FoY Verbs Documentation

version

Erek Starwind

September 10, 2025

Contents

Welcome to the FoY Verbs documentation!

Introduction

FoY Verbs is a template that allows you to recreate those classic SCUMM games, as well as the modern iteration Thimbleweed Park(tm) by Terrible Toybox, Inc.

Setting up the template with all your custom graphics and colors has been simplified a lot. All settings are grouped in the submodule TemplateSettings.asc. There is no need anymore for digging around in the single modules. The messages for unhandled events are defined in the TemplateSettings modules as well.

If you like to create a game with this template, you have to re-think some concepts you use when creating a game with AGS.

The two new concepts are: event handling and default actions, using extensions.

Let's take a look at the event handling, or in other words: stuff that happens after you have clicked on something. Normally you create a function for each event of an object, a hotspot etc. Functions like cup_Interact() or cup_Look(). Using this template you only need one the any click event. Inside these functions you distinguish between the different interactions. So a typical any_click function looks like this:

```
function cup_AnyClick()
{
        // LOOK AT
        if(Verbs.UsedAction(eGA_LookAt)) {
                player.Say("It's a blue cup.");
        }
        // USE
        else if(Verbs.UsedAction(eGA_Use)) {
                player.Say("I'd rather pick it up.");
        }
        // PICKUP
        else if(Verbs.UsedAction(eGA_PickUp)) {
                player.Say("Okay.");
                Verbs.AnyClickWalkLookPick(108, 100, eDir_Up, "You are now mine.",oCup.ID, i
        }
        //USE INV
        else if(Verbs.UsedAction(eGA_UseInv)) {
                Verbs.Unhandled();
        }
        // don't forget this
        else Verbs.Unhandled();
}
```

The function "AnyClickWalkLookPick" is explained in the scripting reference. AnyClickWalkLookPick

So you see, everything is inside a single function, instead 4 seperate functions. Also instead of checking the cursormodes, the function Verbs.UsedAction is called to determine the event/action. The current defined actions are:

```
eGA_LookAt, eGA_TalkTo, eGA_GiveTo, eGA_PickUp, eGA_Use, eGA_Open, eGA_Close, eGA_Push, eGA_
```

For inventory items, it's a little bit different, because there is no any_click event in the room editor. So you first start with "other click on inventory item", which creates the function iCup_OtherClick (in case you have an item, called iCup). Now copy this function name and paste it at other events, like Interact, look, talk and so on. In the end, you only have one function name in all five events. You can also take a look at the sample items.

The second main aspect of the GUI are the extensions. You add an extension to a location (Hotspots, Objects etc.) by editing its description.

For example, let's take an object. In the sample room, the object is called oCup and the description is simply "Cup". When move the cursor over this cup and no extension is defined, the default action will be "look at". Also the corresponding verb button in the gui starts to highlight. Now we can change this behaviour with adding an angeled bracket, followed by one of the following letters:

- n: nothing / default

- g: give to
- p: pick up
- u: use
- o: open
- l: look at
- s: push
- c: close
- t: talk to
- y: pull
- v: variable extension
- e: exit

Let's give oCup the description "Cup>p". Now the right-click action has changed. If you now move the mouse on the cup, the verb button "Pick Up" is highlighted. If you right-click the object, the any_click function from above is called. It checks for the used action and will perform the chosen action. Extensions are also explained in the function reference.

The last thing you should know about, is the global variable "ItemGiven". If you like to give an item to a character, use this variable to check, which item has been given. For example:

```
if (Verbs.UsedAction(eGA_GiveTo))
{
        if (ItemGiven == iCup)
        {
                player.Say("Do you want this Cup?");
                cBman.Say("No, thank you.");
        }
        else if (ItemGiven == iKey)
        {
                player.Say("Is that your key?");
                cBman.Say("Of course. You have it from me.");
        }
        else Verbs.Unhandled();
}
else Verbs.Unhandled();
```

If you need to use "ItemGiven" in other scripts than the global one, you need to import it manually. It's not defined via global variable pane inside the AGS editor.

Using Inventory Items

There are currently three ways of using an inventory item, you can choose from.

