MegaAssets

Table des matières

I – Définition

MegaAssets est une classe possèdant une multitude de fonctions très utiles et interressantes dont le développeur pourra s'en servir pour aller plus vite dans ces programmations. C'est également une bibliothèque de **bas niveau** sur lequel se base les modules (**haut niveau**) pour fournir un bon rendement à leurs utilisateurs. Le développeur peut créé des classes, toutes dérivées de la classe **MegaAssets**.
NB : Ne tentez jamais d'instancier MegaAssets si vous tenez à éviter de maquier la console de votre moteur de rouge.

II – Fonctionnalités disponibles

+ **Dictionary | String get_properties_data (json = false) :** Renvoie les données de toutes les propriétés appartenant au script en question.
  » **bool json :** Voulez-vous renvoyer les données au format (.json) ?

+ **Dictionary | String get_cash (json = false) :** Renvoie les données sur les caches éffectuées au cours de certains traitements.
  » **bool json :** Voulez-vous renvoyer les données au format (.json) ?

+ **static bool is_child_of (nodes, others) :** Détermine si un ou plusieurs noeud(s) sont les enfant(s) d'un ou plusieurs autre(s) noeud(s).
  » **Node | Array nodes :** Contient le(s) premier(s) noeud(s) à ciblé(s).
  » **Node | Array others :** Contient le(s) deuxième(s) noeud(s) à ciblé(s).
  » **bool recursive :** La vérification d'enfant sera t-elle récursive ?

Code : GDScript

```
# Called on first game execution.
func _ready():
    # Check a certains elements.
    print(MegaAssets.is_child_of([$Sprite, self], self.get_viewport(), true));
        # Return true
    print(MegaAssets.is_child_of([$Sprite, self], self.get_viewport(), false));
        # Return false
```

+ **static void output (message, type, object, delay = 0.0) :** Affiche le contenu du paramètre *message* dans la console de l'éditeur.
  » **String message** : Contient le méssage à affiché.
  » **int type** : Contient le type du méssage qui sera affiché. Les valeurs possibles sont :
    -> **MegaAssets.Message.NORMAL ou 0** : Afficher un méssage ordinaire.
    -> **MegaAssets.Message.WARNING ou 1** : Afficher un méssage d'alerte.
    -> **MegaAssets.Message.ERROR ou 2** : Afficher un méssage d'erreur.
  » **Node object** : Quel noeud sera considéré pour éffectuer les différentes opérations ?
  » **float delay** : Quel est le temps mort avant l'affichage du méssage ?

Code : GDScript

```
# Called on first game execution.
func _ready ():
    # Show a normal message.
    MegaAssets.output ("Normal message", MegaAssets.Message.NORMAL, self);
    # Show a warning message.
    MegaAssets.output ("Warning message", MegaAssets.Message.WARNING, self);
    # Show an error message.
    MegaAssets.output ("Error message", MegaAssets.Message.ERROR, self);
```

+ **static bool is_child (nodes) :** Détermine si un ou plusieurs noeud(s) sont considéré(s) comme des enfant(s) dans l'arbre de la scène.
  » **Node | Array nodes** : Contient le(s) noeud(s) à ciblé(s).

Code : GDScript

```
# Called on first game execution.
func _ready ():
    # Check a certains elements are child.
    print (MegaAssets.is_child ([$Sun, $effect, $Camera, self.get_viewport ()]));
        # Return false
    print (MegaAssets.is_child ([$Sun, $effect, $Camera, self]));
        # Return true
```

+ **static bool is_parent (nodes) :** Détermine si un ou plusieurs noeud(s) sont considéré(s) comme des parent(s) dans l'arbre de la scène.
  » **Node | Array nodes** : Contient le(s) noeud(s) à ciblé(s).

Code : GDScript

```
# Called on first game execution.
func _ready():
    # Check a certains elements are child.
    print(MegaAssets.is_parent([self, self.get_viewport()]));
        # Return true
    print(MegaAssets.is_parent([$Sun, $effect, $Camera, self]));
        # Return false
```

+ **static bool is_game_initialised () :** Détermine si le jeu est initialisé ou pas.

+ **static void re_parent (nodes, nparents, object, reversed = false, delay = 0.0, interval = 0.0) :** Change le parent d'un ou de plusieurs noeud(s) donné(s).
  » **NodePath | String | PoolStringArray nodes :** Contient l e(s) chemin(s) de(s) noeud(s) que l'on veut ciblé(s).
  » **NodePath | String | PoolStringArray nparents :** Contient le(s) chemin(s) de(s) future parent(s) du/des noeud(s) ciblé(s).
  » **Node object :** Quel noeud sera considéré pour éffectuer les différentes opérations ?
  » **bool reversed :** Voulez-vous inverser le sens du traitement à éffectué ?
  » **float delay :** Quel est le temps mort avant le changement ?
  » **float interval :** Quel est le délai avant chaque changement ? Notez que ce paramètre est utilisé lorsqu'on éffectue plusieurs changements de parent sur plusieurs noeuds à la fois.

Code : GDScript

```
# Called on first game execution.
func _ready():
    # Let's change a certains elements parent.
    MegaAssets.re_parent(["Camera", "Weapon", "Gun"], "Player", self, true, 0.0, 5.0);
    MegaAssets.re_parent("effect/Navigation", '.', self, false, 15.0);
```

+ **static void dont_destroy_on_load (nodes, obj, reversed = false, delay = 0.0, interval = 0.0) :** Rend indestructible un ou plusieurs noeud(s) donné(s). Ce(s) dernier(s) sera/ont donc toujours présent au changement de scène.
  » **NodePath | String | PoolStringArray nodes :** Contient le(s) chemin(s) de(s) noeud(s) que l'on veut rendre indestructible.
  » **Node obj :** Quel noeud sera considéré pour éffectuer les différentes opérations ?
  » **bool reversed :** Voulez-vous inverser le sens du traitement à éffectué ?
  » **float delay :** Quel est le temps mort avant cette action ?

» **float interval** : Quel est le délai avant chaque changement ? Notez que ce paramètre est utilisé lorsqu'on éffectue plusieurs indestructions sur plusieurs noeuds à la fois.

Code : GDScript

```
# Called on first game execution.
func _ready ():
    # Let's make indestructibles nodes.
    MegaAssets.dont_destroy_on_load (["Camera", "Weapon", "Gun"], self);
    MegaAssets.dont_destroy_on_load ("GameData", self.get_node ("DataManager"));
```

+ static void move_node (node, other, position, object, delay = 0.0) : Déplace un noeud dans un autre noeud.

» **NodePath | String node** : Contient le noeud que l'on veut déplacé.

» **NodePath | String other** : Contient le future parent du noeud à déplacé.

» **int position** : Quelle position hiéarchique occupera le future enfant du paramètre *other* ? Par défaut, l'enfant sera ajouté en dernière position.

» **Node object** : Quel noeud sera considéré pour éffectuer les différentes opérations ?

» **float delay** : Quel est le temps mort avant le déplacement ?

Code : GDScript

```
# Called on first game execution.
func _ready ():
    # Let's move some nodes.
    MegaAssets.move_node ("Camera", "Player", 0, self);
    MegaAssets.move_node ("Navigation/NavigationMeshInstance", '.', 3, self, 5.0);
```

+ static void invoke (mname, params, object, delay = 0.0) : Appel la méthode nommée dans le paramètre *mname* avec un certain temps.

» **String mname** : Contient le nom de la méthode à appelée.

» **Array params** : Contient les values des paramètres de la méthode ciblée.

» **Object object** : Contient l'instance de l'objet possédant la méthode à appelée.

» **float delay** : Quel est le temps mort avant l'appel ?

Code : GDScript

```
# Method1.
func method1 (value): return value;
```

```
# Method2.
func method2 (a: float = 2.569, b: int = 25): print ("The Result is: ", (a + b));
# Called on first game execution.
func _ready ():
    # Let's call some methods.
    MegaAssets.invoke ("method2", [15, 25], self, 2.0);
    print (MegaAssets.invoke ("method1", ["A test for invoke method"], self));
```

+ **void invoke_repeating (mname, params, instance = self, count = 1, interval = 0.0) :** Appelle la méthode nommée dans le paramètre *mname* de façon répétée en fonction du paramètre *count*. Notez qu'une valeur négative au niveau de ce dernier engendrera un appel infini. La valeur nulle n'engendre aucun appelle.
  » **String mname** : Contient le nom de la méthode à appelée.
  » **Array params** : Contient les values des paramètres de la méthode ciblée.
  » **Object instance** : Contient l'instance de l'objet possédant la méthode à appelée.
  » **int count** : Combien de fois la méthode ciblée sera appelée ?
  » **float interval** : Quel est le temps mort avant chaque l'appel ?

Code : GDScript

```
extends MegaAssets;
func method1 (): print ("Call me !");
func method2 (): print ("Repeat call !");
func method3 (): print (" Infinite call !");
# Called on first game execution.
func _ready ():
    # Call method1 once.
    self.invoke_repeating ("method1", [], self);
    # Call method2 5 times.
    self.invoke_repeating ("method2", [], self, 5);
    # Infinite call on method3.
    self.invoke_repeating ("method3", [], self, -1, 1.0);
    # No call.
    self.invoke_repeating ("method1", [], self, 0);
```

+ **bool is_invoking (mname) :** Détermine si une méthode est prise dans le processus d'appel répété provoqué par la fonction *invoke_repeating ()*.
  » **String mname** : Contient le nom de la méthode ciblée.

Code : GDScript

```
extends MegaAssets;
func method1 (): print ("Call me !");
func method3 (): print (" Infinite call !");
# Called on first game execution.
func _ready ():
    # Call method1 once.
    self.invoke_repeating ("method1", [], self);
    # Infinite call on method3.
    self.invoke_repeating ("method3", [], self, -1, 1.0);
    # The method1 is it invoking ?
    print (self.is_invoking ("method1")); # Return false
    # The method3 is it invoking ?
    print (self.is_invoking ("method3")); # Return true
```

+ **void cancel_invoke (mname, callback = {}, delay = 0.0) :** Annule le processus d'appel répété sur une méthode donnée.
  » **String mname** : Contient le nom de la méthode ciblée.
  » **Dictionary callback** : Contient des informations sur la méthode à exécutée lorsque l'invoquation d'une méthode a été éffectuée. Ce dictionaire supporte les clés suivantes :
  - **String | NodePath source** : Contient l'addresse de la méthode à ciblée. La présence de cette clé n'est pas obligatoire. Dans ce cas, on considéra que la méthode référée se trouve sur la référence actuelle.
  - **String method** : Contient le nom de la méthode à exécutée. L'utilisation de cette clé est obligatoire.

  Notez que la méthode à exécutée doit possédée qu'un seul paramètre pour contenir le nom de la méthode ayant été annulée.
  » **float delay** : Quel est le temps mort avant l'annulation ?

Code : GDScript

```
extends MegaAssets;
func method3 (): print (" Infinite call !");
func cancel_msg (method_name):
    # Print a message.
    print ("method3 call is canceled !");
    # The method3 is it invoking again ?
    print ("method3 invoking state: ", self.is_invoking (method_name)); # Return false
# Called on first game execution.
func _ready ():
```

```
# Infinite call on method3.
self.invoke_repeating("method3", [], self, -1, 1.0);
# The method3 is it invoking ?
if self.is_invoking("method3"):
    # Stop method3 call after 10 secondes.
    self.cancel_invoke("method3", Dictionary({method = "cancel_msg"}), 10);
```

+ static Node | Array search (id, ref, prop = 3, count = -1, inv = false, rec = true) : Cherche un ou plusieurs noeud(s) donné(s) dans l'arbre de la scène.

» **String | PoolStringArray | Array id** : Contient le nom/ groupe/classe du/des noeud(s) à recherché(s).

» **Node ref** : Contient le noeud de référence de la recherche.

» **int prop** : Par quel moyen la recherche sera faite ? Les valeurs possibles sont :

-> **MegaAssets.NodeProperty.NAME ou 0** : Trouve un noeud en utilisant son nom.

-> **MegaAssets.NodeProperty.GROUP ou 1** : Trouve un noeud en utilisant le nom de son groupe.

-> **MegaAssets.NodeProerty.TYPE ou 2** : Trouve un noeud en utilisant le nom de sa classe.

-> **MegaAssets.NodeProerty.ANY ou 3** : Trouve un noeud en utilisant l'un des trois moyens cités plus haut.

» **int count** : Combien de résultat seront renvoyés ? Notez qu'une valeur négative à ce niveau engendrera le renvoie de tous les résultats trouvés cours de la recherche.

» **bool inv** : Voulez-vous inverser le sens du traitement à éffectué ?

» **bool rec** : Voulez-vous utiliser un programme récursive pour trouver le(s) noeud(s) recherché(s) ?

+ static bool contains (id, scope, prop = 3, rec = true) : Détermine si le(s) identifiant(s) donné(s) sont tous présent(s) dans un ou plusieurs noeud(s) donné(s) dans l'arbre de la scène.

» **String | PoolStringArray | Array id** : Contient le nom/ groupe/classe du/des noeud(s) à recherché(s).

» **Node | Array scope** : Contient le(s) noeud(s) de référence(s) de la recherche. C'est aussi la portée de la recherche. N'utilisez que dans ce paramètre, des objets dérivant de la classe *Node*.

» **int prop** : Par quelle moyen la recherche sera faite ? Les valeurs possibles sont :

-> **MegaAssets.NodeProperty.NAME ou 0** : Trouve un noeud en utilisant son nom.

-> **MegaAssets.NodeProperty.GROUP ou 1** : Trouve un noeud en utilisant le nom de son groupe.

-> **MegaAssets.NodeProerty.TYPE ou 2** : Trouve un noeud en utilisant le nom de sa classe.

-> **MegaAssets.NodeProerty.ANY ou 3** : Trouve un noeud en utilisant l'un des trois moyens cités plus haut.

» **bool rec** : Voulez-vous utiliser un programme récursive pour trouver le(s) noeud(s) recherché(s) ?

+ **static bool is_indestructible (id, object, property = 3) :** Détermine si un ou plusieurs noeud(s) peut(vent) être détruit au changement d'une scène.
  » **String | PoolStringArray | Array id :** Contient le nom/ groupe/classe du/des noeud(s) à recherché(s).
  » **Node object :** Quel noeud sera considéré pour éffectuer les différentes opérations ?
  » **int property :** Par quelle moyen la recherche sera faite ? Les valeurs possibles sont :
    -> **MegaAssets.NodeProperty.NAME ou 0 :** Trouve un noeud en utilisant son nom.
    -> **MegaAssets.NodeProperty.GROUP ou 1 :** Trouve un noeud en utilisant le nom de son groupe.
    -> **MegaAssets.NodeProerty.TYPE ou 2 :** Trouve un noeud en utilisant le nom de sa classe.
    -> **MegaAssets.NodeProerty.ANY ou 3 :** Trouve un noeud en utilisant l'un des trois moyens cités plus haut.

