

Maria version 2.0 Language Description

This is the official description of Maria Version 2.0. In this document we describe the syntax of the scripting language and the structure of the programs. Anyone interested in writing a Maria script should be familiar with the contents of this document. This is NOT intended to teach programming methods, nor do you need to know this in order to run a Maria script. **Note:** This document is copyright 2006. You are free to distribute this document but not to make changes or additions to it. You may include it with other documents and programs that you or others write as long as it remains in a separate file.

The Design, Scope and Limitations of the Maria Scripting Language

First of all, why another language (scripting or otherwise) in a world full of languages? Maria is a small, task-specific language intended to implement behaviors in simple (read primitive), interactive androids. The language is designed to accommodate advances in technology by using a simplified object-oriented approach that hides low level control inside classes that correspond to parts of the human body. If the android cannot, for whatever reason, accomplish a scripted action, it should fail gracefully; not "die" or begin behaving in an erratic manner. The script itself should use the built-in testing facilities to query the android as to its capabilities; it is, however, up to the script writer to respect the capabilities of the android and work within the confines of those capabilities.

Rather than being strict, I've opted for a fairly flexible interpretation of "object oriented" and have worked towards making the language compact, understandable and easy-to-use rather than a blind adherence to a particular language model. Since Maria scripts are relatively short compared with other languages (Perl, Basic) many of the keywords and operators are fairly verbose. I chose verbose over compact to enhance readability, especially since non-programmers may have an interest in reading the scripts. You'll find that I've freely borrowed elements from Java, C, Smalltalk, Basic, Pascal and other languages. I've also tried to include enough methods (functions) to allow the creation of complex and unique behaviors.

In selecting the level of detail and language terms I've had to make some arbitrary decisions about what was or was not appropriate. I followed a logical, research driven approach but in the end it came down to weighing alternatives. You might disagree with some of the decisions that have been made concerning the implementation of this language and you may be entirely correct. But, and this is a big but, the utility of any language is dependent upon its standard use. Please resist the urge to muck with Maria as this will only break the language and make it difficult for everyone. I've also included some Classes/Instances for specifically for future expansion (not yet implemented) should the need or desire arise.

Finally, the Maria scripting language is only intended for creating behaviors for androids; it is not a general purpose programming language. Please keep this in mind should you have any ideas of trying to extend the language. Here are some aspects of the language and its development/use.

1. Object Oriented

This is a simple object-oriented interpreted scripting language. To simplify things, you may not create or instance new classes; only the classes and instances in the language are available for your use. The reason for this decision is to keep the language simple. Classes in Maria are modelled on the real world; World, Ambiance, Body etc. The classes hide the actual nuts-and-bolts of creating the movement or effect. All the script writer must do is to specify what is required and the interpreter/hardware does the rest. The creator/manufacturer of the android is responsible for the low-level programming of the program/hardware interface in some other language (Java, C++) to allow the interpreter to correctly control the android; this aspect is hidden from the Maria script writer inside the object.

2. The Nature of Human Movement

One thing you'll notice right away is that many of the bodily functions in Maria are encoded as waves with a period, amplitude and duration. This mimics the way the human body works. Heartbeats, brain activity and peristalsis are examples of biological functions that can be described as waves. Waves that represent electrical activity (EEG, ECG) of the nerves tend to be sharp, spike-like waveforms. Muscle movements resulting from these nervous impulses tend to show a more rounded form due to the damping action of the muscle. With Maria, the actual waveform is determined

by the low-level programming done by the creator/manufacturer of the android since this is closely tied to the mechanics of the android and the source of power (electrical, hydraulic, pneumatic).

The androids themselves are stationary and must be placed in position by the user. There are two reasons for this rule: (1) large body movements (force and distance) are required to move adult humans from one position to another. This increases the possibility of injury should something go wrong. (2) the computational aspect of moving a human from one position to another is a problem. It requires large amounts of raw computer power and complex programming. The extensive work on creating walking robots bears this observation out.

