Yaffaif Game Model XML format

For version 0.25

This document collects together information about the format of the XML used to define the initial state of the game. By it's nature it may not be completely up to date at any given time. The JavaDoc is more authoratative, and the source code definitive, but it is (much) harder to infer the format from these sources.

In theory a different game can be written by replacing the model files with new ones, and leaving the Java bits alone. In practice some things are simpler in code, either by creating new factories and items, or by hand-coding behaviour of complex things like puzzles.

Parsing begins with the file `gamef.xml` in the root of the source tree. This must begin with the `<game>` root element. The convention is for areas to be defined in separate files in the `area` subdirectory, then the id path, so the initial village is in the file `area.val1.village.village.xml`.

The various elements are presented in the order and hierarchy they are typically defined in. Note that some elements (like `Item` which can be in a `Location` or `Container`) can occur in multiple places in the hierarchy.

E&OE

XML Elements

1 game – gamef.model.GameSpace

The root element.

Required Attributes:

`playerRefId`	The id of the character the player will start controlling. This id is dereferenced after loading.
`storyVersion`	The version of the story. eg. "0.24"
`title`	The title of the game.

Optional Attributes:

The values of the variables in `gamef.model.Globals` should also be set here by using attribute names of the form "glob.xyz".

`baseDigEff`	The efficiency of the digestive system in thou.
`moneyDenom`	The name of the currency used in game.
`title`	The title of the game.

1.1 chargen – gamef.model.chars.CharGenCtrl

Controls the attributes the player can alter in character generation and the values they can choose from.

1.2 prototype – gamef.model.items.Prototype

A collection of attributes that gets merged with the real attributes of an object when that object specifies a prototype attribute. Used to define common sets of attribute settings.

Required Attributes:

`id` (or `prevId`)	A unique id for this prototype. This will be appended to the containing area's id. The prevId version can be used to refer to an already partially defined instance.

1.3 area – gamef.model.loc.Area

Specifies a group of related areas or locations within the game.

Required Attributes:

`id` (or `prevId`)	A unique id for this location. This will be appended to the containing area's id. The prevId version can be used to refer to an already partially defined instance.

Optional Attributes:

`environment`	One of the enum values of `gamef.model.loc.EnvEn`. Used to control the initial part of the description string amongst other things. If this is not specified then the attribute of the containing area if any will be used.
`inactiveTimeout`	The time in minutes that must pass after the player has left before the area is deactivated
`recover`	The fully qualified class name of the recovery class. If this is not

	specified then the attribute of the containing area if any will be used.

Elements:

`area`	An area that is part of this area
`bounds`	A bounds object for npcs
`faction`	A faction for npcs
`location`	A location that is part of this area

1.3.1 bounds – gamef.model.loc.Bounds

A list of accessible areas and/or locations that can later be applied to an npc. Any areas or locations must already have been created.

Required Attributes:

`id` (or `prevId`)	the id of the bounds object
`areaLoc`	a comma separated list of area and/or location ids that are included as permissible areas

1.3.2 faction – gamef.model.Faction

Defines a faction within the game. Factions can be hostile or not to other people.

Required Attributes:

`id` (or `prevId`)	the id of the faction object
`memberCond`	a boolean expression that determines membership. When evaluating this expression $0 is the faction, and $1 is the person. Example: `species('kobold',$1)&amp;in([val1.mine],$1)`

1.3.3 location – gamef.model.loc.Location

Defines a location (room) within the game.

Required Attributes:

`id` (or `prevId`)	The id of the location object
`name`	A very short name to identify this place on a movement button. Also appears in the leave message: You leave the dell. **Do not**

	include any trailing punctuation.
short	A short, 2 to 4 word, description of the place. Appears in the exit to message: There is a road, South, to the upper valley. Appears in the location header without the article: --Woodland dell--. **Do not include any trailing punctuation.**
desc	A long description of the place shown when first entering and when a description is requested. The game generates the preamble `You are standing` in according to the character's pose and the environment. The game appends a description of all the exits that the character knows about and a list of all the items that the player can see. Avoid mentioning these in the desc text. **End with a full stop. This attribute can be supplied as inline text.**

Optional Attributes:

`environment`	One of the enum values of `gamef.model.loc.EnvEn`. Used to control the initial part of the description string amongst other things. If this is not specified then the attribute of the containing area if any will be used.
`recover`	The fully qualified class name of the recovery class. If this is not specified then the attribute of the containing area if any will be used.

Elements:

`animal`	Zero or more animals at this location
`character`	Zero or more characters at this location
`container`	Zero or more containers at this location
`exit`	Zero or more exits from this location.
`item`	Zero or more items at this location
`multiexit`	Zero or more multi-route exits from this location
`person`	Zero or more person at this location

1.3.3.1 exit – gamef.model.loc.Exit

Specifies an exit from one location to another, exits usually come in pairs. The pair usually have opposite directions, such as: north and south, up and down, but this isn't required.

Required Attributes:

`id` (or `prevId`)	The id of the exit object
`direction`	The direction of the exit from the player's position in the location. One of the enum values of `DirEn`
`name`	A very short name to identify this exit. **Do not include any trailing punctuation.**
`short`	A short description of the exit. Appears in the description message: There is a `wooden door` to the North. **Do not include any trailing punctuation.**
`to`	The id of the location reached by going through the exit. The location id can be relative to the containing area.

Optional Attributes:

`bitting`	Set the bitting of the lock.
`closeable`	`true` if the exit can be closed, such as a door.
`closeAuto`	`true` if the exit closes on it's own
`desc`	A long description of travelling through this exit. This is currently optional. The args to this message are: • $0 the actor leaving, • $1 the location being left, • $2 the exit being used, • $3 the new location, • $4 whether the travel was forced **End with a full stop. This attribute can be supplied as inline text.**
`distance`	The distance to the destination in metres. Use to calculate energy/time expended. Defaults to 1 if not defined.
`height`	The height of the exit.

`hidden`	`true` if the exit cannot be seen. The player will know about exits they have already used.
`lockable`	true if the exit has a lock
`lockAfter`	true if the exit should be locked after passing through
`locked`	true if the exit is locked
`lockedDesc`	The description to be used after trying an exit and finding it locked with a key you don't have.
`lockOnClose`	true if the exit locks when closed.
`oneWay`	true if the exit can only be used from this side
`open`	true if the exit is open. Defaults to false. Does not apply to exits that are not closeable. The open state is shared with the counterpart exit.