Code : GDScript

```
    # Called on first game execution.
    func _ready():
        # Let's make some indestructibles nodes.
        MegaAssets.dont_destroy_on_load (["Camera", "Sprite"], self);
        # The both nodes is it don't destroy ?
        print (MegaAssets.is_indestructible (["Camera", "Sprite"], self)); # Will return false
        # Wait for idle frame.
        yield (self.get_tree (), "idle_frame");
        # The both nodes is it don't destroy ?
        print (MegaAssets.is_indestructible (["Camera", "Sprite"], self)); # Will return true
```

+ **static Node | Array get_indestructible (id, obj, prop = 3, count = -1, inv = false, rec = true) :** Cherche un ou plusieurs noeud(s) indestructible(s) donné(s) dans l'arbre de la scène. Notez que cette méthode ne renvoie que les noeuds se trouvant au sein du noeud *DontDestroyOnLoad*.
  » **String | PoolStringArray | Array id :** Contient le nom/ groupe/classe du/des noeud(s) à recherché(s).
  » **Node obj :** Quel noeud sera considéré pour éffectuer les différentes opérations ?
  » **int prop :** Par quelle moyen la recherche sera faite ? Les valeurs possibles sont :
    -> **MegaAssets.NodeProperty.NAME ou 0 :** Trouve un noeud en utilisant son nom.
    -> **MegaAssets.NodeProperty.GROUP ou 1 :** Trouve un noeud en utilisant le nom de son groupe.
    -> **MegaAssets.NodeProerty.TYPE ou 2 :** Trouve un noeud en utilisant le nom de sa classe.
    -> **MegaAssets.NodeProerty.ANY ou 3 :** Trouve un noeud en utilisant l'un des trois moyens cités plus haut.
  » **int count :** Combien de résultat seront renvoyés ? Notez qu'une valeur négative à ce niveau engendrera le renvoie de tous les résultats trouvés cours de la recherche.
  » **bool inv :** Voulez-vous inverser le sens du traitement à éffectué ?

» **bool rec** : Voulez-vous utiliser un programme récursive pour trouver le(s) noeud(s) recherché(s) ?

Code : GDScript

```
# Called on first game execution.
func _ready():
    # Let's make some indestructibles nodes.
    MegaAssets.dont_destroy_on_load(["Camera", "Sprite"], self);
    # Get insdestuctible "Sprite" node.
    print(MegaAssets.get_indestructible("Sprite", self)); # Will return null
    # Wait for idle frame.
    yield(self.get_tree(), "idle_frame");
    # Get insdestuctibles "Sprite" and "Camera" nodes.
    print(MegaAssets.get_indestructible(["Camera", "Sprite"], self));
```

+ **static Array get_all_indestructible_nodes (object) :** Renvoie tous les noeuds à l'intérieur du noeud ayant le nom *DontDestroyOnLoad*.
  » **Node object** : Quel noeud sera considéré pour éffectuer les différentes opérations ?

+ **static Variant get_data_manager (object) :** Renvoie le noeud où le nom de la classe est *SaveLoadFx* ou nulle si ce dernier n'est pas définit.
  » **Node object** : Quel noeud sera considéré pour éffectuer les différentes opérations ?

+ **static Variant get_audio_manager (object) :** Renvoie le noeud où le nom de la classe est *AudioControllerFx* ou nulle si ce dernier n'est pas définit.
  » **Node object** : Quel noeud sera considéré pour éffectuer les différentes opérations ?

+ **static Variant get_scenes_manager (object) :** Renvoie le noeud où le nom de la classe est *ScenesFx* ou nulle si ce dernier n'est pas définit.
  » **Node object** : Quel noeud sera considéré pour éffectuer les différentes opérations ?

+ **static Variant get_languages_manager (object) :** Renvoie le noeud où le nom de la classe est *LanguagesFx* ou nulle si ce dernier n'est pas définit.
  » **Node object** : Quel noeud sera considéré pour éffectuer les différentes opérations ?

+ **static Variant get_settings_manager (object) :** Renvoie le noeud où le nom de la classe est *SettingsFx* ou nulle si ce dernier n'est pas définit.
  » **Node object** : Quel noeud sera considéré pour éffectuer les différentes opérations ?

+ **static Variant get_controllers_manager (object) :** Renvoie le noeud où le nom de la classe est *ControllersSensorFx* ou nulle si ce dernier n'est pas définit.
  » **Node object** : Quel noeud sera considéré pour éffectuer les différentes opérations ?

+ **static Variant get_cursor_manager (object) :** Renvoie le noeud où le nom de la classe est *CursorFx* ou nulle si ce dernier n'est pas définit.
  » **Node object** : Quel noeud sera considéré pour éffectuer les différentes opérations ?

+ **static Variant get_video_recorder (object) :** Renvoie le noeud où le nom de la classe est *VideoRecorderFx* ou nulle si ce dernier n'est pas définit.
  » **Node object** : Quel noeud sera considéré pour éffectuer les différentes opérations ?

+ **static Variant get_audio_recorder (object) :** Renvoie le noeud où le nom de la classe est *AudioRecorderFx* ou nulle si ce dernier n'est pas définit.
  » **Node object** : Quel noeud sera considéré pour éffectuer les différentes opérations ?

+ **static Variant get_multiplayer_manager (object) :** Renvoie le noeud où le nom de la classe est *MultiplayerFx* ou nulle si ce dernier n'est pas définit.
  » **Node object** : Quel noeud sera considéré pour éffectuer les différentes opérations ?

+ **static bool index_validation (index, size) :** Détermine si un index de position est en dehors des limites d'un tableau ou d'un dictionaire donné.
  » **int index** : Contient la valeur d'un index.
  » **int size** : Contient la taille total des éléments d'un tableau ou d'un dictionaire.

+ **static bool index_validation_msg (index, size, message = "", type = 0) :** Détermine si un index de position est en dehors des limites d'un tableau ou d'un dictionaire donné. Cette méthode affiche le contenu du paramètre *message* en fonction de la valeur du paramètre *type* lorsque la valeur retournée est *False*.
  » **int index** : Contient la valeur d'un index.
  » **int size** : Contient la taille total des éléments d'un tableau ou d'un dictionaire.
  » **String message** : Contient le méssage à affiché.
  » **int type** : Contient le type du méssage qui sera affiché. Les valeurs possibles sont celles définient au sein de la méthode *output ()*.

+ **void instanciate (data, object, delay = 0.0) :** Charge et importe un objet externe (prefab) dans l'arbre d'une scène.
  » **Dictionary data** : Contient toutes les données à propos du future objet ciblé. Les clés prises en charge dans ce dictionaire sont :
    » **String path** : Contient le chemin du future objet à chargé. L'utilisation de cette clé est obligatoire.

» **String | NodePath parent** : Dans quel parent l'objet sera importé ? Si la valeur précisée est invalide ou s'il n'y a aucune valeur, celle se trouvant dans le paramètre *object* sera utiliser.
» **Vector2 | Vector3 position** : Contient la position du future objet à chargé.
» **Vector2 | Vector3 rotation** : Contient la rotation du future objet à chargé.
» **Vector2 | Vector3 scale** : Contient la taille du future objet à chargé.
» **int | float live = -1.0** : Quel est le délai avant la destruction de l'objet après importation ? Si cette clé n'est pas définit ou que vous donnez une valeur négative, l'objet ne se détruirera jamais après avoir été importé. Notez que valeur nulle inhibe l'instance de tout objet.
» **bool visible = true** : Voulez-vous rendre visible votre objet ?
» **bool background = true** : Contrôle le moyen utilisé pour charger un objet. A *true*, le chargement de l'objet s'éffectue en arrière plan sans bloqué le jeu.
» **bool open = true** : Voulez-vous importer automatiquement l'objet après chargement ?
» **bool duplicate = false** : Force la répétition d'un objet. Notez qu'un objet déjà chargé sur un parent ne peut être recharché. C'est pour cela pour avoir quand même le même objet plusieurs fois dans une scène sans avoir à éffectué un rechargement complet de ce dernier, il serais préférable de créer une copy avec une référence différente de l'original.
» **int zindex = -1** : Contrôle la position hiérarchique d'un objet après son chargement.
» **String name** : Souhaitez-vous donner un nom à votre objet après son chargement ?
» **String | PoolStringArray groups** : Souhaitez-vous ajouter de(s) nom(s) de groupe à votre objet après son chargement ?
» **bool global = false** : Désirez-vous affecter les transformations globals de l'objet à naître en fonction des transformations que vous avez données ?
» **Dictionary callback** : Contient des informations sur la méthode à exécutée au cours du chargement de l'objet en question. Ce dictionaire supporte les clés suivantes :
- **String | NodePath source** : Contient l'addresse de la méthode à ciblée. La présence de cette clé n'est pas obligatoire. Dans ce cas, on considéra que la méthode référée se trouve sur la référence donné dans le paramètre *object*.
- **String method** : Contient le nom de la méthode à exécutée. L'utilisation de cette clé est obligatoire.

Notez que la méthode à exécutée doit possédée quatres (04) paramètres à savoir :
» **String id** : Contiendra le nom du fichier ou du l'objet.
» **int progress** : Contiendra la progression actuelle de l'objet en cours de chargement.
» **Variant reference** : Contiendra la référence de l'objet lorsque son chargement aura été éffectué avec succès. Par défaut, vous aurez une valeur nulle.
» **Variant error** : Contiendra l'erreur déclenchée au cours du chargement de l'objet. Ce paramètre vous renvoyera un dictionaire contenant les clés : *message, code* et *type* ou nulle si aucune erreur ne s'est levée durant le chargement de l'objet.

» **Node object** : Quel noeud sera considéré pour éffectuer les différentes opérations ?
» **float delay** : Quel est le temps mort avant l'importation ?

Code : GDScript

```
# Display in real time the loading building progress on editor console.
func load_building (id: String, progress: int, reference, error) -> void:
    # No errors detected.
    if error == null:
        # The destroyed building is loading.
        if reference == null: print (id + " is loading: " + str (progress));
        # The destroyed building is loaded successfully.
        else: print (id + " is loaded successfully: " + str (reference));
    # An error has been thrown.
    else: printerr (error.message);

# Called on first game execution.
func _ready ():
    # Let's load a 3d destroyed building.
    MegaAssets.instanciate (Dictionary ({
        parent = '.',
        path = "res://prefabs/destroyed_building/destroyed_building_02.tscn",
        background = true,
        callback = {
            method = "load_building",
            source = '.'
        },
        live = "-1.0",
        visible = true,
        open = true,
        duplicate = false,
        global = false,
        position = [265, 255],
        rotation = "0",
        scale = "(1.0, 1.0, 1.0)",
        name = "MyAwesomeDestroyBuilding",
        groups = ["BigBuilding", "Destroy"],
        zindex = 0
    }), self, 3.0);
```

+ void instanciates **(data, object, async = false, direction = 0, repeat = 1, delay = 0.0) :** Charge et importe un ou plusieurs objet(s) externe(s) (prefab) dans l'arbre d'une scène.
  » Dictionary **data** : Contient toutes les données à propos des futures objet(s) ciblé(s). Les clés prises en charge dans ce dictionaire sont :
    » String **path** : Contient le chemin du future objet à chargé. L'utilisation de cette clé est obligatoire.
    » String | NodePath **parent** : Dans quel parent l'objet sera importé ? Si la valeur précisée est invalide ou s'il n'y a aucune valeur, celle se trouvant dans le paramètre *object* sera utiliser.
    » Vector2 | Vector3 **position** : Contient la position du future objet à chargé.
    » Vector2 | Vector3 **rotation** : Contient la rotation du future objet à chargé.
    » Vector2 | Vector3 **scale** : Contient la taille du future objet à chargé.
    » int | float **live = -1.0** : Quel est le délai avant la destruction de l'objet après importation ? Si cette clé n'est pas définit ou que vous donnez une valeur négative, l'objet ne se détruirera jamais après avoir été importé. Notez que valeur nulle inhibe l'instance de tout objet.
    » bool **visible = true** : Voulez-vous rendre visible votre objet ?
    » int | float **interval = 0.0** : Quel est l'intervalle de temps avant chaque chargement ?
    » int **count = 1** : Combien de fois l'objet sera charger et importer ? Notez que si vous donné une valeur négative, vous aurez une importation infinie.
    » bool **background = true** : Contrôle le moyen utilisé pour charger un objet. A *true*, le chargement de l'objet s'éffectue en arrière plan sans bloqué le jeu.
    » bool **open = true** : Voulez-vous importer automatiquement l'objet après chargement ?
    » bool **duplicate = false** : Force la répétition d'un objet. Notez qu'un objet déjà chargé sur un parent ne peut être recharché. C'est pour cela pour avoir quand même le même objet plusieurs fois dans une scène sans avoir à éffectué un rechargement complet de ce dernier, il serais préférable de créer une copy avec une référence différente de l'original.
    » int **zindex = -1** : Contrôle la position hiérarchique d'un objet après son chargement.
    » String **name** : Souhaitez-vous donner un nom à votre objet après son chargement ?
    » String | PoolStringArray **groups** : Souhaitez-vous ajouter de(s) nom(s) de groupe à votre objet après son chargement ?
    » bool **global = false** : Désirez-vous affecter les transformations globals de l'objet à naître en fonction des transformations que vous avez données ?
    » Dictionary **callback** : Contient des informations sur la méthode à exécutée au cours du chargement de l'objet en question. Ce dictionaire supporte les clés suivantes :
      • String | NodePath **source** : Contient l'addresse de la méthode à ciblée. La présence de cette clé n'est pas obligatoire. Dans ce cas, on considéra que la méthode référée se trouve sur la référence donné dans le paramètre *object*.
      • String **method** : Contient le nom de la méthode à exécutée. L'utilisation de cette clé est obligatoire.

      Notez que la méthode à exécutée doit possédée quatres (04) paramètres à savoir :
      » String **id** : Contiendra le nom du fichier ou du l'objet.
      » int **progress** : Contiendra la progression actuelle de l'objet en cours de chargement.
      » Variant **reference** : Contiendra la référence de l'objet lorsque son chargement aura été éffectué avec succès. Par défaut, vous aurez une valeur nulle.

» **Variant error** : Contiendra l'erreur déclenchée au cours du chargement de l'objet. Ce paramètre vous renvoyera un dictionaire contenant les clés : *message, code* et *type* ou nulle si aucune erreur ne s'est levée durant le chargement de l'objet.

» **Node object** : Quel noeud sera considéré pour éffectuer les différentes opérations ?

» **bool async** : Devons-nous charger de façon asynchrône le(s) différent(s) objet(s) donné(s) ?

» **int direction** : Le chargement des données suivra quelle direction ? Les valeurs possibles sont :

-> **MegaAssets.Direction.NORMAL ou 0** : Chargement normal des objets.

-> **MegaAssets.Direction.REVERSED ou 1** : Chargement renversé des objets.

-> **MegaAssets.Direction.RANDOM ou 2** : Chargement aléatoire des objets.

-> **MegaAssets.Direction.ALTERNATE ou 3** : Chargement alterné des objets.

» **int repeat** : Combien de fois les objets donnés seront chargés ? Une valeur négative entraînera un chargement infinit et une valeur nulle ne provoquera pas de chargement.

» **float delay** : Quel est le temps mort avant les importations ?

\+ **static bool is_a_number (character) :** Détermine si un caractère est un nombre.

» **String character** : Contient la valeur du caractère en question. Attention ! La taille total du paramètre *character* doit être égale à 1.

\+ **static bool is_full_numbers (string) :** Détermine si une chaîne de caractères est tous en nombre.

» **String string** : Contient la valeur de la chaîne en question.

\+ **static bool is_a_letter (character) :** Détermine si un caractère est une lettre.

» **String character** : Contient la valeur du caractère en question. Attention ! La taille total du paramètre *character* doit être égale à 1.

\+ **static bool is_full_letters (string) :** Détermine si une chaîne de caractères est tous en lettre.

» **String string** : Contient la valeur de la chaîne en question.

\+ **static bool is_upper_case (character) :** Détermine si un caractère est une lettre majuscule.

» **String character** : Contient la valeur du caractère en question. Attention ! La taille total du paramètre *character* doit être égale à 1.

\+ **static bool is_full_upper_case (string) :** Détermine si une chaîne de caractères est tous en majuscule.

» **String string** : Contient la valeur de la chaîne en question.

\+ **static bool is_lower_case (character) :** Détermine si un caractère est une lettre minuscule.

» **String character** : Contient la valeur du caractère en question. Attention ! La taille total du paramètre *character* doit être égale à 1.

+ **static bool is_full_lower_case (string) :** Détermine si une chaîne de caractères est tous en minuscule.
  » **String string :** Contient la valeur de la chaîne en question.

