Godot Deep Dive

Autoloads & Signals

AUCA Gamedev Club — 2026

Docs

Part 1: Autoloads

Global singletons that persist across scenes

🤔 The Problem

Imagine you have a **score** variable in your game.

- Player collects a coin → score increases ✅
- Player moves to the **next level** (scene change) → score resets to 0 ❌
- You need score to **survive scene transitions**

Scene changes destroy all nodes. Your score variable dies with them.

What is an Autoload?

A script (or scene) that Godot **loads automatically** when the game starts.

- 🌍 **Global** — accessible from anywhere via its name
- 🔄 **Persistent** — survives scene changes
- 1️⃣ **Singleton** — only one instance exists
- 📦 **Always loaded** — sits above the scene tree

```
Root
├── GameManager    ← autoload (always here)
├── AudioManager   ← autoload (always here)
└── CurrentScene   ← gets replaced on scene change
    ├── Player
    └── Enemy
```

Setting Up an Autoload

Step 1: Create the script

```gdscript
# game_manager.gd
extends Node

var score: int = 0
var high_score: int = 0
var current_level: int = 1
```

Step 2: Register it

Project → Project Settings → Autoload

Path	Name	Enabled
`res://game_manager.gd`	`GameManager`	✅

Using Autoloads

Now you can access `GameManager` from **any script** in your project:

```gdscript
# In coin.gd
func _on_collected():
    GameManager.score += 10

# In enemy.gd
func _on_killed():
    GameManager.score += 50

# In ui.gd
func _process(_delta):
    $ScoreLabel.text = "Score: %d" % GameManager.score
    $LevelLabel.text = "Level: %d" % GameManager.current_level
```

💡 No need for `get_node()` or exports — just use the autoload name directly!

Practical Autoload Examples

🎵 AudioManager

```gdscript
extends Node

var music_bus := "Music"

func play_sfx(sound: AudioStream):
    var player := AudioStreamPlayer.new()
    player.stream = sound
    add_child(player)
    player.play()
    player.finished.connect(
        player.queue_free
    )
```

💾 SaveManager

```gdscript
extends Node

const SAVE_PATH = "user://save.dat"

func save_game():
    var file = FileAccess.open(
        SAVE_PATH, FileAccess.WRITE
    )
    var data = {
        "score": GameManager.score,
        "level": GameManager.current_level
    }
    file.store_var(data)

func load_game():
    if not FileAccess.file_exists(SAVE_PATH):
        return
    var file = FileAccess.open(
        SAVE_PATH, FileAccess.READ
    )
    var data = file.get_var()
```

⚠️ Autoload Best Practices

- **Keep them focused** — one responsibility per autoload
- **Don't put everything there** — only truly global state
- **Use signals** for communication (we'll cover this next!)
- **Avoid circular dependencies** — autoloads shouldn't depend on scene nodes

✅ Good autoloads

- GameManager (score, lives)
- AudioManager
- SaveManager
- SceneTransition

❌ Bad autoloads

- Player (scene-specific)
- EnemySpawner (scene-specific)
- UIController (changes per scene)
- "GodObject" (does everything)

Part 2: Signals

The Observer pattern, Godot-style

🤔 The Problem (Again)

Your **Player** takes damage. Who needs to know?

- 💊 **HealthBar** — update the display
- 🎵 **AudioManager** — play hurt sound
- 📸 **Camera** — screen shake
- 🎮 **GameManager** — check if player died
- 😈 **Enemy** — celebrate?

```
# ❌ Without signals: Player knows about EVERYONE
func take_damage(amount):
    health -= amount
    $"../HealthBar".update(health)          # tight coupling!
    $"../AudioManager".play("hurt")         # what if path changes?
    $"../Camera".shake(0.5)                 # what if Camera doesn't exist?
    GameManager.check_death(health)         # getting messy ...
```

What is a Signal?

A way for nodes to **emit events** without knowing who's listening.

```
Player says: "Hey, I took damage!"

                ┌──→ HealthBar: "Got it, updating display"
                │
Player ──emit───┼──→ AudioManager: "Playing hurt sound"
                │
                ├──→ Camera: "Shaking screen"
                │
                └──→ GameManager: "Checking if dead"
```

- **Emitter** doesn't know (or care) who's listening
- **Listeners** connect to signals they're interested in
- Adding/removing listeners doesn't change the emitter
- **Decoupled** — nodes are independent

Defining Signals

```gdscript
# Simple signal — no data
signal died

# Signal with parameters
signal health_changed(new_health: int, max_health: int)

# Signal with complex data
signal item_collected(item_name: String, item_value: int)
```

