XVAN 2.6

-- functions --

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

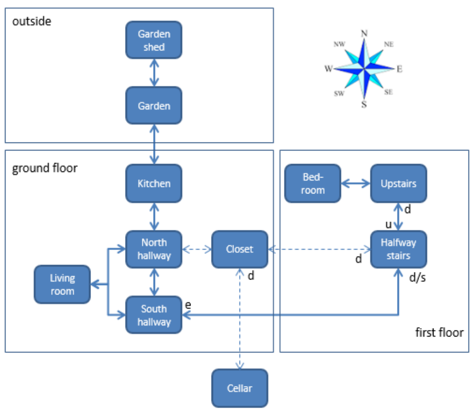


Table of Contents

What's new?

In XVAN version 2.1:

- The score() function was added;
- Wildcards in strings no longer require the %-prefix, they are now put in [ ] ;
- The use of description and attribute parameters in string parameters was added.

In XVAN version 2.11:

- Bugfix: printing an empty description parameter caused a crash;
- Bugfix: [num] and [ord] did not print in strings.

In XVAN version 2.2 there are no new or changed functions.

In XVAN version 2.3 there are no new or changed functions.

In XVAN version 2.3.1:

- The runcommon() function was added;
- The runverb() function was added;
- Added description for function dest().

In XVAN version 2.3.2 there are no new or changed functions.

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;
- The section “And a little bit more about strings” was added. It describes improved string handling.

In XVAN version 2.3.4:

- Added the notimers() function;
- Added the restart() function;
- Added the addjson(), sendjson() and clearjson() functions;
- The try() function has an additional parameter to control the timers;
- The transcript() function no longer overwrites an existing transcript file but appends to it;
- The owns() function now has the possibility to test for the type of containment (in, under, etc).
- Functions getsubject() and getspec() can now preload the preposition.

In XVAN version 2.4:

- The debug function now offers 2 levels of debug information;
- Added the pickone() function;
- Added the functions playmode() and addchoice();
- It is now possible to switch to choice mode, where the user input is limited to one item of a list of predefined options.

Introduction

This document describes the functions that are available in XVAN 2.3.4. Functions are described in terms of purpose, parameters, results and remarks.

A little bit about attribute parameters

Most functions accept attributes as parameters. Attributes can have a variety of types, which would make the parameter overview quite difficult to read;

In order to preserve the readability of the tables with parameter overviews, attributes are not listed in the tables (except for the equal() function).

An attribute may contain one of the following:

- Location;
- Object;
- Timer;
- Description;
- Another attribute;
- Direction;
- Word;
- Value.

For each of the above items, an attribute parameter may be substituted. So in case there is, for example, a location listed as a parameter, this parameter may also be an attribute that holds a location.

In case an attribute contains another attribute, it will retrieve this attribute's value.
When a location or object is stored in an attribute, the attribute can be used to refer to the object's descriptions, flags, triggers or attributes. References can be 1 level deep:

printcr(r_last_visited.d_sys) is a valid statement, whereas
printcr(r_room.r_visitor.d_sys) is not.

The XVAN interpreter keeps track of the contents type of an attribute. E.g. if an attribute is loaded with a value, it cannot be used as a location. The following is invalid and will cause a runtime error:

```
r_attrib = 4503
printcr(r_attrib.d_sys)       # runtime error
```

An attribute does not have to stick with one type. The number attribute in the previous example can be assigned a direction in a new assignment statement.

Attribute parameters have the following possible syntaxes:

Attributes	Syntax	Remark
	local attribute	Will use local attribute from current location or object
	global attribute	Will use global attribute from current location or object
	location.localattribute	Location may not be a wildcard
	object.localattribute	Object may not be a wildcard
	location.globalattribute	location may be a wildcard
	object.globalattribute	object may be a wildcard

And a little bit about "strings"

Text strings are used for descriptions and as function parameters, for print() functions. Several functions for printing are available. In order to limit the number of consecutive print calls in the game source text, wildcards and ids may also be used in text strings.

So, instead of:

```
print("There ")
print(%this.r_to_be_verb)
print(" ")
print(%a)
print(" ")
print(%this)
printcr(" here.")
```

This will do the same:

```
printcr("There [this.r_to_be_verb] [a] [this] here.")
```

The string parameter may contain the following ids and wildcards:

string	Id or wildcard	remarks
	[l_<locname>]	Any location, will print d_sys
	[o_<objname>]	Any object, will print d_sys
	[m_<timer>]	Any timer, will print timer value
	[d_description]	Any description, will print the text **(*)**
	[d_attribute]	Any attribute, will print it's value **(*)**
	[l_location]	The player's current location
	[o_actor]	The actor from the user input
	[o_subject]	The subject from the user input
	[o_spec]	The specifier from the user input
	[action]	The action from the user input
	[this]	The location or object executing this trigger
	[dir]	The direction from the user input
	[prepos]	The preposition from the user input
	[value]	The value from the user input
	[ord]	The ordinal number from the user input
	[the]	Prints 'the' if the location or object following has an article defined with it.
	[a]	Prints 'a' or 'an' if the location or object following has an article defined with it.

(*) In case a local attribute or local description is used without a preceding location or object, the compiler will prefix it with the id of the location or object that is currently being compiled. For strings that are being compiled in verb definitions, there must always be a location or object prefix ,as verbs are self-contained and not associated with a particular location or object.

And a little bit more about strings

(the following section is from the XVAN introduction manual)

So, as we are creating Interactive Fiction works, text will be an important part of the story. Strings can be very lengthy, which makes the story's source code harder to read. XVAN has a number of mechanisms to format text strings so the story source is easier to read, without affecting the way how the string is printed.

Each string must end with a " or a /:
All carriage returns and spaces after a '/' will be ignored up to the next non-<cr>-or-space character.