+ **static PoolIntArray | int get_numbers_from (string) :** Renvoie tous les caractères jugés comme étant des nombres à partir du paramètre *string*.
  » **String string :** Contient la valeur de la chaîne en question.

+ **static PoolStringArray | String get_letters_from (string) :** Renvoie tous les caractères jugés comme étant des lettres à partir du paramètre *string*.
  » **String string :** Contient la valeur de la chaîne en question.

+ **static PoolStringArray | String get_letters_case_from (string, is_lower) :** Renvoie tous les caractères jugés comme étant des lettres minuscules ou majuscules en fonction de *is_lower* à partir du paramètre *string*.
  » **String string :** Contient la valeur de la chaîne en question.
  » **bool is_lower :** Doit-on renvoyé des caractères minuscules ?

+ **static int | String get_hex_symbol (value, is_lower = false) :** Renvoie la valeur hexadécimal correspondant à *value*. La valeur qui sera entrée doit être dans l'intervalle [10 ; 15].
  » **int value :** Contient une valeur entière.
  » **bool is_lower :** Doit-on renvoyé un caractère minuscule ?

+ **static int | String get_int_from_hex (symbol) :** Renvoie la correspondance entière de *symbol*. La valeur qui sera entrée doit être dans l'intervalle [a/A ; f/F].
  » **String symbol :** Contient un symbol hexadécimal.

+ **static Variant any_to_bit (type, bit, security = true) :** Converti n'importe quel type de donnée à n'importe quelle base. Attention ! Les types prises en charge sont : *int, float*, *String*, *Vector2*, *Vector3*, *Array* et *Dictionary*.
  » **Variant type :** Contient une donnée quelconque.
  » **int bit :** Quelle est la base choisie ?
  » **bool security :** Voulez-vous mettre une sécurité au résultat final ?

+ **static Variant any_from_bit (coded_type, bit, security = true) :** Renvoie la valeur du type qui avait été codé à la base *bit* par la méthode *any_to_bit ()*. Cependant, si vous tenez à revoir votre donnée intacte, vous devez utilisez l'une des bases suivantes : 2 à 10 ou 16, tout en sachant que la donnée a été préalablement codée dans l'une des bases recommandées.
  » **String | Array | Dictionary coded_type :** Contient la forme codée de la valeur du type en question.
  » **int bit :** Quelle est la base qui avait été choisie pour le codage ?

» **bool security** : Est-ce que la valeur codée avait-elle été sécurisé ?

Code : GDScript

```
# Called on first game execution.
func _ready():
    # Let's code and decode some variables value.
    print(MegaAssets.any_from_bit(MegaAssets.any_to_bit(true, 8), 8));
    print(MegaAssets.any_from_bit(MegaAssets.any_to_bit(12655666, 3), 3));
    print(MegaAssets.any_from_bit(MegaAssets.any_to_bit(1265.5666, 5), 5));
    print(MegaAssets.any_from_bit(MegaAssets.any_to_bit(Vector2(5.0, -93.2), 16), 16));
    print(MegaAssets.any_from_bit(MegaAssets.any_to_bit(Vector3.DOWN, 9), 9));
    print(MegaAssets.any_from_bit(MegaAssets.any_to_bit("Good boxing !", 10), 10));
    print(MegaAssets.any_to_bit([
        "Rikoudo", 'h', Vector2(45, -96), true, 25.336, 99, false,
        ["edwdrf", true, 9, [Vector2.UP, [Vector3.BACK]]],
        {"a": "b", false: "z", 9: Vector2.AXIS_Y,
        "My dic": {4: -65.02, true: 42, "z": {"x": 22, "y": -66.0225,
        "e": ["awer", "v", [false, 0, -2.15454848]]}}}
    ], 16));
    print(MegaAssets.any_from_bit(MegaAssets.any_to_bit([
        "Rikoudo", 'h', Vector2(45, -96), true, 25.336, 99, false
    ], 6), 6));
```

+ **static String generate_key (max_size = 16) :** Génère de façon aléatoire une clé.
  » **int max_size** : Contient la taille maximale de la clé qui sera générée.

+ **static PoolStringArray generate_keys (key_count, max_size = 16) :** Génère de façon aléatoire des clés.
  » **int key_count** : Contient le nombre total de clés à générées.
  » **int max_size** : Contient la taille maximale des clés qui seront générées.

+ **static String bytes_to_text (bytes) :** Renvoie la correspondance textuelle des *bytes* donnés.
  » **PoolByteArray bytes** : Contient un tableau d'octets.

+ **static ImageTexture get_screen_shot (object, data = {}) :** Renvoie une capture d'écran grâce au viewport de l'application.
  » **Node object** : Quel noeud sera considérer pour éffectuer les différentes opérations ?
  » **Dictionary data** : Contient les différentes configurations à effectuées sur la capture, une fois générée. Ce dictionaire supporte les clés suivantes :

» **Vector2 size = Vector2 (-1, -1)** : Contient la taille de l'image. Si la valeur de l'un des composants de ce paramètre est négative, celle du viewport sera utiliser pour redimensionner l'image.

» **int quality = 2** : Contient la qualité de la résolution de l'image. Les valeurs possibles sont celles définient au sein de classe *Image* de Godot.

» **int skrink = 0** : Contrôle le rétrécissement de l'image.

» **int comp_mode = 1** : Quel mode de compression voulez-vous adopter pour compresser l'image ? Les valeurs possibles sont :
-> **MegaAssets.ImageCompression.NONE ou 0** : Aucune compression ne sera éffectuée.
-> **MegaAssets.ImageCompression.ETC ou 1** : Compression avec en mode *ETC*.
-> **MegaAssets.ImageCompression.ETC2 ou 2** : Compression avec en mode *ETC2*.
-> **MegaAssets.ImageCompression.S3TC ou 3** : Compression avec en mode *S3TC*.

» **int comp_source = 1** : A partir de quelle source de compression l'image sera compresser ? Les valeurs possibles sont celles définient au sein de la classe *Image* de Godot. **N'utilisez cette clé que si l'image doit être compressée**.

» **int format = 6** : Contient le format de l'image. **N'utilisez cette clé que si l'image doit être compressée**. Les valeurs possibles sont :
-> **MegaAssets.ImageFormat.RH ou 0** : Format *RH*.
-> **MegaAssets.ImageFormat.RF ou 1** : Format *RF*.
-> **MegaAssets.ImageFormat.RGH ou 2** : Format *RGH*.
-> **MegaAssets.ImageFormat.RGF ou 3** : Format *RGF*.
-> **MegaAssets.ImageFormat.RGBH ou 4** : Format *RGBH*.
-> **MegaAssets.ImageFormat.RGBF ou 5** : Format *RGBF*.
-> **MegaAssets.ImageFormat.RGBAH ou 6** : Format *RGBAH*.
-> **MegaAssets.ImageFormat.RGBAF ou 7** : Format *RGBAF*.
-> **MegaAssets.ImageFormat.RGBA4444 ou 8** : Format *RGBA4444*.
-> **MegaAssets.ImageFormat.RGBA5551 ou 9** : Format *RGBA55551*.
-> **MegaAssets.ImageFormat.RGBA9995 ou 10** : Format *RGBA9995*.

» **float ratio = 100.0** : Quel taux de compression voulez-vous appliquer à l'image ? **N'utilisez cette clé que si l'image doit être compressée**.

Code : GDScript

```
extends Sprite;
# Called on  first  game execution.
func _ready ():
```

```
        # Let's the current viewport screenshot.
        yield (self.get_tree (), "idle_frame"); yield (self.get_tree (), "idle_frame");
        # Let's change the current sprite texture.
        texture = MegaAssets.get_screen_shot (self);
```

+ **static void create_screen_shot (path, object, data = {}, delay = 0.0) :** Cré sur le disque dure, une capture de l'écran de l'application.
  » **String path :** Contient le chemin de destination de la génération de l'image. Notez que votre image doit être au format (.png).
  » **Node object :** Quel noeud sera considérer pour éffectuer les différentes opérations ?
  » **Dictionary data :** Contient les différentes configurations à effectuées sur la capture, une fois générée. Pour avoir plus d'informations, consulter la documentation sur la fonction *get_screen_shot ()*.
  » **float delay :** Quel est le temps mort avant la création de la capture ?

Code : GDScript

```
    # Called when the node enters the scene tree for the first time.
    func _ready ():
        # Let's create the game screenshot on the disc after 02 secondes.
        MegaAssets.create_screen_shot ("res://screen_shot01.png", self, {}, 2.0);
```

+ **static String hex_to_raw (string) :** Converti une chaîne brute en hexadécimale.
  » **String string :** Contient une chaîne.

+ **static String raw_to_hex (string) :** Converti une valeur hexadécimale en donnée brute.
  » **String string :** Contient une valeur hexadécimale.

Code : GDScript

```
    # Called when the node enters the scene tree for the first time.
    func _ready ():
        # Let's code a string.
        print (MegaAssets.hex_to_raw (MegaAssets.raw_to_hex ("William Hashford")));
            # Will return "William Hashford".
```

+ **static int rotr32 (n, r) :** Renvoie le *rotr32* de deux entiers.
  » **int n :** Contient une value entière.

» **int r** : Contient une value entière.

+ **static int rotl64 (n, r) :** Renvoie le *rotl64* de deux entiers.
» **int n** : Contient une value entière.
» **int r** : Contient une value entière.

+ **static String encrypt (input, key = ”, method = 2, schema = 0) :** Chiffre une donnée avec une certaine clé. Cette fonction supporte plusieurs méthodes de chiffrement.
» **Variant input** : Contient l'entrée à chiffrée. **Ne donnez pas une entrée ayant une taille très long si vous tenez à récupéré votre donnée sans perte au cours du processus de déchiffrement. Il très important que vous comprenez cela**.
» **String key** : Contient la clé à utilisée pour chiffrer l'entrée. Notez que la taille de cette clé doit être égale à l'une des valeurs suivantes : 16 ou 32.
» **int method** : Contient la méthode à utilisée pour le chiffrement. Les valeurs possibles sont :
-> **MegaAssets.EncryptionMethod.AES ou 0** : Chiffrement avec la méthode *AES*.
-> **MegaAssets.EncryptionMethod.ARCFOUR ou 1** : Chiffrement avec la méthode *ARCFOUR*.
-> **MegaAssets.EncryptionMethod.CHACHA ou 2** : Chiffrement avec la méthode *CHACHA*.
» **int schema** : Le chiffrement suivra quel schéma ? Les valeurs possibles sont :
-> **MegaAssets.EncryptionSchema.BASE64 ou 0** : Chiffrement avec le schéma *Base64*.
-> **MegaAssets.EncryptionSchema.HEXADECIMAL ou 1** : Chiffrement avec le schéma *Hexadecimal*.
-> **MegaAssets.EncryptionSchema.RAW ou 2** : Chiffrement avec le schéma *Raw*.

+ **static Variant decrypt (input, key = ”, method = 2, schema = 0) :** Déchiffre une donnée avec une certaine clé. Cette fonction supporte plusieurs méthodes de chiffrement. Assurez-vous que la donnée est été chiffrée au préalablement avec la fonction *encrypt ()*.
  » **String input** : Contient l'entrée déjà chiffrée.
  » **String key** : Contient la clé qui avait été utilisé pour chiffrer l'entrée en question avec la fonction *encrypt ()*. Notez que la taille de cette clé doit être égale à l'une des valeurs suivantes : 16 ou 32.
  » **int method** : Contient la méthode qui avait été utilisé pour chiffrer l'entrée en question avec la fonction *encrypt ()*. Les valeurs possibles sont :
    -> **MegaAssets.EncryptionMethod.AES ou 0** : Déchiffrement avec la méthode *AES*.
    -> **MegaAssets.EncryptionMethod.ARCFOUR ou 1** : Déchiffrement avec la méthode *ARCFOUR*.
    -> **MegaAssets.EncryptionMethod.CHACHA ou 2** : Déchiffrement avec la méthode *CHACHA*.
  » **int schema** : Contient le schéma qui avait été utilisé pour chiffrer l'entrée en question avec la fonction *encrypt ()*. Les valeurs possibles sont :
    -> **MegaAssets.EncryptionSchema.BASE64 ou 0** : Chiffrement avec le schéma *Base64*.
    -> **MegaAssets.EncryptionSchema.HEXADECIMAL ou 1** : Chiffrement avec le schéma *Hexadecimal*.
    -> **MegaAssets.EncryptionSchema.RAW ou 2** : Chiffrement avec le schéma *Raw*.

+ **static String hash_var (input, key = ”, method = 0, schema = 0) :** Hash une donnée avec une certaine clé. Cette fonction supporte plusieurs méthodes de hashage.
  » **Variant input** : Contient l'entrée à hashée.
  » **String key** : Contient la clé à utilisée pour hasher l'entrée. Notez que la taille de la clé ne doit dépassée 16 octets.
  » **int method** : Contient la méthode à utilisée pour le hashage. Les valeurs possibles sont :
    -> **MegaAssets.HashingMethod.SIP ou 0** : Hashage avec la méthode *SIP*.
    -> **MegaAssets.HashingMethod.SHA256 ou 1** : Hashage avec la méthode *SHA256*.
    -> **MegaAssets.HashingMethod.HASHMAC_SHA256 ou 2** : Hashage avec la méthode *HASHMAC_SHA256*.
    -> **MegaAssets.HashingMethod.GODOT_SHA256 ou 3** : Hashage avec la méthode *GODOT_SHA256*.
    -> **MegaAssets.HashingMethod.MD5 ou 4** : Hashage avec la méthode *MD5*.
    -> **MegaAssets.HashingMethod.AES ou 5** : Hashage avec la méthode *AES*.
    -> **MegaAssets.HashingMethod.ARCFOUR ou 6** : Hashage avec la méthode *ARCFOUR*.
    -> **MegaAssets.HashingMethod.CHACHA ou 7** : Hashage avec la méthode *CHACHA*.
  » **int schema** : Le hashage suivra quel schéma ? Les valeurs possibles sont :
    -> **MegaAssets.EncryptionSchema.BASE64 ou 0** : Hashage avec le schéma Base64.
    -> **MegaAssets.EncryptionSchema.HEXADECIMAL ou 1** : Hashage avec le schéma Hexadecimal.
    -> **MegaAssets.EncryptionSchema.RAW ou 2** : Hashage avec le schéma Raw.

+ **static Variant get_variant (string) :** Renvoie le type de valeur correspondant à son format texte. Attention ! Les types prises en charge sont : *int, float, Vector2, Vector3, String, bool, Array et Dictionary*. Notez que l'imbrication de dictionaire et de liste ont été prise en charge.
  » **String string** : Contient la valeur du type en question en chaîne de caractères.

