
# Working Draft 13 March 2001

This version:

http://www.interface.computing.edu.au/documents/VHML/2001/WD−VHML−20010313

Latest version:

http://www.interface.computing.edu.au/documents/VHML

Previous version:

http://www.interface.computing.edu.au/documents/VHML

Editors:

Andrew Marriott
Simon Beard
John Stallo
Quoc Huynh



## Status of this Document

*This section describes the status of this document at the time of its publication. Other documents may supersede this document. The latest status of this document series is maintained at the Curtin InterFace Website.*

This is the 9th March 2001 Working Draft of the "Virtual Human Markup Language Specification".

This working draft relies on several other standards − the various sub−languages of VHML use and extend these standards.

## Abstract

This document describes a Virtual Human Markup Language. The language is designed to accommodate the various aspects of Human−Computer Interaction with regards to Facial Animation, Body Animation, Dialogue Manager interaction, Text to Speech production, Emotional Representation plus Hyper and Multi Media information. [Input here: am I missing any required sub−system?]

It will use / build on existing (de facto) standards such as those specified by the W3C Voice Browser Activity, and will describe new languages to accommodate functionality that is not catered for.

The language will be XML/XSL based and will consist of the following sub−systems:

- DMML Dialogue Manager Markup Language (W3C Dialogue Manager or AIML)
- FAML Facial Animation Markup Language [Any existing standard?]
- BAML Body Animation Markup Language [Any existing standard?]
- SML Speech Markup Language (SSML / Sable)
- EML Emotion Markup Language
- HTML HyperText Markup Language [ or subset only?]

The language will use XML Namespaces for inheritance of existing standards.

Although general in nature, the intent of this language is to facilitate the natural and realistic interaction of a Talking Head or Talking Human with a user via a Web page or application. One specific intended use can be found in the deliverables of the Interface project (http://www.ist−interface.org/).

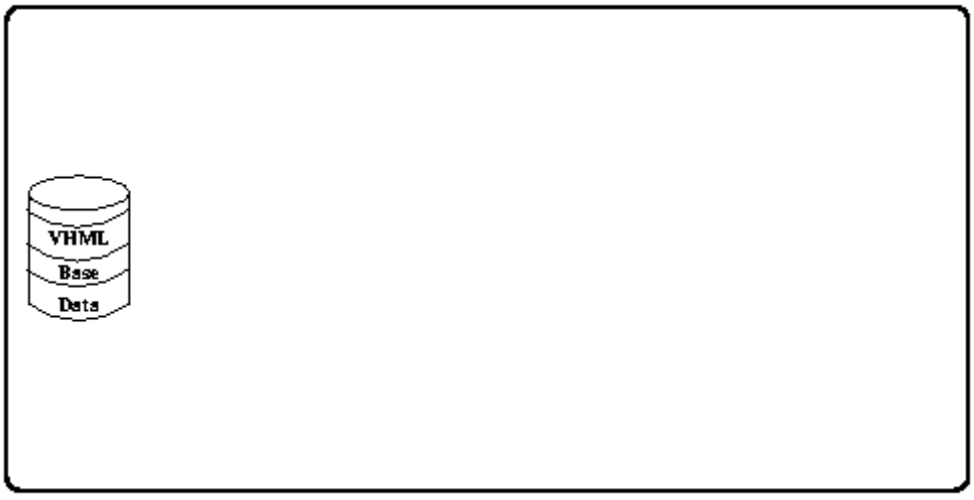



**Figure 1 The user−>Dialogue Manager−>user data flow**

# Table of Contents

## Terminology and Design Concepts

The design and standardization process has adopted the approach of the Speech Synthesis Markup Requirements for Voice Markup Languages published December 23, 1999 by the W3C Voice Browser Working Group.

The following items were the key design criteria.

- *Consistency:* provide predictable control of rendering output across platforms and across speech synthesis implementations.
- *Interoperability:* support use along with other W3C specifications including (but not limited to) the Dialog Markup Language, Audio Cascading Style Sheets and SMIL, etc.
- *Generality:* support rendering output for a wide range of applications with varied graphics capability and speech content.
- *Internationalization:* Enable speech output in a large number of languages within or across documents.
- *Generation and Readability:* Support automatic generation and hand authoring of documents. The documents should be human−readable.
- *Implementable:* The specification should be implementable with existing, generally available technology and the number of optional features should be minimal.

### *Rendering Processes*

A rendering system that supports the Virtual Human Markup Language will be responsible for rendering a document as visual and spoken output and for using the information contained in the markup to render the document as intended by the author.

*Document creation:* A text document provided as input to the system may be produced automatically, by human authoring, or through a combination of these forms. The Virtual Human Markup Language defines the form of the document.

*Document processing:* The following are the nine major processing steps undertaken by a VHML system to convert marked−up text input into automatically generated output. The markup language is designed to be sufficiently rich so as to allow control over each of the steps described below so that the document author (human or machine) can control or direct the final rendered output of the Virtual Human.

a) **XML Parse:** An XML parser is used to extract the document tree and content from the incoming text document. The structure, tags and attributes obtained in this step influence each of the following steps.
b) **Culling of un−needed VHML tags:** For example, at this stage any tags which produce audiowhen the final rendering device/environment does not support audio may be removed. Similarly for other tags. It should be noted that since the timing synchronisation is based upon vocal production, the spoken text may need to be processed regardless of the output device's capabilities.

c) **Structure analysis:** The structure of a document influences the way in which a document should be read. For example, there are common speaking and acting patterns associated with paragraphs and sentences.
   − *Markup support:* Various elements defined in the VHML markup language explicitly indicate document structures that affect the visual and spoken output.
   − *Non−markup behavior:* In documents and parts of documents where these elements are not used, the VHML system is responsible for inferring the structure by automated analysis of the text, often using punctuation and other language−specific data. [How good could we make this?]

d) **Text normalization:** All written languages have special constructs that require a conversion of the written form (orthographic form) into the spoken form. Text normalization is an automated process of the TTS system that performs this conversion. For example, for English, when "$200" appears in a document it may be spoken as "two hundred dollars". Similarly, "1/2" may be spoken as "half", "January second", "February first", "one of two" and so on.
   − *Markup support:* The "say−as" element can be used in the input document to explicitly indicate the presence and type of these constructs and to resolve ambiguities. The set of constructs that can be marked includes dates, times, numbers, acronyms, current amounts and more. The set covers many of the common constructs that require special treatment across a wide number of languages but is not and cannot be a complete set.
   − *Non−markup behavior:* For text content that is not marked with the `say−as` element the TTS system is expected to make a reasonable effort to automatically locate and convert these constructs to a speakable form. Because of inherent ambiguities (such as the "1/2" example above) and because of the wide range of possible constructs in any language, this process may introduce errors in the speech output and may cause different systems to render the same document differently.[What is the BAP equivalent of this text normalisation?]

e) **Text−to−phoneme conversion:** Once the system has determined the set of words to be spoken it must convert those words to a string of phonemes. A *phoneme* is the basic unit of sound in a language. Each language (and sometimes each national or dialect variant of a language) has a specific phoneme set: e.g. most US English dialects have around 45 phonemes. In many languages this conversion is ambiguous since the same written word may have many spoken forms. For example, in English, "read" may be spoken as "reed" (I will read the book) or "red" (I have read the book).
   Another issue is the handling of words with non−standard spellings or pronunciations. For example, an English TTS system will often have trouble determining how to speak some non−English−origin names; e.g. "Tlalpachicatl" which has a Mexican/Aztec origin.
   − *Markup support:* The "phoneme" element allows a phonemic sequence to be provided for any word or word sequence. This provides the content creator with explicit control over pronunciations. The "say−as" element may also be used to indicate that text is a proper name that may allow a TTS system to apply special rules to determine a pronunciation.
   − *Non−markup behavior:* In the absence of a "phoneme" element the TTS system must apply automated capabilities to determine pronunciations. This is typically achieved by looking up words in a pronunciation dictionary and

applying rules to determine other pronunciations. Most TTS systems are expert at performing text−to−phoneme conversions so most words of most documents can be handled automatically.

f) **Prosody analysis:** Prosody is the set of features of speech output that includes the pitch (also called intonation or melody), the timing (or rhythm), the pausing, the speaking rate, the emphasis on words and many other features. Producing human−like prosody is important for making speech sound natural and for correctly conveying the meaning of spoken language.
   − *Markup support:* The "emphasis" element, "break" element and "prosody" element may all be used by document creators to guide the TTS system is generating appropriate prosodic features in the speech output.
   − *Non−markup behavior:* In the absence of these elements, TTS systems are expert (but not perfect) in automatically generating suitable prosody. This is achieved through analysis of the document structure, sentence syntax, and other information that can be inferred from the text input.

g) **Waveform production:** The phonemes and prosodic information are used by the TTS system in the production of the audio waveform. There are many approaches to this processing step so there may be considerable platform−specific variation.
   – *Markup support:* The TTS markup does not provide explicit controls over the generation of waveforms. The "voice" element allows the document creator to request a particular voice or specific voice qualities (e.g. a young male voice). The "audio" element allows for insertion of recorded audio data into the output stream.

h) **Facial and BodyAnimation production:** Timing information will be used to synchronise the spoken text with facial gestures and expressions as well aswith body movements and gestures.

i) Rendering the multiple streams (Audio, Graphics, Hyper and Multi Media) onto the output device(s). XSL Transformation − here or in the earlier stage?