Example:

```
d_long_descr          "This is a very very long description that /
                        goes on and on and on over several /
                        lines in the source file, but it will print /
                        as one line on the screen."
```

Will print as:
This is a very very long description that goes on and on and on over several lines in the source file, but it will print as one line on the screen."

The same effect can be achieved by ending the string with an end quote and start a new string on the next line. Consecutive strings will be combined to 1 string.

Example:

```
d_long_descr          "This is a very very long description that"
                      " goes on and on and on over several"
                       " lines in the source file, but it will print"
                      " as one line on the screen."
```

Will print as:
This is a very very long description that goes on and on and on over several lines in the source file, but it will print as one line on the screen.

String formatting characters:
The following string formatting characters are available:

- \n inserts a carriage return in the string;
- \t inserts a tab in the string;
- \" inserts a quote in the string (as a character, not an end quote);
- \\ inserts a \ in the string.

Example:

```
d_descr                    "This is a string with a \n, a \t, a \" and a \\ in it."

Will print as:
This is a string with a
, a         , a " and a \ in it.
```

XVAN functions

The remainder of this document describes the functions that are available in XVAN.

addchoice()

Purpose:
Add a choice and a response to the choices list.

Parameters:

addchoice()	Parameter 1	Parameter 2	remarks
	string of text	string of text	Delimited by “”

Effects:
Adds parameter 1 to the list of possible choices. When selected, the text string in parameter 2 is returned as user input (as if it were typed from the keyboard).

Remarks:
- If addchoice() is called in interpreter mode there will be no effect.
- Addchoice() is to be used in the t_choice trigger for locations and objects;
- The t_choice triggers will be called in choice and hybrid play mode

addjson()

Note: works only in ifi-xvan version, will be ignored by other releases.

Purpose:
IFI-XVAN maintains a json textstring that can be used by the story to communicate directly with the graphical front-end (GUI). The json string can be controlled by means of the addjson(), sendjson() and clearjson() functions.

This function adds text and or ID to the XVAN internally maintained json string.

Parameters:

addjson()	Parameter 1	Parameter 2	Parameter 3	remarks
	string	location object	--	parameter 2 is optional

Effects:
The text string and object/location ID are added to the json string.

Remarks:
- In case parameter 2 is specified, the location's or object's ID is added to the json string.
- Sample uses:
    - sending an inventory update to the GUI;
    - changing an icon in the gui (e.g. lamp on or off icon);
    - sending a map update to the gui
- The IFI Library contains predefined XVAN verb source code for interacting with the GUI.

agree()

Purpose:
Ends the further execution of a trigger and tells the interpreter that it is ok to offer the action record to other locations and objects for processing.

Parameters:
No parameters.

Effects:
Exits the trigger.

Remarks
This is the default value when returning from a trigger. If neither agree(), disagree() or nomatch() is present at the end of a trigger, the compiler will insert an agree() function.

background()

Purpose:
Change the console window background color.

Parameters:

background()	Parameter 1	Parameter 2	Parameter 3	remarks
	word	--	--	'blue' or 'black'

Effects:
Changes the color of the console window background.

Remarks:
- Blue and black are possible background colors, they must be defined as words in the vocabulary file.
- Selecting blue implies white text.
- Selecting black implies white text.

blockexit()

Purpose:
Removes an existing exit from a location.

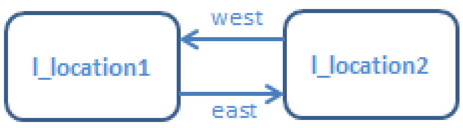
Parameters:

blockexit()	Parameter 1	Parameter 2	Parameter 3	remarks
	location	direction	--	

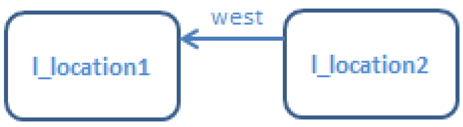
Effects:
Direction is no longer a valid exit from the location

Remarks:
The function works one-way. If there is a ‘way back’ it will remain available.

Example:



After blockexit(l_location1, east)



cansee()

Purpose:
Checks whether parameter 1 can see parameter 2

Parameters:

cansee()	Parameter 1	Parameter 2	Parameter 3	remarks
	location	location	--	
	object	object		

Effects:
Returns true or false.

Remarks:
All combinations of parameter 1 and parameter 2 are allowed.

clearflag()

Purpose:
Sets the value of a flag to 0.

Parameters:

clearflag()	Parameter 1	Parameter 2	Parameter 3	remarks
	local flag	--	--	Will use local flag from current location or object
	common flag	--		Will use common flag from current location or object
	location.localflag	--		Location may not be a wildcard
	object.localflag	--		Object may not be a wildcard
	location.commonflag	--		
	object.commonflag	--		

Effects:
Flag value is 0

Remarks:
--

clearjson()

Note: works only in ifi-xvan version, will be ignored by other releases.

Purpose:
IFI-XVAN maintains a json textstring that can be used by the story to communicate directly with the graphical front-end (GUI). The json string can be controlled by means of the addjson(), sendjson() and clearjson() functions.

This function clears the internally maintained json string.

Parameters:
No parameters.

Effects:
The json string is cleared.

Remarks:
- The IFI Library contains predefined XVAN verb source code for interacting with the GUI.

clearscreen()

Purpose:
Blank the main screen.

Parameters:
No parameters.

Effects:
Fills the main screen with the background color.

Remarks

clearstatus()

Note: works only in Glk version, will be ignored by other releases.

Purpose:
Clear the status window.

Parameters:
No parameters.

Effects:
Fills the status screen with the background color.

Remarks
--

contents()

Purpose:
Manually execute the t_entrance trigger for a location or object's contained objects, but not for the location/object itself.