3. Emergent Behaviors

A few very simple behaviors can combine to create emergent behaviors; that is behaviors that are not explicitly set by the programmer but emerge from the interaction of disparate behaviors through the real world (as opposed to interaction an internal model of the world). In terms of sexual interaction consider how muscular contractions of the bulbocavernosus muscle pull on the clitoris and result in more stimulation and vasocongestion. Greater vasocongestion leads to more vaginal lubrication. Depending on other variables (i.e. penis in the vagina increasing muscular tension) the total behavior can get complex very quickly but it is a result of a small number of simple, primitive behaviors. Emergent behaviors make programming simpler by eliminating the internal state model of the world. Instead of building an internal state model of the world the android reacts to the actual state of the real world as experienced through its sensors.

4. The Maria Manifesto

A manifesto is a declaration of intentions. In the Maria Manifesto the basics of playing "nice" together between android creators and Maria script writers are outlined. Since it is impossible to predict what permutations of scripts/androids will be connected together, the manifesto outlines things in broad terms. This is not a precise document; the intent is to provide a spirit of cooperation rather than law enforcing cooperation.

Creators should not use existing instances to control new or novel features. For example, if you wish to have laser beams emitting from the eyes, don't write an interface that used the leftHand instance to control it! You'll wind up with scripts that only function with your android and cause odd behavior on other ones. Also, scripts from other users may cause odd behavior in your android. Creators should consider whether the new feature needs script control and how necessary it is in the first place. If you feel the feature is of interest and does need direct control from the script, tell us and we'll consider your request for inclusion in future updates of Maria.

Just because the script CAN control a behavior doesn't mean it has to. For example, your android may simulate breathing (chest rising and falling) but this doesn't necessarily require you to implement the **body.respirationRate()** method to control it. What physical qualities you wish to control are up to you. However, you should make it clear to buyers/users of your android, which features exist and whether or not the script can control them. Unconscious behaviors, outside of what is explicitly controlled by the script may be used to give the android a sense of humanity and avoid the uncanny valley (Mori, Masahiro "The Uncanny Valley", *Energy*, 7(4) pp. 33-35). These unconscious behaviors, while not being controlled directly by the script, may track and repond to behaviors that are controlled by the script (implicit control).

For script writers, always use the **test** argument to determine which items are available and controllable from the script. Although creators should provide for graceful failure at the interface/hardware level, this doesn't absolve you of responsibility to determine what physical resources the android has and to use them in an appropriate manner.

Granularity. Android creators should always design the tactile interface so that it gives both coarse as well as fine sensations. In the case of scripts designed to run simpler androids, the script may not make use of fine sensations. Script writers should also make use of the levels keyword early on in the program to determine how fine the feeling is in the android. For a simple switch the levels will be 2 with the nerve either being 0 (no touch) or 100 (touched). The behavior should be consistant whether the nerves are two levels or many levels.

In humans tactile (sense of touch) and proprioception (the sense of dimensional space) provide us with information about our environment. In the case of Maria the script is designed only for tactile sensations. This does not mean that proprioception cannot be used; if it is appropriate proprioception may be used to provide more levels of touch sensation i.e. sensing the position of the jaw in addition to the pressure inside the mouth.

If you're incorporating what most people might think of as "odd" behavior you should explicitly mention this in your description and in the keywords.

How it works

The model is quite simple. The main program is a continuous loop; the code between **main** and **endmain**. Each time through the loop the instance variables are updated indicating the current status of the androids "nervous system". In the case of the proximity system and the speech-to-text system, the instance variables indicate only a **true** or **false**, a method has to be used to get the actual phrase from the speech-to-text engine. To actuate parts of the android use the methods for that part or sub-part. When an actuation method is used it begins its own thread; additional actuations of the same method are queued up and run in sequence.

The structure of a Maria Script and language syntax

The script is composed of a series of statements. Whitespace (blank lines) and comments are used to make the script more readable but are discarded by the interpreter. A statement may take one or more lines but always ends with a semicolon;

Note: In the method descriptions

<level> indicates a numeric value between 0 and 100 unless otherwise indicated.