Code : GDScript

```
# Called when the node enters the scene tree for the first time.
func _ready():
    print(MegaAssets.get_variant("[45, 96.14, true, false, (14.1266, 20.148)]") is Array);
        # Will return True.
    print(MegaAssets.get_variant("58.10") is float); # Will return True.
    print(MegaAssets.get_variant("10") is int); # Will return True.
    print(MegaAssets.get_variant("false") is bool); # Will return True.
    print(MegaAssets.get_variant("(48, 1, 0)") is Vector3); # Will return True.
    print(MegaAssets.get_variant("(−9, −2.1562)") is Vector2); # Will return True.
```

+ **static void serialize (data, path, obj, ky = '', mhd = 7, lv = 1, call = {}, chek = 0, delay = 0.0)** : Sérialise les données se trouvant dans *data* bloquées avec *key* et cré ensuite un fichier sur le disque pour contenir le résultat de la sérialisation. Au cours de la sérialisation les données, peuvent être sécurisées en fonction de la méthode de chiffrement et niveau choisi. Notez qu'au même moment, un checksum est généré par rapport au fichier contenant les données qui ont été sérialisées blockant ainsi toute modification extérieur.
  » **Dictionary data** : Contient toutes les données qui seront sauvegardées. **Ne donnez pas des clés et valeurs ayant une taille très long si vous tenez à récupéré vos données sans perte au cours du processus de désérialisation. Il très important que vous sachez cela**. Vos clés et valeurs sont d'abord converti ensemble en chaîne de caractères avant d'être chiffré.
  » **String path** : Où voulez-vous créer le fichier de sauvegarde ?
  » **Node obj** : Quel noeud sera considéré pour éffectuer les différentes opérations ?
  » **String ky** : Quel est le mot de passe des données ? Notez que la taille de cette clé doit être égale à l'une des valeurs suivantes : 16 ou 32.
  » **int mhd** : Quelle sera la méthode à utilisée pour sécuriser les données ? Les méthodes prises en charge sont :
  -> **MegaAssets.SecurityMethod.NONE ou 0** : Aucune sécurité ne sera appliquée au cours de la sauvegarde des données.
  -> **MegaAssets.SecurityMethod.AES ou 1** : Chiffrement avec la méthode *AES*.
  -> **MegaAssets.SecurityMethod.ARCFOUR ou 2** : Chiffrement avec la méthode *ARCFOUR*.
  -> **MegaAssets.SecurityMethod.CHACHA ou 3** : Chiffrement avec la méthode *CHACHA*.
  -> **MegaAssets.SecurityMethod.BINARY ou 4** : Codage en *Binaire*.
  -> **MegaAssets.SecurityMethod.HEXADECIMAL ou 5** : Codage en *Hexadécimal*.
  -> **MegaAssets.SecurityMethod.OCTAL ou 6** : Codage en *Octal*.
  -> **MegaAssets.SecurityMethod.GODOT ou 7** : Chiffrement avec la méthode *GODOT*.

» **int lv** : Quel niveau de sécurité voulez-vous utiliser ? Les niveaux de sécurité prises en charge sont :
-> **MegaAssets.SecurityLevel.SIMPLE ou 0** : Simple niveau de sécurité.
-> **MegaAssets.SecurityLevel.NORMAL ou 1** : Niveau de sécurité normal.
-> **MegaAssets.SecurityLevel.ADVANCED ou 2** : Niveau de sécurité avancé.

» **Dictionary call** : Contient des informations sur la méthode à exécutée au cours de la sauvegarde d'un fichier. Ce dictionaire supporte les clés suivantes :
- **String | NodePath source** : Contient l'addresse de la méthode à ciblée. La présence de cette clé n'est pas obligatoire. Dans ce cas, on considéra que la méthode référée se trouve sur celle se trouvant dans le paramètre *obj*.
- **String method** : Contient le nom de la méthode à exécutée. L'utilisation de cette clé est obligatoire.

Notez que la méthode à exécutée doit possédée trois (03) paramètres à savoir :
» **String path** : Contiendra le chemin pointant vers le fichier en cours de sauvegarde.
» **int progress** : Contiendra la progression actuelle de fichier en cours de sauvegarde.
» **Variant error** : Contiendra l'erreur déclenchée au cours de la sauvegarde des données. Ce paramètre vous renvoyera un dictionaire contenant les clés : *message, code* et *type* ou nulle si aucune erreur ne s'est levée durant la sauvegarde des données.

» **int chek** : Quelle méthode de chiffrement voulez-vous utiliser pour générer le checksum du fichier de sauvegarde ? Les méthodes disponibles sont :
-> **MegaAssets.Checksum.NONE ou 0** : Aucun checksum ne sera générer.
-> **MegaAssets.Checksum.MD5 ou 1** : Chiffrement avec la méthode MD5.
-> **MegaAssets.Checksum.SHA256 ou 2** : Chiffrement avec la méthode SHA256.

» **float delay** : Quel est le temps mort avant la sérialisation des données ?

Code : GDScript

```
# Save data debug progress.
func save_data_progress (fname: String, progress: int, error) -> void:
    # No errors detected.
    if error == null: print (("Saving Progress of [" + fname + "]: " + str (progress)));
    # Error detected.
    else: printerr (error.message);
# Called when the node enters the scene tree for the first time.
func _ready ():
    # These data will be save into a file .
    var data: Dictionary = {
        a = 15.256,
        b = 9.255,
        c = -456,
        f = false,
        name = MegaAssets.name_generation (5, 7, 3),
```

```
        position = {
            x = -55.1255,
            y = -814.34,
            z = 0.75,
            basis = {
                x = Vector3.UP,
                y = Vector3.DOWN,
                z = Vector3.LEFT,
                w = Vector3.RIGHT
            }
        }
    }
    # Let's serialize some data.
    MegaAssets.serialize(data, "res://save_game.dat", self, "abcdefghijk",
        MegaAssets.SecurityMethod.GODOT, MegaAssets.SecurityLevel.NORMAL, {
        method = "save_data_progress()"
    }, MegaAssets.Checksum.NONE);
```

+ static Dictionary deserialize (path, obj, key = '', mhd = 7, level = 1, call = {}, check = 0) :
Charge un fichier de sauvegarde généré avec la function *serialize()*.
» **String path :** Où se trouve le fichier à chargé ?
» **String key :** Quel était le mot de passe utilisé dans la sauvegarde des données ?
» **Node obj :** Quel noeud sera considéré pour éffectuer les différentes opérations ?
» **int mhd :** Quelle était la méthode utilisée pour sécuriser les données ? Les méthodes prises en charge sont :
-> **MegaAssets.SecurityMethod.NONE ou 0 :** Aucune sécurité ne sera appliquée au cours de la sauvegarde des données.
-> **MegaAssets.SecurityMethod.AES ou 1 :** Chiffrement avec la méthode *AES*.
-> **MegaAssets.SecurityMethod.ARCFOUR ou 2 :** Chiffrement avec la méthode *ARCFOUR*.
-> **MegaAssets.SecurityMethod.CHACHA ou 3 :** Chiffrement avec la méthode *CHACHA*.
-> **MegaAssets.SecurityMethod.BINARY ou 4 :** Codage en *Binaire*.
-> **MegaAssets.SecurityMethod.HEXADECIMAL ou 5 :** Codage en *Hexadécimal*.
-> **MegaAssets.SecurityMethod.OCTAL ou 6 :** Codage en *Octal*.
-> **MegaAssets.SecurityMethod.GODOT ou 7 :** Chiffrement avec la méthode *GODOT*.
» **int level :** Quel était le niveau de sécurité utilisé dans la sauvegarde des données ? Les niveaux de sécurité prises en charge sont :
-> **MegaAssets.SecurityLevel.SIMPLE ou 0 :** Simple niveau de sécurité.
-> **MegaAssets.SecurityLevel.NORMAL ou 1 :** Niveau de sécurité normal.
-> **MegaAssets.SecurityLevel.ADVANCED ou 2 :** Niveau de sécurité avancé.
» **Dictionary call :** Contient des informations sur la méthode à exécutée au cours du chargement d'un fichier. Ce dictionaire supporte les clés suivantes :

- **String | NodePath source** : Contient l'addresse de la méthode à ciblée. La présence de cette clé n'est pas obligatoire. Dans ce cas, on considéra que la méthode référée se trouve sur celle se trouvant dans le paramètre *obj*.
- **String method** : Contient le nom de la méthode à exécutée. L'utilisation de cette clé est obligatoire.

Notez que la méthode à exécutée doit possédée cinq (05) paramètres à savoir :

» **String path** : Contiendra le chemin pointant vers le fichier en cours de chargement.

» **bool is_loading** : Le fichier actuelle est-il réelement en cours de chargement ?

» **int progress** : Contiendra le nombre d'élément(s) déjà chargé(s) en mémoire.

» **Dictionary result** : Contiendra l'ensemble des données chargées à partir du fichier en question.

» **Variant error** : Contiendra l'erreur déclenchée au cours du chargement des données. Ce paramètre vous renvoyera un dictionaire contenant les clés : *message, code* et *type* ou nulle si aucune erreur ne s'est levée durant la sauvegarde des données.

» **int check** : Quelle était la méthode de chiffrement utilisée pour générer le checksum du fichier de sauvegarde ? Les méthodes disponibles sont :

-> **MegaAssets.Checksum.NONE ou 0** : Aucun checksum ne sera générer.

-> **MegaAssets.Checksum.MD5 ou 1** : Chiffrement avec la méthode MD5.

-> **MegaAssets.Checksum.SHA256 ou 2** : Chiffrement avec la méthode SHA256.

Code : GDScript

```
# Load data debug progress.
func load_data_progress (fname: String, is_loading: bool, progress, result, error) -> void:
    # No errors detected.
    if error == null:
        # Is it loading ?
        if is_loading: print (("Loading Progress of [" + fname + "]: " + str (progress)));
        # Otherwise.
        else:
            # Warn the user.
            print ("Loading data successfuly !");
            # Show how many element(s) is loaded in memory.
            print (str (progress) + " data have been loaded in computer memory.");
            # Show all loaded data.
            print ("Data: ", JSON.print (result, "\t"));
    # Error detected.
    else: printerr (error.message);
# Called when the node enters the scene tree for the first time.
func _ready ():
    # Let's deserialize the saved data previewsly.
    MegaAssets.deserialize ("res://save_game.dat", self, "abcdefghijk",
```

```
        MegaAssets.SecurityMethod.GODOT, MegaAssets.SecurityLevel.SIMPLE, {
        method = "load_data_progress ()"
    }, MegaAssets.Checksum.NONE);
```

+ **static Color generate_color (use_alpha = false) :** Génère de façon aléatoire une coleur en suivant la méthode RGBA.
  » **bool use_alpha :** La génération des couleurs doit-elle prendre en charge la transparence ?

+ **static Array get_nodes (nodes_paths, object, reversed = false) :** Renvoie un ou plusieurs noeud à partir de leur chemins d'accès.
  » **PoolStringArray nodes_paths :** Contient les chemins en chaîne de caractères des noeuds ciblés.
  » **Node object :** Quel noeud sera considéré pour éffectuer les différentes opérations ?
  » **bool reversed :** Voulez-vous inverser le sens du traitement à éffectué ?

+ **static void add_children (nodes, other, object, reversed = false, delay = 0.0, interval = 0.0) :** Ajoute un ou plusieurs noeud(s) à un autre noeud. Le noeud ajouté devient ainsi l'enfant de *other*. Notez que le noeud à ajouté ne doit pas avoir au par avant un parent.
  » **PoolStringArray | String | NodePath nodes :** Contient le(s) chemin(s) de(s) noeud(s) que l'on veut ajouté(s).
  » **String | NodePath other :** Contient le chemin du future parent du/des noeud(s) à ajouté(s).
  » **Node object :** Quel noeud sera considéré pour éffectuer les différentes opérations ?
  » **bool reversed :** Voulez-vous inverser le sens du traitement à éffectué ?
  » **float delay :** Quel est le temps mort avant l'ajout ?
  » **float interval :** Quel est le délai avant chaque ajout ?

+ **static PoolStringArray | String name_generation (smin = 3, smax = 7, count : int = 1) :** Génère de façon aléatoire un ou plusieurs nom(s). Notez que si la valeur du paramètre *count* est inférieure ou égale à zéro, vous aurez une valeur nulle.
  » **int smin :** Contient la taille minimale du/des nom(s) à généré(s).
  » **int smax :** Contient la taille maximale du/des nom(s) à généré(s).
  » **int count :** Combien de nom(s) voulez-vous généré(s) ?

+ **static void save_file_config (data, path, object, key = '', security = 0, call = {}, delay = 0.0) :** Sauvegarde un dictionaire dans un fichier de configurations. Notez que le fichier de configurations doit obligatoirement être au format (.cfg).
  » **Dictionary data :** Contient toutes les données qui seront sauvegardées.
  » **String path :** Où voulez-vous créer le fichier de sauvegarde ?
  » **Node object :** Quel noeud sera considéré pour éffectuer les différentes opérations ?
  » **String key :** Quel est le mot de passe des données ?
  » **int security :** Quel niveau de sécurité voulez-vous utiliser ? Les niveaux de sécurité prises en charge sont :

-> **MegaAssets.SecurityLevel.SIMPLE ou 0** : Simple niveau de sécurité.
-> **MegaAssets.SecurityLevel.NORMAL ou 1** : Niveau de sécurité normal.
-> **MegaAssets.SecurityLevel.ADVANCED ou 2** : Niveau de sécurité avancé.

» **Dictionary call** : Contient des informations sur la méthode à exécutée au cours de la sauvegarde d'un fichier. Ce dictionaire supporte les clés suivantes :

- **String | NodePath source** : Contient l'addresse de la méthode à ciblée. La présence de cette clé n'est pas obligatoire. Dans ce cas, on considéra que la méthode référée se trouve sur celle se trouvant dans le paramètre *object*.
- **String method** : Contient le nom de la méthode à exécutée. L'utilisation de cette clé est obligatoire.

Notez que la méthode à exécutée doit possédée trois (03) paramètres à savoir :

» **String path** : Contiendra le chemin pointant vers le fichier en cours de sauvegarde.
» **int progress** : Contiendra la progression actuelle de fichier en cours de sauvegarde.
» **Variant error** : Contiendra l'erreur déclenchée au cours de la sauvegarde des données. Ce paramètre vous renvoyera un dictionaire contenant les clés : *message, code* et *type* ou nulle si aucune erreur ne s'est levée durant la sauvegarde des données.

» **float delay** : Quel est le temps mort avant la sauvegarde des données ?

Code : GDScript

```
# Save game configs debug progress.
func save_configs_progress (fname: String, progress: int, error) -> void:
    # No errors detected.
    if error == null: print (("Saving Progress of [" + fname + "]: " + str (progress)));
    # Error detected.
    else: printerr (error.message);
# Called when the node enters the scene tree for the first time.
func _ready ():
    # These data will be save into a file .
    var data: Dictionary = {
        bases = {
            a = 15.256,
            b = 9.255,
            c = -456,
            f = false,
            name = MegaAssets.name_generation (5, 7, 3)
        },
        position = {
            x = -55.1255,
            y = -814.34,
            z = 0.75,
            basis = {
```

```
                x = Vector3.UP,
                y = Vector3.DOWN,
                z = Vector3.LEFT,
                w = Vector3.RIGHT
            }
        }
    }
    # Let's save a game configurations data.
    MegaAssets.save_config_file (data, "res://game_configs.cfg", self, "abcdefghijk",
        MegaAssets.SecurityLevel.NORMAL, {
        method = "save_configs_progress ()"
    });
```

+ static Dictionary load_file_config (path, object, key = '', security = 0, call = {}) : Charge un fichier de configurations. Notez que le fichier de configurations doit obligatoirement être au format (.cfg). Assurez-vous que le fichier à lire est été créé au préalablement avec la fonction *save_file_config ()*.

» **String path** : Où se trouve le fichier à chargé ?

» **Node object** : Quel noeud sera considéré pour éffectuer les différentes opérations ?

» **String key** : Quel était le mot de passe utilisé dans la sauvegarde des données ?

» **int security** : Quel était le niveau de sécurité utilisé dans la sauvegarde des données ? Les niveaux de sécurité prises en charge sont :

-> **MegaAssets.SecurityLevel.SIMPLE ou 0** : Simple niveau de sécurité.

-> **MegaAssets.SecurityLevel.NORMAL ou 1** : Niveau de sécurité normal.

-> **MegaAssets.SecurityLevel.ADVANCED ou 2** : Niveau de sécurité avancé.

» **Dictionary call** : Contient des informations sur la méthode à exécutée au cours du chargement d'un fichier. Ce dictionaire supporte les clés suivantes :

- **String | NodePath source** : Contient l'addresse de la méthode à ciblée. La présence de cette clé n'est pas obligatoire. Dans ce cas, on considéra que la méthode référée se trouve sur celle se trouvant dans le paramètre *object*.
- **String method** : Contient le nom de la méthode à exécutée. L'utilisation de cette clé est obligatoire.