1.3.3.2 multiexit – gamef.model.loc.MultiExit

A kind of exit that can lead to several places. Examples would be a path you can get lost on (`RND` mode), or an elevator room (`CTRL` mode).

Required Attributes:

`id` (or `prevId`)	The id of the multi-exit object
`direction`	The direction of the exit from the player's position in the location. One of the enum values of `DirEn.` Note that this overrides the direction setting of any included exits.
`illogicalDest`	Short description that replaces the target locations's short description when this multi exit is known. Appears in the description message: A twisty path goes North to the `forest`.

Optional Attributes:

Any of the optional attributes that apply to `exit`, and:

`mode`	Mode of exit selection. One of the enum values of `MultiExitEn` (`CTRL`, `RND`, `SEQ`). Default is `RAND`.
`lruDepth`	Number of last used selections to exclude from the next `RAND`

	selection. Default is zero, so selection is random.
`nextIdx`	The index of the next (first) selection.

Elements:

`exit`	One or more exit definitions. Note that the direction attribute of these exits is ignored.

1.3.3.3 item – gamef.model.items.Item

Any item in the game.

Required Attributes:

`id` (or `prevId`)	The id of the item object
`desc`	Description of this item. If the description starts with a lower case letter the game will generate a preamble of "the *name* is" and add a stop at the end. If the description starts with "+" then the text here is appended to any existing description. **This attribute can be supplied as inline text**
`name`	The name of this item, one or two words long such as "chocolate donut". The last word must be the noun that can be made a plural. Appears in descriptions such as "You see a chocolate donut." Use the "measure" to set the natural quantity. So, setting name=”trousers” and measure=”pair” allows the game to construct "You buy three pairs of trousers."

Optional Attributes:

`adjectives`	Comma separated list of adjectives that can be used to refer to this item. Adjectives may belong to a class introduced with a '$' for example "yellow$colour,bent,curved". Used for the text parser.
`image`	Currently unused. Intended to select an icon image.
`known`	Flag that indicates the player knows this object. Defaults to false.
`lightSource`	Flag indicating that this item emits light. Locations that are dark will require a light source for the player to see.
`manipulated`	**Do not set.** Intended to be used to compact saves.
`measure`	Name of the measure of this item. Measures are things like "pair",

	"bottle", "yard". Used when generating names and plurals like "two bottles of beer" (not "two bottle of beers").
`measureArt`	Flag indicating that the indefinite article ('a' or 'an') is needed with the measure when generating the name. True for cases like "part of a chicken".
`multiple`	Flag to indicate this item represents multiple items. Used in TextBuilder to generate appropriate endings.
`noArt`	Flag to indicate neither article should be used. Set for named people. Defaults to false.
`pluralName`	The plural of the name if this can't be computed using the rules of `Noun`. There aren't many it can't handle, but there are some like woman/women.
`respawn`	Flag to indicate this object should respawn when the player is absent. Default false. A more advanced respawn control is anticipated.
`static`	Flag to indicate that this item cannot be moved. Default false.
`suppress`	Flag to disable this item, effectively making it disappear to the rest of the game until cleared.
`surface`	Flag to indicate this item is a surface on to which other things can be put. If it isn't a surface, things go inside.
`unique`	**Do Not Set.** Flag to indicate that this is the only object of its kind in the location or container. Used to choose the definite or indefinite article. Set by the container.

Other kinds of item can be created by using factories, setting the classname attribute, or the following elements instead of item:

1.3.3.3.1 drink – gamef.model.items.Drink

A drink that you can consume.

Required Attributes:

All the required attributes of Item, and:

kcals	The calorific content of the drink
volCc	The volume of the drink

Optional Attributes:

All the optional attributes of Item, and:

abv	Alcohol by Volume – a measure of how alcoholic a drink is.
flavours	A comma separated list of flavour names from FlavourLib.
fluid	Already set to true because this is a drink

1.3.3.3.2 food – gamef.model.items.Food

A food you can eat.

Required Attributes:

All the required attributes of Item, and:

kcals	The calorific content of the food
volCc	The volume of the food

Optional Attributes:

All the optional attributes of Item, and:

flavours	A comma separated list of flavour names from FlavourLib.
fluid	Already set to false because this is a food

1.3.3.3.3 key – gamef.model.items.Key

A key to a lock.

Required Attributes:

All the required attributes of Item, and:

bitting	The bitting on they key, which must match the lock.

1.3.3.4 container – gamef.model.items.Container

A container is a kind of item that other kinds of item can be put in or on.

Required Attributes:

`id` (or `prevId`)	The id of the container object
`desc`	Description of this item. If the description starts with a lower case

	letter the game will generate a preamble of "the *name* is" and add a stop at the end. If the description starts with "+" then the text here is appended to any existing description. **This attribute can be supplied as inline text**
`name`	The name of this item, one or two words long such as "chocolate donut". The last word must be the noun that can be made a plural. Appears in descriptions such as "You see a chocolate donut." Use the "measure" to set the natural quantity. So, setting name=”trousers” and measure=”pair” allows the game to construct "You buy three pairs of trousers."

Optional Attributes:

The optional attributes of `Item`, and:

`closeable`	Flag to indicate if the container can be closed. Only useful for non-surface containers.
`locked`	Flag to indicate if the container can be locked. Only useful for non-surface containers.
`open`	Flag to indicate if the container is open. Only useful for non-surface containers.
`transparent`	Flag to indicate if the container is “transparent” and the things inside can be seen (but not reached if it is closed). Transparency can be because the container is made of a transparent material, or because it has gaps in it like the bars of a cage.

1.3.3.5 animal – gamef.model.chars.Animal

An animal that only displays basic intelligence. Has a set of standard RPG stats.

Required Attributes:

`id` (or `prevId`)	The id of the animal object
`desc`	Description of this animal. If the description starts with a lower case letter the game will generate a preamble of "the *name* is" and add a stop at the end. If the description starts with "+" then the text here is appended to any existing description. **This attribute can be supplied as inline text**
`name`	The name of this animal, one or two words long such as "chocolate donut". The last word must be the noun that can be made a plural. Appears in descriptions such as "You see a chocolate donut." Use the "measure" to set the natural quantity. So, setting name=”trousers” and measure=”pair” allows the game to construct "You buy three pairs of trousers."

Optional Attributes:

The optional attributes of `Animal`, `Actor`, `Container`, `Item`, and:

dndAc	Set the armour class using Osric/DnD values where 10 is naked, 0 is very good and -10 is basically perfect.
level	Sets the RPG experience level and adjusts hpMax.
mass	Sets the current mass in grams.
stats	A collection of RPG stats – see below for attributes.