<void> indicates no parameters are necessary for this method.

<period> is one cycle of a waveform. integer in milliseconds

<amplitude> is the maximum rise/fall of a waveform, indicated as a relative value 0 to 100, 0 = minimum amplitude
100 = maximum amplitude available

<duration> is the time over which the effect takes place. Integer milliseconds. For continuous (until stopped) use a negative value.

Reserved Words

```
Maria2.01;      # Maria signature. must be first statement in script

persist         # this keyword is used to import and export variables
                  # from the script as text when program is terminated and
                  # import the variables when the program is started again.
                  # variables declared within this block persist from
                  # one script invocation to the next.
                  # Provides a low level of persistence

endpersist     # ends export block

entityname     # quoted text of entity name
description    # holds a description of entity, quoted text only,
                  # no program code

hardware       # a description of the hardware that this program
                  # was (ideally) designed to run on

keywords       # keywords quoted text, comma separated, describing entity
                  # (include sex, sexual orientation etc.)

main           # begins main loop
endmain       # end of main loop

if then        # if statement, begins if block
else          # else statement, must be used inside an if block
endif         # endif, ends if block
```

```

while          # while statement, begins while block
endwhile      # endwhile statement, ends while block
for to by     # for statement (e.g. for int count=1 to 10 by 1)
endfor        # endfor statement, ends for block
void          # void, used in function declarations
              # to indicate no return value

function      # a function declaration, begins function block
return        # return from a function
endfunction   # end of a function block
break         # exit from current loop one level

```

```

# Class or instance method using zero or more parameters
<Class/instance>.<method>(<parameter>, . . .);

```

```

# function call using zero or more parameters
<function name>(<parameter>, . . .);

```

Boolean Operator Types

```

and
or
not equal
equal to
less than
greater than
less than or equal to
greater than or equal to

```

boolean operator types are long for clarity. Many of the people reading
Maria scripts are not programmers and are not used to the short,
cryptic boolean operators used in many popular languages.

Arithmetic Operator Types

```

+      # addition
-      # subtraction
*      # multiplication
/      # division
%      # modulus
^      # exponentiation
=      # assignment operator
(      # left bracket
)      # right bracket

```

Variable Types

```

integer
real
boolean
text
media # primarily implimented for storing sound files for
      # immediate use in voice applications

```

When declaring a variable it may be given a default value. Examples

```

integer tree;
integer bush = 5;
text frankstate = "What is that";

```

Any of these may be made into arrays by using square brackets in the declaration. The size of the array must be used in the declaration. Arrays index begins at 1. Most people (aside from computer scientists) begin a sequence at 1 not zero.

```
integer[23] dayarray;    # this is an array holding integers
dayarray[20] = 349;     # assigning a value to an array element
text[4] sarahsays;     # this is an array holding text strings
boolean[10][6];        # this is a two-dimensional boolean array

# an array may be initialized with a comma separated list
# of values. Values are placed in the array in order,
# any excess values are discarded. Example below
real[5] = 23.7, 89.002, 12.08, 45.01, 13.49;
```

Constants

true
false
null

Universal Method Arguments

These are keywords that can be used as an argument in **any** method.

status - is the method currently executing in a thread. The method replies with true or false or null (if not applicable)

test - tests to see whether this method is available (hardware or software). The method replies with a true or false.

finish - this kills the currently executing thread (if executing) represented by the method and empties the queue. Depending upon the method replies true or false or null

Script Structure

The structure of a Maria script is very simple. Following the **Maria 2.01**; signature line global variables can be declared/initialized, functions declared. Following this section is the **main** script loop. This contains all the code for the program and runs in a loop until script termination. Function declarations should not be placed in **main**, nor should global variables be initialized or declared in **main**

Concurrency

Concurrency is handled implicitly; the script author does not need to explicitly manage it.