[Need info about the FAP and BAP production in here]

### *Document Generation, Applications and Contexts*

There are many classes of document creator that will produce marked−up documents to be spoken by a VHML system. Not all document creators (including human and machine) have access to information that can be used in all of the elements or in each of the processing steps described in the previous section. The following are some of the common cases.

- The document creator has no access to information to mark up the text. All processing steps in the VHML system must be performed fully automatically on *raw text*. The document requires only the containing "vhml" element to indicate the content is to be rendered.
- When marked text is generated programmatically the creator may have specific knowledge of the structure and/or special text constructs in some or all of the document. For example, an email reader can mark the location of the time and date of receipt of email. Such applications may use elements that affect structure, text normalization, prosody, possibly text−to−phoneme conversion, as well as facial or body gestures to gain the user's attention.
- Some document creators make considerable effort to mark as many details of the document to ensure consistent speech quality across platforms and to more

precisely specify output qualities. In these cases, the markup may use any or all of the available elements to tightly control the visual or speech output.

- The most advanced document creators may skip the higher−level markup (Emotions, Facial and body animation tags) and produce low−level VHML markup for segments of documents or for entire documents.

It is important that any XML elements or tags that are part of VHML use existing tags specified in existing (de facto) or developing standards (for example such as HTML or SSML). This will aid in minimising learning curves for new developers as well as maximising opportunites for th emigration of legacy data.

## The Language Structure

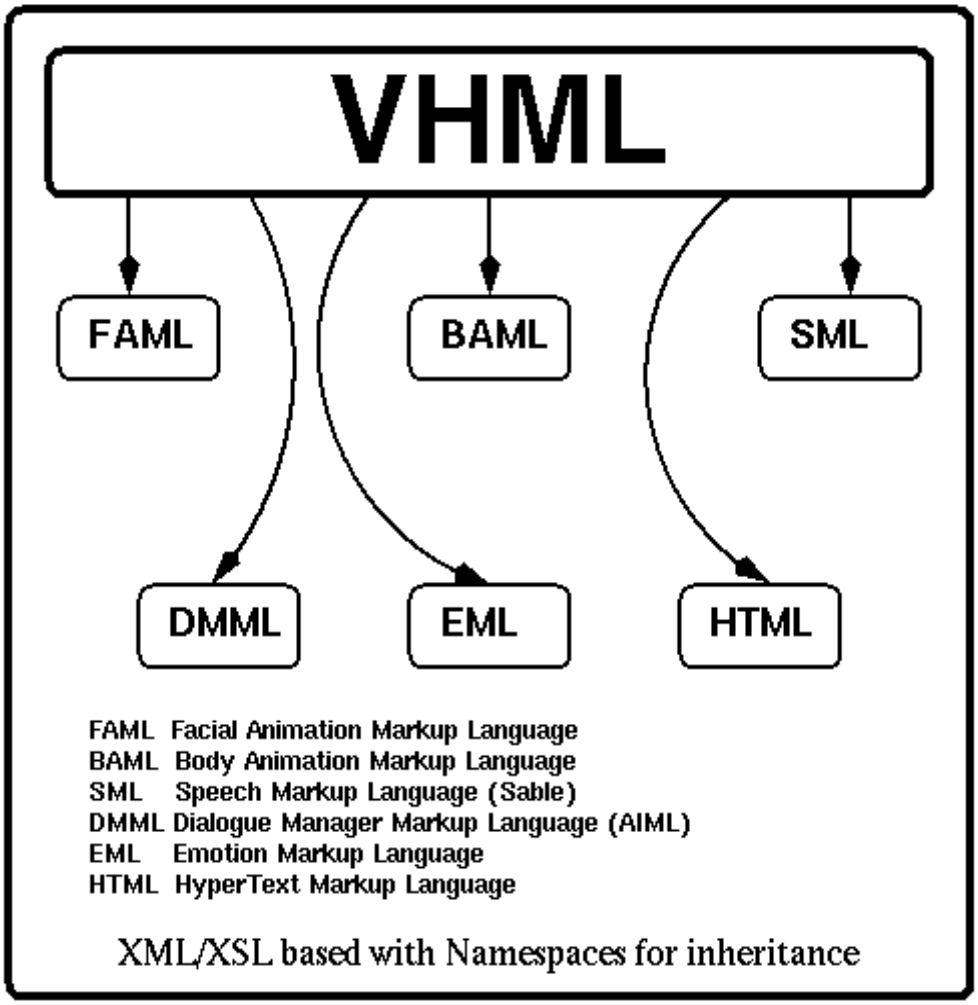



**Figure 2 The VHML Language Structure**

VHML uses the languages shown in Figure2 tofacilitate the direction of a Virtual human interacting with a user via a Web page or stand alone application. In response to a user enquiry, the Virtual human will have to react in a realistic and humane way using appropriate words, voice, facial and body gestures. For example, a Virtual Human that has to give some bad news to the user − "I'm sorry Dave, I can't find that file you want." − mayspeak in a sad way, with a sorry face and with a bowed body stance. In a similar way, a different message may be delivered with a happy voice, a smiley face and with a lively body.

The following sections detail the individual XML based languages which make this possible through VHML.

### *Virtual Human Markup Language (VHML)*

**Root Element**

The Virtual human Markup Language is an XML application. The root element is `vhml`. See the section on Conformance.

```
<?xml version="1.0"?>
<vhml>
      ... the body ...
</vhml>
```

## vhml

*Description:*

Root element that encapsulates all other `vhml` elements.

*Attributes:* none.

*Properties:* root node, can only occur once.

*Example:*

```
<vhml>
      <p>
            <happy>
                  The vhml element encapsulates all other
                  elements
            </happy>
      </p>
</vhml>
```

*Notes:* Should we allow `<viewset>` and `<view>` a la `<frame>` and `<frameset>`? This would allow multiple rendered scenes plus a Virtual Human with an HTML page for hyper information.

**Miscellaneous Elements**

## embed

*Description:*

Gives the ability to embed foreign file types within a VHML document such as sound files, MML files etc., and for them to be processed appropriately.

*Attributes:*

| Name | Description | Values |
|---|---|---|
| `type` | Specifies the type of file that is being embedded. (Required) | audio − embedded file is an audio file.<br>mml − an mml file is embedded.<br>[What values should we have here?] |
| `src` | Gives path to audio file. (Required) | A character string. |

*Properties:* empty.

*Example:*

```
<embed type="mml" src="songs/aaf.mml"/>
```

### *Emotion Markup Language (EML)*

**Emotions**

The following elements will affect the emotion shown by the Virtual Human. These elements will affect the voice, face and body.

**Emotion Default Attributes**

Each element has at least 3 attributes associated with it:

| Name | Description | Values | Default |
|---|---|---|---|
| `intensity` | This value ranges from 0 to−100 and represents a percentage value of the maximum intensity of that particular facial gesture, expression or emotion. | 0 − 100 | 100 |
| `duration` | The duration value represents the time span in seconds or milliseconds that the element expression, gesture or emotion will persist in the Virtual Human animation. | A numeric value representing time (conforms to Times attribute from CSS specification ). | Until closing element |
| `mark` | This attribute can be used to set an arbitrary mark at a given place in the text, so that, for example, an engine can report back to the calling application that it has reached the given location. | Character−string identifier for this tag. | No default − optional attribute |

**Notes:**

EML emotion elements can be placed in sequence to produce a seamless flow from one emotion to the other. Emotion elements can also be blended together at the same instance to produce different expressions and emotions entirely, as desired.

[How would we do this? `Contribution` attributes which are combined to produce 100% emotion? No `contribution` value means 100% of that emotion?]

OTHER EMOTIONS?????

Should the TAG names be nouns (sadness, anger) or verbs (sad, angry)?
Should we also allow subjective durations − short, medium, long − similar to the pause element?

## anger

*Description:*

Simulates the effect of anger on the rendering (i.e. generates a Virtual Human that looks and sounds angry).

*Attributes:* `Default EML Attributes`.

*Properties:* Can contain other non−emotion elements.

*Example:*

```
<anger>
    I would not give you the time of day
</anger>
```

## joy == happy

*Description:*

Simulates the effect of happiness on the rendering (i.e. generates a Virtual Human that looks and sounds joyful).