Notez que la méthode à exécutée doit possédée quatres (04) paramètres à savoir :

» **String path** : Contiendra le chemin pointant vers le fichier en cours de chargement.

» **int progress** : Contiendra la progression actuelle du chargement.

» **Dictionary result** : Contiendra l'ensemble des données chargées à partir du fichier en question.

» **Variant error** : Contiendra l'erreur déclenchée au cours du chargement des données. Ce paramètre vous renvoyera un dictionaire contenant les clés : *message, code* et *type* ou nulle si aucune erreur ne s'est levée durant la sauvegarde des données.

Code : GDScript

```
# Load game configs debug progress.
func load_configs_progress (fname: String, progress: int, result, error) -> void:
    # No errors detected.
    if error == null:
        # Show loading progress.
        print (("Loading Progress of [" + fname + "]: " + str (progress)));
        # The loading is it complete ?
        if progress == 100:
            # Warn the user.
            print ("Loading configs successfuly !");
            # Show all loaded data.
            print ("Data: ", JSON.print (result, "\t"));
    # Error detected.
    else: printerr (error.message);
# Called when the node enters the scene tree for the first time.
func _ready ():
    # Let's load a game configurations data.
    MegaAssets.load_config_file ("res://game_configs.cfg", self, "abcdefghijk",
        MegaAssets.SecurityLevel.NORMAL, {
        method = "load_configs_progress ()"
    });
```

+ static void create_folders_from (path, object, delay = 0.0) : Cré les dossiers néccessaires à l'accès d'un fichier.

» **String path** : Contient un chemin pointant vers un fichier quelconque.

» **Node object** : Quel noeud sera considéré pour éffectuer les différentes opérations ?

» **float delay** : Quel est le temps mort avant la création des dossiers ?

+ static Dictionary any_to_dic (input, reversed = false) : Converti une entrée donnée en dictionnaire.

» **String input** : Contient une valeur quelconque.

» **bool reversed** : Voulez-vous inverser le sens du traitement à éffectué ?

+ static Array any_to_array (input, dic_property = 1) : Converti une entrée donnée en tableau.

» **String input** : Contient une valeur quelconque.

» **int dic_property** : Quelle propriété du dictionaire sera convert en tableau. Les valeurs possibles sont :

-> **MegaAssets.DictionaryProperty.KEYS ou 0** : Cible les clés du dictionaire.

-> **MegaAssets.DictionaryProperty.VALUES ou 1** : Cible les valeurs des clés du dictionaire.

-> MegaAssets.DictionaryProperty.BOTH **ou** 2 : Cible les clés ainsi que leur valeur du dictionaire.

Ce paramètre est à utilisé uniquement lorsque l'entrée est un dictionaire.

+ static void raise_event **(data, scope, mode = 1, recursive = true, delay = 0.0) :** Déclenche un ou plusieurs événement(s) à la fois.

» Array **actions** : Tableau de dictionnaires gérant les configurations liées au différents événements que l'on veut prendre en charge. Le(s) dictionnaire(s) issu(s) de ce tableau, supportent les clés suivantes :

- String **event** : Contient le nom de l'événement à déclenché.
- float **delay** : Quel est le temps mort avant l'appel de l'événement en question.
- Array **params** : Contient les valeurs des différents paramètres de l'événement en question.

» Node **scope** : Contient le noeud à partir duquel le(s) événement(s) sera/ont déclenché(s).

» int **mode** : Contient la portée de l'événement en question. Les valeurs possibles sont :

-> MegaAssets.WornEvents.NONE **ou** 0 : Pas de porté.

-> MegaAssets.WornEvents.CHILDREN_ONLY **ou** 1 : Uniquement les enfants recevront l'événement(s).

-> MegaAssets.WornEvents.PARENTS_ONLY **ou** 2 : Uniquement le(s) parent(s) recevra/ont l'événement(s).

-> MegaAssets.WornEvents.ALL **ou** 2 : Le parent ainsi que les enfants recevront l'événement(s).

» bool **recursive** : La portée de(s) événement(s) sera t-elle récursive ?

» float **delay** : Quel est le temps mort avant le déclenchement ?

+ static String | PoolIntArray get_time_from **(seconds, divider = ' :', parse = false) :** Renvoie le temps correspondant au seconds données au format *HH :MM :SS*.

» int **seconds** : Contient le temps en seconds.

» String **divider** : Contient le séparateur à utilisé.

» bool **parse** : L'activation de ce paramètre obligera la fonction à renvoyé un tableau constitué de trois entiers, dont le premier élément contient l'heure, le second : la minute et le dernier : la seconde.

Code : GDScript

```
# Called when the node enters the scene tree for the first time.
func _ready ():
    # Let's get the real time of 7255 secondes.
    MegaAssets.get_time_from (7255); # Return 02h:00min:55s
```

+ **static String get_type (input) :** Renvoie le type d'un objet en chaîne de caractères.
  » **Variant input** : Contient la valeur d'un type quelconque.

Code : GDScript

```
# Called after game is ready.
func _ready():
    print(MegaAssets.get_type(125)); # Return "int"
    print(MegaAssets.get_type("Mother")); # Return "String"
    print(MegaAssets.get_type(false)); # Return "bool"
    print(MegaAssets.get_type(15.36)); # Return "float"
    print(MegaAssets.get_type(["false", 12, Vector2.UP])); # Return "Array"
    print(MegaAssets.get_type({name = "Mother"})); # Return "Dictionary"
    print(MegaAssets.get_type(null)); # Return "Nil"
```

+ **static bool is_array (input) :** Détermine si une entrée est un tableau.
  » **Variant input** : Contient la valeur d'un type quelconque.

Code : GDScript

```
# Called after game is ready.
func _ready():
    print(MegaAssets.is_array(125)); # Return false
    print(MegaAssets.is_arry(PoolStringArray(["s", "b"]))); # Return true
    print(MegaAssets.is_arry(PoolByteArray([45, 59, 93]))); # Return true
    print(MegaAssets.is_array(PoolIntArray([96, -74, 3]))); # Return true
    print(MegaAssets.is_array(["false", 12, Vector2.UP])); # Return true
    print(MegaAssets.is_array(PoolRealArray([74.36, 95.31]))); # Return true
    print(MegaAssets.is_array(null)); # Return false
```

+ **static Variant file_find (inp, path, key = '', case = false, count = 1, mhd = 5, level = 1) :** Renvoie le ou les lignes contenant la valeur à chercher dans un fichier.
  » **String | PoolStringArray inp** : Que voulez-vous chercher ?
  » **String path** : Où se trouve le fichier en question ?
  » **String key** : Le fichier à ouvrir est-il crypté ?
  » **bool case** : La recherche doit-elle s'éffectuée en ignorant la casse ?
  » **int count** : Combien de résultats seront renvoyés ? Notez qu'une valeur nulle donnera un résultat nulle et une valeur négative obligera la function à renvoyée tous les résultats trouvés au cours de la rechercher.

» **int mhd** : Quelle était la méthode utilisée pour sécuriser le fichier ? Les méthodes prises en charge sont :
-> **MegaAssets.SecurityMethod.NONE ou 0** : Aucune sécurité ne sera appliquée au cours de la sauvegarde des données.
-> **MegaAssets.SecurityMethod.AES ou 1** : Chiffrement avec la méthode *AES*.
-> **MegaAssets.SecurityMethod.ARCFOUR ou 2** : Chiffrement avec la méthode *ARCFOUR*.
-> **MegaAssets.SecurityMethod.CHACHA ou 3** : Chiffrement avec la méthode *CHACHA*.
-> **MegaAssets.SecurityMethod.BINARY ou 4** : Codage en *Binaire*.
-> **MegaAssets.SecurityMethod.HEXADECIMAL ou 5** : Codage en *Hexadécimal*.
-> **MegaAssets.SecurityMethod.OCTAL ou 6** : Codage en *Octal*.
-> **MegaAssets.SecurityMethod.GODOT ou 7** : Chiffrement avec la méthode *GODOT*.
» **int level** : Quel était le niveau de sécurité utilisé pour sécuriser le fichier ? Les niveaux de sécurité prises en charge sont :
-> **MegaAssets.SecurityLevel.SIMPLE ou 0** : Simple niveau de sécurité.
-> **MegaAssets.SecurityLevel.NORMAL ou 1** : Niveau de sécurité normal.
-> **MegaAssets.SecurityLevel.ADVANCED ou 2** : Niveau de sécurité avancé.

+ **static int | PoolIntArray get_index_of (datum, input, reversed = false, recursive = true)** :
Renvoie l'index d'un élément contenu une liste ou un dictionaire.
» **Variant datum** : Contient l'élément dont-on veut récupéré l'index de position.
» **Array | Dictionary | String data** : Contient le conteneur de l'élément ciblé.
» **bool reversed** : Voulez-vous inverser le sens du traitement à éffectué ?
» **bool recursive** : Trouver la position de l'élément en utilisant un programme récursive ?

Code : GDScript

```
# Called after game is ready.
func _ready():
    print(MegaAssets.get_index_of(5, 125)); # Return -1
    print(MegaAssets.get_index_of('h', "Mother")); # Return 3
    print(MegaAssets.get_index_of(12, ["false", 12, Vector2.UP])); # Return 1
    print(MegaAssets.get_index_of("position", {name = "Mother",
        position = Vector3.DOWN})); # Return 1
```

+ **static Variant parse_date (format = "dd-mm-yy", date, to_letter = false, parse = false)** :
Parse une date. Si aucune date n'est renseignée dans le paramètre *date*, la date actuelle du sytème d'exploitation sera utilisée.
» **String format** : Quel format de date voulez-vous ? Notez que le format de votre date doit contenir les caractères : d : pour afficher le jour, m : pour afficher le mois et y : pour afficher l'année. Exemple : "dd-mm-yy", "yy/mm/dd", "mm :yy", "dd yy", etc...
» **Dictionary date** : Contient les données relatives à une date. Ce dictionaire supporte les clés suivant :

» **int day = 1** : Contient le jour.
» **int month = 1** : Contient le mois.
» **int year = 1996** : Contient l'année.
» **bool to_letter** : Voulez-vous avoir le nom du mois correspondant à son numéro ?
» **bool parse** : Obligera la function à retourné un dictionaire contenant en plus des clés de base, une clé de plus appelée : *bixestile*. Cette clé détermine si l'année référencée est bixestile ou non. Elle contient une valeur booléenne.

+ **static Variant run_slot (data, object)** : Exécute tous les actions exigées par le développeur à partir d'un dictionaire.
» **Dictionary data** : Contient tous les configurations relatives à un flux d'exécution. L'utilisation de ce paramètre est déjà décrite au niveau des bases du framework. Précisement le sujet portant sur l'utilisation de la propriété EventsBindings (la section des actions liées à un événement).
» **Node object** : Quel noeud sera considéré pour éffectuer les différentes opérations ?

Code : GDScript

```
var force = -500;
var property = "text";
func parser4 (data = 123): return data;
# Called when the node enters the scene tree for the first time.
func _ready ():
    MegaAssets.run_slot ({
        "action": "property",
        "value": {
            "source": self,
            "action": "parser4()",
            "params": ["?force"]
        }
    });
    print (property); # Will return -500
```

+ **static void run_slots (data, object, delay = 0.0)** : Exécute tous les actions exigées par le développeur à partir d'un tableau de dictionaire.
» **Array | Dictionary data** : Contient toutes les configurations relatives à un ou plusieurs flux d'exécution(s). L'utilisation de ce paramètre est déjà décrite au niveau des bases du framework. Précisement le sujet portant sur l'utilisation de la propriété EventsBindings (la section des actions liées à un événement).
» **Node object** : Quel noeud sera considéré pour éffectuer les différentes opérations ?
» **float delay** : Quel est le temps mort avant les exécutions ?

Code : GDScript

```
var force = -500;
var property = "text";
var dic = {"x": 96};
func parser (): print ("dddd");
func parser4 (data = 123): return data;
# Called when the node enters the scene tree for the first time.
func _ready ():
    MegaAssets.run_slots ([
        {"source": self, "action": "property", "value": "?force"},
        {"action": "parser()"},
        {"action": "dic", "value": {"source": "WorldEnvironment", "action":
            "is_dont_destroy()"}},
        {"action": "force", "value": "?parser4()", "params": ["?property"]}
    ]);
    print (property); # Will return -500
```

+ **static float get_filtered_joy_axis (axis)** : Renvoie la forme filtrée d'un axe d'une manette.
  » **float axis** : Contient la valeur d'une touche d'axe de la manette.

+ **static bool is_key_or_axis_pressed (key_event, device = 0)** : Détermine si une touche est appuyée ou nom. Cette méthode supporte également les touches d'axes.
  » **InputEvent key_event** : Contient une touche d'événement.
  » **int device** : Quelle manette écoutée ? N'utilisez ce paramètre que si le contrôleur est une manette.

+ **static | PoolStringArray get_connected_controllers_names ()** : Renvoie les noms de tous les contrôleurs connectés à l'ordinateur.

+ **static String get_input_key_translation (key_event, object, device = 0, path = "...")** : Renvoie le nom correspondant à la touche appuyée en fonction du contôleur détecté. Notez que cela ne marche pas à tout moment car cela dépend du contôleur utilisé. Plus précisement au niveau de la manette. En cas d'échec de traduction, une valeur universelle sera renvoyée. Cette valeur universelle n'est rien d'autre que le nom que donne Godot à ses touches.
  » **InputEvent key_event** : Contient une touche d'événement.
  » **Node object** : Quel noeud sera considéré pour éffectuer les différentes opérations ?
  » **int device** : Quelle manette écoutée ? N'utilisez ce paramètre que si le contrôleur est une manette.
  » **String path** : Contient le chemin pointant vers le fichier de configuration (.cfg) des manettes non reconnu par cette methode. Pour editer ce ficher, vous devez suivre la nomenclature

suivante :
{gamepads}
NomDeLaManette1 = CategoryDeLaManette1 (Playstation, Xbox, Nitendo, etc...)
NomDeLaManette2 = CategoryDeLaManette2 (Playstation, Xbox, Nitendo, etc...)
...
NomDeLaManetteN = CategoryDeLaManetteN (Playstation, Xbox, Nitendo, etc...)
Les alcolades autour de *gamepads* seront remplacés par des crochets. Comme vous pouvez le constaté, un chemin par défaut est donné. Notez que des catégories de manettes ont été déjà renseignés dans le fichier de configuration que cible cette méthode. La valeur par défaut de ce paramètre est : *res ://addons/mega_assets/ nodes/base/mega_assets/gamepads.cfg*.

+ **static bool is_button (key_event)** : Détermine si la touche appuyée est un bouton ou nom.
  » **InputEvent key_event** : Contient une touche d'événement.

+ **static bool is_axis (key_event)** : Détermine si la touche appuyée est un axe ou nom.
  » **InputEvent key_event** : Contient une touche d'événement.

+ **static String get_keycode_string (key_event, device = 0, mask = false)** : Renvoie le code correspondant à la touche appuyée sous format de chaîne de caractères. Notez que ce qui est renvoyé n'est pas le code ASCII réel de la touche.
  » **InputEvent key_event** : Contient une touche d'événement.
  » **int device** : Quelle manette écoutée ? N'utilisez ce paramètre que si le contrôleur est une manette.
  » **int mask** : Voulez-vous utiliser l'option de masquage pour renvoyer le code des touches de la souris ? N'utilisez ce paramètre que si le contrôleur est une souris.