Class Hierarchy; Classes, Instances and Methods

Note: The class Breasts and the class Chest are the pretty much equal; only one should be used (Breasts for females, chest for males). The class Pubis is the same as class MonsVeneris; use class Pubis for males, class MonsVeneris for females. You'll also note that in cases where a level is the argument in a method that using the argument **level** method replies with current level. The reason for this is to allow rational level changes for example of an erection and to allow the part to "remember" its own state.

World (abstract class)

Instance: none

Inherits from: nothing

Inherited by: everything

DateTime

Instance: none

Inherits from: World

Inherited by: nothing

Instance Methods

weekday(<void>) - replies with today's name (i.e. Sunday) Note: all date time information is taken from the operating system

day(<void>) - replies with today's day of month (i.e. 21)

year(<void>) - replies with current year (i.e. 2006)

month(<void>) - replies with current month (i.e. December)

hour(<void>) - replies with current hour, 0 to 23 (military style time, 2 digits only)

minute(<void>) - replies with current minute past hour-0 to 59 (2 digits only)

runningSeconds(<void>) - replies with time in seconds since this script was started

Variable (abstract class)

Instance: none

Inherits from: World

Inherited by: Integer, Real, Boolean, Text, Media

Instance Methods

get(<text>) - opens an input window on the screen displaying the <text>, gets an answer from stdin (keyboard) and replies with the appropriate variable

put(<var>) - prints to stdout (standard output)

Integer

Instance: none

Inherits from: Variable

Inherited by: nothing

Instance Methods

toReal(<integer>) - replies with real closest to integer

toText(<integer>) - replies with text of integer

Real

Instance: none

Inherits from: Variable

Inherited by: nothing

Instance Methods

toInteger(<real>) - replies with integer from a real

toText(<real>) - replies with text from a real

floor(<real>) - replies with a real, rounds to the nearest integer less than or equal to the argument

ceiling(<real>) - replies with a real, rounds to the nearest integer greater than or equal to the argument

sqrt(<real>) - replies with closest root of the instance

log10(<real>) - replies with logarithm base 10

logn(<real>) - replies with natural logarithm

exp(<real>) - replies with e raised to the xth power

random(<real>) - replies with random number between 0.0 and 1.0

sin(<real>) - replies with sine

cos(<real>) - replies with cosine

tan(<real>) - replies with tangent

sinh(<real>) - replies with hyperbolic sine

cosh(<real>) - replies with hyperbolic cosine

tanh(<real>) - replies with hyperbolic tangent

asin(<real>) - replies with arc sine (inverse sine)

acos(<real>) - replies with arc cosine (inverse cosine)

atan(<real>) - replies with arc tangent (inverse tangent)

Note: All trigonometric methods use radians as the argument unit

Boolean

Instance: none

Inherits from: Variable

Inherited by: nothing

Instance Methods

toText(<boolean>) - outputs "true" or "false" text

Text

Instance: none

Inherits from: Variable

Inherited by: nothing

Instance Methods

toInteger(<text>) - replies with integer of text or NULL

toReal(<text>) - replies with real of text or NULL

length(<text>) - replies with length of text string with an integer, the number of characters including spaces and non-printing characters

subText(<text>, <text>) - replies with place if one string is found in another, 0 if sub text not found

wordCount(<text>) - replies with the number of words (space separated) in the text string

cutText(<text>, <integer>, <integer>) - replies with substring cut from <integer> to <integer> of original text

match(<text>, <text>) - replies with a number between 0 and 100 telling how good a match was made between the text strings. A perfect match (upper and lower case) with replies with 100. Attempts to correct for spelling mistakes (transpositions, phonetic sounds), plural vs. singular, British vs. American spelling, and internal word punctuation and splitting. co-operate, cooperate etc.

