XVAN 2.6

-- syntax --

-- everything is a location, an object or a timer --

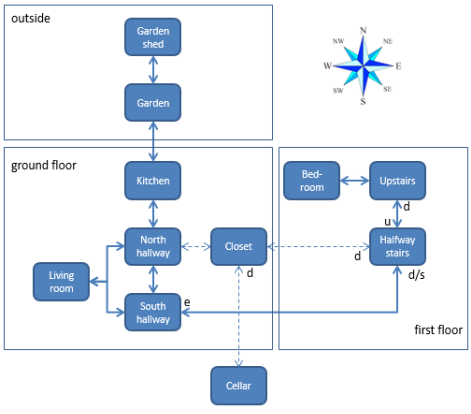

Table of Contents

What's new?

In XVAN version 2.3

- There is no longer an enforced declaration order in the story file.
  Sections for:
  - Common descriptions;
  - Common flags;
  - Common triggers;
  - Locations
  - Objects and
  - Triggers

  may now be used in any order and multiple times in the input file[1] and inserted files. The story title and version are mandatory at the beginning of the input file.
- There is no longer a mandatory vocabulary file. All story code can now be put in one file. For reusability it is recommended to have a separate file with the vocabulary.
- The verb and directions sections from the former vocabulary file may now also be listed multiple times and in any order. As of version 2.2 this was already the case with the other vocabulary sections (nouns, adjectives, etc).
- Keywords that identify sections in the input files must now be prefixed with a '$' character.

In XVAN version 2.3.1

- An issue with the order in which common triggers had to be defined has been fixed. In case common trigger 1 would call common trigger 2, common trigger 2 should be defined before common trigger 1 in the COMMON_TRIGGERS section. If not, common trigger 2 would be compiled as a local trigger.

In XVAN version 2.3.2

- Additional language support (currently Dutch).
- Starter kit 1.0 with predefined actions and dictionary (English and Dutch version). The Starter Kit is not an integral part of the XVAN executables. It is a collection of XVAN source code (verbs, dictionary, triggers etc) to integrate in a story.
- Possibility to redefine already defined verbs and common triggers. This functionality enables to redefine the behavior from verbs and triggers in the Starter Kit, without changing the Starter Kit itself (version control).

In XVAN version 2.3.3

- The XVAN compiler can handle Windows, Linux and macOS text file line delimiters ("\r\n", "\n" and "\r"). It is no longer necessary to convert the source file to the OS that is used;
- The Try() function was added (see XVAN Functions document for a description);
- Improved detection of missing end quotes in strings: each line with a string must now end with either '/' or an end quote. Two consecutive strings will be combined to one string;
- Support for 'all' and 'it', predefined in Starter Kit 1.1. When not using the SK, object o_it with attribute r_it must be defined by the author.

[1] The input file is the story's main file that is entered on the compiler command line. Other files can be included in the input file by using the insert "filename" statement.

In XVAN version 2.3.4

- The LANGUAGE and TITLE section was upgraded to the story info section, with additional options mainly for IFI-XVAN (also see the IFI-XVAN documentation).

In XVAN version 2.4

The story info section now has additional options for the play mode (choice and hybrid).

In XVAN version 2.5

The NOUNS section now allows to define irregular plural forms.

In XVAN version 2.6

Added the possibility to handle nouns that cannot be bound to a real life object (e.g. “get rest”, ”hit the road”).

1. Story file section syntax

Text in ***BLUE BOLD UNDERLINED ITALIC*** are keywords

The individual sections are outlined. Sections may occur multiple times in the input file(s) and in arbitrary order. Exception is the STORY INFO section, which must be the first section in the main input file[2]. For easier maintenance it is recommend to group similar sections (e.g. all locations grouped together), so you know where to look when editing the story.