The attributes that can be set in stats:

stats.fixLevel	Fixed experience level. Does not improve.
stats.hpMax	Sets maximum hit points. Heals to this level.
stats.level	Sets the experience points to the minimum for the given level, allows it to improve.
stats.sexLevel	Sets the sexual experience points to the minimum for the given level, allows it to improve.

Optional Attibutes from Actor:

`bounds`	The id of the bounds for this actor.
`job`	The classname of the current job for this actor. This job will be locked in place until JobList is unlocked.
`location`	**Do not use.** Their current location. This is set by the location they are part of.
`myTime`	**Never use!**
`player`	**Do not use!** Set the player through the GameSpace.
`pose`	Set the pose using a `PoseEn` name: `FLOAT`, `FLY`, `KNEEL`, `LIE`, `SIT`, `STAND`, `SWIM`.
`state`	Set the state uning an `ActStateEn` name: `AWAKE`, `ASLEEP`, `COMA`, `DEAD`, `HOT`, `SEX`. Use with caution!
`subLoc`	The id of a Container the character is in/on. The container must be in this location.
`talkColour`	The colour used in the GUI for this actor's utterances. One of the named colours from `ColourListEn.INK`. A colour will be chosen automatically otherwise.

1.3.3.6 character – gamef.model.chars.IntelPerson

An intelligent NPC (or player).

Required Attributes:

The following attributes are required except when the character has been created by a factory.

`id` (or `prevId`)	The id of the character object.
`height`	Their height in millimetres
`initialMass`	Their initial mass at the start of the game.
`sex`	Their sex using a name from `GenderEn`: `neuter`, `male`, `female`, `herm`

Optional Attributes:

The optional attributes of `Animal`, `Container`, `Item`, and:

`familyName`	The family name of the person
`givenName`	The given name of the person
`homeLocation`	Id of their home location (not used?)
`mass`	**Do not use.** Their current mass
`persName`	Name object
`persName.family`	Family name. Eg. Smith
`persName.given`	Given name. Eg. John
`persName.letters`	Any stuff after their name. Eg. PhD (Cantab)
`persName.nick`	Any nick name. Eg. "the knuckles"
`persName.title`	Title as written. Eg. Doctor, mister, misses, Professor
`persName.titleAbbr`	Title abbreviation. Eg. Dr, Mr, Ms, Prof.
`origShape`	**Do not use.** Set by player character generator. Their original shape.
`origSpecies`	**Do not use.** Set by species.
`sexuality`	Their sexual preference from SexualityEn: `HETRO`, `BICURIOUS`, `BISEXUAL`, `HOMO`, `ASEXUAL`
`species`	Their species from `gamef.model.species.SpeciesEnum`.This sets the species of their whole body to a consistent value.

Elements:

`stats`	One set of body dimensions.

`fatter`	One set of fatter dimensions.
`mind`	One set of mind settings.
`wearing`	One set of clothes to wear.
`jobs`	One set of jobs to do.
`talk`	One set of talk topics.

1.3.3.6.1 stats – gamef.model.chars.BodyStats

Body dimensions when thin.

Required Attributes:

`chestWidthMm`	The width of their chest in millimetres.
`bustExtnMm`	How far their boobs (or moobs) stick out in millimetres.
`waistWidthMm`	The width of their waist in millimetres.
`bellyExtnMm`	How far their belly sticks out in millimetres.
`buttExtnMm`	How far their butt sticks out in millimetres.
`thighDiaMm`	The width of their thighs in millimetres.

1.3.3.6.2 fatter – gamef.model.chars.BodyStats

Body dimensions when fatter. The game will extrapolate from the stats and fatter settings for the current mass. Has the same attributes as stats.

1.3.3.6.3 mind – gamef.model.chars.mind.Mind

The mental state of the character.

Optional Attributes:

`dislike`	Set a flavour that is disliked. Takes two parameters: the flavour from FlavourLib, and the degree to which it is disliked. Recommended values 0 .. 100 (strong dislike).
`drunkTarget`	Define the curve that controls drinking. Give each state (`SOBER`, `TIPSY`, `DRUNK`, `EMOTIONAL`, `PARALYTIC`, `UNCONCIOUS`) a rank between 0 (do not want) and 1000 (my heart's desire). It is assumed that the score for `DEAD` is zero.
`like`	Set a flavour that is liked. Takes two parameters: the flavour from FlavourLib, and the degree to which it is liked. Recommended values 0 .. 100 (strong like).
`massTarget`	The mass the character would like to reach in grams.

1.3.3.6.4 wearing

Specifies what the NPC is wearing at the start of the game.

Elements:

`clothing`	Zero or more items of clothing.

1.3.3.6.4.1 clothing – gamef.model.items.clothes.Clothing

An item of clothing. Most clothing will be created using the clothing factory class for simplicity.

Required Attributes:

`id` (or `prevId`)	The id of the item object
`colour`	The main colour of the clothing.
`desc`	Description of this clothing. The description field is used as a long version of the name. It should not include information from elsewhere, such as main colour or material. The game will generate a preamble of "You look at the *(damaged) colour material*" then append this description. Afterwards it may add "that you are wearing", "that she is wearing". Next comes the descExtra. Finally if if is broken, dirty, or undone. If the description starts with "+" then the text here is appended to any existing description. **This attribute can be supplied as inline text**
`layer`	One of the `ClothLayerEn` names determines the layer of the clothing: `SKIN`, `UNDER`, `LOWER`, `UPPER`, `TOP`, `ACCESSORY`
`name`	The name of this item, one or two words long such as "chocolate donut". The last word must be the noun that can be made a plural. Appears in descriptions such as "You see a chocolate donut." Use the "measure" to set the natural quantity. So, setting name="trousers" and measure="pair" allows the game to construct "You buy three pairs of trousers."
`part`	The `ClothPartEn` name that indicates which parts of the body are covered up by the clothing.

Optional Attributes:

The optional attributes of Item, and:

`acThou`	The armour class in thou. 0 is as good as naked, 1000 is "perfect".
`broken`	True if the clothing is broken. Broken clothing falls off.
`damaged`	True if the clothing is damaged.

descExtra	Additional description of the clothing. During expansion $0 is who is looking, $1 is the clothing. It should be a complete sentance.
design	The design on the clothing, such as “gingham”, “paisley”
fastener	One of the FastnerEn names. Default is NONE. Other options are BUCKLE, BUTTON, CLASP, HOOK, LACE, LACES, OPEN, PERMENANT, TOGGLE, ZIP, ZIPS
fitted	**Do not use.** Assigned by the game. Clothing is created in an unfitted state and becomes fitted when it is first worn.
material	One of the MaterialEn names. Default is CLOTH.
numFastener	The number of fasteners.
numUndone	The number of undone fasteners.
stretchy	The degree to which the clothing stretches in thou 0 is perfectly elastic, 1000 is unchanging.
transparent	True if what is underneath can be seen. So a string vest, fishnets, sandals, or organza is ‘transparent’.