1. **"Use" only**

   For this, you need to add the use-extention ">u" to the description of the item and an event function for "Interact inventory item". If you have followed the instructions in this manual you probably already have it there. This option might come handy for a watch. Clicking on it always gives you the current time. You can not give it away or use it with different items.

2. **"Use" and "Use with"**

   Here you need to remove the use-extension from the description, but still keep the event function. This allows the player to directly use the item by clicking on the "use"-verb first. Directly clicking the items results in "use-with". Sticking to the watch-example: using the watch with the verb-button sets an alarm. Clicking directly on it in the inventory results in "use with", so you can use the watch with a shelf to hide it there. But please note that it might be hard for the player to understand, that using the verb button and using the inventory directly are two different things.

3. **“Use with” only**

   For the last option, you need to remove the use-extension and remove the event function. Yep, that’s right: on the right side of “Interact inventory item” is no function at all. If you then use the item, whether it’s via the verb-button or a direct click, the action always stays “use with”.

Exit Extensions

You can add an exit extension to hotspots and objects. Clicking on such a hotspot will make the player walk to it and change the room afterwards. There are several advantages compared to the usual methods like ‘screen edges’ or stand-on hotspot functions:

- works with objects and vertical hotspots (like cave entrances)
- supports double click to skip the walking
- optional walking off the screen: if you set the exit hotspot towards a screen edge, you can make the player leave the screen and change the room after that.

This is how it works: First of all create your hotspot and let it have the ‘>e’ extension. Now switch over to the events (that little flash) and add the Usermode_1 hotspot event. Eventually you’ll end in the room script with a function called ‘hExit_Mode8’. In that function, all you have to do is to script the room change. e.g.

```
player.EnterRoom(1, 76, 111, eDir_Right, true);
```

This function is almost similar to the AGS function player.ChangeRoom, you can look it up in the function reference below. If you want the player to leave the screen, you have to change the extension of the hotspot. These ones are possible:

- el: left
- er: right
- eu: up
- ed: down

If you have an exit on the right side of your screen and want the player to leave the screen on that side, your hotspot description should be called:

```
Exit>er
```

Now the character will walk to the clicked location and keeps on walking for another 30 extra pixels. That offset can be changed in the script header. If you simply call your hotspot:

```
Exit>e
```

No additional walking will occur. This is useful for exits not being at the screen border. There’s also an example in the second room of the demo template.

Language & Translation

Currently the GUI supports German, French, Spanish, Italian, Portuguese and Dutch. If you like to help translating this template, please drop me a PM at the AGS Forums.

If you like to create your game in a different language than English, you need to setup a different default language. In the TemplateSettings.asc module you’ll find the line:

```
Verbs.VerbGuiOptions[eVerbGuiTemplateLanguage] = eLangEN;
```

At the time of writing, valid values are: eLangEN, eLangDE, eLangES, eLangFR, eLangIT, eLangPT, eLangNL. Setting this variable to one of these values will translate all your GUIs, including all provided dialogs, unhandled events and verb actions.

If you are providing a multilanguage game, which default language is English instead, you will probably create one or more AGS translation files. Sadly AGS won’t automatically change GUI graphics by the language preference set by running the game setup, but we got you covered with this: after generating a new language, tell your translators to look out for the line.

```
GUI_LANGUAGE
```

Now simply put a translation for that line chosing between *EN, DE, ES, FR, IT, PT or NL*. For example:

```
GUI_LANGUAGE
DE
```

Now the GUI will be set to the corresponding language when the user selects a different language by running game setup.

Math and Helper Functions

Verbs.Distance Offset GetButtonAction DisableGui EnableGui IsGuiDisabled GlobalCondition InitGuiLanguage HandleInvArrows SetDoubleClickSpeed

Distance

```
float Verbs.Distance(int x1, int y1, int x2, int y2);
```

Returns the distance between two coordinates

Offset

```
int Verbs.Offset(int point1, int point2);
```

Returns the offset between to two given values.

GetButtonAction

```
int Verbs.GetButtonAction(int action);
```

Returns the connected action of a verb button. The actions for the verb buttons are not “hard-wired” inside the GUI-script, but defined in the function SetButtonAction.

See also: SetActionButtons, AdjustLanguage

DisableGui

```
void Verbs.DisableGui();
```

This functions disables the GUI and hides it.