Parameters:

contents()	Parameter 1	Parameter 2	Parameter 3	remarks
	location			
	object			

Effects:
Trigger t_entrance will be executed for the contained objects of Parameter 1 but not for parameter 1 itself.
Contents() has the following possible return values:

- In case all responses to t_entrance were either agree() or nomatch() the entrance() function will return true.
- In case one of the objects or the location returned disagree(), the entrance() function will return false.

Remarks:

- t_entrance is a system defined trigger. It is available by default with empty code. The story author can redefine the t_entrance trigger in location and object definitions. It is the responsibility of the story author to call the entrance() function.
- In case t_entrance must be called for the Parameter 1 location or object as well as for its contained objects, function entrance() must be used.

count()

Purpose:
Determine the number of objects in a location or object contains that comply with the value of the flag parameter.

Parameters:

count()	Parameter 1	Parameter 2	Parameter 3	Parameter 4	remarks
	location	Common flag	0 or 1	[level]	
	object				

Effects:
Returns a number.

Remarks:
- The level parameter indicates the number of levels to include under the location or object in parameter 1;
- Parameter 1 is not included in the count;
- Level has default value 1;
- Level value 0 denotes all levels.

Example 1: count(l_room, f_takeable, 1) will return the number of objects that are owned by l_room with common flag f_takeable set to 1.

Example 2: count(o_bed, f_takeable, 1, 2) will return the number of objects that are max 2 levels in the containment tree below o_bed and with common flag f_takeable set to 1.

debug()

Purpose:
Turn on debug mode.

Parameters:

debug()	Parameter 1	Parameter 2	Parameter 3	remarks
	number	--	--	Number can be 0, 1 or 2

Effects:
Debug(1) prints indented messages when entering and leaving triggers.
Debug(2): debug(1), plus additionally prints function calls (function name and parameter values).
Debug(0): turns off debug mode.

Remarks:
- To include debug information with a story, the story must be compiled with the “-d” option (without quotes) on the command line;
- When no debug information is stored with the story, a message will be printed;
- To show that a story contains debug information, the user input prompt will change from “> “ to “debug> “

dest()

Purpose:
Find the destination when going in a certain direction without actually going there.

Parameters:

dest()	Parameter 1	Parameter 2	Parameter 3	remarks
	location	direction	--	
	object			

Effects:
Returns the location when traveling from the location in parameter in the direction in parameter 2..

Remarks:
When parameter 1 is an object, the object's location will be used.

disagree()

Purpose:
Ends the further execution of a trigger and tells the interpreter not to offer the action record to other locations and objects for processing.

Parameters:
No parameters.

Effects:
Exits the trigger.
Further handling of the action record will be terminated.

Remarks
--

distance()

Purpose:
Determine the distance between locations or objects.

Parameters:

distance()	Parameter 1	Parameter 2	Parameter 3	remarks
	location	location		
	object	object		

Effects:
Returns an integer that denotes the number of moves between parameter 1 and parameter 2.

Remarks:
- All combinations of parameter 1 and parameter 2 are allowed;
- In case of an object parameter, the object will be replaced by the location it is contained in;
- Only valid exits are considered when calculating the distance (e.g. flags that prohibit exiting a location are not take into account);
- When no route exists, the value '-1' is returned.

div()

Purpose:

Integer division

Parameters:

div	Parameter 1	Parameter 2	Parameter 3	remarks
	attribute	attribute	attribute	
	timer	timer	timer	
		value	value	

Effects:

Calculates the integer division of parameter 2 and parameter 3 and stores the result in parameter 1.

Remarks:

Syntax:

- par1 = par2 div par3.
- If parameter 2 and/or 3 are attributes, they must have values (you cannot divide locations, directions, etc, only numbers).
- Parameter 1 previous contents is replaced by the result of the division.
- All combinations of parameter 1, parameter 2 and parameter 3 are allowed.
- To obtain the remainder of the division, use function rem().
- Examples:
  2 div 3 = 0
  5 div 3 = 1
  9 div 3 = 3
  11 div 3 = 3

entrance()

Purpose:
Execute the t_entrance trigger for a location or object's contained objects, as well as for the location/object itself.

Parameters:

entrance()	Parameter 1	Parameter 2	Parameter 3	remarks
	location			
	object			

Effects:
Trigger t_entrance will be executed for the Parameter 1 location or object and its contained objects. Entrance() has the following possible return values:

- In case all responses to t_entrance were either agree() or nomatch() the entrance() function will return true.
- In case one of the objects or the location returned disagree(), the entrance() function will return false.

Remarks:

- t_entrance is a system defined trigger. It is available by default with empty code. The story author can redefine the t_entrance trigger in location and object definitions. It is the responsibility of the story author to call the entrance() function.
- In case t_entrance must be called for the contained objects but not for the parameter 1 itself, the function contents() must be used.

For an example of the use of the entrance() function, see the example for the exit() function.

equal()

Purpose:
Checks whether parameter 1 and parameter 2 have identical values

Parameters:

equal()	Parameter 1	Parameter 2	Remark
	location	location	
	object	object	
	local attribute	Local attribute	Will use local attribute from current location or object
	global attribute	global attribute	Will use global attribute from current location or object
	location.localattribute	location.localattribute	Location may not be a wildcard
	object.localattribute	object.localattribute	Object may not be a wildcard
	location.globalattribute	location.globalattribute	
	object.globalattribute	object.globalattribute	
	timer	timer	Will use the timer value
	direction	direction	
	word	word	
	value	value	

Effects:
Returns true or false.

Remarks:

- All combinations of parameter 1 and parameter 2 are allowed.
- In case parameter 1 and parameter 2 are of different types, the function will return false;
- In case a parameter is an attribute, the attribute's type and value will be used in the comparison.

exit()

Purpose:
Execute the t_exit trigger for location /object and all its contained objects.

Parameters:

exit()	Parameter 1	Parameter 2	Parameter 3	remarks
	location			
	object			

Effects:
Trigger t_exit will be executed for the Parameter 1 location or object and its contained objects.
Exit() has the following possible return values:

- In case all responses to t_exit were either agree() or nomatch() the exit() function will return true.
- In case one of the objects or the location returned disagree(), the exit() function will return false.