Story info section

TITLE	"Sample story for XVAN 2.3.4"	
AUTHOR	“author name”	
ORGANIZATION	“organization name”	
COVERTEXT	“text for story’s cover page”	
CREDITS	“credits text”	
CHOICE_MODE		# optional
HYBRID_MODE		# optional
VERSION	"version text with format 1.0.0"	
ANDROID_MKT	“url to app in Play Store”	# optional
IOS_MKT	“url to app in App Store”	# optional
BACKIMAGE	“text with file path to background image for game cover”	# optional
EFFECT	“text with file path to shader to display on cover page”	# optional
NO_SIDEBAR		# optional
NO_TEXTINPUT		# optional
NO_COMPASS		# optional
PRIMARY_COLOR	“theme primary color, e.g. deep orange”	# optional
AUTOLINK	# switch on autolink option for GUI, in XVAN default off	
XVAN_LANGUAGE	<language>	
STORY_LANGUAGE	<language>	

Keywords other than TITLE, AUTHOR, VERSION, CHOICE_MODE, HYBRID_MODE and LANGUAGE are only used by IFI-XAN (XVAN version with graphical user interface). For more information on possible values, consult the IFI-XVAN documentation.

The language keywords and languages may be entered in the supported languages, currently English and Dutch:
XVAN_LANGUAGE, XVAN_TAAL
STORY_LANGUAGE, STORY_TAAL
english, eng, engels
dutch, nl, nederlands

[2] The main input file is the file that is entered with the compile command.

If a language keyword is omitted, it defaults to English.

For IFI_XVAN, the keywords CHOICE_MODE and HYBRID_MODE determine the values for sidebar, textinput and compass.

CHOICE_MODE and HYBRID_MODE will turn on the compass and sidebar, regardless of the setting in the story info setting.

Insert section

```
$insert          "sample.lib"     # the file sample.lib will be processed now.
```

Verb section

```
$VERB maximum_size_verb
PROLOGUE
      <prologue code>
EPILOGUE
      <epilogue code>
<string-1>
<string-n>
AMBIGUITY_RULES
   if <condition> then score(number) endif
 END_RULES
      <default code for commands in string-1 .. string-n>
DEFAULT
      <default default verb code>
ENDVERB

$VERB minimum_size_verb ENDVERB

$REDEFINE_VERB existing_verb_name
PROLOGUE
      <prologue code>
EPILOGUE
      <epilogue code>
<string-1>
<string-n>
AMBIGUITY_RULES
   if <condition> then score(number) endif
 END_RULES
      <default code for commands in string-1 .. string-n>
DEFAULT
      <default default verb code>
```

ENDVERB

Note: $redefine_verb is used to redefine a verb that is in the library with new behavior without having to edit the library (for easier library version control)

Direction section

$DIRECTIONS
north ***SYNONYM*** n,
south ***SYNONYM*** s,
east ***SYNONYM*** e,
west ***SYNONYM*** w,
northeast ***SYNONYM*** ne,
northwest ***SYNONYM*** nw,
southeast ***SYNONYM*** se,
southwest ***SYNONYM*** sw,
up ***SYNONYM*** u ***SYNONYM*** high,
down ***SYNONYM*** d ***SYNONYM*** low,
in, out, left, right, forward, back

Noun section

$NOUNS
Define nouns here:
noun1, noun2, noun3 ***SYNONYM*** noun4, noun5 ***PLURAL*** noun6, ...

Note: for defined nouns, plural that ends with “s” or “es” (dutch: “s”, “n” or “en” will automatically be handled by the interpreter. Other plural forms must be defined by adding after the noun with the PLURAL keyword (e.g. ox PLURAL oxen).

Adjective section

$ADJECTIVES
Define adjectives here:
adjective1, adjective2, adjective3 ***SYNONYM*** adjective4, ...

Preposition section

$PREPOSITIONS
Define prepositions:
preposition1, preposition2, ...

Adverb Section

$ADVERBS
Define adverbs here:
adverb1 ***SYNONYM*** adverb2 ***SYNONYM*** adverb3, adverb4, ...