See also: IsGuiDisabled, EnableGui

EnableGui

```
void Verbs.EnableGui();
```

This function enables the GUI again.

See also: IsGuiDisabled, DisableGui

IsGuiDisabled

```
bool Verbs.IsGuiDisabled();
```

Returns true, if the GUI is currently disabled, false otherwise

See also: DisableGui

GlobalCondition

```
int Verbs.GlobalCondition(int parameter);
```

Used to check for conditions that are used many times in the script. For example, it's used to check, if the mouse cursor is in the inventory and the mode walk or pickup are selected. Returns 1, if the condition is true and 0 otherwise.

InitGuiLanguage

```
void Verbs.InitGuiLanguage();
```

This is a helper function to set the correct sprites for the verb GUI.

SetDoubleClickSpeed

```
void Verbs.SetDoubleClickSpeed(int speed)
```

Defines the double click speed

HandleInvArrows

```
void Verbs.HandleInvArrows()
```

Takes care of showing or hiding the inventory scroll sprites.

Unhandled Events

In order to give a the player a feedback for actions the author hasn't thought of, unhandled events come into play. With a single function, you can achieve something like "That doesn't work" or "I can't pull that", which makes a game much more authentic and alive. The messages itself are defined outside of this function, initially in TemplateSettings.asc

Unhandled

```
void Verbs.Unhandled(int door_script);
```

Use this function at the end of your any_click functions in order to cause default reactions. For example:

```
function cChar_AnyClick()
{
        if (Verbs.UsedAction(eGA_LookAt)) player.Say("He looks like he is hungry.");
        else Verbs.Unhandled();
}
```

In this example, you get a default reaction for everything but look at. The optional parameter is only used internally to make the function work with the door scripts.

Action Functions

These functions are mainly used to control the verb buttons.

UsedAction IsAction SetActionButtons SetDefaultAction SetAction SetAlternativeAction CheckDefaultAction UpdateActionBar

UsedAction

```
void Verbs.UsedAction (Action test_action);
```

Used to determine, which action has been selected by the player. Instead of checking cursor modes, this function is used.

IsAction

```
bool Verbs.IsAction(Action test_action);
```

Used to check, if the current action is the one, given in the parameter.

SetActionButtons

```
void Verbs.SetActionButtons(Action action, int btn_ID, int sprite, int sprite_highlight, cha
```

This functions connects the verb buttons with the action and is also used to assign / change the graphics of the verb buttons.

See also: AdjustLanguage

SetDefaultAction

```
void Verbs.SetDefaultAction(Action def_action);
```

Used to define, which action is being used, if no verb has been clicked. Usually this is “walk to”.

SetAction

```
void Verbs.SetAction(Action new_action);
```

Since the cursor modes are bypassed, this function defines the current action. Among other things, this function is called by clicking a verb button.

SetAlternativeAction

```
void Verbs.SetAlternativeAction(char extension, Action alt_action);
```

This function makes the right-click shortcuts work. If you use extensions like “>p” (e.g. pickup), this function makes sure, that the correct verb button is highlighted.

See also: CheckDefaultAction

CheckDefaultAction

```
void CheckDefaultAction();
```

This function checks for a given extension in hotspots, objects and characters. If there isn’t an extension, a default action is given, e.g. “Talk to” if the mouse is over a character. In case of a given extension, the default actions are being overridden. It is also defined here, which letters are causing what default action. See the chapter Extensions for more details.

See also: Extensions

UpdateActionBar

```
void UpdateActionBar();
```

This function is used to show and update the status bar. It checks for an extension, triggers the translation and renders the results on screen.

See also: TranslateAction, RemoveExtension

ToogleGuiStyle

```
void ToogleGuiStyle(int enable_new);
```

Switches between classic Scumm mode and new one.

Player functions

FreezePlayer UnfreezePlayer SetPlayer EnterRoom

FreezePlayer

```
vpod Verbs.FreezePlayer();
```

Use this function to prevent the player from moving by the following movement functions of the template.

See also: UnfreezePlayer

UnfreezePlayer

```
void Verbs.UnfreezePlayer();
```

Use this function to undo the FreezePlayer function and let the characters move again.