Remarks:

- t_exit is a system defined trigger. It is available by default with empty code. The story author can redefine the t_exit trigger in location and object definitions. It is the responsibility of the story author to call the exit() function.

Example of a trigger that moves the o_player object around, using the exit() function:

```
TRIGGERS
“[dir]”   -> t_move                  # dir is a wildcard to denote a direction

t_move
   if valdir(l_location, dir) then
      if exit(l_location) then       # call all t_exit triggers
        move(o_player, dir)          # move updates current location
        entrance(l_location)         # call all t_entrance triggers
      endif
   else
      printcr(“You can’t go that way.”)
      nomatch()                      # let other objects react.
   endif
   agree()
```

firstdir()

Purpose:
Determine the direction of the first move to a location.

Parameters:

firstdir()	Parameter 1	Parameter 2	Parameter 3	remarks
	location	location		
	object	object		

Effects:
Returns the direction for the first move when traveling from parameter 1 to parameter 2.

Remarks:
- All combinations of parameter 1 and parameter 2 are allowed;
- In case of an object parameter, the object will be replaced by the location it is contained in;
- Only valid exits are considered when determining the direction (e.g. flags that prohibit exiting a location are not take into account);
- When no route exists, the result %none is returned.

getspec()

Purpose:
Get information on the specifier needed to complete an action.

Parameters:
No parameters.

Effects:
Asks the user to enter the specifier necessary to process his input.

Remarks
Getspec() will typically be used in verb code.

Example of verb code:

```
…..
"Unlock [o_door]"
printcr("What do you want to unlock [the] [o_door] with?)
getspec()
```

After running getspec() a new action record will be created and offered for processing.

From the player perspective it will look like:

```
> unlock the door
How do you want to unlock the door?
>with the key
Ok, the door is now unlocked.
```

The above example assumes the location will have a trigger for "unlock [o_door] with [o_key]" to print the last line.

getsubject()

Purpose:
Get information on the subject needed to complete an action.

Parameters:
No parameters.

Effects:
Asks the user to enter the subject necessary to process his input.

Remarks
Getsubject() will typically be used in verb code.

Example of verb code:

```
…..
“take ”
printcr(“What do you want to take?)
getsubject()
```

After running getsubject() a new action record will be created and offered for processing.

From the player perspective it will look like:

```
> take
What do you want to take?
> old brass key
old brass key, taken.
```

The above example assumes the location will have a trigger for “take [o_key]” to print the last line.

goto()

Purpose:
Let's object travel to the location, using the possible exits that the locations offer.

Parameters:

goto()	Parameter 1	Parameter 2	Parameter 3	remarks
	object	location	[moves]	Parameter 3 is optional with default value 1

Effects:
This function will try to generate a path from the Parameter 1 object's current location to the parameter 2 location. If such a path exists, the object will be moved the number of moves as indicated by Parameter 3.

Remarks:

- In order to move around a non-player character, his function can be used together with a timer that fires every move. The timer should call a trigger from the npc object that contains the goto() function. The trigger must test whether the target location has been reached and then stop the timer (or set a new location).
- When using this function, it should be tested thoroughly. Depending on the map layout, it is possible that a loop occurs. If this happens the map layout must be changed to prevent looping.
- In case no route can be found, the interpreter will print a runtime error 'no route found' message.

Example of trigger that moves object o_fred to location l_dest. Based on a timer m_move_fred that fires at value 1 and calls trigger t_move.

```
t_move
  if equal(owner(o_fred), l_dest) then
# fred has reached the destination
 stopcounter(m_move_fred)
    # insert here what Fred must do when he reaches his destination
  else
# Fred is not yet at l_dest
# Move Fred to the next location on the route to l_dest
goto(o_fred, l_dest)
    # reset the timer for the next round, it fires at value 1
    setcounter(m_move_fred, 0)
  endif
  agree()
```

hitanykey()

Purpose:
Pause further execution until the player hits a key.

Parameters:
No parameters.

Effects:
Continues when the player hits a key.

Remarks
The function does not return the value of the key that was hit.

indent()

Purpose:
Add whites spaces before the next string that is printed

Parameters:

indent()	Parameter 1	Parameter 2	Parameter 3	remarks
	number	--	--	Positive or negative value
	--	--	--	

Effects:
The interpreter has an internal indent value that is set to 0 when the story starts.
When called with a value in Parameter1, the internal indent value is increased by the value of Parameter 1.
When called with no parameters, the interpreter will print the number of white spaces indicated by the interpreter's internal indent value.

Remarks:
- Negative value for parameter 1 will decrease the number of white spaces.
- In case the indent value would become negative, it will be set to zero.
- Multiple calls of indent() will add up (or decrease) the white spaces.

islit()

Purpose:
Check whether a location or an object is a source of light (i.e. it emits light) or is in the line of sight of a light source.

Parameters:

islit()	Parameter 1	Parameter 2	Parameter 3	remarks
	location			
	object			

Effects:
Returns true or false.

Remarks:

- This function is also used by the interpreter to determine whether items that the human player refers to are visible. Not visible, means the interpreter will print a message like “You cannot see a …. here” and then stop processing the user input.
- To exclude objects from the interpreter’s visibility check, the system defined common flag f_bypass for the applicable object must be set. An example of the use of f_bypass.

f_bypass flag of lamp not set

```
> turn off the lamp
It is now pitch black.
> turn on the lamp
You cannot see a lamp here
```

f_bypass flag of lamp set

```
> turn off the lamp
It is now pitch black.
> turn on the lamp
You are in a twisty passage.
```

isobject()

Purpose:
Check something is an object..