+ **void apply_root_motion (anim, node, delta, gravity = 9.8, delay = 0.0)** : Déplace un noeud de type *KinematicBody* à partir des transformations du bone racine d'un squelette d'animation.
  » **Node | String | NodePath anim** : Contient l'instance d'un noeud de type *AnimationTree*.
  » **Node | String | NodePath node** : Contient l'instance d'un noeud de type *KinematicBody*.
  » **float delta** : Contient le temps écoulé depuis la dernière trame du jeu.
  » **float gravity** : Quelle sera la force gravitationnelle à appliquée sur l'objet affecté par les transformations du bone racine ?
  » **float delay** : Quel est le temps mort avant la mise à jour des transformations de l'objet ciblé ?

+ **static void apply_camera_effect (pvt, eft, obj, lyt = 0, fus = 0, delay = 0.0)** : Applique un éffet de caméra sur l'objet sélectionné.
  » **String | NodePath pvt** : Contient l'instance d'un noeud de type *Sprite* ou *TextureRect*.
  » **int eft** : Quel éffet voulez-vous appliqué ? Les valeurs possibles sont celles définient au sein du module *CameraEffectsFx* (Au niveau du champ *Effect*).
  » **Node obj** : Quel noeud sera considéré pour éffectuer les différentes opérations ?

» **int lyt** : Voulez-vous donner une disposition à votre éffet ? Les valeurs possibles sont celles définient au sein du module *CameraEffectsFx* (Au niveau du champ *Layout*).
» **int fus** : Shouhaitez-vous modifier de façon dynamique la taille de votre éffet ? Les valeurs possibles sont celles définient au sein du module *CameraEffectsFx* (Au niveau du champ *FullScreen*).
» **float delay** : Quel est le temps mort avant l'application de l'éffet en question ?

+ **void apply_occlusion_culling (cam, nodes, delta, acy = 1000.0, scans = 10.0, delay = 0.0)** : Utilise une caméra pour déterminer si les objets prises en charge vont s'afficher ou non. Cette méthode est très utile pour l'optimisation de la mémoire graphique. Notez que le(s) objet(s) que vous choisirez doivent des instances des types suivants : *KinematicBody*, *RigidBody*, *Area*, *StaticBody*, *SoftBody*.
» **Node | String | NodePath cam** : Contient l'instance d'un noeud de type *Camera*.
» **Array nodes** : Contient les noeuds sous l'emprise de l'occlusion.
» **float delta** : Contient le temps écoulé depuis la dernière trame du jeu.
» **float acy** : Contient la précision de l'occlusion.
» **float scans** : Contient la fréquence du scan à éffectuée par seconde par l'occlusion.
» **float delay** : Quel est le temps mort avant le démarrage de l'occlusion des noeuds ?

+ **static void apply_std_effect (pvt, eft, obj, geo = true, idx = 0, next = false, delay = 0.0)** : Applique un éffet standard à l'objet sélectioné.
» **String | NodePath pvt** : Contient l'instance d'un noeud de type *GeometryInstance*, *Node2D* ou *Control*.
» **int eft** : Quel éffet voulez-vous appliqué ? Les valeurs possibles sont celles définient au sein du module *StandardEffectsFx* (Au niveau du champ *Effect*).
» **Node obj** : Quel noeud sera considéré pour éffectuer les différentes opérations ?
» **bool geo** : Voulez-vous que l'éffet à appliqué soit mis sur la propriété *material_override* ? Notez que la désactivation de cette option ne peut qu'être fait que sur un noeud de type *MeshInstance*.
» **bool next** : Voulez-vous que l'éffet à appliqué soit mis sur la valeur de la propriété *NextPass* ? Le comportement de cette option est récursive.
» **int idx** : Contient l'index de position du matériel qui sera affecté par l'éffet. Cette propriété est à utilisée uniquement sur un noeud de type *MeshInstance*.
» **float delay** : Quel est le temps mort avant l'application de l'éffet en question ?

+ **static bool is_range (value, minimum, maximum)** : Détermine si une valeur appartient à l'intervalle précisé.
» **float value** : Contient une valeur.
» **float minimum** : Contient la valeur minimum de l'intervalle.
» **float maximum** : Contient la valeur maximum de l'intervalle.

Code : GDScript

```
    # Called after game is ready.
    func _ready ():
        print (MegaAssets.is_range (125, 25.36, 524.36)); # Return true
        print (MegaAssets.is_range (69, 14.019, 36.216)); # Return false
```

+ static Transform get_ik_look_at (skn, bme, tar, obj, dir = 2, oft = Vector3.ZERO, fze = 0) :
Renvoie la transformation de l'os à partir de la position de sa cible. Cette méthode utilise la fonction *look_at ()* pour suivre l'objet donné.

» **String | NodePath skn :** Contient l'instance d'un noeud de type *Skeleton*.

» **String bme :** Contient le nom de la section (bone) à ciblée.

» **String | NodePath tar :** Contient l'instance d'un noeud de type *Spatial*.

» **Node obj :** Quel noeud sera considéré pour éffectuer les différentes opérations ?

» **int dir :** Quel axe sera choisi pour pister la cible référée ? Les valeurs possibles sont :

-> **MegaAssets.Axis.X ou 1 :** L'axe des absisses.
-> **MegaAssets.Axis.Y ou 2 :** L'axe des ordonnés.
-> **MegaAssets.Axis.Z ou 3 :** L'axe des côtes.
-> **MegaAssets.Axis._X ou 4 :** L'opposé de l'axe des absisses.
-> **MegaAssets.Axis._Y ou 5 :** L'opposé de l'axe des ordonnés.
-> **MegaAssets.Axis._Z ou 6 :** L'opposé de l'axe des côtes.

» **Vector3 oft :** Contient le décalage de la rotation du bone. Utilisez cet paramètre pour ajuster l'angle de la section si la valeur par défaut ne répond pas à vos attentes.

» **int fze :** Quel axe bloqué au cours du pistage ? Les valeurs possibles sont :

-> **MegaAssets.Axis.NONE ou 0 :** Aucun blockage.
-> **MegaAssets.Axis.X ou 1 :** Blockage de l'axe des absisses.
-> **MegaAssets.Axis.Y ou 2 :** Blockage de l'axe des ordonnés.
-> **MegaAssets.Axis.Z ou 3 :** Bockage de l'axe des côtes.
-> **MegaAssets.Axis._X ou 4 :** Blockage de l'opposé de l'axe des absisses.
-> **MegaAssets.Axis._Y ou 5 :** Blockage de l'opposé de l'axe des ordonnés.
-> **MegaAssets.Axis._Z ou 6 :** Blockage de l'opposé de l'axe des côtes.
-> **MegaAssets.Axis.XY ou 7 :** Bockage des axes x et y.
-> **MegaAssets.Axis.XZ ou 8 :** Bockage des axes x et z.
-> **MegaAssets.Axis.YZ ou 9 :** Bockage des axes y et z.
-> **MegaAssets.Axis._XY ou 10 :** Bockage de l'opposé des axes x et y.
-> **MegaAssets.Axis._XZ ou 11 :** Bockage de l'opposé des axes x et z.
-> **MegaAssets.Axis._YZ ou 12 :** Bockage de l'opposé des axes y et z.
-> **MegaAssets.Axis.XYZ ou 13 :** Bockage des axes x, y et z.
-> **MegaAssets.Axis._XYZ ou 14 :** Bockage de l'opposé des axes x, y et z.

Code : GDScript

```
var bone_name: String = "head";
# Called every frame. "delta" is the elapsed time since the previous frame.
func _process (_delta):
    var idx: int = $Armature/Skeleton.find_bone (bone_name);
    var rest: Transform = MegaAssets.get_ik_look_at (
        "Armature/Skeleton", bone_name, "target", self, MegaAssets.Axis.Y,
        Vector3 (0.0, -180.0, 0.0), MegaAssets.Axis.NONE
    );
    $Armature/Skeleton.set_bone_global_pose_override (idx, rest, 1.0, true);
```

+ **void bind_prop (data, delay = 0.0)** : Ajoute une propriété dans la liste des propriétés d'un script. N'appelez cette méthode que dans la fonction virtuelle : *_init ()*.
Avant de continuer, vous devez savoirs que les clés suivantes peuvent être manipulées dans l'utilisation des clés : *showif, disableif, require, changed, min, max, button, clone et notification*.
» **String | bool statement** : Contient la déclaration du développeur. Vous avez la possibilité de donné un booléen tout comme une chaîne de caractères. Si vous donnez une chaîne, alors elle devra remfermer une condition qui est exactement comme les déclarations que vous faites lorsque vous définissez une condition en programmation, mais à quelques différences près.
Exemple :
"**propriete1 > 56 || propriete2 != ?propriete3 and !propriete4**"
"**methode1() == not option**"
"**propriete5 <= ?methode2() and propriete4 or !methode1()**"
"**StringProperty == null**" => Vérifiera si la chaîne *StringProperty* est vide ("").
Le point d'interrogation signifie que l'on veut aller chercher la valeur que renvoie une autre propriété ou méthode prédéfinit. Les expressions mathématiques et les appels de méthodes paramétrable ne sont pas prises en charge dans une déclaration. Toute fois, si vous voulez recupérer la valeur d'une clé ou d'un index de position contenu dans un dictionaire ou tableau vous devez adopter la nomenclature : *NomDeLaPropriete.LeNomDeLaCle/IndexDePosition*.
Exemple : "**tableau1.2 > ?tableau2.0 || dictionaire1.2 != ?dictionaire3.level**"
Cependant, n'utilisé cette clé que si vous manipulez les clés suivantes : *showif, disableif, clone, require et notification*.

» **Array | Dictionary actions** : Contient les différentes configurations sur la manière dont les actions seront exécutées à la vérification de certaines conditions. Si vous donnez un tableau, alors celui-ci devra que contenir des dictionaires supportant les mêmes clés que si vous passez un dictionaire ordinaire. Si vous donnez un dictionaire, alors il supportera les clés suivantes :
• **String slot** : Contient le nom de la méthode ou propriété à ciblée lorsque les conditions d'exécution sont validées.

- **Variant value** : Cette clé, est à utilisée uniquement lorsque l'action à éffectuée est sur une propriété. Elle contient la valeur à affectée à la propriété en question, si vous ne passé pas un dictionaire. Dans le cas contraire, vous serez dans l'obligation de redéfinir la propriété avec toutes les clés que vous utilisez à la création d'une propriété via la méthode *bind_prop ()*.

- **Array params** : Contient les valeurs des différents paramètres de la méthode ciblée. Cette clé, est à utilisée uniquement lorsque l'action à éffectuée est sur une méthode. Le remplissage de ce tableau, doit respecté l'ordre d'alignement des paramètres de la méthode en question. Inutile d'utiliser cette clé, si la méthode n'a pas de(s) paramètre(s) ou peut prendre aucun argument(s). Notez que vous avez la possibilité de récupérer la valeur d'une autre propriété ou méthode grâce au mots clés spéciaux (*?NomDeLaPropriete ou ?NomDeLaMethode()*). Cette clé est uniquement présente sur les clés : *changed, actions* et *value*.

- **String message** : Contient le méssage à affiché lorsque les conditions d'exécution sont validées.

- **int type = 2** : Contient le type du méssage qui sera affiché. Les valeurs possibles sont celles définient au sein de la méthode *output ()*.

» **String callback** : Quel est le nom de la méthode à exécutée ? Cette clé est uniquement présente sur les clés : *changed* et *value*.

» **Array | Dictionary data** : Contient les configurations d'un ou de plusieurs propriétés d'un script. Si vous passez en paramètre une liste, celle-ci devra que contenir des dictionaires. Le(s) dictionaire(s) à utilisé(s) acceptera/ont les clés suivantes :

» **String source** : Contient le nom de la proprété à créée ou son chemin d'accès. L'utilisation de cette clé est obligatoire.

» **Variant value** : Contient la valeur de la propriété en question. Adaptez la valeur de votre propriété en fonction du type choisi. Il est important de le faire si vous ne voulez pas avoir des résultats bizzards. Si vous passez une chaîne de caractères comme valeur, vous aurez la possibilité de récupérer la valeur d'une autre propriété ou du résultat que renvoie une méthode.
Exemple :
?property affectera à la propriété ciblée, la valeur de property.
?method() affectera à la propriété ciblée, ce que retourne method.
Vous avez également la possibilité donner un dictionaire supportant les clés : *callback* et *params*.

» **String | PoolStringArray attach** : Voulez- vous attacher des propriétés à cette propriété ? Pour éffectuer un attachement, vous devez donner le(s) nom(s) des propriété(s) dont le comportement dépend de cette propriété.
Exemple : Soit propriete1, propriete2, propriete3 et propriete4 quatres proprétés du script. Les comportements des propriétés 2 et 3 dépendent de la valeur de propriete1 et celui de propriete4, celle de propriete3. Ainsi, la valeur de la clé *attach* de propriete1 ciblera les noms des propriétés 2 et 3 ; celui de propriete3 ciblera le nom de propriete4. Notez que l'utilisation de cette clé optimise l'exécution des configurations éffectuées au niveau toutes les propriétées du script. Il formellement conseilé de l'utiliser pour éviter les exécutions hors du domaine prédéfinit.

» **int type = 0** : Quel type de donnée sera contenu dans votre propriété ? Les valeurs possibles sont celles de Godot.

» **int hint = 0** : Quel est l'indice de la propriété ? Les valeurs possibles sont celles de Godot.

» **int usage = 71** : Quel est l'usage de la propriété dans l'éditeur ? Les valeurs possibles sont celles de Godot.

» **int stream = 2** : Par quel moyen on aura accès à la propriété ? Les valeurs possibles sont :
-> **MegaAssets.PropertyAccessMode.READ_ONLY ou 0** : On accèdera à la propriété qu'en lecture seule.
-> **MegaAssets.PropertyAccessMode.WRITE_ONLY ou 1** : On accèdera à la propriété qu'en écriture seule.
-> **MegaAssets.PropertyAccessMode.BOTH ou 2** : On accèdera à la propriété en lecture et écriture.

• **bool saveable = false** : Voulez-vous configurez la propriété afin qu'elle puisse répondre à la sauvegarde et au chargement des données du script ? N'utilisez cette propriété que si votre script hérite de la classe Saveable ou *Recordable*.

• **String title** : Voulez-vous changer la valeur du texte inscrite sur la catégory *Script Variables* ?

• **Vector2 | Vector3 range** : Mets des valeurs limit sur une propriété définie comme étant un réel ou entier ? La première valeur est celle minimale de la propriété. La seconde, celle maximale. Si vous utilisez une vecteur trois dimension, la troisième valeur représente le pas à ajouté lorsqu'on modifie la valeur de la propriété en question.

• **String hint_string** : Cette clé vous permet de grouper plusieurs valeurs suivant des cas spécifiques. Son comportement est la même que celle de Godot.

» **bool visible = true** : La propriété sera t-elle visible parmit celles du script ?

» **bool enabled = true** : La propriété sera t-elle activée parmit celles du script ?

» **bool duplicate = false** : Va t-on dupliquer la dernière valeur de l'élément de la propriété à chaque ajout ? Cette fonctionalité ne fonctionne que les listes.

» **int index = -1.0** : Contrôle la position qu'occupe une propriété au sein de la présentation des différentes variables du script.

» **bool private = false** : La propriété sera t-elle privée ? A ce stade, vous n'avez plus la possibilité de récupérer, ni même de modifier la valeur de cette dernière.

» **String | Dictionary button** : Cré une propriété booléenne ayant le comportement d'un bouton.

» **String | Dictionary changed** : Ecoute la valeur de la propriété et décide en cas de changement d'exécuter un callback donné par le dévoloppeur. Si vous donnez une chaîne de caractères, alors celle-ci doit référée le nom de votre callback (méthode à exécutée lorsque la valeur de la propriété change). Si votre méthode supporte deux ou plusieurs argument(s), la valeur du premier et celle du second seront respectivement remplacées par le nom de la propriété et sa valeur. N'oubliez pas ce détail.