*Attributes:* `Default EML Attributes`. .

*Properties:* Can contain other non−emotion elements.

*Example:*

```
<joy>
    I have some wonderful news for you.
</joy>
```

## neutral

*Description:*

Gives a neutral intonation to the Virtual Human's appearance and sound..

*Attributes:* `Default EML Attributes`. .

*Properties:* Can contain other non−emotion elements.

*Example:*

```
<neutral>
    I can sometimes sound non−commital like this.
</neutral>
```

## sadness

*Description:*

Simulates the effect of sadness on the rendering (i.e. generates a Virtual human that looks and sounds sad).

*Attributes:* Default EML Attributes. .

*Properties:* Can contain other non−emotion elements.

*Example:*

```
<sadness>
    Honesty is hardly ever heard.
</sadness>
```

## fear

*Description*:

Simulates the effect of fear on the rendering (i.e.generates a Virtual Human that looks and sounds afraid).

*Attributes*: Default EML Attributes. .

*Properties*: Can contain other non−emotion elements.

*Example*:

```
<fear>
    I am afraid of flying.
</fear>
```

## disgust

*Description*:

Simulates the effect of disgust on the rendering (i.e. generates a Virtual Human that looks and sounds disgusted).

*Attributes*: Default EML Attributes. .

*Properties*: Can contain other non−emotion elements.

*Example*:

```
<disgust>
    How could you eat Roquefort cheese!
</disgust>
```

## surprise

*Description*:

Simulates the effect of surprise on the rendering (i.e. generates a Virtual Human that looks and sounds surprised).

*Attributes*: Default EML Attributes. .

*Properties*: Can contain other non−emotion elements.

*Example*:

```
<surprise>
    I did not expect to find that in my lasagne!
</surprise>
```

## dazed

*Description*:

Simulates the effect of being dazed on the rendering (i.e. generates a Virtual Human that looks and sounds dazed).

*Attributes*: Default EML Attributes. .

*Properties*: Can contain other non−emotion elements.

*Example*:

```
<dazed>
    Did you get the number of that truck?
</dazed>
```

## confused

*Description*:

Simulates the effect of confusion on the rendering (i.e. generates a Virtual Human that looks and sounds confused).

*Attributes*: Default EML Attributes. .

*Properties*: Can contain other non−emotion elements.

*Example*:

```
<confused>
    If this is Tuesday, then this must be
    Linköping.
</confused>
```

## bored

*Description*:

Simulates the effect of boredom on the rendering (i.e. generates a Virtual Human that looks and sounds bored).

*Attributes*: Default EML Attributes. .

*Properties*: Can contain other non−emotion elements.

*Example*:

```
<bored>
    Writing specifications is real fun.
</bored>
```

**Other Virtual Human Emotional Responses**
The following elements will accommodate other well known human emotional reactions. These will affect the voice, face and body of the Virtual Human.

[Should these be EML?]

**Notes:**
1: The timing is such that the action is performed at the place where the element is (i.e.depends on what has been spoken/acted out before this element is met.) This must take into account **Text Normalisation** differences between what the text is and what is actually spoken.

```
A <smile intensity="50" duration="5000/>
little dog goes into
<head_left_roll intensity="40" duration="1200"/>
<agree intensity="30" duration="1200"/>
a saloon in the Wild West, and
<head_right_roll intensity="60" duration="1000"/>
<agree intensity="30" duration="1000"/>
<head_left intensity="40" duration="1000"/> beckons
to the bartender.
```

2: These elements also have `intensity` and `duration` attributes as for the EML elements. The `duration` must be specified.

## agree

*Description*:

The `agree` element animates a nod of the Virtual Human. The agree element animation is broken into two sections: the head raise and then the head lower.
Observations have shown that there is a raise of the head before the nod is initiated. The `agree` element mimics this and 10 percent of the `duration` for the `agree` element is allocated for the head raise, with an intensity of 10 percent of the authored `intensity` value; the other 90 percent is allocated to the head lower.
The `agree` element can typically be used to gesture "yes" or "agreement". Only the vertical angle of the head is altered during the element animation, the eye gaze is still focused forward.
[Body animation for this element?]
[Should % be an attribute?]

*Attributes*: `Default EML Attributes`.
`duration` must have a value.

*Properties*: none (Atomic element).

*Example:*

```
That's certainly <agree duration="1000"/>right
          Olly.
```

## disagree

*Description*:

The `disagree` element animates a shake of the head. The element animates two shakes, a single shake is considered to be a head movement from the left to the right.
The `disagree` element can be used as a facial gesture for "no" or "disagree".
The element only affects the horizontal displacement of the head and no other facial features are affected.
Animation involves moving first to the left, then right, repeated and then returning to the central plane.
[Body animation for this element?]
[Other attributes? − # of shakes, left or right first?]

*Attributes*: `Default EML Attributes`.
`duration` must have a value.

*Properties*: none (Atomic element).

*Example:*

```
I <disagree duration="2000"/> will not have that
smelly cheese on my spaghetti
```

## emphasis

*Description*:

The `emphasis` element is very similar in animation to the `agree` element. The difference being the `emphasis` element incorporates a lowering of the eyebrow into the nod itself as described by Pelachaud and Prevost (1995). This serves to further emphasize or accentuate words in the spoken text.
The `emphasis` element similarly has raise and lower stages as found in the `agree` element animation. It is noted however that the eyebrow are lowered at the same rate as the nod and if a different intensity of eyebrow lowering is needed the `emphasis` element can be used in conjunction with the `brow_down` element to produce an emphasis animation with a greater lowering of the eyebrow or a more subtle one.
[Body animation for this element?]

*Attributes*: `Default EML Attributes`.
`duration` must have a value.

*Properties*: none (Atomic element).

*Example:*

```
I <emphasis duration="500"/> will not buy this record, it is scratched.
```

## smile

*Description*:

The `smile` element, as the name suggest animates the expression of a smile into the Talking Head animation.
The mouth is widened and the corners pulled back towards the ears. The larger the `intensity` value for the `smile` element, the greater the intensity of the smile. However a value too large, produces a rather "cheesy" looking grin and can look disconcerting or phony. This however can be used to the animator's advantage, if a mischievous grin or masking smile is required.
The `smile` element is generally used to start sentences and is used quite often when accentuating positive or cheerful words in the spoken text (Pelachaud and Prevost, 1995).

*Attributes*: `Default EML Attributes`.
`duration` must have a value.

*Properties*: none (Atomic element).

*Example:*

```
<smile duration="5000"/> Potatoes must be almost as good as chocolate to eat!
```

## shrug

*Description*:

The `shrug` element animation mimics the facial and body expression "I don't know".
A facial shrug consists of the head tilting back, the corners of the mouth pulled downward and the inner eyebrow tilted upwards and squeezed together.
A body shrug consists of [INFO needed here please.]

*Attributes*: `Default EML Attributes`.
`duration` must have a value.

*Properties*: none (Atomic element).

*Example:*

```
<shrug duration="5000"/>I neither know nor care!
```

**Emotional Markup Language Examples**

```
<?xml version="1.0"?>
<!DOCTYPE vhml SYSTEM "./vhml-v01.dtd">

<vhml>

    <p>
     <angry>Don't tell me what to do</angry>

     <happy>I have some wonderful news for you</happy>

     <neutral>I am saying this in a neutral voice</neutral>

     <sad>I can not come to your party tomorrow</sad>
    </p>
</vml>
```

### *Facial Animation Markup Language (FAML)*

**Emotion Default Attributes**

Each element has at least 3 attributes associated with it:

| Name | Description | Values | Default |
|---|---|---|---|
| `intensity` | This value ranges from 0 to−100 and represents a percentage value of the maximum intensity of that particular facial gesture, expression or emotion. | 0 − 100 | 100 |
| `duration` | The duration value represents the time span in milliseconds that the element expression, gesture or emotion will persist in the Virtual Human animation. | A numeric value representing time in milliseconds. | Must be specified |
| `mark` | This attribute can be used to set an arbitrary mark at a given place in the text, so that, for example, an engine can report back to the calling application that it has reached the given location. | Character− string identifier for this tag. | No default − optional attribute |

**Direction/Orientation**

The following elements affect the direction or orientation of the head and the eyes (directions are wrt Talking Head).

The animation of the head movement can be broken down into three main parts: pitch, yaw and roll.

The pitch affects the elevation and depression of the head in the vertical field. The yaw affects the rotational angle of the head in the horizontal field and roll affects the axial angle. The combination of these three factors allow full directional movement for the animation of the Talking Head.

**Notes**

1: There are 12 main elements that control and animate the direction and orientation of the Talking Head. [Should we have independent eye/head movement?]

2: It is noted that the eyes and head move at the same rate during the animation of the looking elements.

