XVAN Library 1.4

-- Library –

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

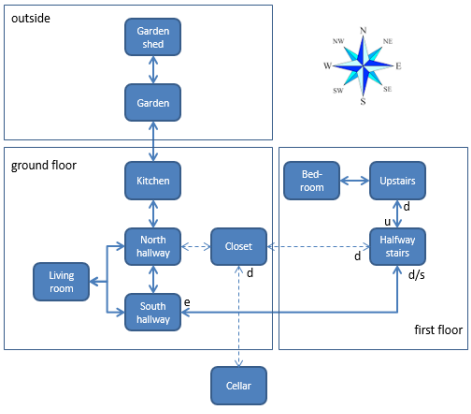

Table of Contents

What's new?

The XVAN Library version 1.3 has been tested with the XVAN IF Authoring System version 2.4.

In Library version 1.1:

- Support for 'all' and 'it'.

In Library version 1.2:

- Verbs "use" and "unused" were added (for use with IFI-XVAN);
- Verbs "sit", "knock", "poke", put, jump, sing, cry, yell, say, touch and xyzzy were added;
- Verb "restart" was added;
- Use of function notimers() for actions: save, restore, score, transcript and verbose;
- Some small changes in some actions;
- Some changes to o_all object;
- Some changes to o_it object;
- New file ifi-actions.lib for use with IFI-XVAN.

In Library version 1.3

- The Starter Kit is now called Library;
- Verbs "exits", "goto", "kick", "remember", "thank", and "undo" were added;
- Common description d_remember was added;
- Common flags f_supporter and f_fixed were added;
- Common trigger t_reveal was added;
- Common attribure r_nr_to_reveal was added.

In Library version 1.4

- New implementation of the undo function.

Introduction

XVAN is an Interactive Fiction authoring system: a compiler, an interpreter and an authoring language. The XVAN distribution comes with a set of sample stories and a tutorial on how to create an XVAN story from scratch.

Why do I need the Library?
Actually you don't. The compiler, interpreter and a simple text editor like notepad, textedit or vi are all that is needed to start creating stories.
But the Library offers a head start to begin writing XVAN stories. It has definitions for actions (verbs) commonly used in IF-works, definitions for mandatory objects, as well as a basic dictionary and a set of attributes, flags and triggers that you most likely will need in your stories.

Everything in the Library is 'normal' XVAN code. It can be edited, deleted, expanded or replaced as you wish, as long as you comply with the XVAN language syntax.

The remainder of this document describes the contents of the Library:

- Actions (in XVAN actions are called verbs);
- locations;
- objects;
- timers;
- artefacts (descriptions, flags, attributes and triggers);

The annex describes how to include the Librray in your story.

In case you are unfamiliar with XVAN artifacts, each artifact section starts with a short explanation of the artifact. But you may also want to read the XVAN Introduction document for a more detailed description.

Verbs

What is a verb?
In XVAN, verbs can be more than just words. A verb may contain instructions to be executed as a default handler for user input. Default in the sense that no locations or objects in the story have code to handle the situation.

An example:

```
$VERB break
  “break [o_subject]”
      printcr(“Trying to break [the] [o_subject] is not particularly helpful.”)
ENDVERB
```

So, whenever a “break something” command from the user is not handled in the story, the verb will print the message that it’s not very helpful to do this.

The following verbs are available in the Library:

The verbs are listed in alphabetical order.

