DynamicGuiDocs Documentation

Release latest

Dec 12, 2018

Gui Concepts

DynamicGui currently support yaml, json, xml and hocon for gui configuration. To write guis for DynamicGui you will need to know one the listed languages.

CHAPTER 1

Functions

Functions are the building blocks for customizable guis in DynamicGui. With functions you can customize guis and slots. Below are the built-in functions for DynamicGui, addons may add more. If you are interested in making functions take a look at the developer docs.

1.1 Broadcast

The broadcast function is used to send a message to all players on the server.

Usage:

```
broadcast: This is a test message!
```

Broadcasts the message "This is a test message!" to all players.

1.2 Check Level

Check a player's exp level.

Usage:

```
checklevel: 1000
```

The function would require the player to have level 1000.

1.3 Execute Command As Console

Executes a command for the player as console.

Usage:

```
executec: say Hello from the server!
```

Executes the say command from console.

1.4 Execute Command As Player

Makes a player execute a command.

Usage:

```
executep: spawn
```

Makes a player execute the spawn command.

1.5 Gui open

Makes a player open a gui.

Usage:

```
gui: test
```

Makes a player open a gui named “test”.

1.6 No Permission

Checks if a player does not have a permission.

Usage:

```
nopermission: some.permission
```

Checks if the player does not have the permission “some.permission”.

1.7 Pay with money

Allows a player to pay money.

Usage:

```
moneywithdraw: 1000
```

Makes the player pay 1000 if they have the balance avaliable.

1.8 Permission

Checks if the player has a permission.

Usage:

```
permission: some.permission
```

Checks if the player has the permission "some.permission".

1.9 Player Message

Sends the player a message.

Usage:

```
pmsg: Hello!
```

Sends the player the message "Hello!".

1.10 Remove Slot

Removes the current slot.

Usage:

```
removeslot: this
```

Removes the slot from which the function is called.

1.11 Send player to server

Sends the player to a server.

Usage:

```
send: testserver
```

Sends the player to the server "testserver".

1.12 Set data for slot

Set data for the current slot, can be used in 1.12 and below.

Usage:

```
setdata: 1
```

Sets the data value of the current slot to 1.

1.13 Set enchants for slot

Sets enchants for the current slot. Check here for the enchantment enums

Usage:

```
setenchants: DURABILITY,1
```

Sets the current slot to have level 1 durability.

1.14 Set lore for slot

Set lore for the current slot.

Usage:

```
setlore: test lore
```

Sets the lore for the current slot to “test lore”.

Also supports multi-line lore.

Usage:

```
setlore: test;lore
```

Sets the lore for the current slot to “test” on the first line and “lore” on the second.

1.15 Set name for slot

Set name for the current slot.

Usage:

```
setname: name
```

Sets the name for the current slot to “name”.

1.16 Set type for slot

Set type for the current slot.

Usage:

```
settype: STONE
```

Sets the type for the current slot to “STONE”.

1.17 Sound

Plays a sound.

Look here for sound enums for 1.9+.

Look here for sound enums for 1.8.

Usage:

```
sound: LAVA,1.0,0.5
```

Sends a lava sound to the player with 1.0 volume and 0.5 pitch.

1.18 Statistic

Get a player's statistics.

Look here for statistics.

Usage:

```
statistic: MINE_BLOCK,DIRT,10
```

Checks if the player has mined at least 10 dirt blocks.

CHAPTER 2

Slots

Slots have a number of components that can be assigned to them. Below is an example of a basic slot.

```
'0': #Index to be placed at, starting at 0
  icon: "DIRT" #Icon for the slot
```

More complex example of a slot.

```
'0': #Index to be placed at, starting at 0
  amount: 2 #2 of the item, default is 1
  icon: "DIRT" #Icon for the slot
  name: "&cDirt but red" #Name supports color codes
  lore: #Lore for the item
  - "&cThere is dirt in this slot"
  - "&fLore can also have multiple lines!"
  functions: #Functions to run on click
  - "broadcast: &fThis is a simple test"
```

Example of using fail functions.

```
'0': #Index to be placed at, starting at 0
  icon: "DIRT" #Icon for the slot
  functions: #Functions to run on click
  - "pay: 1000"
  pay-failfunctions: #Didn't have 1000 to pay
  - "pmsg: You do not have enough money"
```

CHAPTER 3

Guis

Guis like slots can have functions and failfunctions. Below is an example of a gui. If you do not know how any function works take a look at the function documentation.

```
gui-title: "&c&lTest GUI"
rows: 5
mode: "set" #Set will set items at the index, starting at 1
            #Where as 'add' will add items into the gui instead of setting at a␣
↪specific index
close: false #Whether or not slots will close the gui, false will not close the gui
locations:
- "0,64,0,world" #Locations the gui can be clicked on in the world
npc-ids:
- '0' #Npc ids that the gui will be registered to
alias:
- "test" #Command aliases

functions:
- "permission: gui.test" #Permission needed to open the gui
- "sound: LAVA,1.0,1.0" #Sound will be played if player has the permission

permission-failfunctions:
- "pmsg: &4You do not have access to this gui!"
#Player will be messaged if they do not have access
#Any function can be used in this section
#If any function fails in this area the functions will not continue
#past the failed function

'0':
  icon: "EXP_BOTTLE"
  name: "&6Level test"
  lore:
  - "&cYou have to have at least level 1337 for this to pass"
  functions:
  - "checklevel: 1337" #Checks to see if the user has 1337 exp
```