» **Variant dropdown** : Souhaitez-vous transformer la propriété en une énumération ? Notez que si vous donnez un dictionaire, vous aurez la possibilité d'utiliser les clés suivantes :

- **Variant value** : Contient le(s) différente(s) valeur(s) de la liste déroulante.
- **int behavior** : Quel sera le comportement de la liste déroulante. Les valeurs possibles sont :
  -> **MegaAssets.NaughtyAttributes.INPUT_MAP ou 0** : Récupère la liste des entrées définient au sein de Godot.
  -> **MegaAssets.NaughtyAttributes.SIGNALS ou 1** : Récupère la liste des signaux définient au sein du script.
  -> **MegaAssets.NaughtyAttributes.TAGS ou 2** : Récupère la liste des groupes auquel appartient le noeud en question.
  -> **MegaAssets.NaughtyAttributes.METHODS ou 3** : Récupère la liste de toutes les méthodes appartenant au script.
  -> **MegaAssets.NaughtyAttributes.TYPES ou 4** : Récupère la liste des types de base de Godot.
  -> **MegaAssets.NaughtyAttributes.OPERATORS ou 5** : Récupère la liste des opérateurs de Godot.
  -> **MegaAssets.NaughtyAttributes.MOUSE_CONTROLS ou 6** : Récupère la liste de toutes les touches de la souris. Notez que vous pouvez également récupéré les codes des touches en mettant la clé *keycodes* sur true. C'est la même chose au niveau des touches de la manette et celles du clavier.

-> **MegaAssets.NaughtyAttributes.GAMEPAD_CONTROLS ou 7** : Récupère la liste de toutes les touches de la manette.
-> **MegaAssets.NaughtyAttributes.DESKTOP_RESOLUTIONS ou 8** : Récupère la liste des résolutions d'écran possibles des ordinateurs de bureau. Notez que vous pouvez également récupéré les résolutions en mettant la clé *sizes* sur true. C'est la même chose au niveau des résolutions de l'ipad, l'iphone et de l'android.
-> **MegaAssets.NaughtyAttributes.IPAD_RESOLUTIONS ou 9** : Récupère la liste des résolutions d'écran possibles des portables ipad.
-> **MegaAssets.NaughtyAttributes.IPHONE_RESOLUTIONS ou 10** : Récupère la liste des résolutions d'écran possibles des portables iphone.
-> **MegaAssets.NaughtyAttributes.ANDROID_RESOLUTIONS ou 11** : Récupère la liste des résolutions d'écran possibles des portables android.
-> **MegaAssets.NaughtyAttributes.KEYBOARD_CONTROLS ou 12** : Récupère la liste de toutes les touches du clavier.
-> **MegaAssets.NaughtyAttributes.SYSTEM_DIR ou 13** : Récupère la liste des chemins disponibles sur le système d'exploitation installé. Notez que vous pouvez également récupéré les chemins en mettant la clé *paths* sur true.
-> **MegaAssets.NaughtyAttributes.GAME_CONTROLLERS ou 14** : Récupère la liste des commandes connectées au jeu. Notez que cette option écouté les commandes connectées à l'ordinateur pour donner en temps réel les commandes disponibles. Notez que les comportements d'une liste déroulante sont prioritaire devant la valeur même de cette dernière.

» **String | Dictionary showif** : Détermine si une propriété sera visible dans les variables du script ou pas en fonction de la condition imposée par le développeur. IDEM pour la clé *disableif* qui s'occupe de l'activation ainsi que de la désactivation d'une propriété.

» **Array | Dictionary clone** : Détecte la présence de doublons en fonction de l'identifiant qui lui a été donné. Si vous donnez un tableau, alors celui-ci devra que contenir des dictionaires supportant les mêmes clés que si vous passez un dictionaire ordinaire. Si vous donnez un dictionaire, alors il supportera les clés suivantes :
- **Variant id** : Contient un identifiant dont-on souhaite vérifié des duplications au sein de ces valeurs.
- **int limit = 1** : Quel est le critère de répétition des valeurs de l'identifiant choisi ?

» **Dictionary | Vector2 | Vector3 | float | int min** : Force la valeur de la propriété à restée à celle minimale imposée. Notez que cette clé n'est utilisée que si le type de la propriété est un entier, réel, vecteur 2d ou vecteur 3d. Si vous donnez un dictionaire, alors il supportera les clés suivantes :
- **int | float | Vector2 | Vector3 value** : Contient la valeur minimale de la propriété.
- **int index = -1** : Quelle valeur voulez-vous ciblée ? N'utilisez cette clé que si la propriété que vous manipulez est un vecteur 2d ou vecteur 3d.

Même concepte pour clé *max*.

» **Dictionary | Array | String require** : Ajoute une certaine contrainte à la valeur d'une propriété. Si vous donnez une chaîne de caractères, elle sera considérée comme un méssage d'erreur à affiché tanqu'on ne changera pas la valeur initiale de la propriété. Si vous donnez un tableau, alors celui-ci devra que contenir des dictionaires supportant les mêmes clés que si vous passez un dictionaire ordinaire.

» **Dictionary | Array notification** : Ecoute les notifications de l'éditeur pour ensuite exécuter les configurations éffectuées à son égard. Si vous donnez un tableau, alors celui-ci devra que contenir des dictionaires supportant les mêmes clés que si vous passez un dictionaire ordinaire. Si vous donnez un dictionaire, alors il supportera les clés suivantes :
- **int what** : Quelle notification voulez-vous écouter ?

» **float delay** : Quel est le temps mort avant l'ajout de la propriété en question ?

Code : GDScript

```
# Slot1 method definition.
func slot1 (value): print ("Boolean: ", value);
# Slot2 method definition.
func slot2 (value): print (value);
# Slot3 method definition.
func slot3 (param): return param;
# Consts definition.
var CONST1 = false; var CONST2 = "OS dropdown";
# Called on class initialisation .
func _init ():
    # Change script title .
    self.bind_prop ({title = "ModuleType", index = 0});
    # Adding a boolean property.
    self.bind_prop ({
        source = "Features/Range", type = TYPE_REAL, value = 1, range = Vector2 (0, 5),
        attach = "range"
    });
    # Adding a boolean property.
    self.bind_prop ({
        source = "Features/Boolean", type = TYPE_BOOL, value = "?CONST1",
        changed = {callback = "slot1", params = null}, attach = "integer"
    });
    # Adding an integer property.
    self.bind_prop ({
        source = "Features/Integer", type = TYPE_INT, value = 0, attach = "integer",
```

```
        min = {value = 0, actions = {slot = "slot2 ()", params = "Min value of integer"}},
        max = {value = 10, actions = {slot = "slot2 ()", params = "Max value of integer"}},
        showif = {statement = "boolean"},
        notification = [{
            what = NOTIFICATION_ENTER_TREE,
            actions = {slot = "slot2 ()", params = "Start Game",
            message = "Enter tree", type = Message.NORMAL}
        }, {
            what = NOTIFICATION_EXIT_TREE,
            actions = {slot = "slot2 ()", params = "Stop Game",
            message = "Exit tree", type = Message.NORMAL}
        }]
});
# Adding an OS dropdown property.
self.bind_prop ({
        source = "Features/OS Dropdown",
        dropdown = ["Desktop", "Ipad", "Iphone", "Android"],
        value = 0, disableif = "not nodepath is Camera",
        attach = ["selecTED Os", "Resolutions dropdown"]
});
# Adding a screen resolutions dropdown.
self.bind_prop ({
        source = "Resolutions Dropdown", value = 0,
        dropdown = {behavior = NaughtyAttributes.DESKTOP_RESOLUTIONS},
        disableif = "not nodepath is Camera", attach = "resolutions dropdown",
        require = [{
            statement = "os dropdown == 0",
            actions = {
                slot = "resolutions dropdown",
                value = {
                    dropdown = {behavior = NaughtyAttributes.DESKTOP_RESOLUTIONS}
                }
            }
        }, {
            statement = "OS dropdown == 1",
            actions = {
                slot = "resolutions dropdown",
                value = {
                    dropdown = {behavior = NaughtyAttributes.IPAD_RESOLUTIONS}
                }
            }
        }, {
            statement = "oS DROPdown == 2",
```

```
            actions = {
                slot = "resolutions dropdown",
                value = {
                    dropdown = {behavior = NaughtyAttributes.IPHONE_RESOLUTIONS}
                }
            }
        }, {
            statement = "OS DROPDOWN == 3",
            actions = {
                slot = "resolutions dropdown",
                value = {
                    dropdown = {behavior = NaughtyAttributes.ANDROID_RESOLUTIONS}
                }
            }
        }]
});
# Adding a node path.
self.bind_prop ({
    source = "NodePath", index = 0, type = TYPE_NODE_PATH, value = NodePath (''),
    attach = ["NODEPATH", "OS DropDOWN", "RESOlutiONS drOPDown"],
    require = [{
        statement = "!nodepath is Camera",
        actions = {
            message = "You should donate an instance of camera.",
            type = Message.ERROR
        }
    }, {
        statement = "Nodepath is Camera",
        actions = {
            message = "NodePath property value is [OK]",
            type = Message.NORMAL
        }
    }]
});
# Adding an array.
self.bind_prop ({
    source = "Array", type = TYPE_ARRAY, value = Array ([]), duplicate = true,
    attach = "array"
});
# Adding a reset value button.
self.bind_prop ({
    source = "ResetValues",
    button = {
```

```
                actions = {slot = "reset_props_value ()", params = ["ResetVALUES", null]}
            }
        });
        # Adding a selected os name listener.
        self.bind_prop ({
            source = "Selected OS", type = TYPE_STRING, value = String ("No OS seclected"),
            attach = "selected os",
            require = {
                statement = "os dropdown >= 0",
                actions = {
                    slot = "sEleCtED Os",
                    value = {
                        value = {
                            callback = "get_prop",
                            params = [{callback = "slot3", params = "?CONST2"}, true]
                        }
                    }
                }
            }
        });
    # Calls on editor notification.
    func _notification (what): self.listen_notifications (what);
    # Gets all script variables list.
    func _get_property_list (): return self.get_properties ();
    # Calls to return property value changed.
    func _get (property): return self.get_prop (property);
    # Calls to set property editor value.
    func _set (property, value): self.set_prop (property, value, true);
```

+ **void listen_notifications (what, delay = 0.0)** : Ecoute les notifications de l'éditeur pour pouvoir ensuite déclenché les configurations éffectuées à propos de la clé *notification* sur les variables d'un script.
  » **int what** : Contient la notification de l'éditeur à écoutée.
  » **float delay** : Quel est le temps mort avant l'écoute de chaque notification de l'éditeur.

+ **static Variant get_id_value_of (id, input, index = -1, count = -1, rev = false, rec = true)** : Renvoie la ou les valeur(s) associée(s) à l'identifiant contenu dans un dictionaire ou tableau.
  » **Variant id** : Contient l'identifiant contenue dans le dictionaire ou tableau à utilisé.
  » **Variant input** : Contient un tableau ou dictionaire.
  » **int index** : Contient l'index de position de la valeur de l'identifiant référé.
  » **int count** : Combien de résultat aurons nous en sorti ? Une valeur négative entraînera le renvoie de tous les résultats trouvés au cours du traitement.

» **bool rev** : Souhaitez-vous inverser l'ordre des traitements à éffectués ?
» **bool rec** : Renvoyer le(s) valeur(s) de l'identifiant en utilisant un programme récursive ?

+ **void reset_props_value (trigger, prop = null, delay = 0.0)** : Rénitialise la valeur d'un ou de plusieurs propriétés.
» **String trigger** : Contient le nom de la propriété ayant déclenché cette méthode. C'est un peu bizzard pour vous, mais lorsqu'il s'agira de rénitialiser toutes les propriétés, celui-ci deviendra incontournable pour éviter que l'éditeur ne se ferme de lui même.
» **Variant prop** : Contient le(s) nom(s) de la ou des propriété(s) à rénitialisée(s). Si aucune propriété n'est référée, alors toutes les propriétés disponibles seront rénitialisée(s).
» **float delay** : Quel est le temps mort avant la rénitialisation de la ou des propriétés ciblée(s) ?

+ **Array get_properties (invert = false)** : Renvoie la liste de toutes les propriétés définient au sein du dictionaire *__props__*. Ce dictionaire est une propriété spéciale définie dans la classe **MegaAssets**. Ainsi, tous script héritant de cette classe possède cette dernière avec laquelle interagiront les méthodes *bind_prop ()*, *get_properties ()*, *reset_props_value ()*, *destroy_props ()*, *override_prop ()* et *listen_notification ()*.
» **bool invert** : Voulez-vous inverser l'ordre d'alignement des propriétés ?

+ **void destroy_props (source, delay = 0.0)** : Détruit un ou plusieurs propriété(s) associée(s) à un script.
» **String | PoolStringArray source** : Contient les noms de toutes les popriétés à détruire.
» **float delay** : Quel est le temps mort avant le(s) destructions ?

+ **void override_prop (data, prop_name, delay = 0.0)** : Redéfinit une propriété donnée.
» **Dictionary data** : Contient les nouvelles configurations relatives au propriété à redéfinir.
» **String prop_name** : Contient le nom de la propriété à redéfinir.
» **float delay** : Quel est le temps mort avant la redéfinition ?

+ **static String any_to_str (input)** : Converti une entrée en chaîne de caractères. Attention, le comportement de cette méthode est différente lorsque l'entrée fournit est un tableau.
» **Variant input** : Contient une valeur à convertir.

+ **static String | Array array_invert (array)** : Inverse l'ordre d'alignement des éléments contenu dans un tableau. Cette méthode supporte également une chaîne de caractères.
» **Variant array** : Contient un tableau à inversé.

+ **static Variant get_clones_of (id, input, limit = 1)** : Quelles sont les valeurs dont le nombre d'occurence est immédiatement supérieur à la limite imposée ?
» **Variant id** : Contient l'identifiant contenue dans le dictionaire ou tableau à utilisé.
» **Variant input** : Contient un tableau ou dictionaire.

» **int limit** : Quel est critère de répétition des valeurs ?

+ **static Variant set_id_value_of (id, value, input, index = -1, typed = false, rec = true)** : Modifie la valeur d'un identifiant dans un dictionaire ou tableau. L'influence de cette méthode est récursive à l'entrée donnée.
  » **Variant id** : Contient l'identifiant contenue dans le dictionaire ou tableau à utilisé.
  » **Variant value** : Contient la nouvelle valeur à affectée à l'identifiant en question.
  » **Variant input** : Contient un tableau ou dictionaire.
  » **int index** : Parmit les doublons de l'identifiant sélectionés lequel ciblé ?
  » **bool typed** : Devons nous changer la valeur de l'identifiant en se basant sur le type de sa valeur ?
  » **bool rec** : Modifier le(s) valeur(s) de l'identifiant en utilisant un traitement récursive ?

+ **static PoolStringArray | PoolVector2Array get_desktop_resolutions (to_string = true)** : Renvoie les résolutions possibles d'écran que peuvent avoir les ordinateurs.
  » **bool to_string** : Le résultat sera t-elle sous forme de chaine de caractères ?

+ **static PoolStringArray | PoolVector2Array get_ipad_resolutions (to_string = true)** : Renvoie les résolutions possibles d'écran que peuvent avoir les portables Ipad.
  » **bool to_string** : Le résultat sera t-elle sous forme de chaine de caractères ?

+ **static PoolStringArray | PoolVector2Array get_iphone_resolutions (to_string = true)** : Renvoie les résolutions possibles d'écran que peuvent avoir les portables Iphone.
  » **bool to_string** : Le résultat sera t-elle sous forme de chaine de caractères ?

+ **static PoolStringArray | PoolVector2Array get_android_resolutions (to_string = true)** : Renvoie les résolutions possibles d'écran que peuvent avoir les portables Android.
  » **bool to_string** : Le résultat sera t-elle sous forme de chaine de caractères ?