3: All combinations of the above directional elements allow the head to have full range of orientation. A combination of the `<look_left/>` and `<look_up/>` elements will enable the head to look to the top left in the animation sequence, whilst `<look_right/>` `<look_down/>` will enable the head to look to the bottom right.

4: The `eye_xxx` directional elements allow four independent directions for eye movement. This entails movement in the vertical and horizontal planes. As with head directional elements, the elements can be combined together to provide full range of eye gaze even those not humanly possible. It is however noted that the eyes cannot be animated independently of each other. [ Is this a problem????  We could use the `which` attribute of `eyebrow_up`]

## anger

*Description:*

Inherited from EML.

## joy == happy

*Description:*

Inherited from EML.

## neutral

*Description:*

Inherited from EML.

## sadness

*Description:*

Inherited from EML.

## fear

*Description*:

Inherited from EML.

## disgust

*Description*:

Inherited from EML.

## surprise

*Description*:

Inherited from EML.

## dazed

*Description*:

Inherited from EML.

## confused

*Description*:

Inherited from EML.

## bored

*Description*:

Inherited from EML.

## look_left

*Description*:

Turns both the eyes and head to look left.

*Attributes*: `Default FAML Attributes`.
`duration` must have a value.

*Properties*: none (Atomic element).

*Example*:

```
<look_left duration="1000"/>Cheese to the left of
me!
```

## look_right

*Description*: Turns both the eyes and head to look right.

*Attributes*: `Default FAML Attributes`.
`duration` must have a value.

*Properties*: none (Atomic element).

*Example*:

```
<look_right duration="800"/>Cheese to the right of
me!
```

## look_up

*Description*:

Turns both the eyes and head to look up.

*Attributes*: `Default FAML Attributes`.
`duration` must have a value.

*Properties*: none (Atomic element).

*Example*:

```
<look_up duration="5000"/>Dear God, is there no
escaping this smelly cheese?
```

## look_down

*Description*:

Turns both the eyes and head to look down.

*Attributes*: `Default FAML Attributes`.
`duration` must have a value.

*Properties*: none (Atomic element).

*Example*:

```
<look_down duration="1000"/>Perhaps it is just my
feet!
```

## head_left

*Description*:

Only the head turns left, the eyes remain looking forward.

*Attributes*: `Default FAML Attributes`.
`duration` must have a value.

*Properties*: none (Atomic element).

*Example*:

```
<head_left duration="2000" intensity="30"/>What, no
potatoes?
```

## head_right

*Description*:

Only the head turns right, the eyes remain looking forward.

*Attributes*: `Default FAML Attributes`.
`duration` must have a value.

*Properties*: none (Atomic element).

*Example*:

```
<head_right duration="100"/>Where is the chocolate?
```

## head_up

*Description*:

Only the head turns upward, the eyes remain looking forward.

*Attributes*: `Default FAML Attributes`.
`duration` must have a value.

*Properties*: none (Atomic element).

*Example*:

```
<head_up intensity="100" duration="1000"/>You are an
insolent swine!
```

## head_down

*Description*:

Only the head turns downward, the eyes remain looking forward.

*Attributes*: `Default FAML Attributes`.
`duration` must have a value.

*Properties*: none (Atomic element).

*Example*:

```
<head_down duration="2500"/>Are you happy now?
```

## eyes_left

*Description*:

Only the eyes turn left, the head remains looking forward.

*Attributes*: `Default FAML Attributes`.
`duration` must have a value.

*Properties*: none (Atomic element).

*Example*:

```
<eyes_left duration="1000"/>There is the door,
please use it.
```

## eyes_right

*Description*:

Only the eyes turn right, the head remains looking forward.

*Attributes*: `Default FAML Attributes`.
`duration` must have a value.

*Properties*: none (Atomic element).

*Example*:

```
<eyes_right duration="1000"/>Stand still laddie!
```

## eyes_up

*Description*:

Only the eyes turn upward, the head remains looking forward.

*Attributes*: `Default FAML Attributes`.
`duration` must have a value.

*Properties*: none (Atomic element).

*Example*:

```
<eyes_up intensity="75" duration="1000"/>Not that
turnip!
```

## eyes_down

*Description*:

Only the eyes turn downward, the head remains looking forward.

*Attributes*: `Default FAML Attributes`.
`duration` must have a value.

*Properties*: none (Atomic element).

*Example*:

```
<eyes_down duration="1000"/>Sorry seems to be the
hardest word.
```

## head_left_roll

*Description*:

The `roll` element animates the roll of the Talking Head in the axial plane. Roll, although subtle in normal movement, is essential for realism.
This element allows the author to script roll movement in the Talking Head, typically in conjunction with other elements, such as nodding and head movements, to add further realism to the Talking Head.

*Attributes*: `Default FAML Attributes`.
`duration` must have a value.

*Properties*: none (Atomic element).

*Example*:

```
<head_left_roll duration="1000"/>Way over yonder.
```

## head_right_roll

*Description*:

The `roll` element animates the roll of the Talking Head in the axial plane. Roll, although subtle in normal movement, is essential for realism.
This element allows the author to script roll movement in the Talking Head, typically in conjunction with other elements, such as nodding and head movements, to add further realism to the Talking Head.

*Attributes*: `Default FAML Attributes`.
`duration` must have a value.

*Properties*: none (Atomic element).

*Example*:

```
<head_right_roll duration="800"/>What a strange
sight!
```

**EyeBrows**

**Notes:**
1: The eyebrow movement element enables the author to script certain eyebrow movements to accentuate words or phrases. MPEG−4 separates the eyebrow into 3 regions, inner, middle and outer. The eyebrow elements affect all three regions of the eyebrow to animate movement.

[individual sections to be moved independently???]
[Should we mention MPEG−4?]

## eyebrow_up

*Description* :

vertical eyebrow movement upwards.

*Attributes*: `Default FAML Attributes`.
`duration` must have a value.

| Name | Description | Values | Default |
|---|---|---|---|
| `which` | which eyebrow to move | both<br>right<br>left | both |

*Properties*: none (Atomic element).

*Example*:

```
<eyebrow_up which="left" duration="1000"/>
Fascinating Captain.
```

## eyebrow_down

*Description*:

vertical eyebrow movement downwards.

*Attributes*: `Default FAML Attributes`.
`duration` must have a value.

| Name | Description | Values | Default |
|---|---|---|---|
| `which` | which eyebrow to move | both<br>right<br>left | both |

*Properties*: none (Atomic element).

*Example*:

```
<eyebrow_down duration="1000"/>I am not happy with
you!
```

## eyebrow_squeeze

*Description*:

Squeezing of the eyebrow together.

*Attributes*: `Default FAML Attributes`.
`duration` must have a value.

*Properties*: none (Atomic element).

*Example*:

```
<eyebrow_squeeze duration="1000"/>Oooh, that's
difficult.
```

**Blinks/Winks**

**Notes**

# blink

*Description*:

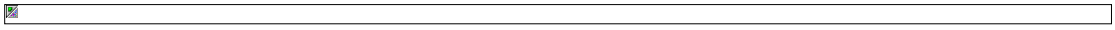
The `blink` element animates a blink of both eyes in the Talking Head animation.
The `blink` element only affects the upper and lower eyelid facial features of the head. By alternating the `intensity` value, the amount of eye closure is affected in the animation. An `intensity` value of 50 denotes 50 percent of the max amplitude for the blinking element, and as such the animation would only reflect half blinking where only half of the eyeball is covered.

*Attributes*: `Default FAML Attributes`.
`duration` must have a value.
[Attributes for left/right start time?]

*Properties*: none (Atomic element).

*Example*:

```
He gave a <blink intensity="10" duration="500"/>
blink, then a <right_wink duration="500"/> wink and
laughed.
```



# double_blink

*Description*:

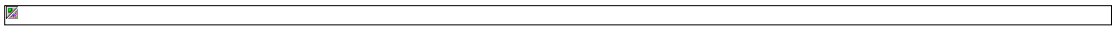
Not all blinks in humans are singular. Observation has shown that double blinking is quite common and can precede changes in emotion or denote sympathetic output.

*Attributes*: `Default FAML Attributes`.
`duration` must have a value.
[Attributes for left/right start time?]

*Properties*: none (Atomic element).

*Example*:

```
<double_blink duration="20"/>What a surprise!!
```



## left_wink

*Description*:

Animates a wink of the left eye. The wink is not just the blinking of one eye, but the head pitch, roll and yaw is affected as well as the outer eyebrow and cheek. The combination of these animated features add to the realism of the wink itself.

*Attributes*: `Default FAML Attributes`.
`duration` must have a value.

*Properties*: none (Atomic element).