Media

Instance: none

Inherits from: Variable

Inherited by: nothing

Instance Methods

toText(<integer>) - replies with original URL as text string

display(<media>) - display media if possible to standard output

Person (abstract class)

Instance: none

Inherits from: world

Inherited by: Body Sense Intelligence BodyPart

Body

Instance: body

Inherits from: Person

Inherited by: nothing

Instance Methods

respirationRate(<level>) - sets respiration rate, using the argument **level** method replies with current level, 0 is normal resting respiration (about 12 breaths/minute), 100 is maximum respiration (usually about 30 per minute). May also control heartbeat.

arch(<level>) - sets arch of back 0 to 100 (speed/arch), using the argument **level** method replies with current level

scent(<level>) - sets scent (perfume) level, using the argument **level** method replies with current level

toy(<level>) - allows control of a toy, using the argument **level** method replies with current level, 0 toy is off, 100 full on

Note: Describe scent (perfume type) and toy type in **hardware** for proper set-up

Intelligence

Instance: intelligence

Inherits from: Person

Inherited by: nothing

Instance Methods

script(<URL>) - tries to load AI script in AIML format, replies with true if successful or false

query(<text>) - query to AI engine, method replies with <text>

Note: The standard intelligence engine is based upon the ALICE AI engine

Sense

Instance: sense

Inherits from: Person

Inherited by: nothing

Instance Variables

proximityInput - **true** or **false** is there something near the android (within a meter or so)

listenInput - **true** when the speech-to-text engine has translated a phrase, reset to **false** when **heard()** method accessed

Instance Methods

phrases(<URL>) - file with phrases to be listened for, replies with true if successful or false

heard(<void>) - returns a text string of last speech heard

voice(<URL>) - tries to load voice file (diphone database), replies true if successful or false

voiceparam(<text>) - tries to load voice parameters, replies true if successful or false

say(<level>) -sets sound level, 50 is normal volume, 100 is maximum volume, 0 is minimum volume

say(<media sound>) -plays named sound clip through the entity

say(<text>) - plays text through entity using text-to-speech processor (if available)

Note: The standard text-to-speech output is based on the MBROLA project, initiated by the TCTS Lab of the Faculté Polytechnique de Mons (Belgium)

BodyPart (abstract class)

Instance: none

Inherits from: Person

Inherited by: Head, Mouth, Thorax, RightArm, RightHand, LeftArm, LeftHand, Breasts, Chest, Abdomen, Pelvis, MonsVeneris, Vulva, Vagina, Pubis, Scrotum, Penis, Anus, RightLeg, RightFoot, LeftLeg, LeftFoot

Class Variables

self - (the sensation for the total body part instance) sensation level at this moment

Instance Methods

sensitivity(<instance variable>) - replies with sensitivity (number of discrete values) available for that variable i.e. for a simple switch activated nervous system a 2 (2 discrete values or 0 or 1). This is dependent upon the actual sensory mechanism (sensors) and is set by the program and hardware. Cannot be reset or changed by the script.