Verb	Synonym(s)	Description
are		
ask		Ask someone about something.
break		Break an item.
close		Close an item.
cry		Cry.
do		
does		
drop		Drop an item.
examine	**x, investigate**	Examine an item.
exits		Lists the available exits from the current location. When a destination has been visited before, its name is listed as well.
get	**take, grab**	Get an item.
give		Give an item to someone/something.
go		Move through the world.
goto		Travel to a location that has been visited before.
hang		Hang an item on/in/behind/... another item.
has		
have		
help		Turn on/off syntax examples for verbs.
inventory	**i**	List the player’s possessions.

is		
jump		Jump.
kick		Kick something.
kill		Kill someone.
knock		Knock on things.
listen		Listen, or listen to something.
lock		Lock an item.
look	**l**	Give long description of room and items in it.
move		Move an item.
open		Open an item.
poke		Poke (in) things.
put		Put an item in/on/… something.
quit	**q**	Quit the game.
read		Read an item.
remember		Remember something.
restart		Restart the game from the beginning.
restore		Restore previously saved progress.
save		Save current progress.
say		Say something to someone.
score		Print the player's current score and the maximum possible score.
sing		Sing.
sit		Sit on things.
smell		Smell or smell something.
tell		Tell someone about something.
testmode		Read user input from a file instead of the keyboard.
thank	**thanks**	Thank someone.
throw		Throw an item in a direction or at/in/… another item.
tie		Tie an item to another item.
touch		Touch an item.
transcript		Log commands and replies in a textfile transcript.txt
turn		Turn an item.
undo		Tells that we don't support undo.
unlock		Unlock an item.
untie		Untie an item from another item.
unuse		Sent by the IFI-XVAN GUI when an icon is dropped on an empty space. Default behavior is drop.
use		Sent by the IFI-XVAN GUI when a hyperlink is clicked. Default behavior is examine.
verbose	**v**	Toggle between always printing long room descriptions or only at first entrance.

wait	**z**	Wait 1 or more turns.
wear		Wear an item.
xyzzy		Tribute to Colossal Cave Adventure.
yell		Yell.

A more detailed overview of verbs, the input sentences they support and the flags and attributes they use is available in a separate document: “XVAN Library verbs”.

If your story needs additional verbs that are not in the Library, these can just be defined in the game source itself or in an external file that can be referenced from the game file.

Locations

What is a location?
In XVAN, the world is made up of locations. A location can be seen as a room with exits connecting it to other rooms. The player can move from location to location.

The Library version 1.3 has no predefined locations.

Objects

What is an object?
An object is an item that exists in the XVAN world. Unlike a location, it need not be fixed in place, it can be manipulated by the user (taken, opened, closed, destroyed,…). An object is always contained in another object or in a location. Moving upwards in an object's containment tree, you will always end up in a location as the top level.

The Library comes with five objects:

Object	Purpose	Remark
o_player	Represents the main character that is directed by the person playing the story.	The compiler will throw an error if there is no o_player object defined.
o_nst	No-such-thing object, used when resolving ambiguities in user input.	The compiler will throw an error if there is no o_nst object defined.
o_status_window	Display a status window in the upper part of the text window.	Only used in the Glk version of the interpreter.
o_all	Allows to refer to all objects in scope with certain verbs.	Works for following commands: "take all" 'take all from <something>" "drop all" "drop all <prepos><something>"
o_it	Allows to refer to a previously mentioned location or object.	Is updated after every move, with the value of o_subject. Can be loaded with other values by clearing flag o_it.f_update_it and then assigning the desired value to attribute o_it.r_it Value must be a location or object.

If your story needs additional objects that are not in the Library, these can just be defined in the game source itself or in an external file that can be referenced from the game file.

Special note for object o_all
Commands with 'all' use common triggers t_take and t_drop that are defined in the Library in the all-object section. These are basic triggers with only necessary checks. In case an object in the game needs extra checks for take or drop, a local trigger with the same name t_get or t_drop must be defined in this object.

Timers

What is a timer?
A timer is used to schedule future events, so you don't have to check for it every turn. A timer has a value and counts up or down from that value. The timer is updated automatically when it's on. Once a preset threshold is reached, the timer will fire a trigger. An example of the use of a timer is modeling a battery for a flashlight.

The Library comes with one timer:

Timer	Purpose	Remark
m_init	Used to start the story. The timer has value 0 and fires at 1. The trigger that is started is o_player.t_init (see artefacts section)	

If your story needs additional timers that are not in the Library, these can just be defined in the game source itself or in an external file that can be referenced from the game file.