*Example*:

```
Nudge, nudge, <left_wink duration="500"/> wink,
<left_wink duration="2000"/>wink.
```

## right_wink

*Description*:

Animates a wink of the right eye. the wink is not just the blinking of one eye, but the head pitch, roll and yaw is affected as well as the outer eyebrow and cheek. The combination of these animated features add to the realism of the wink itself.

*Attributes*: `Default FAML Attributes`.
`duration` must have a value.

*Properties*: none (Atomic element).

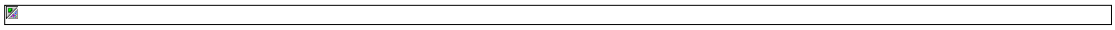
*Example*:

```
Nudge, nudge, <left_wink duration="500"/> wink,
<right_wink duration="2000"/>wink.
```

### *Hyper Text Markup Language (HTML)*

[Should we translate HTML into the ACSS as shown or only allow a minimum subset of well formed HTML?]

```
H1, H2, H3,
H4, H5, H6    { voice−family: paul, male; stress: 20; richness: 90 }
H1            { pitch: x−low; pitch−range: 90 }
H2            { pitch: x−low; pitch−range: 80 }
H3            { pitch: low; pitch−range: 70 }
H4            { pitch: medium; pitch−range: 60 }
H5            { pitch: medium; pitch−range: 50 }
H6            { pitch: medium; pitch−range: 40 }
LI, DT, DD    { pitch: medium; richness: 60 }
DT            { stress: 80 }
PRE, CODE, TT { pitch: medium; pitch−range: 0; stress: 0; richness: 80 }
EM            { pitch: medium; pitch−range: 60; stress: 60; richness: 50 }
STRONG        { pitch: medium; pitch−range: 60; stress: 90; richness: 90 }
DFN           { pitch: high; pitch−range: 60; stress: 60 }
S, STRIKE     { richness: 0 }
I             { pitch: medium; pitch−range: 60; stress: 60; richness: 50 }
B             { pitch: medium; pitch−range: 60; stress: 90; richness: 90 }
U             { richness: 0 }
A:link        { voice−family: harry, male }
A:visited     { voice−family: betty, female }
A:active      { voice−family: betty, female; pitch−range: 80; pitch: x−high }
```



### *Body Animation Markup Langauge (BAML)*

[Input here please, what extra markup is needed?]
1: Movement
2: Stance
3: Uses EML
4: Gestures

## anger

*Description:*

Inherited from EML.

## joy == happy

*Description:*

Inherited from EML.

## neutral

*Description:*

Inherited from EML.

## sadness

*Description:*

Inherited from EML.

## fear

*Description*:

Inherited from EML.

## disgust

*Description*:

Inherited from EML.

## surprise

*Description*:

Inherited from EML.

## dazed

*Description*:

Inherited from EML.

## confused

*Description*:

Inherited from EML.

## bored

*Description*:

Inherited from EML.

### *Dialogue Manager Markup Language (DMML)*

**Dialogue Manager Response**

This language covers the Dialogue Manager's **response** only, not the pattern matching or the overall Knowledge base format.

[Since this work has already begun we need to talk about a preferred subset of AIML that can be used for the DMML.]

Therefore, the AIML tags,

| | |
|---|---|
| `<alice></alice>` | root element of Alice |
| `<category></category>` | categorization of an Alice topic. |
| `<pattern></pattern>` | the user input pattern. |
| `<template>`XXXX`<template>` | the marking of the DM's response |

are not part of DMML.

The XXXX in the above is covered by DMML.. For example, in the Alice fragment:

```
<template>
     My name is <getvar name="botname"/>.
     What is your name?
</template>
```

the DMML would handle the plain text "My name is ", the XML element "`<getvar name="botname"/>`" and the trailing text ".  What is your name?".

**List of DMML elements:**

<star/> indicates the input text fragment matching the pattern '*' or '_'.

<that></that> If previous bot reply matches the THAT this event is fired.
<that/> = <that><star/></that>
<justbeforethat> </justbeforethat>
<justthat> </justthat>

<person2> X </person2> change X from 1st to 2nd person
<person2/> = <person2><star/></person2>
<person> X </person> exchange 1st and 3rd person
<person/> = <person><star/></person>
<srai> X </srai> calls the pattern matches recursively on X.
<sr/> =<srai><star/></srai>

<random> <li>X1</li><li>X2</li> </random> Say one of X1 or X2 randomly
<system>X</system> tag to run the shell command X
<think> X </think> tag pair is to evaluate the AIML expression X, but "nullify" or hide the result from the client reply.

<gossip> X </gossip> Save X as gossip.

<getvar name = "Name Of Variable" default="Default if no variable found"/>
and
<setvar name = "Name Of Variable"> Set it to this </setvar>

**Recognised variable names**
The recognised variable names are:

| preferred | legacy equivalent name | deprecated Atomic tag |
|---|---|---|
| DMbirthplace | botbirthplace | <birthplace/> |
| DMbirthday | botbirthday | <birthday/> |
| DMmaster | botmaster | <botmaster/> |
| DMboyfriend | botboyfriend | <boyfriend/> |
| DMband | botband | <favorite_band/> |
| DMbook | botbook | <favorite_book/> |
| DMcolor | botcolor | <favorite_color/> |
| DMfood | botfood | <favorite_food/> |
| DMmovie | botmovie | <favorite_movie/> |
| DMsong | botsong | <favorite_song/> |
| DMfun | botfun | <for_fun/> |
| DMfriends | botfriends | <friends/> |
| DMgender | botgender | <gender/> |
| DMgirlfriend | botgirlfriend | <girlfriend/> |
| DMmusic | botmusic | <kind_music/> |
| DMlooks | botlooks | <look_like/> |
| DMname | botname | <name/> |
| DMsize | botsize | <getsize/> |
| question | | <question/> |
| name | | <getname/> |
| topic | | <gettopic/> |
| age | | <get_age/> |
| gender | | <get_gender/> |
| has | | <get_has/> |
| he | | <get_he/> |
| ip | | <get_ip/> |
| it | | <get_it/> |
| location | | <get_location/> |
| she | | <get_she/> |
| they | | <get_they/> |
| we | | <get_we/> |
| dialogueManagerName | | |
| dialogueManagerwhoami | | |
| dialogueManagerGender | | |
| dialogueManagerHisHer | | |
| dialogueManagerHimHer | | |
| dialogueManagerHeShe | | |
| dialogueManagerMaster | | |

dialogueManagerBirthPlace
dialogueManagerBirthDay
dialogueManagerAge
dialogueManagerDescription

dialogueManagerFavouriteColour
dialogueManagerFavouriteSport
dialogueManagerFavouriteFood
dialogueManagerFavouritePainter
dialogueManagerFavouriteArtist
dialogueManagerFavouriteBook
dialogueManagerFavouriteMovie
dialogueManagerFavouriteMusic
dialogueManagerFavouriteSong
dialogueManagerFavouriteAlbum

dialogueManagerPurpose
dialogueManagerHomeURL

### *Speech Markup Language (SML)*

The following list is a description of each of SML's elements. As with any XML element, all SML elements are case sensitive; therefore, all SML elements must appear in lower case, otherwise they will be ignored.

**Speech markup Language default Attributes**

| Name | Description | Values | Default |
|---|---|---|---|
| mark | This attribute can be used to set an arbitrary mark at a given place in the text, so that, for example, an engine can report back to the calling application that it has reached the given location. | Character−string identifier for this tag. | No default − optional attribute |

## xml:lang

*Description*:

Following the XML convention, languages are indicated by an `xml:lang` attribute on the enclosing element with the value following RFC 1766 to define language codes. Language information is inherited down the document hierarchy, i.e. it has to be given only once if the whole document is in one language, and language information nests, i.e. inner attributes overwrite outer attributes.

*Example:*

```
<vhml xml:lang="en-US">
  <paragraph>I don't speak Japanese.</paragraph>
  <paragraph xml:lang="ja">
        Nihongo-ga wakarimasen.
   </paragraph>
</vhml>
```

*Notes:*

*1:* The speech output platform determines behavior in the case that a document requires speech output in a language not supported by the speech output platform. This is currently only one of two allowed exceptions to the conformance criteria.

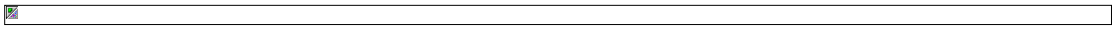
*2:* There may be variation across conformant platforms in the implementation of `xml:lang` for different markup elements. A document author should beware that intra-sentential language changes may not be supported on all platforms.

*3:* A language change often necessitates a change in the voice. Where the platform does not have the same voice in both the enclosing and enclosed languages it should select a new voice with the inherited voice attributes. Any change in voice will reset the prosodic attributes to the default values for the new voice of the enclosed text. Where the `xml:lang` value is the same as the inherited value there is no need for any changes in the voice or prosody.