Head

Instance: head

Inherits from: Bodypart

Inherited by: nothing

Instance Methods

eyesOpen(<boolean>) - sets eyes opened or closed, replies with current state

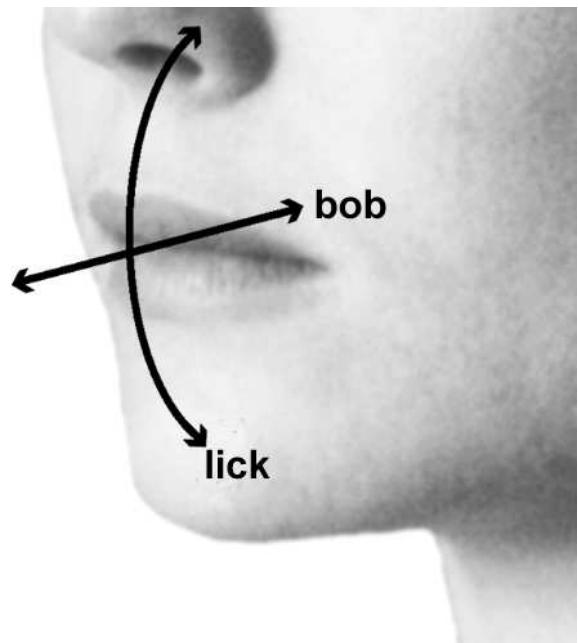
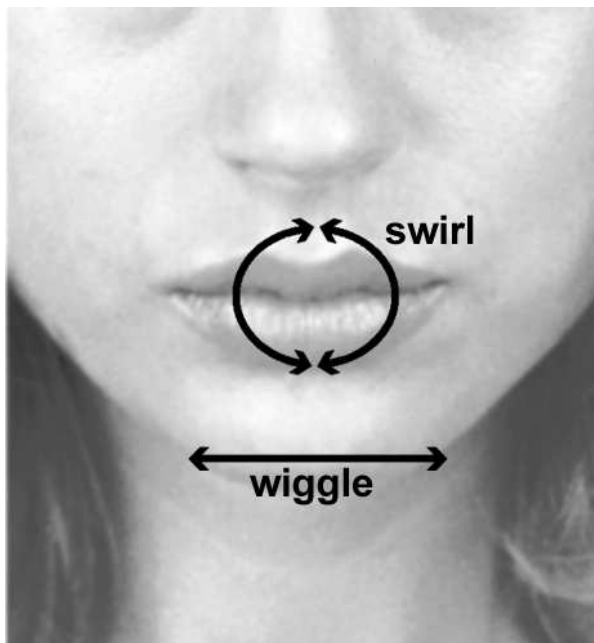
lick(<period>, <amplitude>, <duration>) - linear motion up and down

swirl(<period>, <amplitude>, <duration>) - rotational movement around the axis of the mouth

wiggle(<period>, <amplitude>, <duration>) - side to side movement of head

bob(<period>, <amplitude>, <duration>) - back and forth movement of head

Note: Use head movement in combination with mouth. See diagram below.



Mouth

Instance: mouth

Inherits from: BodyPart

Inherited by: nothing

Instance Variables

lips - sensation at this moment calculated by program

tongue - sensation at this moment calculated by program

Instance Methods

fellatio(<void>) - mouth ready for fellatio. Use other methods for actual stimulation

cunnilingus(<void>) - mouth ready for cunnilingus. Use other methods for actual stimulation

mouthOpen(<level>) - fully open 100, fully closed 0, using the argument **level** method replies with current level

tongue(<period>, <amplitude>, <duration>) - in and out motion of tongue from mouth

nibble(<period>, <amplitude>, <duration>) - light pressure with teeth for stimulation

suck(<period>, <amplitude>, <duration>) - slight vacuum, see **fellatio()** and **cunnilingus()**

kiss(<period>, <amplitude>, <duration>) - lips pursed, slight suction, slight pressure

hum(<period>, <amplitude>, <duration>) - low frequency audible stimulation from deep in the throat

salivation(<level>) - 100 is maximum salivation set by hardware, using the argument **level** method replies with current level

Thorax

Instance: thorax

Inherits from: Bodypart

Inherited by: nothing

RightArm

Instance: rightArm

Inherits from: Bodypart

Inherited by: nothing

massage(<period>, <amplitude>, <duration>) - circular motion to massage chest/breasts/back of partner.

rake(<period>, <amplitude>, <duration>) - linear (back and forth) motion. light pressure to stimulate a partner

jerk(<period>, <amplitude>, <duration>) - hand movement at right angle to fingers, fingers closed around object

RightHand

Instance: rightHand

Inherits from: BodyPart

Inherited by: nothing

Instance Variables

thumb - sensation at this moment calculated by program

indexFingerTip - sensation at this moment calculated by program

middleFingerTip - sensation at this moment calculated by program

Instance Methods

finger(<period>, <amplitude>, <duration>) - motion of fingers to stimulate the genitals of partner. Hand open.

squeeze(<period>. <amplitude>. <duration>) - squeeze fingers around object. Parameters describe pressure exerted on object by squeezing

LeftArm

Instance: leftArm

Inherits from: Bodypart

Inherited by: nothing

massage(<period>, <amplitude>, <duration>) - circular motion to massage chest/breasts/back of partner.