1.3.3.6.5 jobs – gamef.model.chars.job.JobList

A list of the jobs the character can do.

Elements:

timing	Zero or more job timings.

1.3.3.6.5.1 timing – gamef.model.chars.job.JobEntry

Definition of when an NPC should be doing a job.

Required Attributes:

allDays	True to do the job on every day
allTime	True to do the job thought a day
days	Control which days to do the job on with a comma separated string. all turns all days on, a three letter day name toggles the day. Examples tue,thu - do the job on Tuesday and Thursday all,fri – do the job every day except friday
priority	The base prority of the job in thou. If more than one job is scheduled the higher priority one will be done.
startStop	Set the hours in the day to do the job. Use the format hh:mm-hh:mm. If the job continues past midnight 18:00-04:00 works fine.
weekdays	True to work weekdays

`weekend`	True to work weekends

Elements:

`job`	Definition of a job to carry out during the given times

1.3.3.6.5.1.1 job – gamef.model.chars.job.JobBase

Defines the job to be done at the given time. Different jobs have different attributes. In general jobs will be created from the job factory.

Required Attributes:

`id`	the id for this job

Explore:

Move about, picking up things occasionally.

`pick`	the probability of picking something up in thou
`travel`	the probability of moving to a new location in thou

GuardPatrol:

Patrol a fixed route, waiting at each location, and challenging foes.

`dwellMins`	the time to spend at each patrol location in mins
`foeTest`	an expression to determine if someone is a foe that should be attacked. `$0` is the person, `$1` is guard
`route`	a comma separated list of location ids to visit in order. If there's not a direct connection between one room and the next a route will be found.It is assumed that the route is circular and the guard will go from the last entry to the first. If a route cannot be found then the guard returns to the first location and start over.

Idle:

Wander about, eating when hungry. Does not stray out of defined bounds.

`foodLoc`	a location where food may be found

Sleep:

Go to sleep, rest, and recover hp.

`bed`	the id of the character's bed (optional)
`loc`	the location to sleep in

1.3.3.6.6 talk – gamef.model.talk.TalkList

A list of all the questions and answers that can be used with an NPC.

Elements:

`question`	Zero or more questions the PC can ask, or NPC responses

1.3.3.6.6.1 question – gamef.model.talk.QuAndAn

Specifies a question and it's responses.

Required Attributes:

`id`	The id for this question. The convention is to begin true question-answers with 'q', reply only with 'r', and job-related ones with 'j'.
`question`	The text of the question. $0 is the person asking the question. $1 is the person answering the question. $2 is null, and is notionally the narrator. Use {talk,0,1} for the direct speech of the questioner. Use {narrate,0} for the narrative voice. **End with normal punctuation or `{se}`. This attribute can be supplied as inline text.**

Optional Attributes:

`condition`	An expression that must be true for the question to be offered to the player. $0 is the person asking the question. $1 is the person answering the question.
`enabled`	Flag that must be true to offer the question to the player. Defaults to true.
`random`	Flag to indicate a random answer should be used. Defaults to false so the answers are used in order.
`short`	Short text to put on the UI button after the style when offering this question to the player. This is prefixed by the style.
`style`	One of the `QuStyleEn` names. The style affects the presentation on the button, and which verb helpers are used for this action when parsing textual input. The default is ASK.

topicNouns	A list of nouns associated with the indirect object (the thing being talked about) of this question. Nouns should be comma separated. The noun text begins with the noun singular. It may optionally have a '/' and the plural version. It may then optionally have a '$' and a sub category.

Styles:

ASK	Ask npc about xyz Question npc about xyz Talk (to) npc about xyz	AskHelper
ASKTELL	Ask npc about xyz Question npc about xyz Talk (to) npc about xyz Tell npc about xyz	AskTellHelper
ASKFOR	Ask npc for xyz Talk (to) npc about xyz	AskForHelper
BUY	Buy xyz Purchase xyz Buy xyz from npc Purchase xyz from npc	BuyTalkHelper
GIVE	Give xyz to npc	
GIVESHOW	Give xyz to npc Show xyz to npc	
SHOW	Show xyz to npc	
TELL	Tell npc about xyz Talk (to) npc about xyz	TellHelper
NO	Say no	NoHelper
YES	Say yes	YesHelper

Elements:

answer or sell or work	Zero or more answers the NPC can give.

1.3.3.6.6.1.1 answer – gamef.model.talk.Answer

An reply to a question posed by the PC (or another NPC). Also used to define statements made by NPCs.

Required Attributes:

format	The text of the response. $0 is the person asking the question. $1

	is the person answering the question. $2 is null, and is notionally the narrator. Use `{talk,1,0}` for the direct speech of the answerer. Use `{narrate,0}` for the narrative voice. **End with normal punctuation or `{se}`. This attribute can be supplied as inline text.**

Optional Attributes:

`disableId`	Id of a question to disable so it is no longer offered to the player after this answer is given.
`enableId`	Id of a question to enable so it is offered to the player after this answer is given.

Elements:

`change`	Zero or more changes to make to the model when the answer is given.

1.3.3.6.6.1.1.1 change – gamef.model.talk.Change

A change to make to the model when then answer is given.

Required Attributes:

Preferred:

`expr`	An expression that changes the model. $0 is the questioner, $1 is the answerer

Or:

`objExpr`	An `reflection-expr` that identifies an object and setter method
`paramsExpr`	Optional parameters to the setter method

1.3.3.6.6.1.2 sell – gamef.model.talk.Sell

A kind of answer that costs money and results in an item being added to the questioner's inventory if they can afford it. Note that if it can't be afforded the enableId, disableId, and changes are not applied.

Required Attributes:

The required attributes of `Answer`, and

`cost`	The cost of the item. May be zero.
`lessMoney`	Text to output if you cannot afford the item.
`part`	The name of the part for the factory class to make
`partFactory`	The name of the factory that creates the new item

Optional Attributes:

The optional attributes of `Answer`, and

`newDesc`	A replacement description for the factory item.
`newFlav`	A list of flavours to apply to the facotry item if it is a Consumable
`newName`	The new short name of the item.