*4:* All elements should process their contents specific to the enclosing language. For instance, the phoneme, emphasis and break element should each be rendered in a manner that is appropriate to the current language.

*5:* Unsupported languages on a conforming platform could be handled by specifying nothing and relying on platform behavior, issuing an event to the host environment, or by providing substitute text in the Markup Language.

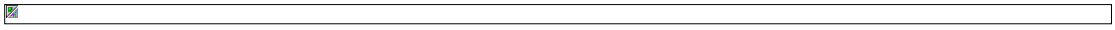
[Should this be for all markups? Body Language as well?]



**anger**

*Description:*

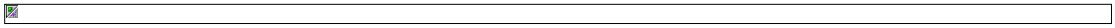
Inherited from EML.



**joy == happy**

*Description:*

Inherited from EML.



**neutral**

*Description:*

Inherited from EML.

**sadness**

*Description:*

Inherited from EML.

**fear**

*Description*:

Inherited from EML.

**disgust**

*Description*:

Inherited from EML.

**surprise**

*Description*:

Inherited from EML.

**dazed**

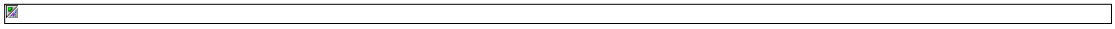
*Description*:

Inherited from EML.

**confused**

*Description*:

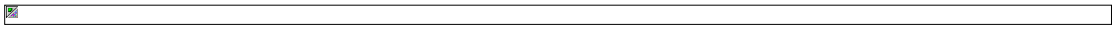
Inherited from EML.



**bored**

*Description*:

Inherited from EML.



## p == paragraph

*Description:*

Element used to divide text into paragraphs. Can only occur directly within a `vhml` element. The `p` element wraps emotion elements.

*Attributes:* none.

*Properties:* Can contain all other elements, except itself and vhml.

*Example:*

```
<p>
      <sad>Today it's been raining all day,</sad>
      <happy>
            But they're calling for sunny skies tomorrow.
      </happy>
</p>
```

*Notes:*

*1:* For brevity, the markup supports `<p>` as an exact equivalent of `<paragraph>`. (Note: XML requires that the opening and closing elements be identical so `<p>` text `</paragraph>` is not legal.).

2: The use of `paragraph` elements is optional. Where text occurs without an enclosing paragraph element the speech output system should attempt to determine the structure using language−specific knowledge of the format of plain text.

## s == sentence

*Description:*

Element used to divide text into sentences. Can only occur directly within a `vhml` element.

*Attributes:* none.

*Properties:* Can contain all other elements, except itself and vhml.

*Example:*

```
<p>
      <sentence>Today it's been raining ,</sentence>
      <happy>
            But they're calling for sunny skies tomorrow.
      </happy>
</p>
```

*Notes:*

*1:* For brevity, the markup also supports `<s>` as exact equivalent of `<sentence>`. (Note: XML requires that the opening and closing elements be identical so `<s>` text `</sentence>` is not legal.). Also note that `<s>` means "strike−out" in HTML 4.0 and earlier, and in XHTML−1.0−Transitional but not in XHTML−1.0−Strict.

2: The use of the `sentence` element is optional. Where text occurs without an enclosing sentence element the speech output system should attempt to determine the structure using language−specific knowledge of the format of plain text.

## say-as

*Description*:

The say-as element indicates the type of text construct contained within the element. This information is used to help specify the pronunciation of the contained text. Defining a comprehensive set of text format types is difficult because of the variety of languages that must be considered and because of the innate flexibility of written languages.

*Attributes*:

The say-as element has been specified with a reasonable set of format types. Text substitution may be utilized for unsupported constructs.

The type attribute is a required attribute that indicates the contained text construct. The format is a text type optionally followed by a colon and a format. The base set of type values, divided according to broad functionality, is as follows:

***Pronunciation Types***

- **acronym**: contained text is an acronym. The characters in the contained text string are pronounced as individual characters.

```
<say-as type="acronym"> USA </say-as>
<!-- U. S. A. -->
```

***Numerical Types***

- **number**: contained text contains integers, fractions, floating points, Roman numerals or some other textual format that can be interpreted and spoken as a number in the current language. Format values for numbers are:
  **"ordinal"**, where the contained text should be interpreted as an ordinal. The content may be a digit sequence or some other textual format that can be interpreted and spoken as an ordinal in the current language; and
  **"digits"**, where the contained text is to be read as a digit sequence, rather than as a number.

```
Rocky <say-as type="number"> XIII </say-as>
<!-- Rocky thirteen -->
Pope John the <say-as type="number:ordinal"> VI
</say-as>
<!-- Pope John the sixth -->
Deliver to <say-as type="number:digits"> 123
</say-as> Brookwood.
<!-- Deliver to one two three Brookwood-->
```

***Time, Date and Measure Types***

- **date**: contained text is a date. Format values for dates are:
  **"dmy"**, **"mdy"**, **"ymd"** (day, month , year), (month, day, year), (year, month, day)
  **"ym"**, **"my"**, **"md"** (year, month), (month, year), (month, day)
  **"y"**, **"m"**, **"d"** (year), (month), (day).
- **time**: contained text is a time of day. Format values for times are:
  **"hms"**, **"hm"**, **"h"** (hours, minutes, seconds), (hours, minutes), (hours).
- **duration**: contained text is a temporal duration. Format values for durations are:
  **"hms"**, **"hm"**, **"ms"**, **"h"**, **"m"**, **"s"** (hours, minutes, seconds), (hours, minutes), (minutes, seconds), (hours), (minutes), (seconds).
- **currency**: contained text is a currency amount.
- **measure**: contained text is a measurement.
- **telephone**: contained text is a telephone number.

```
<say-as type="date:ymd"> 2000/1/20 </say-as>
<!-- January 20th two thousand -->
Proposals are due in <say-as type="date:my">
5/2001 </say-as>
<!-- Proposals are due in May two thousand
and one -->
The total is <say-as type="currency">
$20.45</say-as>
<!-- The total is twenty dollars and
forty-five cents -->
```

When multi-field quantities are specified ("dmy", "my", etc.), it is assumed that the fields are separated by single, non-alphanumeric character.

***Address, Name, Net Types***

- **name**: contained text is a proper name of a person, company etc.
- **net**: contained text is an internet identifier. Format values for net are:
  **"email"**, **"uri"**.
- **address**: contained text is a postal address.

```
<say-as type="net:email"> road.runner@acme.com
</say-as>
```

- The `sub` attribute is employed to indicate that the specified text replaces the contained text for pronunciation. This allows a document to contain both a spoken and written form.

```
<say-as sub="World Wide Web Consortium"> W3C
</say-as>
<!-- World Wide Web Consortium -->
```

*Notes:*

*1:* The conversion of the various types of text and text markup to spoken forms is language and platform-dependent. For example, `<say-as type="date:ymd">` 2000/1/20 `</say-as>` may be read as "January twentieth two thousand" or as "the

twentieth of January two thousand" and so on. The markup examples above are provided for usage illustration purposes only.

2: It is assumed that pronunciations generated by the use of explicit text markup always take precedence over pronunciations produced by a lexicon.

## phoneme

*Description*:

The phoneme element provides a phonetic pronunciation for the contained text. The phoneme element may be empty. However, it is recommended that the element contain human−readable text that can be used for non−spoken rendering of the document. For example, the content may be displayed visually for users with hearing impairments.

*Attributes:*

The alphabet attribute is an optional attribute that specifies the phonetic alphabet.

- **ipa**: The specified phonetic string is composed of symbols from the International Phonetic Alphabet (IPA).
- **worldbet**: The specified phonetic string is composed of symbols from the Worldbet (Postscript) phonetic alphabet.
- **xsampa**: The specified phonetic string is composed of symbols from the X−SAMPA phonetic alphabet.

The ph attribute is a required attribute that specifies the phoneme string.

*Example:*

```
<phoneme alphabet="ipa" ph="t&#x252;m&#x251;to&#x28A;">
tomato </phoneme>
<!−− This is an example of IPA using character
entities −−>
```

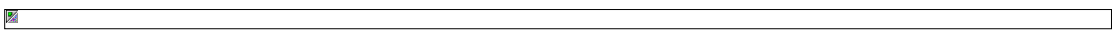
*Notes:*

*1:* Characters composing many of the IPA phonemes are known to display improperly on most platforms. Additional IPA limitations include the fact that IPA is difficult to understand even when using ASCII equivalents, IPA is missing symbols required for many of the world's languages, and IPA editors and fonts containing IPA characters are not widely available.