rake(<period>, <amplitude>, <duration>) - linear (back and forth) motion. light pressure to stimulate a partner

jerk(<period>, <amplitude>, <duration>) - hand movement at right angle to fingers, fingers closed around object

LeftHand

Instance: leftHand

Inherits from: BodyPart

Inherited by: nothing

Instance Variables

thumb - sensation at this moment calculated by program

indexFingerTip - sensation at this moment calculated by program

middleFingerTip - sensation at this moment calculated by program

Instance Methods

finger(<period>, <amplitude>, <duration>) - motion of fingers to stimulate the genitals of partner. Hand open.

squeeze(<period>. <amplitude>. <duration>) - squeeze fingers around object. Parameters describe pressure exerted on object by squeezing

Breasts

Instance: breasts

Inherits from: Bodypart

Inherited by: nothing

Instance Variables

nippleRight - includes both the right nipple and the right areola (the dark area surrounding the nipple)

nippleLeft - includes both the left nipple and the left areola (the dark area surrounding the nipple)

Instance Methods

nipples(<level>) - accepts one parameter, level 0 to 100, erection of nipples, using the argument **level** method replies with current level

swell(<level>) - accepts one parameter level a to 100, swelling of breasts, using the argument **level** method replies with current level

Chest

Instance:chest

Inherits from: BodyPart

Inherited by: nothing

Instance Variables

nippleRight - includes both the right nipple and the right areola (the dark area surrounding the nipple)

nippleLeft - includes both the left nipple and the left areola (the dark area surrounding the nipple)

Instance Methods

nipples(<level>) - accepts one parameter, level 0 to 100, erection of nipples, using the argument **level** method replies with current level

Abdomen

Instance: abdomen

Inherits from: Bodypart

Inherited by: nothing

Pelvis

Instance: pelvis

Inherits from: Bodypart

Inherited by: nothing

Instance Methods

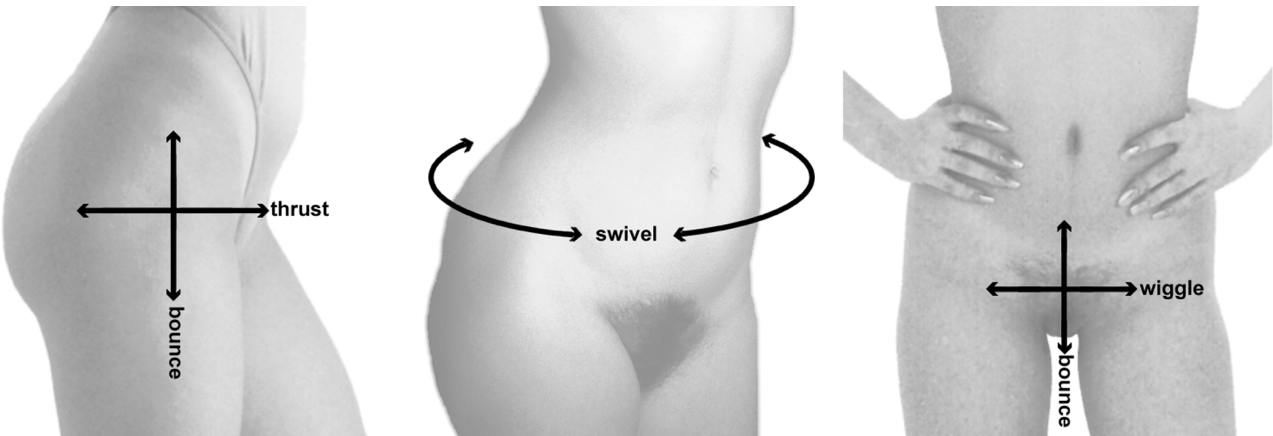
thrust(<period>, <amplitude>, <duration>) - see description below

swivel(<period>, <amplitude>, <duration>) - see description below

wiggle(<period>, <amplitude>, <duration>) - see description below

bounce(<period>, <amplitude>, <duration>) - see description below

Note: Pelvis is used to provide motion for the sex organs which are part of it. Bounce is a back and forth motion of the pelvis along the axis of the vagina/rectum. Swivel is a rotation of the hips around the axis of the vagina/rectum. Wiggle is a side-to-side motion of the hips. Thrust is a back and forth motion of the pelvis along the axis of the erect penis (standing at a right angle to the body). All of these motions can be used by either sex. See diagram below.