See also: FreezePlayer

SetPlayer

```
void SetPlayer(Character*ch);
```

Usage:

```
cEgo.SetPlayer();
```

Similar to the AGS function Character.SetAsPlayer(). The difference is, that make the previous character clickable again, whereas the new character gets unclickable.

EnterRoom

```
void EnterRoom(this Character*, int newRoom, int x, int y, eDirection dir, bool onWalkable);
```

Usage:

```
cEgo.EnterRoom(1,15,15,eDir_Left,true);
```

Similar to the AGS function Character.ChangeRoom. The difference is, that you can also define, it which direction the character should look. Using this function makes the character turn to the direction, mentioned above.

Semi-blocking movement functions

Semi-blocking means, that you can cancel the movement, but certain code is only executed, after the character has actually reached it's goal. To archive this, these functions are called inside an if-clause.

Example:

```
if(Verbs.MovePlayer(20,20)) Display("The player has reached the destination.");
```

If the player's character reaches the coordinates 20,20, the message "I'm there" is being displayed. If the movement is being cancelled by a mouseclick, the message doesn't appear.

MovePlayer MovePlayerEx GoToCharacter GoToCharacterEx NPCGoToCharacter AnyClickMove AnyClickWalk AnyClickWalkLook AnyClickWalkLookPick AnyClickUseInv GoTo WalkOffScreen SetApproachingChar

MovePlayer

```
int Verbs.MovePlayer(int x, int y);
```

Moves the player character around on walkable areas, a wrapper for MovePlayerEx. Returns 1, if the character has reached it's goal and 0 if the movement has been cancelled before.

See also: MovePlayerEx

MovePlayerEx

```
int Verbs.MovePlayerEx(int x, int y, WalkWhere direct);
```

Move the player character to x,y coords, waiting until he/she gets there, but allowing to cancel the action by pressing a mouse button. Returns 1, if the character hasn't cancelled the movement and 0 if the movement has been cancelled before. 2 is returned, if the characters has actually reached it's goal: eg. if a walkable area is being removed while the player is still moving.

GoToCharacter

```
int Verbs.GoToCharacter(Character*charid, eDirection dir, bool NPCfacesplayer, int blocking)
```

The same as GoToCharacterEx, just with the one character being the player and a default offset of x=35px and y=20px. Returns 1, if the character has reached it's goal and 0 if the movement has been cancelled before.

See also: GoToCharacterEx

GoToCharacterEx

```
int Verbs.GoToCharacterEx(Character*chwhogoes, Character*ch, eDirection dir, int xoffset, in
```

Goes to a character staying at the side defined by 'direction': 1 up, 2 right, 3 down, 4 left and it stays at xoffset or yofsset from the character. blocking: 0=non-blocking; 1=blocking; 2=semi-blocking Returns 1, if the character has reached it's goal and 0 if the movement has been cancelled before.

See also: GoToCharacter, NPCGoToCharacter

NPCGoToCharacter

```
int Verbs.NPCGoToCharacter(Character*charidwhogoes, Character*charidtogoto, eDirection dir,
```

The same as GoToCharacterEx, just with an default offset of x=35 and y=20 Returns 1, if the character has reached it's goal and 0 if the movement has been cancelled before.

See also: GoToCharacterEx

AnyClickMove

```
int Verbs.AnyClickMove(int x, int y, eDirection dir);
```

Moves the player character to the coordinates given in the parameters. If the player reaches the destination, it's turns to the given direction. Returns 1, if the character has reached it's goal and 0 if the movement has been cancelled before. You can use this kind of functions (including the movePlayer function which is called by this function), to check if the player actually reached it's destination. For example:

```
if (Verbs.AnyClickMove(130,110,eDir_Left) == 1 ) player.Say("I've reached the place.");
```

So the Message is only displayed, if the movement hasn't been cancelled.

See also: MovePlayer, MovePlayerEx

AnyClickWalk

```
int Verbs.AnyClickWalk(int x, int y, eDirection dir);
```

This function is almost similar to AnyClickMove. But it's only called, if the current action is eGA_WalkTo.