*2:* Entity definitions may be used for repeated pronunciations. For example:

```
<!ENTITY uk_tomato "t&#x252;m&#x251;to&#x28A;">
... you say <phoneme ph="&uk_tomato;"> tomato </phoneme>
I say...
```

*3:* In addition to an exhaustive set of vowel and consonant symbols, IPA supports a syllable delimiter, numerous diacritics, stress symbols, lexical tone symbols, intonational markers and more.



## voice

*Description:*

The `voice` element is a production element that requests a change in speaking voice.

*Attributes:*

- `gender`: optional attribute indicating the preferred gender of the voice to speak the contained text. Enumerated values are: **"male"**, **"female"**, **"neutral"**.
- `age`: optional attribute indicating the preferred age of the voice to speak the contained text. Acceptable values are of type **(integer)**
- `category`: optional attribute indicating the preferred age category of the voice to speak the contained text. Enumerated values are: **"child"** , **"teenager"** , **"adult"**, **"elder".**
- `variant`: optional attribute indicating a preferred variant of the other voice characteristics to speak the contained text. (e.g. the second or next male child voice). Acceptable values are of type **(integer).**
- `name`: optional attribute indicating a platform−specific voice name to speak the contained text. The value may be a space−separated list of names ordered from top preference down. Acceptable values are of the form **(voice−name−list)**.

*Examples:*

```
<voice gender="female" category="child">
Mary had a little lamb,
</voice>
<!−− now request a different female child's
voice −−>
<voice gender="female" category="child" variant="2">
It's fleece was white as snow.
</voice>
<!−− platform−specific voice selection −−>
<voice name="Mike">
I want to be like Mike.
</voice>
```

*Notes:*

*1:* When there is not a voice available that exactly matches the attributes specified in the document, the voice selection algorithm may be platform−specific.

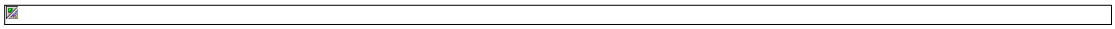
*2:* Voice attributes are inherited down the tree including to within elements that change the language.

```
<voice gender="female">
  Any female voice here.
  <voice category="child">
    A female child voice here.
    <paragraph xml:lang="ja">
      <!−− A female child voice in Japanese. −−>
    </paragraph>
  </voice>
</voice>
```

*3:* A change in voice resets the prosodic parameters since different voices have different natural pitch and speaking rates. Volume is the only exception. It may be possible to preserve prosodic parameters across a voice change by employing a style sheet. Characteristics specified as "+" or "−" voice attributes with respect to absolute voice attributes would not be preserved.

*4:* The `xml:lang` attribute may be used specially to request usage of a voice with a specific dialect or other variant of the enclosing language.

```
<voice xml:lang="en-cockney">Try a Cockney voice
(London area).</voice>
<voice xml:lang="en-brooklyn">Try one New York
accent.</voice>
```



## emphasis

*Description:*

The `emphasis` element requests that the contained text be spoken with emphasis (also referred to as prominence or stress). The synthesizer determines how to render emphasis since the nature of emphasis differs between languages, dialects or even voices.
See also `emphasise_syllable`

*Attributes*:

- `level`: the **"level"** attribute indicates the strength of emphasis to be applied. Defined values are **"strong"**, **"moderate"**, **"none"** and **"reduced"**. The default level is **"moderate"**. The meaning of **"strong"** and **"moderate"** emphasis is interpreted according to the language being spoken (languages indicate emphasis using a possible combination of pitch change, timing changes, loudness and other acoustic differences). The **"reduced"** level is effectively the opposite of emphasizing a word. For example, when the phrase "going to" is reduced it may be spoken as "gonna". The **"none"** level is used to prevent the speech synthesizer from emphasizing words that it might typically emphasize.

*Examples:*

```
That is a <emphasis> big </emphasis> car!
That is a <emphasis level="strong"> huge </emphasis>
bank account!
```

## break

*Description:*

The `break` element is an empty element that controls the pausing or other prosodic boundaries between words. The use of the break element between any pair of words is optional. If the element is not defined, the speech synthesizer is expected to automatically determine a break based on the linguistic context. In practice, the `break` element is most often used to override the typical automatic behavior of a speech synthesizer.
See also `pause` element.

*Attributes*:

- `size`: the **"size"** attribute is an optional attribute having one of the following relative values: **"none"**, **"small"**, **"medium"** (default value), or **"large"**. The value **"none"** indicates that a normal break boundary should be used. The other three values indicate increasingly large break boundaries between words. The larger boundaries are typically accompanied by pauses.
- `time`: the **"time"** attribute is an optional attribute indicating the duration of a pause in seconds or milliseconds. It follows the "Times" attribute format from the Cascading Style Sheet Specification. e.g. "250ms", "3s".

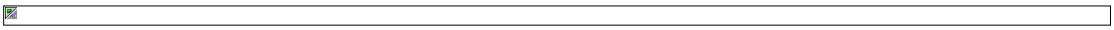
*Examples:*

```
Take a deep breath <break/> then continue.

Press 1 or wait for the tone. <break time="3s"/>
I didn't hear you!
```

*Notes:*

*1:* Using the `size` attribute is generally preferable to the `time` attribute within normal speech. This is because the speech synthesizer will modify the properties of the break according to the speaking rate, voice and possibly other factors. As an example, a fixed 250ms pause (placed with the `time` attribute) sounds much longer in fast speech than in slow speech.



## prosody

*Description*:

The `prosody` element permits control of the pitch, speaking rate and volume of the speech output.
See also `pitch` element.

*Attributes*:

- `pitch`: the baseline pitch for the contained text in Hertz, a relative change or values **"high"**, **"medium"**, **"low"**, **"default"**.
- `contour`: sets the actual pitch contour for the contained text. The format is outlined below.
- `range`: the pitch range (variability) for the contained text in Hertz, a relative change or values **"high"**, **"medium"**, **"low"**, **"default"**.
- `rate`: the speaking rate for the contained text, a relative change or values **"fast"**, **"medium"**, **"slow"**, **"default"**.
- `duration`: a value in seconds or milliseconds for the desired time to take to read the element contents. Follows the Times attribute format from the Cascading Style Sheet Specification. e.g. "250ms", "3s".
- `volume`: the volume for the contained text in the range 0.0 to 100.0, a relative change or values **"silent"**, **"soft"**, **"medium"**, **"loud"** or **"default"**.
  Note, this element sets *only* the volume, and does not change voice quality (e.g. quiet is not a whisper).

### *Relative values*

Relative changes for any of the attributes above are specified as floating−point values: "+10", "−5.5", "+15.2%", "−8.0%". For the pitch and range attributes, relative changes in semitones are permitted: "+5st", "−2st". Since speech synthesizers are not able to apply arbitrary prosodic values, conforming speech synthesis processors may set platform−specific limits on the values. This is the second of only two exceptions allowed in the conformance criteria for a VHML processor.

```
The price of XYZ is <prosody rate="−10%">
<say−as type="currency">$45</say−as></prosody>
```

### *Pitch contour*

The pitch contour is defined as a set of targets at specified intervals in the speech output. The algorithm for interpolating between the targets is platform−specific. In each pair of the form `(interval,target)`, the first value is a percentage of the period of the contained text and the second value is the value of the `pitch` attribute (absolute, relative, relative semitone, or descriptive values are all permitted). Interval values outside 0% to 100% are ignored. If a value is not defined for 0% or 100% then the nearest pitch target is copied.

```
<prosody contour="(0%,+20)(10%,+30%)(40%,+10)">
        good morning
</prosody>
```

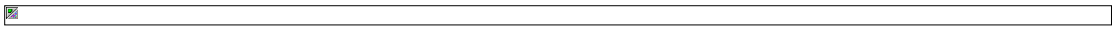
*Notes:*

*1:* The descriptive values ("high", "medium" etc.) may be specific to the platform, to user preferences or to the current language and voice. As such, it is generally preferable to use the descriptive values or the relative changes over absolute values.

*2:* The default value of all prosodic attributes is no change. For example, omitting the `rate` attribute means that the rate is the same within the element as outside.

*3:* The `duration` attribute takes precedence over the `rate` attribute. The `contour` attribute takes precedence over the `pitch` and `range` attributes.

*4:* All prosodic attribute values are indicative: if a speech synthesizer is unable to accurately render a document as specified it will make a best effort (e.g. trying to set the pitch to 1Mhz, or the speaking rate to 1,000,000 words per minute.)



## audio

*Description:*

The `audio` element supports the insertion of recorded audio files and the insertion of other audio formats in conjunction with synthesized speech output. The audio element may be empty. If the audio element is not empty then the contents should be the marked−up text to be spoken if the audio document is not available. The contents may also be used when rendering the document to non−audible output and for accessibility.

*Attributes:*

The required attribute is `src`, which is the URI of a document with an appropriate mime−type.