MonsVeneris

Instance: monsVeneris

Inherits from: BodyPart

Inherited by: nothing:

Vulva

Instance: vulva

Inherits from: BodyPart

Inherited by: nothing

Instance Variables:

labiaMajora - large external lips of the vulva

labiaMinora - smaller lips of the vulva

clitoralHood - the loose fold of skin covering the glans clitoris

clitoris - the entire clitoris including the glans clitoris and the body (buried under the mons)

urethralMeatus - the urinary outlet, located between the glans clitoris and vagina and the labia minora

Instance Methods:

labiaSwell(<level>) - swelling of the labia (minora and majora), 100 is maximum swell

clitoralErection(<level>) - erection, 100 is maximum, using the argument **level** method replies with current level

Vagina

Instance: vagina

Inherits from: BodyPart

Inherited by: nothing

Instance Variables:

introitus - the external opening of the vagina

urethralSponge - the tissue surrounding the urethra, towards the front of the body, can be felt as a rounded mass in the vagina (also called the G-spot)

deep - the area past the urethral sponge to the cervix

Instance Methods

vasocongestion(<level>) - swelling/tightness of the vagina due to increased vasocongestion, 100 is maximum, using the argument **level** method replies with current level.

tenting(<level>) - vaginal tenting caused by the elevation of the uterus, using the argument **level** method replies with current level

intoitus(<period>, <amplitude>, <duration>) - the action of the muscles at the opening of the vagina (pc and others)

deep(<period>, <amplitude>, <duration>) - the action of the deep muscles of the vagina

lubrication(<level>) - 100 is maximum lubrication set by hardware, using the argument **level** method replies with current level

Pubis

Instance: pubis

Inherits from: BodyPart

Inherited by: nothing

Scrotum

Instance: scrotum

Inherits from: BodyPart

Inherited by: nothing

Instance Variables:

raphe - the "seam" running down the center of the scrotum, dividing left and right

Instance Methods:

testesElevation(<level>) - elevation of testes, 100 furthest away from body, 0 closest to body (most proximal), using the argument **level** method replies with current level

Penis

Instance: penis

Inherits from: BodyPart

Inherited by: nothing

Instance Variables:

urethralMeatus - the opening at the tip of the penis

glans - the entire glans penis (head of penis)

frenulum - the loose fold of skin underneath the glans penis

foreskin - the loose sleeve of skin covering the glans penis

shaft - the shaft of the penis from the glans to the body

base - the area where the penis attaches to and enters the body

Instance Methods:

erection(<level>) - erection, 100 is maximum, using the argument **level** method replies with current level

ejaculation(<period>, <amplitude>, <duration>) - used for both lubrication and ejaculation, maximum amplitude for ejaculation

Anus

Instance: anus

Inherits from: BodyPart

Inherited by: nothing

Instance Variables

perineum - area between vulva/anus or scrotum/anus

sphincter - the anal sphincter

rectum - the rectum, internal area beyond the anus

Instance Methods:

sphincter(<period> , <amplitude> , <duration>) - contraction of anal sphincter muscle

rectum(<period> , <amplitude> , <duration>) - contraction of internal muscles
(muscles used for defecation)

RightLeg

Instance: rightLeg

Inherits from: Bodypart

Inherited by: nothing

RightFoot

Instance: rightFoot

Inherits from: Bodypart

Inherited by: nothing

LeftFoot

Instance: leftFoot

Inherits from: Bodypart

Inherited by: nothing

LeftLeg

Instance: leftLeg

Inherits from: Bodypart

Inherited by: nothing