Artefacts

The Library comes with the following artefacts:

Descriptions

What is a description?
A description is an artefact that can hold a string of text. IF works generally have large chunks of text, which makes the code less readable (print statements can become very long and spread out over several lines). By storing the text in descriptions, all text for a specific object can be moved to a central place in the object code, which makes the code better readable.

Descriptions have following syntax:

```
d_description1   "This is a description text."
```

Following descriptions are available from the Library:

All descriptions listed are so called common descriptions – each location and object has them – unless stated otherwise.

Description	Purpose	Remark
d_sys	Describes how the player can refer to an object or a location.	d_sys may have more than 1 text string if an object has more than 1 description.
d_exa	Description for examining an object.	
d_entr_long	Long room description.	
d_entr_short	Short room description.	
d_remember	Memory description.	

If your story needs additional common descriptions that are not in the Library, these can just be defined in the game source itself or in an external file that can be referenced from the game file.

```
$COMMON_DESCRS
    d_descr1, d_descr2, …
```

Flags

What is a flag?
A flag is used to remember yes/no situations: can something be locked, is something locked, is something open, is something alive, …. As an example: the verb 'lock' uses an object's flag to check whether the command to lock something actually makes sense.

A flag definition looks like:

```
f_flag = <1 or 0>
```

Generally, value 1 (set) means yes and 0 (clear) means no.

Functions setflag() and clearflag() are used to manipulate flags.

Following flags are available from the Library:

All flags listed are so called common flags – each location and object has them – unless stated otherwise.

Flag	Purpose	Default	Remark
f_alive	Tells if an object is alive.	0	
f_any	Used to group objects for synchronize() and count() functions.	1	
f_bypass	Tells the interpreter to always consider this object when mapping user input to objects.	0	System defined.
f_container	Tells if an object can contain other objects.	0	
f_first	Used to determine when to print the starting text of the item list.	1	Local flag defined with the mandatory player object.
f_hidden	Tells the interpreter to never consider this object when mapping user input to objects.	0	System defined.
f_fixed	Tells whether an object is fixed in place.		
f_lit	To determine whether an object is lit.	0	System defined.
f_lockable	Tells if an object can be locked.	0	Says nothing about the object actually being locked.
f_locked	Tells if an object is locked.	0	
f_may_save	Tells if the game may be saved.	1	Local flag defined with the mandatory player object.
F_may_undo	Tells if the last move may be undone	1	Local flag defined with the mandatory player object.
f_opaque	If no, object is see through.	0	System defined.
f_open	Tells if an object is open.	0	
f_openable	Tells if an object can be opened.	0	Says nothing about the object actually being open.
f_scenery		0	
f_seenbefore	If yes, short room description will be printed.	0	

f_supporter	Tells is an object is a supporter (i.e. something can be put on it).		
f_swap	If there is more than 1 system description, the interpreter will refer to a location or object with the same description as the player used.	0	System defined.
f_takeable	Tells if an object can be picked up.	0	
f_verb_help	Used to determine if verb syntax examples are active.	1	Local flag defined with the mandatory player object.
f_verbose	Tells whether to always print long room descriptions	0	Local flag defined with the mandatory player object.
f_wearable	Tells whether an item can be worn	0	
f_worn	Tells whether an item is actually worn.	0	

If your story needs additional common flags that are not in the Library, these can just be defined in the game source itself or in an external file that can be referenced from the game file.

```
$COMMON_FLAGS
    f_flag1 = 0
    f_flag2 = 1
    …
```

Attributes

Why do we need attributes?
An attribute is used to remember information other than yes/no. Attributes can contain pretty much any type of information: location, object, number, description, another attribute, …

The "=" operator is used to assign a value to an attribute:

```
r_attribute = <value>
```

Following attributes are available from the Library:

All attributes listed are so called common attributes – each location and object has them – unless stated otherwise.