(continues on next page)

(continued from previous page)

```
  - "broadcast: &cLevel check has passed, you have at least level 1337"
'1':
  icon: "BEDROCK"
  name: "&fConsole command test"
  functions:
  - "executec: say Broadcasting from console, this test passed"
'2':
  icon: "PAPER"
  name: "&fGui test"
  functions:
  - "gui: blank"
'3':
  icon: "BARRIER"
  name: "&cClose Test"
  close: true #Since close is set to true, this will override the gui close setting
'4':
  icon: "WORKBENCH"
  name: "&6Broadcast Test!"
  lore:
  - "Used for testing message broadcasting"
  functions:
  - "broadcast: &cBroadcast test is working!" #Broadcasts when clicked
'5':
  icon: "STONE"
  name: "&4Set data test"
  functions:
  - "setdata: 1" #Sets the data to 1 when clicked
'6':
  icon: "ENCHANTMENT_TABLE"
  name: "&bEnchanting test"
  functions:
  - "setenchants: DURABILITY,1" #Adds durability 1 to the item when clicked
'7':
  icon: "REDSTONE"
  name: "&cName test"
  functions:
  - "setname: &fName test passed" #Sets the name of the item when clicked
  - "broadcast: Name test passed" #Broadcasts if the name was set
'9':
  icon: "SAND"
  name: "&aRemove test"
  functions:
  - "removeslot: this" #Removes the item when clicked
  - "broadcast: &bRemove passed" #Broadcasts if the removal was successful
'10':
  icon: "DIRT"
  name: "&aType test"
  functions:
  - "settype: GRASS" #Sets the item type to grass
  - "broadcast: &aType test passed" #Broadcasts if the item type was updated
'11':
  icon: "DIAMOND_BLOCK"
  name: "&2Money Pay test"
  functions:
  - "moneywithdraw: 100"
  - "broadcast: &aPay test passed"
'13':
```

(continues on next page)

(continued from previous page)

```
  icon: "SAND"
  name: "&aRemove load test"
  load-functions: #Fuctions that happen on load, before the gui opens
  - "removeslot: this" #Removes this slot
  - "broadcast: &bRemove load test passed" #Broadcasts if successfully removed
  functions:
  - "broadcast: Why is this broadcasting, you got removed" #Function that can be ran␣
↪if the slot does not get removed
```

CHAPTER 4

Gui Examples

CHAPTER 5

Complex Problems

There are many complex problems that can be solved with using DynamicGui.

5.1 Removing a slot

CHAPTER 6

Remote Guis

Remote guis are treated the same as regular guis but are loaded from a remote location, e.g. a webserver. These guis have to be specified in the config.yml under the "remote-guis" section. If the location of a remote gui cannot be resolved on runtime then an older backup of the file will be loaded. Below is an example of a remote gui being loaded from Github.

```
remote-guis:
  remote: #Remote identifier
    file-name: "remote.yml" #File name
    url: "https://raw.githubusercontent.com/ClubObsidian/DynamicGUIExamples/master/
↪examples/remote.yml"
```

CHAPTER 7

Function Api

The function manager is what is responsible for the registration of functions. Below you will find out how to write functions and add the functions to the function manager so they can be used in guis.

As a note any built-in functions should be platform independent, if you want more information on this see the implementing a server type documentation. This does not apply to addon plugins.

7.1 Making a function

All functions have to extend the Function class. The function class allows you to work with the player and manipulate guis.

Below is an example of a simple function that... todo

7.2 Function owners

Most functions do not need to worry about the owner of the function. However there are certain operations where it would be useful to know the owner of the function. In DynamicGui function owners are split up slots and guis. Slots refer to the individual slots in a gui and a gui is the whole gui including the slots that make up the gui.

7.3 Making a slot only function

Slot functions are slot only and are to be used if a functions can only interact with a slot.

7.4 Making a gui only function

Gui functions are gui only and can be used if a function needs to manipulate the gui on creation and on opening.

7.5 Adding a function

CHAPTER 8

Gui Api

The gui manager contains the registration of all guis. With the gui api you can check if a player has a gui open and open a gui for a player.

CHAPTER 9

Implementing a server type

Server types are the underlying implementation for DynamicGui to share a partial codebase. Wrapper classes are written to handle shared code such as Gui, Slot and Function. Events are to be fired off in their native implementation and then handled by Trident listeners.

9.1 Player Wrapper

9.2 Inventory Manager

9.3 ItemStack Manager

9.4 Events