Emitting Signals

```gdscript
# player.gd
var health: int = 100
var max_health: int = 100

func take_damage(amount: int):
    health -= amount
    health_changed.emit(health, max_health)
    if health <= 0:
        died.emit()
```

Connecting Signals

Method 1: In code (recommended)

```gdscript
# health_bar.gd
func _ready():
    # Find the player and connect to its signal
    var player = get_node("../Player")
    player.health_changed.connect(_on_health_changed)

func _on_health_changed(new_health: int, max_health: int):
    # Update the health bar
    value = float(new_health) / float(max_health) * 100
    if new_health < max_health * 0.25:
        modulate = Color.RED  # flash red when low
```

Method 2: In the Editor

1. Select the **Player** node
2. Go to **Node** tab → **Signals**
3. Double-click `health_changed`
4. Pick the target node & method

Built-in Signals

Godot nodes come with **tons** of useful signals:

Common Node Signals

```
# Node
ready
tree_entered
tree_exiting

# Timer
timeout

# Area2D
body_entered(body)
body_exited(body)
area_entered(area)
```

UI Signals

```
# Button
pressed

# LineEdit
text_changed(new_text)
text_submitted(text)

# AnimationPlayer
animation_finished(anim_name)

# HTTPRequest
request_completed(result, code,
    headers, body)
```

💡 **Tip:** Always check the **Node → Signals** tab in the editor to see available signals!

Signals + Autoloads = 🔥

The **real power** comes from combining both patterns:

```gdscript
# game_manager.gd (autoload)
signal game_over
signal score_changed(new_score: int)
signal level_completed(level: int)

var score: int = 0:
    set(value):
        score = value
        score_changed.emit(score)

# player.gd — emit through autoload
func die():
    GameManager.game_over.emit()

# ANY script can listen — no node references needed!
# ui.gd
func _ready():
    GameManager.score_changed.connect(_on_score_changed)
    GameManager.game_over.connect(_on_game_over)

func _on_score_changed(new_score):
    $ScoreLabel.text = str(new_score)

```

Real-World Example: Coin Pickup

Let's put it all together:

```gdscript
# game_manager.gd (autoload)
signal coin_collected(total: int)

var coins: int = 0

func add_coin():
    coins += 1
    coin_collected.emit(coins)
```

```gdscript
# coin.gd
func _on_body_entered(body):
    if body.is_in_group("player"):
        GameManager.add_coin()
        queue_free()
```

```gdscript
# coin_counter_ui.gd
func _ready():
    GameManager.coin_collected.connect(
        _update_display
    )
    _update_display(GameManager.coins)

func _update_display(total: int):
    text = "🪙 %d" % total
```

```gdscript
# achievement_manager.gd
func _ready():
    GameManager.coin_collected.connect(
        _check_achievements
    )

func _check_achievements(total: int):
    if total >= 100:
        unlock("coin_master")
```

⚠️ Signal Best Practices

- **Name signals as past tense events:** `health_changed`, `died`, `item_collected`
- **Keep parameters minimal** — pass only what listeners need
- **Disconnect when needed:** `signal.disconnect(callable)` to prevent leaks
- **Use** `Callable` for lambdas: `signal.connect(func(): print("hi"))`
- **One-shot connections:** `signal.connect(callback, CONNECT_ONE_SHOT)`
- **Don't chain signals** excessively — it becomes hard to debug

🐛 Debugging tip

```
# Print when signal fires — great for debugging
my_signal.connect(func(args): print("Signal fired: ", args))
```

Quick Comparison

Feature	Direct Call	Signal
Coupling	Tight 🔗	Loose 🔓
Emitter knows listener?	Yes	No
Multiple listeners?	Manual	Automatic
Easy to add new listeners?	Requires changes	Just connect
Debugging	Easier to trace	Harder to trace
Performance	Slightly faster	Slightly slower

Rule of thumb: Use signals when multiple systems need to react to the same event

Recap 🎯

Autoloads

- Global singletons
- Persist across scenes
- Access by name: `GameManager.score`
- Good for: game state, audio, saves, transitions
- Set up: Project Settings → Autoload

Signals

- Event-driven communication
- Decoupled architecture
- Define: `signal my_event`
- Emit: `my_event.emit()`
- Connect: `node.my_event.connect(func)`

Combined: Autoload signals = global event bus 🚌

Let’s Code! 🎮

Time for a hands-on exercise

1. Create a `GameManager` autoload with score + lives
2. Add signals: `score_changed`, `life_lost`, `game_over`
3. Build a simple scene that uses both patterns
4. Bonus: Add an `AudioManager` autoload

Questions? 🤔

Resources:

- Godot Docs: Singletons (Autoload)
- Godot Docs: Signals
- GDQuest: Signals Guide