Attribute	Purpose	Default	Remark
r_do	Storing an item's conjugation for 'to do'	does	
r_have	Storing an item's conjugation for 'to have'	has	
r_is	Storing an item's conjugation for 'to be'.	is	
r_key	Store he key that unlocks a locked item.	none	
r_max_score	The maximum score for the story.	0	Local attribute defined with the mandatory player object
r_nr_to_reveal	The number of items that must be printed when opening a container.	0	Local attribute defined with the mandatory player object
r_preposition	Building correct sentences when referring to an item's position.	none	System defined.
r_score	The player's current score.	0	Local attribute defined with the mandatory player object

If your story needs additional common attributes that are not in the Library, these can just be defined in the game source itself or in an external file that can be referenced from the game file.

```
$COMMON_ATTRIBUTES
    r_attr1 = <value>
    r_attr22 = <value>
    …
```

Triggers

What is a trigger?
A trigger is a small program that is executed based on the input from the person playing the story. XVAN has a number of functions that can be used in triggers.

An example of a trigger is:

```
t_hidden_passage
  printcr("Moving the rock reveals a dark passage down!")
  printcr("Your score just went up by 50 points.")
  newexit(%this, down, l_cave)
  o_player.m_score += 50
  agree()
```

Explanation: this trigger creates a new exit from the current location downwards to the cave. The player's score is increased with 50 points.

Printcr() and newexit() are examples of XVAN functions. An overview of XVAN functions is available in the document: “XVAN functions”.

We link the trigger to the applicable user input through a statement like:

```
“move [o_rock]” ->  t_hidden_passage
```

This tells the interpreter to fire the trigger when the user command resolves to “move rock”.

The Library comes with the following triggers:

Description	Purpose	Remark
t_init	Starts the story. Prints the opening message and does some other stuff.	This is a local trigger defined in the player object’s trigger section.
t_move	Lets the player object travel through the world.	This is a local trigger defined in the player object’s trigger section.
t_entrance	The entrance() function calls the t_entrance trigger for each object in scope. The entrance() function is usually called when the player enters a new location.	
t_i	Prints the item’s description for the inventory command.	In case an item must respond to the inventory command, the statement “inventory” ->t_i must be in the item’s trigger section.
t_reveal	Is used for printing a list of items separated by commas.	Attribute r_nr_to_reveal must be set to the correct value.

If your story needs additional common triggers that are not in the Library, these can just be defined in the game source itself or in an external file that can be referenced from the game file.

```
$COMMON_TRIGGERS
    t_trigger1
        <trigger code>

    t_trigger2
        <trigger code>
```

Templates

The Library can be used with any IF work created with XVAN. This gives a head start[1], you now only have to think about locations, objects and timers.

Note: use the template files that come with the Library. Copy/pasting the templates from the text in this document may contain formatting characters that will cause compile errors.

Verbs

A verb can be defined in the story file with following syntax:

```
$VERB name
  PROLOGUE
        <actions to be performed before anything else for this verb is done>

  EPILOGUE
        <actions to be performed after everything for this verb has been done>

  "user command to respond to"
     <…code…>

  "user command to respond to"
     <…code…>

  DEFAULT
     <code to execute wen nothing else fired>

ENDVERB
```

Some remarks:

- User input is offered to locations and objects first. If the input is not handled there, then it is offered to the verb. Exception: the verb prologue – when present – is always executed first.

The Library comes with 48 verbs and some synonyms.

[1] When your story progresses you may want to (most certainly will) add additional sections with verbs, dictionary words and artefacts specific for your story.