See also: MovePlayer, MovePlayerEx, AnyClickMove

AnyClickWalkLook

```
int Verbs.AnyClickWalkLook(int x, int y, eDirection dir, String lookat);
```

This function moves the player character to the given location, turns it to the given direction and lets it say the message, given in the string.

See also: AnyClickWalk

AnyClickWalkLookPick

```
int Verbs.AnyClickWalkLookPick(int x, int y, eDirection dir, String lookat, int objectID, In
```

This function starts the same as any_click_walk_look. If an object ID > 0 has been given, this object is set invisible. Afterwards the inventory item is going to be added to the player's inventory and if there's an audioclip in the parameters, that one is played too.

The function return 0 if the action has been cancelled, before the player has reached the coordinates. 1 is returned if the player has reached the given destination, but has not picked up the item. 2 is returned, if the item has been picked up.

See also: AnyClickWalkLook, AnyClickWalk

AnyClickUseInv

```
int Verbs.AnyClickUseInv (InventoryItem*item, int x, int y, eDirection dir);
```

This function moves the player to the given destination. It returns 0, if the action is unhandled, 1 is returned, if the action is handled, but has been cancelled. 2 is returned, if everything went fine. A possible usage is:

```
if (Verbs.AnyClickUseInv(iWrench,100,130,eDir_Left) == 2 ) player.Say("I will now repair thi
```

See also: AnyClickWalkLook, AnyClickWalk

GoTo

```
int Verbs.GoTo(int blocking);
```

Go to whatever the player clicked on. This function is used to intercept a walk-to event and check if the player has reached it's goal. E.g. this is used in the exit extension processing. blocking: 0=non-blocking; 1=blocking; 2=semi-blocking (default)

See also: MovePlayer, WalkOffScreen

WalkOffScreen

```
void Verbs.WalkOffScreen();
```

Handles the action of hotspots or objects with the exit extension ('>e'). Take a look at chapter about extensions to see what this function does.

See also: Extensions

SetApproachingChar

```
void Verbs.SetApproachingChar(bool enable);
```

If set to true, the player walks to other chars before talking or giving items. This behaviour is initially defined in the guiscript, this function is used to change it during runtime.

Door functions

SetDoorState GetDoorState InitObject AnyClick AnyClickSpecial

This template implements a clever door scripting system, which is a real timesaver if you use a lot of doors. It uses a hotspot for the closed door and a non-clickable object, to show the opened door. If you enter a room the first time, you have to set up its containing doors:

```
function room_FirstLoad()
{
        // Lock door on startup when entering the room
        Doors.SetDoorState(20, 2);
        Doors.InitObject(20, oDoor.ID);
}
```

This will set up a door with the id 20 to the state 2, locked. With “Doors.InitObject”, you connect the object, displaying a sprite of an opened door, with the door ID. Now let’s take a look at your hotspot’s function:

```
function hDoor_AnyClick()
{
        if (Doors.AnyClick(20, oDoor.ID, 61, 104, eDir_Left, 1, 115, 135, eDir_Down)==0) Ver
}
```

This function is explained in detail later in this document. But for starters, this is all you have to do in the room script. And looks much harder than it is, just take a look at the sample room, supplied with this template.

If you want to have the script to show the correct default actions, you also need to change a line in the gui-script: so look for a function, called VariableExtensions. In Verbs.VariableExtensions, look at this line:

```
if (r==1 && h == 1)
Verbs.OpenCloseExtension (20);
```

This tells the script, that Room 1 contains a hotspot with the id 1. This is connected to a door a door with the id 20. So now, the right-click should suggest open/close, depending on the door’s state.

To define the messages, the player character says, when approaching a door, you can access the array holding those messages anytime. Initially they are setup in the TemplateSettings.asc module.

SetDoorState

```
void Doors.SetDoorState(int door_id, int value);
```

Call this function to set a door state for the given door_id. A door can have 3 different states:

- 0 = The door is closed
- 1 = The door is open
- 2 = The door is closed and locked

See also: GetDoorState, InitObject

GetDoorState

```
int Doors.GetDoorState(int door_id);
```

Returns the current state of a door.