Parameters:

isobject()	Parameter 1	Parameter 2	Parameter 3	remarks
	location			
	object			
	local attribute			
	common attribute			

Effects:
Returns true when parameter 1 is an object, otherwise returns false.

Remarks:
--.

move()

Purpose:
Move an object into another object or a location.

Parameters:

Function()	Parameter 1	Parameter 2	Parameter 3	remarks
	object	location	[word]	
	object	object		
	object	direction		

Effects:

- Parameter 1 object is moved into Parameter 2 location.
- Parameter 1 object is moved into parameter2 object.
- Parameter 1 object is moved in the direction indicated by Parameter 2.
- Parameter 1's r_preposition attribute is updated with the value in parameter 3.

Remarks:

- Parameter 3 is optional. If not used, parameter 1's current preposition attribute will remain unchanged.
- It is up to the author to ensure that parameter 1 is a preposition. Any word value will be accepted.
- When using the direction parameter, it is up to the author to ensure that direction is an exit for the object’s current location.
- move() is not to be confused with the goto() function.

newexit()

Purpose:
Create a new exit between 2 locations.

Parameters:

newexit()	Parameter 1	Parameter 2	Parameter 3	remarks
	location	direction	location	

Effects:
A passage from Parameter 1 location to Parameter 3 location with the direction as specified in parameter 2.

Remarks:
The created passage is one-way.

nomatch()

Purpose:
Informs the interpreter that it should discard the fact that there was a matching action record.

Parameters:
No parameters.

Effects:
Exits the trigger.
Interpreter will continue as if there was no match. Nomatch() will also count for the contained objects.

Remarks

- This function is used to exit a trigger and leave the option open to execute the verb default code (in case none of the other locations and objects reacts to the action record).
- When a location or an object return nomatch(), the interpreter will not offer the action record to its contained objects for processing.

notimers()

Purpose:
Prevents the timers from being fired at the end of a turn.

Parameters:
No parameters.

Effects:
Timers will not be fired and associated triggers will not be executed.

Remarks

- This function is intended to be used with meta-commands like save, restore, verbose, etc. These commands are ‘outside’ the story world and therefore should not trigger times events like increasing the number of moves, battery power, npc’s moving around etc.
- The notimers() function is already in the code for applicable commands in the XVAN Library.

owner()

Purpose:
Get an object's containing location or object.

Parameters:

owner()	Parameter 1	Parameter 2	Parameter 3	remarks
	object			

Effects:
Returns the location or object that contains the Parameter 1 object

Remarks:
--

owns()

Purpose:
Check whether an object is contained in a location or object.

Parameters:

owns()	Parameter 1	Parameter 2	Parameter 3	Parameter 4	remarks
	location	object	[depth]	[word]	
	object				

Effects:
Returns true or false.

Remarks:
- The depth-parameter indicates how many levels of containment must be searched.
- Depth = 0 indicates all levels.
- Default value for depth is 1 level.
- The word -parameter must be a preposition

Example:
After $\boxed{\text{move(o\_toaster, o\_counter, on)}}$,

$\boxed{\text{owns(o\_counter, o\_toaster)}}$ will return true,

$\boxed{\text{owns(o\_counter, o\_toaster, on)}}$ will return true,

$\boxed{\text{owns(o\_counter, o\_toaster, under)}}$ will return false.

pickone()

Purpose:
Randomly select one item of a list.

Parameters:

pickone()	Parameter 1	Parameter 2	Parameter n	remarks
	Anything but a string	Anything but a string	Anything but a string	Parameters may be of different types.

Effects:
Returns one of the parameters in the parameter list.

Remarks:
There is no maximum number of parameters.

playmode()

Purpose:
Switches playmode.

Parameters:

playmode()	Parameter 1	Parameter 2	Parameter n	remarks
	word	--	--	Interpreter, choice or hybrid

Effects:
Interpreter mode: user input via keyboard . For IFI-XVAN links, compass etc work as per the settings in the story file.
Choice mode: user input via selecting a choice from a list of choices. For IFI-XVAN links, compass and sidebar are enabled in this mode.
Hybrid mode: only for IFI-XVAN. In the side bar a shortcut list will be displayed, but keyboard input is also enabled.

Remarks:
The list of possible choices is generated by t_choice triggers from locations and objects. At the beginning of each turn, the play mode is checked and if it is choice or hybrid the t_choice triggers for the current location and contained objects will be executed. Choices can be added to the list by using the addchoice() function.

print()

Purpose:
Print text on the screen.

Parameters:

owns()	Parameter 1	Parameter 2	remarks
	String of text		Delimited by “”
	commondescription		
	localdescription		
	location.commondecscription		
	object.commondescription		
	location.localdescription		Location may not be a wildcard
	object.localdescription		Object may not be a wildcard
	location		Will print last used location.d_sys
	object		Will print last used object.d_sys
	timer		Will print timer value

Effects:
Prints the Parameter 1 (content) on the screen.

Remarks:

- Print does not print a carriage return at the end of the text. If a <cr> must be printed at the end of the text, use function printcr().
- For information about using parameters in strings, please refer to page 9 (ctrl click)

printbold()

Note: works only in Glk and Linux console version, will be ignored by other releases.

Similar to print, but prints in **boldface**.

printitalic()

Note: works only in Glk and Linux console version, will be ignored by other releases.

Similar to print, but prints in *italic*.

printcr()

Similar to print(), but prints a carriage return at the end of the printed text.

printcrbold()

Note: works only in Glk and Linux console version, will be ignored by other releases.

Similar to printcr, but prints in **boldface**.

printcritalic()

Note: works only in Glk and Linux console version, will be ignored by other releases.

Similar to printcr, but prints in *italic*.

printcrstatus()

Note: works only in Glk version, will be ignored by other releases.

Similar to printcr, but text will be printed in the status window.

Remarks:

- Use function Setcursor() to position the cursor in the status window.

printstatus()

Note: works only in Glk version, will be ignored by other releases.