Locations

A location can be defined in the story file with following syntax:

```
$LOCATION l_location-identifier          # location names must start with 'l_'
  DESCRIPTIONS
     d_sys            "system description 1", "system description 2"
     d_entr_long      "long description, printed when entering the location"
     d_entr_short     "short description"
     d_other          "other descriptions"

  FLAGS
     f_flag1 = 0
     f_flag2 = 1

  ATTRIBUTES
     r_attribute1 = <attribute value>
     r_attribute2 = <attribute value>

  TRIGGERS
     "some command that the player typed"   -> t_trigger1
     "teleport to [l_starship]"             -> t_trigger2

     t_trigger1
        # code for trigger 1, sample below
        printcr("You have started trigger 1.")
        agree()

     t_trigger2
        move(o_player, l_starship)
        entrance(l_starship)          # call t_entrance for starship and contained objects
        agree()
  END_LOC
```

Some remarks:

- Lines starting with '#' are comments.
- Location identifiers must start with 'l_'
- l_location-identifier is only used in XVAN code to refer to the location.
- d_sys descriptions are used to map user commands to the location.
- d_sys text strings therefore must obey XVAN English grammar rules and must be in the dictionary

Objects

An object can be defined in the story file with following syntax:

```
$OBJECT l_object-identifier       # location names must start with 'o_'
  DESCRIPTIONS
      d_sys             "system description 1", "system description 2"
      d_entr_long       "long description, printed when entering the object's location"
      d_entr_short      "short description"
      d_exa             "printed when the object is examined"
      d_other           "other descriptions"

  CONTAINED in l_location-identifier

  FLAGS
      f_flag 1      = 0
      f_takeable = 1   # may be picked up

  ATTRIBUTES
      r_attribute1 = <attribute value>
      r_attribute2 = <attribute value>

  TRIGGERS
      "some command that the player typed"   -> t_trigger1
      "inventory"                            ->t_i       # common trigger not defined here

      t_trigger1
          # code for trigger 1, sample below
          printcr("You have started trigger 1.")
          agree()
END_OBJ
```

Some remarks:

- The CONTAINED section tells where the object is located (location or other object). Other prepositions than 'in' may also be used. The preposition will be stored in the object's r_preposition attribute, so it can be used in trigger code for printing, testing etc.
- The t_i trigger is not defined with the object because it is a common trigger (see artifacts section). You may define a t_i trigger locally in the object's trigger section if you need behavior that is different from the common trigger. The local trigger will then override the common trigger.

Timers

A timer can be defined with following syntax:

```
m_timer-identifier                                  # timer names must start with 'm_'
  value         <number>
  step          <number>                            # number to increase/decrease by
  direction     <up / down>
  interval      <number>                            # update interval (1 updates every turn)
  state         go / stop
  trigger_at    <number> [or_more] / [or_less]      # value when to execute the trigger
  execute       <trigger>
```

Some remarks:

- Because a timer is defined stand alone and not within a location or an object, the trigger to be executed must be preceded by the location or object to execute it for. E.g. o_lamp.t_empty.
- The state can be manipulated with functions starttimer() and stoptimer().

Redefining verbs and common triggers

Suppose you need a verb or a trigger from the Library to behave different than is coded in the Library.

As an example, let's assume you want the command 'x' not to be a synonym for 'examine' as defined in the Library, but to be a command to return the player's to his previous location.

If you just add a new verb "$VERB x", the XVAN compiler will throw a multiple defined verb error, because 'x' is already defined as a verb.

One way to handle this, would be the change the code in the Library:

```
$VERB examine SYNONYM x

    < …code… >

ENDVERB
```

would become:

```
$VERB examine

    < …code… >

ENDVERB
```

And you would define a new verb in your story:

```
$VERB x

    move(o_player, o_player.r_previous)
    entrance(o_player.r_previous)

ENDVERB
```

Although this works perfectly well, it introduces a version control issue with the Library as there is now a Library version specifically for this story. When distributing the sources of your story, you must also distribute the modified Library.

To prevent this issue, from XVAN 2.3.2 it is possible to *redefine* verbs and common triggers. In the example above it suffices to just redefine the verb 'x'.

```
$REDEFINE_VERB x

    move(o_player, o_player.r_previous)
    entrance(o_player.r_previous)

ENDVERB
```

The XVAN compiler will remove ‘x’ as a synonym for examine and assign it to the new code. It is also possible to create synonyms with a *redefine*.