0 = The door is closed 1 = The door is open 2 = The door is closed and locked

See also: SetDoorState, InitObject

InitObject

```
void Doors.InitObject(int door_id, int act_object);
```

Used to set up the corresponding object, used by the door with the given id. If the state of the door is closed, the object will be invisible. Otherwise, the object will be shown. The object stays unclickable all the time.

See also: SetDoorState

AnyClick

```
int Doors.AnyClick(int door_id, int act_object, int x, int y, eDirection dir, int nr_room, i
```

This function is used in the room script in combination with the door hotspot. Parameters:

- door_id: The door id, you have defined
- act_object: The object, containing the open sprite
- x,y: the walk-to point of the door (please don't use the built in "walk-to coordinates" feature of the room editor.
- dir: the direction, the player's character should face, after it reached x,y
- nr_room: if the door is opened and walking through it, the player is being send to this room
- nr_x,nr_y: the x,y coordinates of inside of the new room
- nr_dir: after the room change, the player faces this direction

This is the main function of the door scripts. With this you connect the hotspot with the door and the player's action. If you have defined default door sounds, these are also being called in this function. Also you can't unlock a door with this function. You need Doors.AnyClickSpecial for that.

See also: AnyClickSpecial

AnyClickSpecial

```
int Doors.AnyClickSpecial(int door_id, int act_object, int x, int y, eDirection dir, int nr_
```

This function extends any_click_door with the following parameters:

- opensound: custom sound to be played, when the door is being opend
- closesound: custom sound to be played, when the door is being closed
- key: the id of the inventory item, that can unlock the door, -1 masterkey, -2 if the door cannot be unlocked
- closevalue: default 0 (closed), but you can also set 2 (locked).

See also: AnyClickAnyClick

Translation

TranslateAction AdjustLanguage AdjustGUIText

To make the verbs work with translations, strings are being used to define the button graphics, hotkeys and so on. If you like to customize your game or get it translated, you need to take a closer look at the function AdjustLanguage.

TranslateAction

```
void Verbs.TranslateAction(int action, int tr_lang);
```

This function defines the text for the verb buttons, e.g. if you click on the talk verb button, “Talk to” is being displayed in the action/status bar. The second parameter defines the returned language. If you want to customize this text, you have to edit this function.

AdjustLanguage

```
void Verbs.AdjustLanguage();
```

This function has to be called from inside the template’s game_start() function. It sets up everything related to the verb buttons, so you need to take a look at this, if you want to customize your GUI. It is also import to understand, how this function works, if you want to get you game translated. If you take a closer look at this function, you will notice the following lines:

```
Verbs.SetActionButtons(eGA_Open,    0, 59, 60, 'q');
Verbs.SetActionButtons(eGA_Close,   1, 61, 62, 'a');
Verbs.SetActionButtons(eGA_GiveTo,  2, 63, 64, 'z');
```

and so on.

Your verb buttons are initialized here, by calling the function SetActionButtons. The parameters define the following: * Defined Action/Verb * GUI-button ID * Spriteslot normal * Sprite slot highlighted * Keyboard-Shortcut.

This line

```
Verbs.SetActionButtons(eGA_GiveTo,  2, 63, 64, 'z');
```

tells the AGS:

- We want to define a button for the verb “Give”
- The buttons has the GUI-ID 2. If you take a look at the GUI “gMaingui”, you can see several buttons. The one with the ID 2 will be used for the action you define here.
- The button will use the spriteslot 63 as the default graphic and
- spriteslot 64, if it’s highlighted. This can be a little bit confusing, since if you look at gMaingui, those graphics have already been assigned. But you also need to define the graphics slots in this function, because eventually these are the ones being used.
- The last parameter defines the hotkey for this action.

You might wonder, why this function overrides the values of gMaingui. But in some other languages the translation for use could be a very long word, so you might want to swap it with something else. E.g. in german “use” means “Benutze”, so you need more space for the verb. But “pick up” can be translated to “nimm”, so you save some space here.

Unlike the 9Verb MI-Style template, the fonts can be customized and overwritten in AdjustGUIText().

See also: SetActionButtons

AdjustGUIText

```
void Verbs.AdjustGUIText();
```

This function will also be called inside the template’s game_start() function. Here you can modify the labels of your GUI buttons for the options GUI. In case you need it, you can also define alternative fonts for different languages.