Similar to print, but text will be printed in the status window.

Remarks:

- Use function Setcursor() to position the cursor in the status window.

quit()

Purpose:
Stop playing the story and exit the program.

Parameters:
No parameters.

Effects:
Ends the story playing session.

Remarks:
--

rem()

Purpose:
Remainder part of division

Parameters:

rem	Parameter 1	Parameter 2	Parameter 3	remarks
	attribute	attribute	attribute	
	timer	timer	timer	
		value	value	

Effects:
Calculates the remainder of the integer division of parameter 2 and parameter 3 and stores the result in parameter 1.

Remarks:
Syntax:

- par1 = par2 rem par3.
- If parameter 2 and/or 3 are attributes, they must have values (you cannot divide locations, directions, etc, only numbers).
- Parameter 1 previous contents is replaced by the remainder of the division.
- All combinations of parameter 1, parameter 2 and parameter 23are allowed.
- To obtain the quotient of the division, use function div().
- Examples:
  2 rem 3 = 2
  5 rem 3 = 2
  9 rem 3 = 0
  11 rem 3 = 2

restart()

Purpose:
Start the game from scratch.

Parameters:
No parameters.

Effects:
The same as quitting the game and starting it again.

Remarks:
- The restart() function re-reads all dynamic data from the story file to restore the initial game situation. If you add more code after the restart() function, the behavior may be unexpected because the system variables have been reset to their initial values;
- If transcript mode was active, it will remain active after a restart.

restore()

Purpose:
Restores a session previously saved by the save() function.

Parameters:
No parameters.

Effects:
Restores the story playing session to the situation saved earlier by the save() function.

Remarks:
This function will search for the file save.dat in the current directory.

rnd()

Purpose:
Calculate a random number.

Parameters:

rnd()	Parameter 1	Parameter 2	Parameter 3	remarks
	value	value	--	Value must be a number

Effects:
Will return an integer between Parameter 1 and Parameter 2 (borders included).

Remarks:
Example: rnd(1,3) will return either 1, 2 or 3.

runcommon()

Purpose:
Run common trigger code from a local trigger.

Parameters:
No parameters.

Effects:
When called from a local trigger, this function will run the local trigger's common trigger code (if any) and then return to the trigger.

Remarks:

- Runcommon() is a test function, it must be used in an *"if runcommon() then"* clause;
- Common trigger code is made to be executed when there is a matching action record for the user input but no local trigger. Runcommon() allows to execute the common trigger code from within the local trigger;
- Calling runcommon() looks a bit like returning nomatch() from a trigger, but the difference is that runcommon() will return to the trigger it was called from, whereas nomatch() won't;
- Runcommon() cannot be called from verb code or common trigger code;
- If runcommon() is called from a local trigger that has no corresponding common trigger, a runtime error will be thrown by the interpreter;
- Typical use for runcommon() can be where an object will do everything that's in the common trigger code but needs a few lines extra.

runverb()

Purpose:
Run verb code from a trigger.

Parameters:
No parameters.

Effects:
When called from a trigger, this function will run the verb code for the current action and then return to the trigger.

Remarks:

- Runverb() is a test function, it must be used in an *“if runverb() then”* clause;
- Verb code is made to be executed when there is no matching action record for the user input. Runverb() allows to execute the verb code from within a trigger;
- Runverb() looks a bit like returning nomatch() from a (common) trigger, but the difference is that runverb() will return to the trigger it was called from, whereas nomatch() won’t;
- Runverb() cannot be called from verb code;
- Typical use for runverb() can be where an object will do everything that’s in the verb code but needs a few lines extra. E.g. dropping an item (which can be handled by the verb) and after that setting some object specific flag that is not in the verb code.

save()

Purpose:
Save the current story progress.

Parameters:
No parameters.

Effects:
Saves the current progress so it can be restored later by using the restore() function.

Remarks:
This function will save the current story progress in file save.dat in the current directory.

sendjson()

Note: works only in ifi-xvan version, will be ignored by other releases.

Purpose:
IFI-XVAN maintains a json textstring that can be used by the story to communicate directly with the graphical front-end (GUI). The json string can be controlled by means of the addjson(), sendjson() and clearjson() functions.

This function sends the internally maintained json string to the GUI.

Parameters:
No parameters.

Effects:
The json string is sent.

Remarks:
- The IFI Library contains predefined XVAN verb source code for interacting with the GUI.

score()

Purpose:
Give instructions to parser in case of ambiguity when parsing user input .

Parameters:

score()	Parameter 1	Parameter 2	Remark
	number	--	

Effects:
Certain combinations of actor, subject and specifier get higher priority.

Remarks:
- This function can only be used in the AMBIGUITY_RULES section for verbs.
- Example: when parsing subjects for a 'drop' command, subjects that are held by the player get a higher score than subjects that are not.

```
VERB drop
 "drop [o_subject]"
  AMBIGUITY_RULES
   if owns(o_actor, o_subject) then score(5)
  END_RULES
  move(o_subject, owner(o_subject))
ENDVERB
```

- Disambiguation is used for the parser to make a guess what the player might have meant. In the example, if the player deliberately enters a drop command for something he's not carrying, there will be no disambiguity and hence no ambiguity rules will be executed. The verb default code should check for this. In the example:

```
VERB drop
 "drop [o_subject]"
  AMBIGUITY_RULES
   if owns(o_player, o_subject) then score(5)
  END_RULES
  if not(owns(o_player, o_subject) then
   print("But you are not holding [the] [o_subject])
  else
move(o_subject, owner(o_subject))
  endif
ENDVERB
```

setcursor()

Note: works only in Glk version, will be ignored by other releases.

Purpose:
Position the cursor in the statuswindow.

Parameters:

setcursor()	Parameter 1	Parameter 2	Remark
	x-position	y-position	both parameters must be numbers

Effects:
Positions the cursor in the statuswindows on position (x,y).