Articles section

$ARTICLES
Define articles here:
article1, article2, ...

Question word section

$Q_WORDS
Examples of question words are: where, who, how, which etc
Define question words here:
qword1, qword2, qword3, ...

Conjunction section

$CONJUNCTION
An example of a conjunction is the word 'and'.
Define conjunctions here:
conjunction1, conjunction2, ...

Common descriptions section

$COMMON_DESCRS

Define common descriptions here, with syntax:
d_descr1, d_descr2, d_descr3 .....

Common flags section

$COMMON_FLAGS

# Define common flags here with syntax:
f_flag1 = <value> # value is either 0 or 1
f_flag2 = <value>

Common attributes section

$COMMON_ATTRIBUTES

# Define common attributes here with syntax:
r_attribute1 = <value>
r_attribute2 = <value>

Common triggers section

$COMMON_TRIGGERS

Define common triggers here with syntax:
t_trigger1
<code>
t_trigger2
<code>

$REDEFINE_TRIGGERS

Redefine existing common triggers here with syntax:
t_trigger1
<code>
t_trigger2
<code>

Note: $redefine_trigger is used to redefine a trigger that is in a library or the Library with new behavior without having to edit the library (for easier library version control)

Locations section

```
# Define locations here with syntax:
$LOCATION l_location1
<location body, see annex 2 for an example>
END_LOC
```

Objects section

```
#Define objects here with syntax:
$OBJECT o_object1
<object body, see annex 3 for an example>
END_OBJ
```

Timers section

```
$TIMERS
# Define timers here with syntax:
m_maximal_timer
    value           <number>
    step            <number>
    direction       <up / down>
    interval        <number>
    state           <go / stop>
    trigger_at      <number>or_more / or_less
    execute         <trigger>       # preceded by location or object (e.g. o_object1.t_trigger1)

m_minimal_timer
    value           <number>
```

2. A sample XVAN location

Text in ***BLUE BOLD UNDERLINED ITALIC*** are keywords

$LOCATION l_panel_room
DESCRIPTIONS

d_sys "the panel room", "the lever room"

d_entrance "As you enter the wall seems to close behind you."

d_longlight "You are in a square room. The walls are made of a smooth /
material. You cannot see a direct way out."

d_longdark "You don't know where you are. It is pitch black. You can /
make out the outlines of a panel in the north wall. There /
seems to be a light source behind the panel, which shines /
through the edges."

d_dark "It takes some time for your eyes to adapt to the sudden /
darkness. After some time, you can make out the outlines /
of a panel in the north wall. There seems to be a light source /
behind the panel, which shines through the edges."

EXITS

n -> l_room1
e -> l_room3
sw -> l_room7

TRIGGERS

"look" -> t_look
"turn off [o_lamp]" -> t_off
"pull [o_lever]" -> t_pull_lever

```
t_pull_lever
    # Strings may also be printed directly, instead of using a description
    printcr("As you pull down the you feel some resistance...")
    printcr("You pull it all the way down and let go. It slowly returns to the up position")
    if valdir(l_room2, w) then
        blockexit(l_room2, w)
        printcr("The exit in the west wall disappears.")
    else
        newexit(l_room2, w, l_room4)
        printcr("An exit just appeared in the west wall!")
    endif
    disagree()
```

```
t_entrance
    if islit(o_player) then
        print(d_entrance)
        printcr(d_longlight)
    else
        printcr(d_longdark)

t_look
    if cansee(o_player, l_location) then        # l_location is player's current location
        printcr(d_longlight)
    else
        printcr(d_longdark)

t_off
    printcr(d_dark)
```

END_LOC

3. A sample XVAN object

Text in ***BLUE BOLD UNDERLINED ITALIC*** are keywords

This is the player character that represents the user playing the story.