E.g:

```
$REDEFINE_VERB x SYNONYM exit SYNONYM out

    move(o_player, o_player.r_previous)
    entrance(o_player.r_previous)

ENDVERB
```

Analogue to $REDEFINE_VERB there is also $REDEFINE_TRIGGER, which redefines a common trigger.

IFI-XVAN

As of XVAN version 2.3.4, a version with a graphical user interface (GUI) is available: IFI-XVAN[2]. IFI-XVAN consist of a back-end and a front-end, the back-end being XVAN and the front-end being the Brahman GUI developed by Strandgames. XVAN and GUI communicate by exchanging text strings according to the JSON-format.

The IFI Library contains several verbs, flags, attributes and triggers that are used by XVAN to communicate with the GUI. E.g. sending possible exits, updating the player's current location, sending file names from pictures etc.

Note: for information on how to "ifify" an XVAN story, please refer to the IFI-XVAN document.

At the end of each turn, all necessary updates are sent to the GUI. This is an automatic process that does not require any coding by the author, as long as you use the IFI Library.

Following locations, objects, timers and artefacts are in the IFI library:

Verbs

Verb	Synonym(s)	Description
ifi_exits	--	Sends the current location's exits to the GUI.
ifi_items	--	Sends the player's inventory to the GUI.
Ifi_people	--	Sends a list of characters the player has already met to the GUI.
ifi_loc	--	Sends the player's current location's id to the GUI.
ifi_map	--	Sends the story's map data to the GUI.
ifi_picture	--	Sends the current locatoi's background picture to the GUI.
Ifi_update_gui	--	Executes all of the above verbs.

Locations

Locations	Purpose	Remark
I_json	Home for several flags, triggers, attributes etc.	

Objects

The ifi library has no predefined objects.

[2] IFI stands for Interactive Fiction Interface.

Timers

Timer	Purpose	Remark
m_ifi	Triggers the t_ifi trigger from l_json at the end of each move.	

Descriptions

Description	Purpose	Remark
d_map_backimage	Used to store pathname of the story's opening screen background image.	Path is always \images\coverimage.png relative to the datadir. This is a local description with location l_json.

Flags

The ifi library has no predefined flags.

Attributes

Attribute	Purpose	Default	Remark
r_gx	A location's x-coordinate on the map.	0	
r_gy	A location's y-coordinate on the map.	0	
r_ifi_picture	Hold a location's picture file path.	""	
r_ifi_icon	Hold an object's icon file path.	""	
r_ifi_maplevel	Height level on a 3D map	0	Attribute with l_json object.
r_ifi_maptext	Header text for the map	0	Attribute with l_json object. Advice is to define descriptions in the story and use t_entrance triggers to assign the right description to r-ifi_maptext.

Triggers

Description	Purpose	Remark
t_ifi	Execute all necessary ifi-actions at the end of a move.	Is a local trigger with location l_json.
t_ifi_items	If the object is in the player's inventory, the object id and icon path will be sent	

	to the GUI to display in the inventory.	
t_ifi_items_id	Sends the object id to the GUI to display on the map.	Only applies to objects that are directly in the location (and not in other objects).
t_ifi_people	If the player has met the object before, it will send the object's id and icon to the GUI.	Tests common flag f_seenbefore from the character.
t_ifi_place	Sends location information to the GUI.	Position on the map, exits, name, id, contained items. Used to draw the map.
t_ifi_exits	Sends exit information for a location	Used to render the clickable compass rose on the GUI.

Annex: using the Library in your story

The XVAN Library is a single file called something like “XVAN Library x-y.lib”.

The file must be placed in the same folder as your story file(s). In your story file, insert the .lib file with the line:

- $insert ".\\XVAN Library x-y.lib" or
- $insert “.//XVAN Librray x-y.lib”