1.3.3.6.6.1.3 work – gamef.model.talk.Work

A kind of answer where the questioner ends up doing some work. The messages output are defined by a Java class, as it's rather complicated. In general there will be a start message, possibly some NPC messages, and a stop message.

Required Attributes:

Note that it is not necessary to define the format attribute.

`earn`	The amount of money to earn
`mins`	The length of the task
`workClass`	The classname of to instantiate to describe the start and end of the work. This class must implement `WorkIf`.

Optional Attributes:

The optional attributes of Answer.

1.4 quest – gamef.model.quest.Quest

Definition of a quest for the player.

A quest fails if any of its non-optional parts are failed. A quest succeeds if all of its non-optional parts are successful. A part can also have the state "over" that means it is finished but doesn't contribute to success or failure.

Required Attributes:

`id` (or `prevId`)	The id of the quest object
`exp`	The experience awarded for completing the quest
`name`	A short name for the quest.

Elements:

`part`	One or more parts (steps) in the quest

1.4.1 part – gamef.model.quest.QuestPart

Definition of a step in a quest.

Required Attributes:

`id` (or `prevId`)	The id of the quest part object
`name`	A short name for the sub-quest.

Optional Attributes:

`failChanges`	A comma separated list of changes to the quest parts to be applied if the quest part is failed. `+id` starts another quest part `-id` stops another quest part `!id` fails another quest part
`failure`	A boolean expression that returns true when this part is failed.
`optional`	`true` if this part is optional to succeeding or failing the main quest.
`prime`	`true` if this part should be started when the quest is started.
`succChanges`	A comma separated list of changes to the quest parts to be applied if the quest part is completed successfully. `+id` starts another quest part `-id` stops another quest part `!id` fails another quest part
`success`	A boolean expression that returns true when this part is complete.

2 File format

The files have to be fully compliant XML, rather than sloppy HTML style. The parser is very unforgiving! As usual where the characters '<', '>' are needed (eg. in expression strings) they should be replaced by their character entities `<` and `>`.

2.1 Comments

Standard XML comments can be used: start with `<!--` and end with `-->`.

2.2 Includes

To include one XML file in another, which is pretty much essential to keep things readable:

- The including file must have `xmlns:xi="http://www.w3.org/2001/XInclude"` as an attribute of its root element.
- At the point where the included file is to be read add something like `<xi:include href="jar:/area/val1/valley.xml"/>`
- The included file should have the preamble `<?xml version="1.0" encoding="UTF-8"?>`

It is suggested that each area and each character should be defined in their own file.

2.3 Forward references

It is impossible to avoid having to use forward references when defining exits from one location to another – one of the locations will not be defined yet. The work around is to create the object initially with just an `id` attribute, then return to it later using the `prevId` attribute.

Example:

```
<!-- Forward refs -->
<location id="tunnel"/>
<location id="entCham"/>
...
<location prevId=”tunnel”>
      <exit
            id="exEnt"
            direction="n"
            to="entCham"
      />
</location>

<location prevId=”entCham”>
      <exit
            id="exTun"
```

```
            direction="n"
            to="tunnel"
        />
    </location>
```

Note: forward references can span mutiple source files as required.

2.4 Creation order

The XML is read and the corresponding objects are created in the order that they appear. Specifically:

- The object is created when the corresponding element is encountered.
- Any attributes of that element are applied.
- Then any sub-objects are created and added to the object.
- Any free text is applied to the object.

Because of this, it is not possible for an attribute to use a reference to the id of one of its sub-objects, as they won't exist when the attribute is applied.

2.5 Creation control

By default, the parser creates new objects of the types given for each element. However, this can be overriden by the following faux-attributes:

`class`	An object of the given fully qualified class name is created using its default constructor. Then the setSpace(GameSpace) method is then invoked if it is derived from GameBase.
`factory`	Specifies the name of a factory to create the object. The fully qualified classname of the factory is gamef.factory.NameFactory.
`facpart`	The name of a method in the factory class to invoke to create the object
`prevId`	The object is already created with the given id – no new object is required. See 2.3 Forward references
`prototype`	Merge the attibutes contained in the named prototype with the ones specified for the element, then continue normal attribute processing. Note: a prototype can specify class, factory, and facpart!

Note:

- That factory and facpart must be used together.
- Using class overrides factory/facpart.

- When using forward references class, factory, facpart, prototype must be specified in the initial creation if they are to have any effect.

2.6 Attributes

Attributes to an element are applied by using reflection to find setters for the atribute names. Attribute names can include the '.' character so setters on deeper objects can be used.

2.7 Inline text

Text inside an element, but outside other elements is inline text. This text is concatenated together and whitespace reduced and is the inline text. Any inline text can be used to set a specific attribut, usually the description attribute.

2.8 Ids

Ids form a notional tree structure. When an id is assigned to an object it becomes *parent-id*.*my-id*. Creating a Food with the id "cake" in the location "val1.home.kitchen" causes its id to become "val1.home.kitchen.cake". The id does not change if the player picks it up and moves it somewhere else; it stays the same for the duration.

The root GameSpace keeps a map of ids to game objects, which is how lookup by id works. If the player were to eat the above mentioned cake, then it is removed from the game. However, it may persist in the id map until the object is garbage collected sometime later. Before the game is saved an attempt is made to force the garbage collection of game objects.

3 Expressions

The following is an attempt to explain how to write expressions. There are actually two separate expression parsers in the game: one works on the fly from the expression string, the other pre-compiles the expression into objects. Both are intended to work the same way, but... some of this may be a work in progress.