$OBJECT o_player
DESCRIPTIONS
    d_sys   "you", “me”
    d_init   "Short sample story for X_VAN2.2"

CONTAINED in l_room0

FLAGS
    f_alive   = 1
    f_may_save   = 1

ATTRIBUTES
    r_to_be_verb   = are
    r_last_loc   = l_room0
    r_getal   = none

TRIGGERS
    "save"   -> t_save
    "restore"   -> t_restore
    "[dir]"   -> t_move
    "follow [o_npc]"   -> t_follow
    "where is [o_subject]"   -> t_where

```
t_init
# This trigger initializes things. It is started by timer
# m_init that goes off right away.
    printcr(d_init)
    printcr("")
    entrance(owner(%this))
    stoptimer(m_init)
    starttimer(m_moves)

t_entrance
    agree() # prevents calling default t_entrance for player.
```

```
t_save
    if testflag(f_may_save) then
        save()
    else
        printcr("We don't allow to save at this point ... ")
    endif
    disagree()

t_restore
    restore()

t_move
    if valdir(l_location, dir) then
        if exit(l_location) then
            # exit() returns true if the location and all objects in location
            # allow the player to leave the location. The exit() function
            # calls all t_exit triggers.
            move(o_player, dir) # move updates current location
            entrance(l_location)
        endif
    else
        nomatch()  # let other objects react.
    endif
    agree()

t_follow
    if equal(o_npc.r_last_loc, l_location) then
        move(o_player, o_npc.r_last_dir)  # also updates l_location
        entrance(l_location)
    else
        printcr("You cannot follow [the] [o_npc]")
    endif
    disagree()

t_where
    printcr(owner(o_subject))
    # may also use printcr(owner(o_subject).d_sys), this will print the first d_sys from
    # the location or object's definition. Without .d_sys and f_swap set it will print the
    # last d_sys that was referred to.
    disagree()
```

END_OBJ

4. A sample XVAN verb

Text in ***BLUE BOLD UNDERLINED ITALIC*** are keywords

$VERB ask
Verbs have a scope. Default scope is CURR_LOC_ONLY (the current location and the objects
that are in it and visible). Other options are ACTOR_ONLY (the actor and everything he
carries and is visible) or ALL_LOCS (every location and object in the game).
The scope determines the locations and objects that are considered by the interpreter during
parsing the user's input.

SCOPEALL_LOCS # may ask about anything, not limited to contents of the current location
PROLOGUE
Actor is going to ask something to the subject, so he must be able to see him.
```
    if not(equal(o_subject, %none)) then
        if not(cansee(o_actor, o_subject)) then
            printcr("But [o_actor] can´t see [the] [o_subject] here!")
                disagree() #stop
```

following are the inputs that the verb will respond to, provided that none of the locations
or object has responded to the user input.

```
"ask"
    printcr("You have to be more specific than that!")
    disagree()
```

Next we have 2 inputs that have the same response. If there is no code between two or more
text strings, they will all be registered with the same action record.

```
"ask [o_subject] about [o_spec]"                # here, actor is the player object
"[o_actor], ask [o_subject] about [o_spec]"     # here, there is another actor
    AMBIGUITY_RULES
        # If there is ambiguity about the specifier, the parser must consider specifiers that
        # the actor has seen before.
        # This code will only be run if the specifier cannot be uniquely identified from the
        # user input
        if testflag(o-spec.f_seen_before) then score(5)
    END_RULES
    # Because of the scope, we must check whether
    # actor and subject can see each other.
    if not(cansee(o_actor, o_subject)) then
        print("But  [the] [o_subject] [o_subject.r_to_be_verb] not here!")
    else
        if testflag(o_subject.f_alive) then
            printcr("It's just an ordinary [o_spec]. There's nothing more to tell about it. ")
```

```
        else
            printcr("You're talking to [a] [o_subject]! Are you feeling ok?")
        endif
    endif
    disagree()
```

DEFAULT

```
# The default section serves as a last resort.
printcr("That will not work.")
```

ENDVERB

5. XVAN interpreter flow