Remarks:

- In case the cursor is positioned 'outside' of the status window, subsequently printed text will not be visible.

setflag()

Purpose:
Set the value of a flag to 1.

Parameters:

setflag()	Parameter 1	Parameter 2	Parameter 3	remarks
	commonflag			
	localflag			
	location.commonflag			
	object.commonflag			
	location.localflag			Location may not be a wildcard.
	object.localflag			Object may not be a wildcard.

Effects:
The Parameter 1 flag will be set to 1.

Remarks:
--

shuffle()

Purpose:
Randomly change the order of a location or object's contained objects list.

Parameters:

shuffle()	Parameter 1	Parameter 2	Parameter 3	remarks
	location	--	--	
	object	--	--	

Effects:
Randomizes the response order from contained objects.

Remarks:
Action records are offered to a location or object's contained objects according to the contained_objects_list (which is an internal XVAN variable). This function will change the order of the list randomly, thus randomizing in which order the contained objects will respond.

starttimer()

Purpose:
Start a timer.

Parameters:

starttimer()	Parameter 1	Parameter 2	Parameter 3	remarks
	timer			

Effects:
Timer status will be set to start.

Remarks:
The timer will start from its current value.

stoptimer()

Purpose:
Stop a timer.

Parameters:

stoptimer()	Parameter 1	Parameter 2	Parameter 3	remarks
	timer			

Effects:
Timer status will be set to stop.

Remarks:
--

synchronize()

Purpose:
Execute a common trigger for objects that are contained in a location or object and that comply with the value of the flag parameter.

Parameters:

synchronize()	Parameter 1	Parameter 2	Parameter 3	Parameter 4	Parameter 5
	location	common trigger	common flag	0 or 1	[level]
	object				

Effects:
Returns the number of the objects that executed their trigger.

Remarks:
- The level parameter indicates the number of levels to include under the location or object in parameter 1;
- Parameter 1 is not included;
- Level has default value 1;
- Level value 0 denotes all levels.
- Objects whose trigger returns nomatch() will not be counted.
- In case a trigger returns disagree(), execution of the synchronize() function will stop for the remaining objects (= normal behavior for disagree() function.)

Example 1: synchronize(l_room, t_take_by_fred , f_takeable, 1) will execute trigger t_take_by_fred for the objects that are owned by l_room with common flag f_takeable set to 1. It will return the number of objects that actually executed the trigger.

testflag()

Purpose:
Test whether a flag has been set (value = 1).

Parameters:

testflag()	Parameter 1	Parameter 2	Parameter 3	remarks
	commonflag			
	localflag			
	location.commonflag			
	object.commonflag			
	location.localflag			Location may not be a wildcard.
	object.localflag			Object may not be a wildcard.

Effects:
Returns true or false.

Remarks:
Example from a trigger using the testflag() function.

```
t_open
  if testflag(this, f_locked) then
    printcr(“But [the] [this] is locked. You must unlock it first.”)
  endif
  agree()
```

testmode()

Purpose:
Read user input from a file rather than from the keyboard.

Parameters:
No parameters.

Effects:
The interpreter will start reading the input from file "testinput.txt". After the end of the file has been reached, the interpreter will switch back to keyboard input.

Remarks:
The input file must be in the same directory as the interpreter executable.

text()

Purpose:
Change the text color.

Parameters:

text()	Parameter 1	Parameter 2	Parameter 3	remarks
	word	--	--	'blue' or 'black'

Effects:
Changes the text color.

Remarks:

- Blue and black are possible text colors, they must be defined as words in the vocabulary file.
- Selecting blue implies white background.
- Selecting black implies white background.

transcript()

Purpose:
Copy all user input and interpreter output to a file.

Parameters:
No parameters.

Effects:
The interpreter will copy the user input and the interpreter output to file "transcript.txt". Calling transcript() when it is already active will turn it off.

Remarks:

- The output file is created in the directory where the interpreter executable is located;
- If the output file already exists, the output will be added at the end of the file.

trigger()

Purpose:
Execute a trigger.

Parameters:

trigger()	Parameter 1	Parameter 2	Parameter 3	remarks
	commontrigger			
	localtrigger			
	location.commontrigger			
	object.commontrigger			
	location.localtrigger			Location may not be a wildcard.
	object.localtrigger			Object may not be a wildcard.

Effects:
Executes the Parameter 1 trigger.

Trigger() has the following possible return values:

- If the Parameter 1 trigger returned agree() or nomatch(), the trigger() function will return true.
- If the parameter 1 trigger returned disagree() the trigger() function will return false.

Remarks:
This function must be used in an IF-clause

try()

Purpose:
Tries to execute a command in a text string for the given scope.

Parameters:

try()	Parameter 1	Parameter 2	Parameter 3	Parameter 4	remarks
	location	0 or 1	0 or 1	string of text	
	object				
	common attribute				
	local attribute				
	location.commonattribute				
	location.localattribute				
	object.commonattribute				
	object.localattribute				

Effects:
Creates an action record from the text string parameter 3 and offers it to the location/object in parameter 1 for execution.

Remarks:
- This function must be used in an IF-clause;
- The action record is offered to parameter 1 and all objects that it contains (unless the scope of the verb in parameter 3 is all_locs);
- Parameter 2 tells if output must be suppressed (1 means mute);
- Parameter 3 tells if the timers must be fired after the execution ends (1 means fire);
- Parameter 4 must have the syntax as used in the trigger section of locations/objects;
- Parameter 4 must be a complete command (strings like "get" will throw an error[1]
  Example: suppose we have an npc walking around who will try to take anything from a chest:

```
t_clear_chest
  if cansee(o_npc, o_chest) then
      if try(o_npc, 1, 0, “[o_npc], open [o_chest]”) then
          if not(try(o_npc, 1, 0, “[o_npc], take [o_all] from [o_chest]”) then
              disagree()
```

The advantage is we don't have to test anything: verb open will check if chest is already open, will check if npc has the key when the chest is locked, will unlock and open it, etc. And because we mute any replies there will be no messages printed on the screen. It will all go unnoticed by the player.