3.1 Syntax

The table below describes the syntax:

Name	BNFish syntax	Examples
expression	*or-expr*	`x`
or-expr	*and-expr* *and-expr* `|` *or-expr*	`x` `x | y | z`
and-expr	*eq-expr* *eq-expr* `&` *and expr*	`x` `x & y & z`

eq-expr	*rel-expr*
	rel-expr `==` *rel-expr*
	rel-expr `!=` *rel-expr*
		`x`
		`x == y`
		`x != y`

rel-expr	*add-expr*
	add-expr `>` *add-expr*
	add-expr `>=` *add-expr*
	add-expr `<=` *add-expr*
	add-expr `<` *add-expr*
		`x`
		`x > y`
		`x >= y`
		`x <= y`
		`x < y`

add-expr	*mul-expr*
	mul-expr `+` *add-expr*
	mul-expr `-` *add-expr*
		`x`
		`x + y + z`
		`x – y - z`

mul-expr	*unary-expr*
	unary-expr `*` *mul-expr*
	unary-expr `/` *mul-expr*
	unary-expr `%` *mul-expr*
		`x`
		`x * y * z`
		`x / y / z`
		`x % y % z`

unary-expr	*term*
	`!`*term*
	`-`*term*
		`x`
		`!x`
		`-x`

term	*int-literal*
	str-literal
	enum-literal
	bool-literal
	`(` *expression* `)`
	function-call
	reflect-expr
		`42`
		`'hello'` or `"hello"`
		`AWAKE`
		`true`
		`( x )`
		`isDead(x, y)`
		`$0`

function-call	*identifier arg-list*	`isDead($0)`

arg-list	`()`
	`(`*arg*`)`
	`(`*arg*`, `*arg*`, ... )`
		`()`
		`(x)`
		`(x, y)`

arg	*expression*	`x`
reflect-expr	*reflect-root* *reflect-root* `.` *reflect-path*	`$0` `$0.body.isMale`

reflect-root	`$`*int-literal*
	`/`
	`[`*id*`]`
		`$0`
		`/`
		`[val1.home.chicken]`

reflect-path	*reflect-method* *reflect-method*`.`*reflect-path*	`body` `body.head.hair.lengthMm`
reflect-method	*identifier* *identifier arg-list*	`body` `rollIdx(10)`
id	*identifier* *identifier* `.` *id*	`val1` `val1.mine.entrance`
identifier	*[a-zA-Z][0-9a-zA-Z]**	`legLength2`
str-literal	`‘` *chars* `‘` `“` *chars* `“`	`‘hello’` `“goodbye”`
int-literal	*[0-9][0-9]**	`42`
enum-literal	*identifier*	`ASLEEP`
bool-literal	`true` `false`	`true` `false`

NOTES:

String literals can include and escape character ‘\’ that prevents the next character matching the delimiter. To include a ‘\’ write ‘\\’.

The syntax mostly follows Java except:

- The inclusion of reflection expressions that operate on the GameSpace
- There are no bitwise operators
- There is no character type.
- A single ‘&’ or ‘|’ represent logical and and logical or
- Enum literals don’t need to specifiy their class.

Operators associate left to right so 2 + 3 * 4 is 14, not 20.

Evaluation does not occur where it can be determined the rest of the expression has no impact on the result.

The ‘+’ operator can operate on mathematical expressions, or concatenate strings

3.2 Type Conversion

The expression evaluation cannot type check until runtime, which is unfortunate. It also plays fairly fast and loose with conversions.

3.2.1 Conversion to Boolean

- A boolean is unchanged
- An integer converts to true if it is non-zero, false if it is zero
- An enum converts to true if it is non-null, false if it is null
- A string converts to true if it is “true” or “yes”; false otherwise
- Any other object converts to true if it non-null, false otherwise
  An expression like $0 & $0.isMale ‘safely’ tests the $0 is not null and is male.

3.2.2 Conversion to Integer

- A boolean true becomes 1, false becomes 0.
- An integer is unchanged
- An enum is converted to it’s ordinal value
- A string is parsed for an integer
- An object becomes 1 if it is not null, 0 if it is null

3.2.3 Conversion to Enum

- A boolean ???
- An integer ???
- An enum is unchanged
- A string is parsed for the enum name
- An object ???

3.2.4 Conversion to String

- A boolean is converted to “true” or “false”
- An integer is converted to a String in base 10

- An enum is converted to its name
- A string is unchanged
- A GameBase object is converted to its id
- An object has its toString method called
  This may not be desireable; most toString methods produce debug information

3.2.5 Conversion to Object

- A boolean is converted to a Boolean
- An integer is converted to an Integer object
- An enum is unchanged
- A string isunchanged
- An object is unchanged

3.3 Reflection

Reflection expressions refer to objects in the GameSpace.

The GameSpace itself is the root element, and is refered to by ‘/’. Any object in the game space can be refered to by its id in the form [id]. A array of parameters are provided to any evaluation, with the meaning determined by the caller. The objects in this array are referred to by ‘$’ and the index, so $0, $1, and so on.

The reflection path is made up of a series of reflection methods. Each method is called in turn, and the result of one is the starting point for the next.

A method with no parameters is assumed to be a getter, so “.body” invokes “getBody()”. If there is no getter with that name then the method body() will be invoked if it exists.

A method with parameters is invoked with the name given, so .setLength(3*60) will invoke the setLength(int) method with a parameter of 180. Type conversion is attempted to convert the parameter types to the available methods of that name. Where the method name matches various methods with different parameter types the selected method will be the one that has the same number of parameters and each one can be converted.

Where a method doesn’t exist or the parameters can’t be appropriately converted a fault occurs.

3.4 Functions

The game has a number of built in functions:

3.4.1 has(container, object)

Tests to see if the container has the given object in it.

3.4.2 in(container, object)

Determines if the first parameter contains, recursively, the second. The first parameter must be an Area, Location, or a Container. The second must be an Item.

3.4.3 isDead(animal)

Determines if the first parameter is dead. The first parameter must be an Animal or something derived from Animal: Person, IntelPerson.

3.4.4 species(speciesEn, person)

Determines if a person is a specific species. The first parameter must be the name of a species. The second must be a Person.

4 Text Subsititution

Text subsitution in messages is done with the TextBuilder class (where various methods are called for the different parts), and the TextFormatter class which parses its format input and calls the methods of TextBuilder. The description of the format patterns here applies to TextFormatter.

In general messages are written so they can be used by the PC or an NPC. This means the same message has to cover both cases:

4.1 Do’s

- Be aware that in Yaffaif text is typically used for both the player and NPCs. Write out text with alternatives so you know where the tags must be used (and where you don't need to bother): "You/Penny check/s your/her wallet/purse; it is empty."
- Take care that you don't include description of what is in the characters mind for NPCs. Use the {pc} tag around text that would not make sense when the player is watching an NPC.
- Take care when adding your own adjectives to those built by parser tags. The formatter does buffer up adjectives and will re-order them into something approaching normal English usage, but only if you use the {adj} tag. So "{adj,blue,#60}{part,$0,ball,"#n"} will result in "two blue balls".
- Be aware that the parser will choose verb forms to go with the person or plurality of the subject if you use the {verb} tag.
- Use tags where they are needed. Don't bother if they are not.

