAS>
```

When the `CLASS` attribute value is `literal`, the letters, digits, and other characters of the contained text should be spoken individually. In English, this is effectively doing spelling. This is useful for speaking many acronyms and for speaking numbers as digits. For example:

```
<SAYAS CLASS="literal">JSML</SAYAS>
<!-spoken as J S M L -->

<SAYAS CLASS="literal">12</SAYAS><!--spoken as one two-->

<SAYAS CLASS="literal">100%</SAYAS> <!--might be spoken
as one zero zero percent sign-->
```

When the `CLASS` attribute value is `number`, the contained text should be pronounced as a number. For example:

```
<SAYAS CLASS="number">12</SAYAS> <!--spoken as twelve-->
```

2. Words with the same spelling but different pronunciations. For example, “I will ***read*** it.” and “I have ***read*** it.”

5.1.3 PHON (Phonetic Pronunciation)

The `PHON` attribute uses the International Phonetic Alphabet (IPA) character subset of Unicode to define a sequence of sounds. IPA characters are represented by codes from `\u0250` to `\u02AF`, by modifiers from `\u02B0` to `\u02FF`, by diacritics from `\u0300` to `\u036F`, and by certain Latin, Greek and symbol characters from the range `\u0000` to `\u017F`. Details of the Unicode IPA support are provided in *The Unicode Standard, Version 2.0* (The Unicode Consortium, Addison-Wesley Developers Press, 1996).

The following examples are equivalent:

```
<SAYAS PHON="foʊnɛtɪks"> phonetics </SAYAS>

<SAYAS PHON="\u0066\u006F\u028A\u006E\u025B
\u0074\u026A\u006B\u0073"> phonetics </SAYAS>
```

Note that sounds from outside the language of the synthesizer may not be reproduced accurately.

5.1.4 Nesting

Elements cannot be nested within the contents of a `SAYAS`.

Legal example:

```
<PROS RATE="-30%"><SAYAS SUB="sun dot com">sun.com
</SAYAS></PROS>
```

Illegal example:

```
<SAYAS SUB="sun dot com"><PROS RATE="-30%">sun.com
</PROS></SAYAS>
```

5.2 EMP

`EMP`	Element that specifies a level of emphasis for the contained text (if used as a container element) or the following word (if used as an empty element). `LEVEL="moderate"` is the default attribute.
`LEVEL`	Required attribute that indicates the level of emphasis. Values: `strong`, `moderate`, `none`, or `reduced`. `LEVEL="moderate"` is the default attribute.
`MARK`	Optional attribute that requests a notification when the synthesizer's production of audio reaches this element's contained text. Its value is the text to be made available when the notification occurs.

The `EMP` element specifies that a range of text should be spoken with emphasis. The `LEVEL` attribute's values are `strong` (for strong emphasis), `moderate` (for some emphasis), `none` (for no emphasis), and `reduced` (for a reduction in emphasis).

For example:

```
Clap your <EMP>hands.</EMP>

Clap your <EMP LEVEL="moderate">hands.</EMP>
```

The `EMP` element can also be an empty element, where it specifies that the immediately following text[3] is to be emphasized.

The following examples have the same effect as above:

```
Clap your <EMP/>hands.

Clap your <EMP LEVEL="moderate" MARK="hands"/> hands.
```

3. The meaning of “immediately following text” is language dependent. English speech synthesizers will emphasize the next word.

5.3 BREAK

`BREAK`	Empty element that marks a break in the speech. `SIZE="medium"` is the default attribute if neither `SIZE` or `MSECS` is provided.
`MSECS`	Optional attribute having a value of an integral number of milliseconds.
`SIZE`	Optional attribute having one of the following relative values: `none`, `small`, `medium`, or `large`.
`MARK`	Optional attribute that requests a notification when the synthesizer's production of audio reaches this element's contained text. Its value is the text to be made available when the notification occurs.

The `BREAK` element is an empty element that is used to mark phrasing boundaries in the speech output. To indicate what type of break is desired, the element can include a `SIZE` attribute or a `MSECS` attribute, but not both. A `SIZE` attribute indicates a break that is relative to the characteristics of the current speech, and a `MSECS` attribute indicates a pause for an absolute amount of time.

Where possible, the break should be defined by a `SIZE` rather than a `MSECS`, because, in most languages, breaks are produced by special movements in pitch, by timing changes, and often with a pause. Those factors are significantly affected by speaking context. For example, a 300 millisecond break in fast speech sounds more significant than it does in slow speech.

Examples:

```
<BREAK/>

<BREAK SIZE="small" MARK="145"/>

<BREAK MSECS="300"/>
```

5.4 PROS

PROS	Element that specifies prosodic information for the contained text. At least one of the RATE, PITCH, RANGE, and VOL attributes is required.
RATE	Optional numeric attribute that sets the speaking rate in words per minute. See the text following this table for the type of values allowed.
VOL	Optional numeric attribute that sets the output volume on a scale of 0.0 to 1.0 where 0.0 is silence and 1.0 is maximum loudness. See the text following this table for the type of values allowed.
PITCH	Optional numeric attribute that sets the baseline pitch in Hertz. See the text following this table for the type of values allowed.
RANGE	Optional numeric attribute that sets the pitch range in Hertz. See the text following this table for the type of values allowed.
MARK	Optional attribute that requests a notification when the synthesizer's production of audio reaches this element's contained text. Its value is the text to be made available when the notification occurs.

The PROS element provides prosody control for JSML. Prosody is a collection of features of speech that includes its timing, intonation and phrasing. Proper control of prosody can improve the understandability and naturalness of speech. They are better viewed as being "hints" to the synthesizer. Most of the attributes of the PROS tag accept numeric values. These values are floating point numbers of the form 23, 10.8, or -0.55.

The RATE attribute is defined in words per minute and can have values of the following forms:

n	Sets the speaking rate to n
+n	Increases the speaking rate by n
-n	Decreases the speaking rate by n
+n%	Increases the speaking rate by n percent
-n%	Decreases the speaking rate by n percent
reset	Sets the speaking rate to the default

For example,

```
<PROS RATE="150">text at 150 words per minute</PROS>
```

The `VOL` attribute can have values of the following forms:

n	Sets the volume to n (between 0.0 and 1.0 inclusive)
+n	Increases the volume by n (to a final value no larger than 1.0)
-n	Decreases the volume by n (to a final value no smaller than 0.0)
+n%	Increases the volume by n percent
-n%	Decreases the volume by n percent
reset	Sets the volume to the default

The `PITCH` and `RANGE` attributes can have values of the following forms:

n	Sets the baseline pitch or pitch range to n Hertz
+n	Increases the baseline pitch or pitch range by n Hertz
-n	Decreases the baseline pitch or pitch range by n Hertz
+n%	Increases the baseline pitch or pitch range by n percent
-n%	Decreases the baseline pitch or pitch range by n percent
reset	Sets the baseline pitch or pitch range to the default

Musically-inclined developers might think of pitch in semitones and octaves. A semitone rise in pitch is approximately +5.9% and a semitone drop is -5.6%. A two-semitone shift is +12.2% or -10.9%. A one-octave shift (12 semitones) is 100% or -50%, that is, doubling or halving pitch.[4]

While speaking a sentence, pitch moves up and down in natural speech to convey extra information about what is being said. The baseline pitch represents the normal minimum pitch of a sentence. The pitch range represents the amount of variation in pitch above the baseline. Setting the baseline pitch and pitch range can affect whether speech sounds monotonous (small range) or dynamic (large range).

4. Percentages for 1 to 12 semitone pitch rises are +5.9%, +12.2%, +18.9%, +26.0%, +33.5%, +41.4%, +50%, +58.7, +68.2%, +78.2%, +88.8%, +100%.
Decreases are -5.6%, -10.9%, -15.9%, -20.6%, -25.1%, -29.3%, -33.3%, -37.0%, -40.5%, -43.9%, -47.0%, -50.0%.

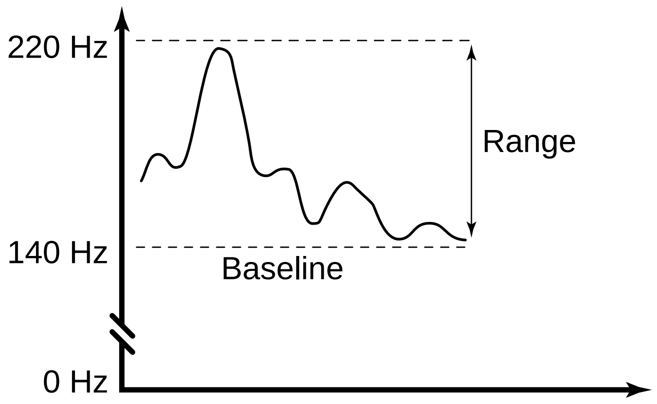

Figure 4 Baseline Pitch and Pitch Range

Normal baseline pitch for a female voice is between 140Hz and 280Hz, with a pitch range of 80Hz or more. Male voices are typically lower: baseline of 70-140Hz, with a range of 40-80Hz.

Note that in all cases, relative values increase the portability of JSML across speaking voices and synthesizers. Relative settings allow users to apply the same JSML to different voices (*e.g.*, male and female voices with very different pitch ranges) and to set a local preference for speaking rate. For example, some users set the speaking rate very high (300 words per minute or faster) so they can listen to a lot of text very quickly.

Example:

```
The <EMP/>ACME Trading Corporation, <PROS
RANGE="-30%">which supplies cartoon goods,</PROS> was
purchased yesterday for <PROS RATE="-20%" VOL="+15%">
$2,060,000 </PROS> by <EMP> Road Runner </EMP>
Incorporated.
```

6.0 Other Elements

6.1 MARKER

`MARKER`	Empty element that requests a notification when the synthesizer's production of audio reaches the marker. The `MARK` attribute is required.
`MARK`	Required attribute having a value of the text to be made available when a marker event occurs.

The `MARKER` element requests a notification from the speech synthesizer to the application when the `MARK` is reached during the synthesizer's production of audio for the text.

Example:

```
Answer <MARKER MARK="yes_no_prompt"/> yes or no.
```

6.2 ENGINE

`ENGINE`	Container element that provides information from the required `DATA` attribute to the synthesizer identified by the required `ENGID` attribute
`ENGID`	Identifier for a speech synthesizer or a comma-separated set of speech synthesizer identifiers.
`DATA`	Required attribute having a value of the information for the synthesizer.
`MARK`	Optional attribute that requests a notification when the synthesizer's production of audio reaches this element's contained text. Its value is the text to be made available when the notification occurs.

This `ENGINE` element allows applications to utilize a synthesizer's special capabilities. The element provides information, the value of the `DATA` attribute, to any speech synthesizers that are identified by the `ENGID` attribute. The information is generally a command in an engine-specific syntax.

`ENGINE` is a container element that is treated specially by a speech synthesizer that matches any engine specified in the `ENGID`. A matching engine should substitute the `DATA` for the text contained within the element. Other engines should ignore the `DATA` and instead process the contained text. For example, given the code

```
I am <ENGINE ENGID="Acme Voice" DATA="Mr. Acme"> someone
else</ENGINE>
```

an Acme voice synthesizer will say “I am Mr. Acme” and all other speech synthesizers will say “I am someone else.” A JSML document can contain ENGINE elements for any number of synthesizers. These elements can be nested.

Less-than signs (“<”) or ampersands (“&”) in a DATA attribute must be escaped to avoid being mistaken for JSML (see *Escaping/Quoting Text* on page 7).

For example;

```
<ENGINE ENGID="Croaker 1.0" DATA="&lt;ribbit=1>"
MARK="frog start"> no frog sound </ENGINE>
```