See also: AdjustLanguage

Extensions

RemoveExtension AddExtension Extension ExtensionEx OpenCloseExtension VariableExtensions

Extensions are used to define the default action for the right-click. You can add extensions to characters, hotspots, objects and inventory items. To add an extension, e.g. chose an object in the room editor and take a look at the description (not the name). In the sample room, we have an object, called Cup. In addition to the name we have an angle bracket and the letter p:

```
Cup>p
```

The bracket acts as a seperator for the extension, the letter tells the script, which default action to use. By default, the template knows about the following extensions:

- n: nothing / default
- g: give to
- p: pick up
- u: use
- o: open
- l: look at
- s: push
- c: close
- t: talk to
- y: pull
- v: variable extension
- e: exit

If you like to customize or add these extensions, take a look at the function CheckDefaultAction.

You don't have to add an extension for every object and hotspot. The template also adds some default actions on it's own. The default action for Characters is "talk to", for Hotspots and Objects, it's look at. Inventory items are handled a little differently, the right-click always causes "look at", no matter what. If you left-click an item, it's usually "use with". But if you have added the extension "u", the action will be simply "use". Clicking the verb button "use" and the item afterwards would cause the same action. But it could seem a little bit unpredictable, whether an item can used by a verb button or not. With this shortcut you can make things a little bit easier. You can see this behaviour in the sample room, when opening the letter. Otherwise you would have needed something else to interact with it. But with the use-extension, it is getting opened by a single left-click. The exit extension is covered in the following chapter.

See also: CheckDefaultAction

RemoveExtension

```
void Verbs.RemoveExtension();
```

Used to remove the extension from a location (Hotspots, Objects etc.), so it doesn't get displayed in the status bar.

AddExtension

```
void Verbs.AddExtension(char extension);
```

Used to add a default extension in case the location doesn't have one.

Extension

```
char Verbs.Extension();
```

Returns the first extention of a location.

ExtensionEx

```
char Verbs.ExtensionEx(int index, String name);
```

Returns the n-th extension of the given string. This is currently used for exit extensions.

OpenCloseExtension

```
void Verbs.OpenCloseExtension(int door_id);
```

Used in combination with the door scripts. This function returns a close extension, if the door with the given id is open and vice versa.

VariableExtensions

```
void Verbs.VariableExtensions();
```

This function is called, if you have have set “v” as an extension for a certain location. Currently it is used for the OpenClose extension, but of course you can add your own variable extensions here, for example “turn on / turn off”.

Changes

What has changed between 9-Verbs MI-Style and FoY Verbs? Continue reading if you plan to upgrade your project to this template.

1. All commands are grouped inside structs to provide some sort of namespace. This makes it easier to distinct template functions and AGS functions. In the past you may have written your scripts like this:

   ```
   if ( MovePlayer(160,160) == 1 ) player.Say("I'm there!");
   ```

   In FoY Verbs you write it like this:

   ```
   if ( Verbs.MovePlayer(160,160) == 1 ) player.Say("I'm there!");
   ```

   Not much has changed, you just have to put *Verbs.* in front of according template functions.

   Currently these two structs are included in this template: * Verbs - contains all template functions * Doors - contains all door scripts and functions
2. All functions are now in CamelCase and start with a capital letter. Some functions have been renamed to match this convention. E.g. any_click_move turned into AnyClickMove.
3. Saving an loading is now completely different. If you want to continue using your old textbox based GUIs, you need to import them from your former project. Also the function GetLucasSavegameListBox is not provided anymore by this template.
4. All actions in the enum start with eGA (enum Global Action). Therefore eMA_WalkTo turned into eGA_WalkTo and eMA_Default is now eGA_Default
5. The game_start function (or better: it’s contents) moved from the globalscript to verbgui.asc
6. SetActionButtons now takes real parameters instead of an endless space-seperated string. If you need custom fonts, you now define them in AdjustGUIText().
7. The on_key_press function moved from the globalscript to verbui.asc
8. Internal variables have been renamed, e.g. GSloctype is now location_type.

Legacy Layer

I decided to remove this feature as most people prefer not to switch the template in the middle of a project.

Indices and tables

- **genindex**

Index