(it's a bit more complicated, e.g. what to do when the player is in the same room as the chest? But this example illustrates the purpose of try().

[1] Assuming the Get verb will ask "what do you want to get"and then call GetSubject().

undo()

Purpose:
Revert to the game state after the previous turn.

Parameters:

undo()	Parameter 1	Parameter 2	Parameter 3	remarks
	[word]	--	--	clear

Effects:
Restores the situation from before the last move.

Remarks:

- The number of turns that can be undone varies, depending on the information that is stored. This is game dependent and is determined by the interpreter;
- The undo buffer is circular. When it gets full, the oldest move is deleted from the buffer;
- The undo history is not stored with save games;
- Meta commands like save, transcript, restart etc cannot be undone;
- The parameter is optional and may only have value "clear".
- When "clear" is specified, the undo history is erased. This option can be used e.g. to prevent trial-and-error guessing.

valdir()

Purpose:
Check whether a direction is a valid exit.

Parameters:

valdir()	Parameter 1	Parameter 2	Parameter 3	remarks
	location	direction		
	commonattribute	commonattribute		
	localattribute	localattribute		
	location.commonattribute	location.commonattribute		
	object.commonattribute	object.commonattribute		
	location.localattibute	location.localattibute		Location may not be a wildcard.
	object.localattribute	object.localattribute		Location may not be a wildcard.

Effects:
Returns true or false.

Remarks:
All combinations from parameter 1 and parameter 2 are allowed.

wait()

Purpose:
Wait one or more moves

Parameters:

wait()	Parameter 1	Parameter 2	Parameter 3	remarks
	number	--	--	

Effects:
The number of moves indicated in Parameter 1 will pass without further user input. For each move, the timers will be processed.

Remarks:
--

yesno()

Purpose:
Ask the player to enter yes or no.

Parameters:
No parameters.

Effects:
Returns true if the player enters yes.

Remarks
- Input is not case sensitive
- This function loops until the players enters one of the following inputs:
  - "yes"
  - "y"
  - "no"
  - "n"

+

Purpose:
Add parameters

Parameters:

+	Parameter 1	Parameter 2	Parameter 3	remarks
	attribute	attribute	attribute	
	timer	timer	timer	
		value	value	

Effects:
Parameter 2 and 3 are added and the result is stored in parameter 1.

Remarks:
Syntax:

- par1 = par2 + par3.
- If parameter 2 and/or 3 are attributes, they must have values (you cannot add locations, directions, etc, only numbers).
- Parameter 1 previous contents is replaced by the result of the addition.
- All combinations of parameter 1, parameter 2 and parameter 3 are allowed.

+=

Purpose:
Add parameters

Parameters:

+=	Parameter 1	Parameter 2	Parameter 3	remarks
	attribute	attribute	--	
	timer	timer		
		value		

Effects:
Parameter 2 is added to parameter 1.

Remarks:
Syntax:

- par1 += par2.
- Parameter 1 must contain a value.
- If parameter 2 is an attribute, it must have a value (you cannot add locations, directions, etc, only numbers).
- Parameter 1 previous contents is replaced by the result of the addition.
- All combinations of parameter 1 and parameter 2 are allowed.

-

Purpose:
Subtract parameters

Parameters:

-	Parameter 1	Parameter 2	Parameter 3	remarks
	attribute	attribute	attribute	
	timer	timer	timer	
		value	value	

Effects:
Parameter 3 is subtracted from parameter 2 and the result is stored in parameter 1.

Remarks:
Syntax:
- par1 = par2 - par3.
- If parameter 2 and/or 3 are attributes, they must have values (you cannot subtract locations, directions, etc, only numbers).
- Parameter 1 previous contents is replaced by the result of the subtraction.
- All combinations of parameter 1, parameter 2 and parameter 3 are allowed.

-=

Purpose:
Subtract parameters

Parameters:

-=	Parameter 1	Parameter 2	Parameter 3	remarks
	attribute	attribute	--	
	timer	timer		
		value		

Effects:
Parameter 2 is subtracted from parameter 1.

Remarks:
Syntax:

- par1 -= par2.
- Parameter 1 must contain a value.
- If parameter 2 is an attribute, it must have a value (you cannot subtract locations, directions, etc, only numbers).
- Parameter 1 previous contents is replaced by the result of the subtraction.
- All combinations of parameter 1 and parameter 2 are allowed.

*

Purpose:
Multiply parameters

Parameters:

*	Parameter 1	Parameter 2	Parameter 3	remarks
	attribute	attribute	attribute	
	timer	timer	timer	
		value	value	

Effects:
Parameter 2 and 3 are multiplied and the result is stored in parameter 1.

Remarks:
Syntax:

- par1 = par2 * par3.
- If parameter 2 and/or 3 are attributes, they must have values (you cannot multiply locations, directions, etc, only numbers).
- Parameter 1 previous contents is replaced by the result of the multiplication.
- All combinations of parameter 1, parameter 2 and parameter 3 are allowed.

*=

Purpose:
Multiply parameters

Parameters:

*=	Parameter 1	Parameter 2	Parameter 3	remarks
	attribute	attribute	--	
	timer	timer		
		value		

Effects:
Parameter 1 is multiplied by parameter 2.

Remarks:
Syntax:

- par1 *= par2.
- Parameter 1 must contain a value.
- If parameter 2 is an attribute, it must have a value (you cannot multiply locations, directions, etc, only numbers).
- Parameter 1 previous contents is replaced by the result of the multiplication.
- All combinations of parameter 1 and parameter 2 are allowed.