4.2 Format Patterns

The format is written as a string. In the simple case the characters are echoed to the output. However, the following characters have a special meaning:

- **** escapes the next character
- **{** Introduces a command
- **,** is equivalent to {comma}
- **!** is equivalent to {exclaim}
- **?** is equivalent to {query}
- **.** is equivalent to {stop}

The formatter automatically adds spaces between commands and other text.

4.3 Commands

Format	Example	Function
{a}	a, an	insert the correct indefinite article for the next word
{aa}	a, an	insert the correct indefinite article for the next word even if it is an item unique in its setting
{add,param}		insert the param converted to a string
{adj,adjective,priority}	adjective	queues an adjective to be output in an ordered fashion before the next noun. Priority is used to get a sensible order (highest values emitted first) such as "some good peaty 70% proof 12 year old amber Scotch whisky". This is particularly useful to combine with adjectives added by the part command. Use '#' to introduce numeric literals for priority. • 30 - material, such as wooden, metallic • 40 - origin, such as American • 60 - colour, such as red, blue, pale • 70 - age, such as young, old, new, ancient, six-year-old • 80 - measurable size and shape, such as wealthy, large, round, and physical properties such as speed • 90 - subjective opinion specific to the noun such as tasty food, clever dog • 100 - subjective opinion not specific to the noun such as good, bad • 200 - determiners and other things not actually adjectives such as "some", "ten"
{bold}	none	following text will be rendered in bold weight font
{boldEnd}	none	following text will be rendered in normal weight font
{choice,param,n1,part1,n2,part2...}	part1, part2, ...	Takes the integer value of the parameter and then conditionally outputs the matching part. The part chosen is the first one whose numeric value is greater than or equal to the parameter (choice parts need to be in ascending order). The parts can contain other commands, but the ',' character will end the part (use {comma} instead).
{comma}	,	Inserts a comma
{dot}	.	Inserts an dot, without the normal stop

		behaviour.
{ellipsis}	...	Inserts an ellipsis, without the normal stop behaviour.
{exclaim}	!	Inserts a exclamation mark, capitalises the first word of the sentence
{indent}	Text	Inserts spaces (normally at the start of the line), capitalises the first word of the sentence
{if,param,truepart[,falsepart]}	truepart or falsepart	Takes the boolean value (non zero is considered true for ints) of the parameter and then conditionally outputs the true part or the optional false part. The parts can contain other commands, but the ',' character will end the part (use {comma} instead).
{length,param}	6'2"	Inserts the param (in millimetres) converted using the current units
{money,param}	three coins	Inserts the param (in coins) converted using the game currency
{moneys,param}	three coins	Inserts the param (in coins) converted using the game currency. Makes the coins the subject.
{narrate,2nd}	N/A	Sets the context for narration about the 2nd person. References to the second person become 'you'. No reference is treated as 'I' or 'me'.
{nl}	/n	Inserts a newline (CR/LF) into the stream. Starts next with a capital.
{npc,actor,truepart[,falsepart]}	truepart or falsepart	Determines if the actor is a non-player character and then conditionally outputs the true part or the optional false part. The parts can contain other commands, but the ',' character will end the part (use {comma} instead).
{num,param}	109	Inserts the param as a number
{obj}	the chicken, a chicken, you, Paul	Inserts text to refer to the object. If the object is the player then "you" is used.
{obj,param}	the rat, a rat, Peter	Sets the object to the parameter, then inserts text to refer to the object. The parameter must be an Item (or derived class).
{oneof,parts,part1,part2...}	part1 or part2 or ...	Takes a random number between 0 and parts then conditionally outputs the matching part. The part chosen is the one whose index is equal to the random number. The parts can contain other commands, but the ',' character will end the part (use {comma} instead).
{part,param,partname,style	body part	Inserts a description of a body part. The style

}	description	controls the detail that is output.
{parts,param,partname,style}	body part description	Inserts a description of a body part. Sets subject variables so correct verb endings are used.
{pc,actor,truepart[,falsepart]}	truepart or falsepart	Determines if the actor is a player character and then conditionally outputs the true part or the optional false part. The parts can contain other commands, but the ',' character will end the part (use {comma} instead).
{posadj}	your, her, his, hir, its, their	Inserts the possessive adjective for the subject
{posadjname}	Peter's, your, her, his, hir, its, their	Inserts the name or possessive adjective for the subject
{posadjobj}	your, her, his, hir, its, their	Inserts the possessive adjective for the object
{posadjobjname}	Paul's, your, her, his, hir, its, their	Inserts the name or possessive adjective for the object
{pronom}	you, he, she, shi, it, they	Inserts the nominative pronoun (for the subject)
{proobj}	you, him, her, hir, it, them	Inserts the objective pronoun (for the object)
{query}	?	Inserts a question mark, capitalises the first word of the sentence
{reflexpro}	yourself, himself, herself, hirself, themselves	Inserts the reflexive pronoun for the subject
{reflexproobj}	yourself, himself, herself, hirself, themselves	Inserts the reflexive pronoun for the object
{rnd,limit,n1,part1,n2,part2...}	part1 or part2 or ...	Takes a random number between 0 and limit-1 then conditionally outputs the matching part. The part chosen is the one whose numeric value is less than or equal to the random number (choice parts need to be in ascending order). The parts can contain other commands, but the ',' character will end the part (use {comma} instead).
{se}	"	Inserts closing speech marks
{sr}	"	Inserts opening speech marks for resuming speech. No capital is forced.
{ss}	"	Inserts opening speech marks. The next word is capitalised.
{stop}	.	Inserts a full stop, capitalises the first word of the sentence
{setobj,param}		Sets the object of the sentence. Produces no

		output, so you can use {proobj} and so on. The parameter must be an Item or dervied class.
{setsubj,param}		Sets the subject of the sentence. Produces no output, so you can use {pronom} and so on. The parameter must be an Item or dervied class.
{subj}	you, the rat, a rat, Peter	Inserts text to refer to the subject. If the player is the subject then "you" is used.
{subj,param}	the rat, a rat, Peter	Sets the subject to the parameter, then inserts text to refer to the subject. The parameter can be an Item (or derived class) or a TextBuilder. If it is a TextBuilder, then the subject is *temporarily* changed to that of the TextBuilder to facilitate sub-phrases like: `{setsubj,0} {posadj}{subj,1} {verb,jiggle}as{pronom} {verb,walk}.` which might generate "Your fat arse jiggles as you walk." or "Sam's swollen breasts jiggle as she walks."
{talk,1st,2nd}	N/A	Sets the context for 1st person talking to the 2nd. References to the first person become 'I' or 'me'. References to the second person become 'you'.
{the}	the	Inserts the definite article, prevents use of indefinite article
{tobe}	am, are, is	Inserts the correct part of the irregular verb to be
{verb,(root|param)}	like, likes	Inserts the version of the verb for the subject. Automatically generates the third person singular version. The arg may be a string or a parameter.
{verb,(root|param),(third|param)}	like, likes	Inserts the version of the verb for the subject. The third person singular version is provided as a second parameter. The arg may be a string or a parameter.
{volume,param}	1800cc	Inserts the param (in cc) converted to a volume using the current units
{weight,param}	14st 5lb	Inserts the param (in grams) converted to a weight using the current units