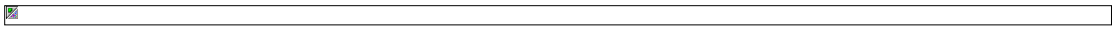
*Examples:*

```
<!−− Empty element −−>
Please say your name after the tone. <audio
src="beep.wav"/>
<!−− Container element with alternative text
−−>
<audio src="prompt.au">What city do you want to fly
from?</audio>
```

*Notes:*

*1:* The `audio` element is not intended to be a complete mechanism for synchronizing synthetic speech output with other audio output or other output media (video etc.). Instead the `audio` element is intended to support the common case of embedding audio files in voice output.

*2:* The alternative text may contain markup. The alternative text may be used when the audio file is not available, when rendering the document as non−audio output, or when the speech synthesizer does not support inclusion of audio files.



## mark

*Description:*

A `mark` element is an empty element that places a marker into the output stream for asynchronous notification. When audio output of the TTS document reaches the mark, the speech synthesizer issues an event that includes the required `name` attribute of the element. The platform defines the destination of the event. The `mark` element does not affect the speech output process.

*Attributes:*

The required attribute is `name`, which is a character string.

*Examples:*

```
Go from <mark name="here"/> here, to <mark name="there"/>
there!
```

*Notes:*

*1:* When supported by the implementation, requests can be made to pause and resume at document locations specified by the `mark` values.

*2:* The mark `name` is not required to be unique within a document.

## emphasise_syllable == emphasize_syllable

*Description:*

Emphasizes a syllable within a word.

*Attributes:*

| Name | Description | Values |
|---|---|---|
| target | Specifies which phoneme in contained text will be the target phoneme. If target is not specified, default target will be the first phoneme found within the contained text. | A character string representing a phoneme symbol. Uses the MPRA phoneme set. |
| level | The strength of the emphasis. (Default level is weak). | weakest, weak, moderate, strong. |
| affect | Specifies if the element is to affect the contained text's phoneme pitch values, or duration values, or both. (Default is pitch only). | p − affect pitch only.<br>d − affect duration only.<br>b − affect both pitch and duration. |

*Properties:* Cannot contain other elements.

*Example:*

```
I have told you <emph affect="b"
      level="moderate">so</emph> many times.
```

## pause

*Description:*

Inserts a pause in the utterance.

*Attributes:*

| Name | Description | Values |
|---|---|---|
| length | Specified the length of the utterance using descriptive value. | short, medium, long. |
| msec | Specifies the length of the utterance in seconds or milliseconds. | A positive number. |
| smooth | Specifies if the last phonemes before this pause need to be lengthened slightly. | yes, no (default = yes) |

*Properties:* empty.

*Example:*

```
I'll take a deep breath <pause length="long"/> and try
it again.
```

## pitch

*Description:*

Element that changes pitch properties of contained text.

*Attributes:*

| Name | Description | Values |
|---|---|---|
| middle | Increases/decreases pitch average of contained text by N% | (+/−)N%, highest, high, medium, low, lowest. |
| range | Increases/decreases pitch range of contained text by N%. | (+/−)N% |

*Properties:* Can contain other non−emotion elements.

*Example:*

```
'Not I', <pitch middle="-20%">said the dog</pitch>
```

## Conformance

**This section is Normative.**
The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this conformance section are to be interpreted as described in RFC 2119

### *Conforming Virtual Human Markup Document Fragments*

A Virtual Human markup document fragment is a *Conforming XML Document Fragment* if it adheres to the specification described in this document including the DTD (see Document Type Definition) and also:

- (relative to XML) is well−formed.
- if all non−Virtual Human namespace elements and attributes and all `xmlns` attributes which refer to non−Virtual Human namespace elements are removed from the given document, and if an appropriate XML declaration (i.e., `<?xml...?>`) is included at the top of the document, and if an appropriate document type declaration which points to the Virtual Human DTD is included immediately thereafter, the result is a valid XML document.
- conforms to the following W3C Recommendations:
    - the XML 1.0 specification (Extensible Markup Language (XML) 1.0).
    - (if any namespaces other than Virtual Human markup are used in the document) Namespaces in XML.

The Virtual Human Markup Language or these conformance criteria provide no designated size limits on any aspect of Virtual Human markup documents. There are no maximum values on the number of elements, the amount of character data, or the number of characters in attribute values.

### *Conforming Stand−Alone Virtual Human Markup Language Documents*

A file is a *Conforming Stand−Alone Virtual Human Markup Language Document* if:

- it is an XML document.
- its root element is a ’speak’ element.
- it conforms to the criteria for Conforming Virtual Human Markup Language Fragments.

### *Conforming Virtual Human Markup Language Processors*

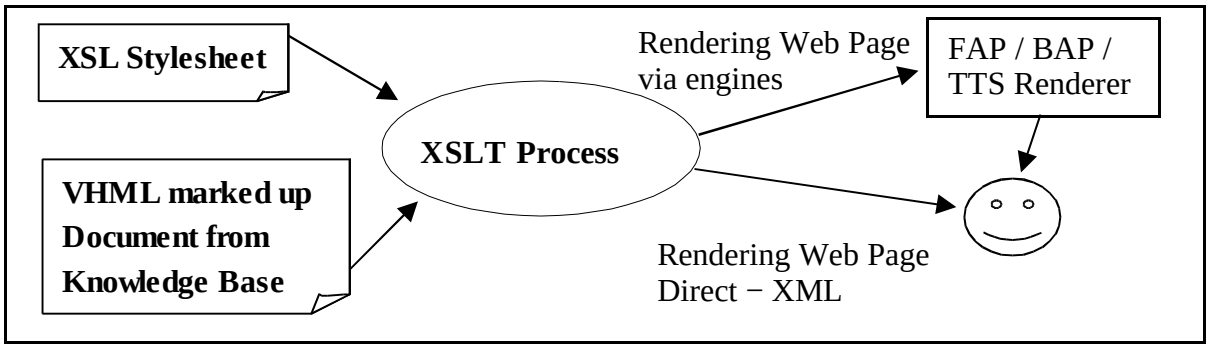
A Virtual Human Markup Language processor is a program that can parse and process Virtual Human Markup Language documents.

In a *Conforming Virtual Human Markup Language Processor*, the XML parser must be able to parse and process all XML constructs defined within XML 1.0 and XML Namespaces.

A Conforming Virtual Human Markup Language Processor **must** correctly understand and apply the command logic defined for each markup element as described by this document. Exceptions to this requirement are allowed when an xml:lang attribute is utilized to specify a language not present on a given platform, and when a non−enumerated attribute value is specified that is out−of−range for the platform. The response of the Conforming Virtual Human Markup Language Processor in both cases would be platform−dependent.

A Conforming Virtual Human Markup Language Processor **should** inform its hosting environment if it encounters an element, element attribute, or syntactic combination of elements or attributes that it is unable to support. A Conforming Virtual Human Markup Language Processor **should** also inform its hosting environment if it encounters an illegal Virtual Human document or unknown XML entity reference.

## The Rendering



## References

**Normative.**


Java Speech API Markup Language

http://java.sun.com/products/java−media/speech/forDevelopers/JSML/index.html JSML is an XML specification for controlling text−to−speech engines. Implementations are available from IBM, Lernout & Hauspie and in the Festival speech synthesis platform and in other implementations of the Java Speech API.

S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, Harvard University, March 1997


**Informative.**


SABLE

http://www.research.att.com/~rws/Sable.v1_0.htm SABLE is a markup language for controlling text to speech engines. It has evolved out of work on combining three existing text to speech languages: SSML, STML and JSML. Implementations are available for the Bell Labs synthesizer and in the Festvial speech synthesizer. The following are two of the papers written about SABLE and its applications:

- SABLE: A Standard for TTS Markup, Sproat et. al. (http://www.research.att.com/~rws/SABPAP/sabpap.htm)
- SABLE: an XML−based Aural Display List For The WWW, Sproat and Raman. (http://www.bell−labs.com/project/tts/csssable.html)

Spoken Text Markup Language

(http://www.cstr.ed.ac.uk/publications/1997/Sproat_1997_a.ps) STML is an SGML language for controlling text to speech engines developed jointly by Bell Laboratories and by the Centre for Speech Technology Research, Edinburgh University.

Microsoft Speech API Control Codes

(http://www.microsoft.com/iit/) SAPI defines a set of inline control codes for manipulating speech output by SAPI speech synthesizers.

VoiceXML Prompts

(http://www.voicexml.com/) The Voice XML specification for dialog systems development includes a set of prompt elements for generating speech synthesis and other audio output that are very similar to elements of JSML and SABLE.

Pelachaud, C. and Prevost, S. (1995) Talking heads: Physical, *linguistic and cognitive issues in facial animation*. Course Notes for Computer Graphics International ’95.

## Acknowledgements

This document was ripped off from various sources as a Working Draft.