4.4 Parameters

Parameters can be specified in a number of ways and object parameters may be followed by '.' and and a getter method name to retrieve sub-parts of the original parameter:

Format	Example	Function

/	/.glob.version	Refers to the current game space (established by context)
$n	$1.stats.hp	Refers to the supplied parameter array by index
n	1.stats.hp	Refers to the supplied parameter array by index
[id]	[val1.village.pub.arthur].stats.hp	Refers to the game object with the supplied id
#n	#101	An integer value (not a parameter index)
text	gruntbuggly	A string value

NOTE: At some point the commands and parameters will be evaluated using the main expression handler as a comma separate list of args, which will change this syntax. The main impact will be the an int-literal will refer to an integer, not a parameter like ‘$’ and parameters can include proper expressions. Avoid using an int to refer to a parameter index.

4.5 Parts

The following body parts are understood: arm, ball, belly, bust, butt, clit, digit, ear, eye, female (genitals), hair, hand, leg, male (genitals), mouth, muzzle, nipple, scrotum, tail, thigh, torso, waist. Also the following non-body parts: name.

4.5.1 Style strings

Style strings are made up of a sequence of characters that identify what to output. If a style character is not appropriate no output is produced for that character. In general use characters in the order the output is needed. Noun modifiers 'e' and 's' must preceed 'n' if they are used. Adjectives may be re-ordered to match adjective priorities. The following style characters apply to body parts:

Letter	Example	Function
1	... eye ...	request singular noun
2	... cocks ...	request plural noun. Singular will still be used if there is only one.
#	... two ...	number of items
?	N/A	stop output if no adjectives are available from the style string
a	... huge ...	size adjective
b		unused
c	... red ...	colour adjective
C	... DD ...	Bust: cup size adjective
d	... fox-like ...	add species adjective if it is different from parent part
e	... love-pillows ...	allow euphemism to replace noun. Must preceed 'n'
f		unused
g		unused
G	... four inch ...	Male genitals: knot width

h		unused
i		unused
j		unused
k	... fur covered ...	add skin type if different from parent part
K	... fur covered ...	add skin type always
l	... 20cm ...	length in current units
m	... anxious ...	emotive state
n	... feet ...	the noun (or euphemism if 'e' proceeds 'n')
o		unused
p		unused
q		unused
r		unused
s	N/A	make noun subject of sentence, must preceed 'n'. Verbs will match plurality of the selected noun
t	... fox-like ...	add species related description
u		unused
v	... badly nicked and drooping at the tips	add text to follow "noun is/are"
w	10lb	add weight in current units
W	... two inch ...	add width in current units
x	... delicate ...	add extended adjectives
y		unused
z	... with black roots tied back into a ponytail	extend the description using the withExtend and participleExtend. Use after 'n'
Z	"... with ... doodaded ... that is/are ...."	extend description to the max. uses withExtend, participleExtend and if verbExtend is available adds it after "that {tobe}". Use sparingly. Use after 'n'.

The following apply to the "name" part:

Letter	Example	Function
f	Broadbent	family name (surname)
g	Tim	given name
l	PhD BSc	letters after name
n	Slasher	nickname
s	N/A	make subject of sentence
t	Dr.	Title abbreviation
T	Doctor	Title in full

4.6 Writing Speech

Correct forms:

You say, "I love you."

In TextFormatter:

```
{subj,$0}{verb,say},{ss}{talk,$0,$1){subj}{verb,love}{proobj}.{se}
```

In some cases this can be simplified. If we know $0 is always the player then the verb form will not change, and normally the object pronoun won't either (unless the player is incredibly narcisistic and talking to themselves where the reflexive-pronoun will be needed anyway).

```
{subj,$0}say,{ss}{talk,$0,$1){subj}love you.{se}
```

In TextBuilder:

```
subj(person).verb("say").comma()
.ss().talk(person, lover)
.subj().verb("love").obj(lover).stop()
.se().finish();
```

You say, "I love you. I love you so much."

```
subj(person).verb("say").comma()
.ss().talk(person, lover)
.subj().verb("love").obj(lover).stop()
.subj().verb("love").obj(lover).add("so much").stop()
.se().finish();
```

"He is very clever, you know."

```
ss().talk(speaker, listener).setSubj(other)
.proNom().toBe().add("very clever").comma()
.setSubj(listener).proNom().verb("know").stop()
.se().finish();
```

"He thinks it's a more respectable job," said Jo.

```
ss().talk(speaker, listener).setSubj(other)
.proNom().verb("think").add("it's a more respectable job").comma()
.se().narrate(player).setSubj(speaker)
.verb("said", "said").subj().stop().finish();
```

"Can I come in?" he asked.

```
ss().talk(other, listener).setSubj(other)
.add("can").proNom().verb("come").add("in").query()
.se().narrate(player).setSubj(other)
.proNom().verb("asked", "asked").stop().finish();
```

"Just a moment! she shouted.

```
ss().add("just a moment").exclaim()
```

```
.se().narrate(player).setSubj(other)
.proNom().verb("shouted", "shouted").stop().finish();
```

"Thinking back," she said, "I did not expect to win."

```
ss()
.add("thinking back").comma()
.se().narrate(player).setSubj(other)
.proNom().verb("said", "said").comma()
.sr().talk(other, listener)
.subj(other).verb("did", "did").add("not expect to win").stop()
.se().finish();
```

"That," he said, "is nonsense."

```
ss()
.add("that").comma()
.se().narrate(player).setSubj(other)
.proNom().verb("said", "said").comma()
.sr()
.add("is nonsense").stop()
.se().finish();
```

Not normally used (first person narration):

"I do not agree," I replied.

```
ss().talk(speaker, listener).setSubj(speaker)
.proNom().verb("do").add("not agree").comma()
.se().ctx(speaker, listener).setSubj(speaker)
.proNom().verb("replied", "replied").stop().finish();
```

Table of Contents