+ **static Variant get_initialised_type (input)** : Renvoie la version initialisée de la valeur donnée en paramètre en fonction de son type de contenu. Les types *Rid* et *Object* ne sont prises en charge dans cette méthode.
  » **Variant input** : Contient une valeur quelconque. Attention, le type de donnée référée doit être parmit les type de base.

Code : GDScript

```
# Called after game is ready.
func _ready():
    print(MegaAssets.get_initialised_type(125)); # Return 0
    print(MegaAssets.get_initialised_type("Mother")); # Return ""
```

```
print (MegaAssets.get_initialised_type (false)); # Return false
print (MegaAssets.get_initialised_type (15.36)); # Return 0.0
print (MegaAssets.get_initialised_type (["false", 12, Vector2.UP])); # Return []
print (MegaAssets.get_initialised_type ({name = "Mother"})); # Return {}
print (MegaAssets.get_initialised_type (Vector2 (0.36, -96.14))); # Return (0.0, 0.0)
```

+ **static PoolStringArray get_godot_base_types ()** : Renvoie la liste de toutes les types de bases prises en charge dans Godot.

+ **static PoolStringArray get_godot_operators ()** : Renvoie la liste de toutes les opérateurs de Godot.

+ **static PoolStringArray | PoolIntArray get_mouse_controls (keycode = false)** : Renvoie la liste de toutes les touches de la souris.
  » **bool keycode** : Désirez-vous recevoir uniquement les codes ascii des touches de la souris ?

+ **static PoolStringArray | PoolIntArray get_joystick_controls (keycode = false)** : Renvoie la liste de toutes les touches de la manette.
  » **bool keycode** : Désirez-vous recevoir uniquement les codes ascii des touches de la manette ?

+ **static PoolStringArray | PoolIntArray get_keyboard_controls (keycode = false)** : Renvoie la liste de toutes les touches du clavier.
  » **bool keycode** : Désirez-vous recevoir uniquement les codes ascii des touches du clavier ?

+ **static bool is_undefined (target, object)** : La ou les propriétés ou la ou les méthodes données sont-elles toutes définient dans l'instance de l'objet référée ?
  » **String | PoolStringArray target** : Quelles sont le(s) des méthodes ou des propriétés à cherchées ?
  » **Object object** : Contient l'instance d'un élément de type *Object*.

Code : GDScript

```
# Create some variables.
var player_life = 25;
var player_energy = 69;
# Create a method.
func parser (): print ("Hello world !");
# Called after game is ready.
func _ready ():
    print (MegaAssets.is_undefined ("player_life", self)); # Return false
```

```
print (MegaAssets.is_undefined ("life", self)); # Return true
print (MegaAssets.is_undefined ("parser", self)); # Return false
print (MegaAssets.is_undefined (["parser", "player_energy"], self)); # Return false
print (MegaAssets.is_undefined (["mana", "piece"], self)); # Return true
```

+ **static String get_object_prefix (object)** : Renvoie les informations de bases sur l'instance d'un objet.
  » **Object object** : Contient l'instance d'un élément de type *Object*.

+ **static String get_joy_category (jname, object, path = "...")** : Renvoie le nom de la catégory associé au nom de la manette référé.
  » **String jname** : Contient le nom de la manette en question.
  » **Node object** : Quel noeud sera considéré pour éffectuer les différentes opérations ?
  » **String path** : Contient le chemin pointant vers le fichier de configuration (.cfg) des manettes non reconnu par cette methode. Pour editer ce ficher, vous devez suivre la nomenclature suivante :
  {gamepads}
  NomDeLaManette1 = CategoryDeLaManette1 (Playstation, Xbox, Nitendo, etc...)
  NomDeLaManette2 = CategoryDeLaManette2 (Playstation, Xbox, Nitendo, etc...)
  ...
  NomDeLaManetteN = CategoryDeLaManetteN (Playstation, Xbox, Nitendo, etc...)
  Les alcolades autour de *gamepads* seront remplacés par des crochets. Comme vous pouvez le constaté, un chemin par défaut est donné. Notez que des catégories de manettes ont été déjà renseignés dans le fichier de configuration que cible cette méthode. La valeur par défaut de ce paramètre est : *res ://addons/mega_assets/ nodes/base/mega_assets/gamepads.cfg*.

+ **static Variant get_detected_controllers_names (limit = -1)** : Renvoie la liste des commandes détectées dans l'intervalle de détection imposé par le développeur.
  » **int limit** : Quelle est la limite de détection des commandes ? Une value nulle inhibe tout détection. Une valeur positive génère un intervalle de détection et une valeur négative rend ilimitées les détections.

+ **static Dictionary get_game_statistiques (object)** : Renvoie les caractéristiques générales liées au jeu.
  » **Node object** : Quel noeud sera considéré pour éffectuer les différentes opérations ?

+ **static Dictionary get_computer_statistiques ()** : Renvoie les caractéristiques générales liées à l'ordinateur ainsi qu'a son système d'exploitation.

+ **static void debug_data (visible, object, data = {}, options = {}, filter = null, delay = 0.0)** :
Affiche en temps réel les différentes données contenues dans le paramètre *data*. N'appelez cette méthode que si le jeu en cours d'exécution.
» **bool visible** : Devons-nous afficher les données issues du débogage ?
» **Node object** : Quel noeud sera considéré pour éffectuer les différentes opérations ?
» **Dictionary data** : Contient les différentes données à affichées pour le débogage.
» **Dictionary options** : Contient les différentes configurations pour la gestion de l'affichage des données. Ce dictionaire supporte les clés suivantes :
» **bool vspace = false** : Voulez-vous ajouter une space verticale sur l'ensemble des données matérialisées ?
» **int mrgtop = 0** : Ajoute une marge en haut de l'ensemble des données représentées.
» **int mrgleft = 0** : Ajoute une marge à gauche de l'ensemble des données représentées.
» **int mrgright = 0** : Ajoute une marge à droite de l'ensemble des données représentées.
» **int mrgbottom = 0** : Ajoute une marge en bas de l'ensemble des données représentées.
» **int charcount = -1** : Combien de caractères voulez-vous afficher sur l'ensemble des données affichées ?
» **bool uppercase = false** : Souhaitez-vous mettre en majuscule l'ensemble des données repésentées ?
» **Color color = Color.white** : Contrôle le couleur de toutes les données matérialisées par le débogage.
Notez que vous avez la possibilité de personnalisé les propriétés de chaque donnée repésentée. Il suffit simplement de préciser le nom de la clé correspondant à la donnée à personnalisée et de lui affectée un dictionaire supportant les clés : *mrgtop, mrgbottom, mrgleft, mrgright, color, charcount et uppercase*.
» **Array | String | PoolStringArray filter** : Contient le(s) clé(s) des éléments qui seront repésentés au cours du débogage.
» **float delay** : Quel est le temps mort avant la matérialisation des données ?

Code : GDScript

```
    # Called every frame. "delta" is the elapsed time since the previous frame.
    func _process (delta):
        # Show game statistiques.
        MegaAssets.debug_data (true, self, {
            fps = Engine.get_frames_per_second (),
            "Window Width": OS.window_size.x
        }, {
            mrgtop = 15, mrgleft = 15, color = Color.chartreuse,
            fps = {color = Color.red},
            "Window Width": {mrgtop = 2, mrgleft = 5}
        });
```

+ static void apply_game_settings (settings, object, nodes, delay = 0.0) : Modifie les configurations globales du jeu. N'appelez cette méthode que si le jeu en cours d'exécution.
  » Dictionary settings : Contient les différentes paramètres pour la gestion des configurations globales du jeu. Ce dictionaire supporte les clés suivantes :
    » float volume : Contrôle le volume des éffets sonores du jeu. Ces valeurs sont dans l'intervalle de [0.0 ; 100.0]. Notez qu'il faut que les éffets sonores doivent être permit avant de pouvoir varier leur volume.

    » int quality = 0 : Contrôle le degré de qualité du jeu. Les valeurs possibles sont :
      -> MegaAssets.GameQuality.LOW ou 0 : Mauvaise qualité.
      -> MegaAssets.GameQuality.MEDIUM ou 1 : Qualité moyen.
      -> MegaAssets.GameQuality.HIGH ou 2 : Bonne qualité.

    » int orientation = 0 : Voire la documentation de Godot sur la propriété *ScreenOrientation*.

    » bool borderless = false : Voulez-vous supprimer le cadre définissant la fenêtre du jeu ? En d'autres termes, souhaitez-vous supprimer la barre de titre ?

    » bool foreground = false : Voulez-vous mettre en premier plan la fenêtre du jeu ?

    » bool resizable = true : Voulez-vous permettre le redimentionnement de la fenêtre du jeu ?

    » Vector2 resolution = Vector2 (1024, 600) : Contrôle la taille de la fenêtre du jeu.

    » Vector2 minsize = Vector2 (0, 0) : Quelle est la résolution minimale de la fenêtre du jeu ? Si ne serais que l'une des valeurs de ce vecteur est négative ou nulle, on considera que la valeur minimale de cette dernière est nulle. Ne donnez pas une résolution suppérieur à celle de la fenêtre du jeu.

    » Vector2 maxsize = Vector2 (0, 0) : Quelle est la résolution maximale de la fenêtre du jeu ? Si ne serais que l'une des valeurs de ce vecteur est inférieur à la résolution actuelle de la fenêtre du jeu, on considera que la taille maximale est égale à celle actuelle au sein de la fenêtre du jeu. Ne donnez pas une résolution suppérieur à ce que peut supporté l'écran de votre machine.

    » Vector2 position = Vector2 (-1, -1) : Quelle est la position de la fenêtre du jeu ? Si ne serais que l'une des valeurs de ce vecteur est négative, cette dernière sera automatiquement centrée sur l'écran à la valeur ne respectant pas les conditions d'une résolution à la première exécution du jeu.

    » bool maximize = false : Voulez-vous maximiser la taille de la fenêtre du jeu ? Si cette fonctionnalité est activée, la fenêtre du jeu sera redimentionnée à la résolution maximale de l'écran.

» **bool fullscreen = false** : Contrôle le mode plein écran du jeu.

» **int brightness = 13** : Contrôle la luminosité de l'écran dans le jeu. Ces valeurs sont dans l'intervalle de [0 ; 100].

» **int contrast = 13** : Contrôle l'effet subjectif d'une apposition quantitative de couleurs. Ces valeurs sont dans l'intervalle de [0 ; 100].

» **int saturation = 13** : Contrôle la pureté de la couleur de l'écran dans le jeu. Ces valeurs sont dans l'intervalle de [0 ; 100].

» **bool vsync = true** : Voulez-vous activer la synchrônisation verticale au cours de l'exécution du jeu ?

» **bool vsyncompositor = false** : Voulez- vous activer la synchrônisation verticale par le biais d'un compositeur au cours de l'exécution du jeu ? Dans ce cas, le compositeur de fenêtre du système d'exploitation sera utilisé lorsque le jeu est uniquement en mode fenêtré. Notez que cette option n'est activée que sur un système d'exploitation de type Windows et est également expérimentale et destinée à atténuer le bégaiement ressenti par certains utilisateurs. Cependant, certains utilisateurs ont subi une réduction de moitié de la fréquence d'images (par exemple, de 60 FPS à 30 FPS) lors de leur utilisation.

» **bool keepscreen = true** : Désirez-vous garder l'écran actif lorsque le jeu est en cours d'exécution.

» **bool optimization = false** : Voulez-vous optimiser l'utilisation du processeur ? A ce niveau, le rafraîchissement de l'écran n'est fait que si cela est nécessaire afin de réduire la consommation de la batterie. Utile pour les appareils portables.

» **Node object** : Quel noeud sera considéré pour éffectuer les différentes opérations ?

» **Array | String | NodePath | PoolStringArray nodes** : Contient les différent(s) noeud(s) à ciblé(s) pour l'application de la valeur de certaines clés du dictionaire fournit. Cet paramètre supporte les noeuds de type *Camera* et *WorldEnvironment* pour mettre à jour la valeur des propriétés *Saturation*, *Contrast*, *Brightness* et *Mode* ; *AudioStreamPlayer*, *AudioStreamPlayer2D* et *AudioStreamPlayer3D* pour mettre à jour la valeur des propriétés *VolumeDB* et *MaxDB*.

» **float delay** : Quel est le temps mort avant l'application des configurations ?

\+ **static String get_os_dir (path)** : Renvoie le chemin réel pointant vers un dossier se trouvant sur le système d'exploitation.

» **int path** : Quel dossier souhaitez-vous ciblé ? Les valeurs possibles sont :

-> **MegaAssets.Path.GAME_LOCATION ou 0** : Cible le dossier racine du jeu.

-> **MegaAssets.Path.OS_ROOT ou 1** : Cible le dossier racine du système d'exploitation installé.

-> **MegaAssets.Path.USER_DATA ou 2** : Cible le dossier racine des données de l'utilisateur.
-> **MegaAssets.Path.USER_ROOT ou 3** : Cible le dossier racine de l'utilisateur.
-> **MegaAssets.Path.USER_DESKTOP ou 4** : Cible le bureau du système d'exploitation.
-> **MegaAssets.Path.USER_PICTURES ou 5** : Cible le dossier *Images* du système d'exploitation.
-> **MegaAssets.Path.USER_MUSIC ou 6** : Cible le dossier *Musiques* du système d'exploitation.
-> **MegaAssets.Path.USER_VIDEOS ou 7** : Cible le dossier *Vidéos* du système d'exploitation.
-> **MegaAssets.Path.USER_DOCUMENTS ou 8** : Cible le dossier *Documents* du système d'exploitation.
-> **MegaAssets.Path.USER_DOWNLOADS ou 9** : Cible le dossier *Téléchargements* du système d'exploitation.

+ **void pixels_adjustment ()** : Surveille les changements de taille de fenêtre et les poignées pour mettre à l'échelle de l'écran du jeu avec un nombre entier exact, des multiples d'une résolution de base en mémoire. Utile pour les jeu de pixélisation 2D.

+ **static void draw_line_3d (data, object, live = -1.0, delay = 0.0)** : Déssine une ligne en 3d à partir des différentes données fournit.
» **Dictionary data** : Contient les différentes configurations sur le trâcage et le comportement de la ligne. Le dictionaire supporte les clés suivantes :
» **NodePath | String parent** : A ligne à trâcée sera l'enfant de quel parent ? Si la valeur précisée est invalide ou s'il n'y a aucune valeur, la valeur se trouvant dans le paramètre *object* sera utiliser.

» **Variant points** : Contient les coordonnées des différents points sur qui définisseront la trajectoire que prendra la ligne à trâcée. Notez que vous avez la possibilité de donner un l'instance d'un noeud de type *RayCast*.

» **String id** : Contient le nom de la ligne à trâcée. Si l'identifiant n'a pas été précisé, "LineRenderer" est attribué par défaut.

» **int mesh = 4** : Quelle mesh utilisée pour trâcer la ligne ? Notez que les valeurs possibles sont celles de Godot.

» **int freeze = 0** : Quel axe bloqué au cours du déssin ? Les valeurs possibles sont :
-> **MegaAssets.Axis.NONE ou 0** : Aucun blockage.
-> **MegaAssets.Axis.X ou 1** : Blockage de l'axe des absisses.
-> **MegaAssets.Axis.Y ou 2** : Blockage de l'axe des ordonnés.
-> **MegaAssets.Axis.Z ou 3** : Bockage de l'axe des côtes.
-> **MegaAssets.Axis.XY ou 7** : Bockage des axes x et y.
-> **MegaAssets.Axis.XZ ou 8** : Bockage des axes x et z.
-> **MegaAssets.Axis.YZ ou 9** : Bockage des axes y et z.

-> **MegaAssets.Axis.XYZ ou 13** : Bockage des axes x, y et z.

» **Color | ShaderMaterial | SpatialMaterial skin** : Voulez-vous donner un teint à votre ligne ?

» **bool visible = true** : Contrôle la visibilité de la ligne.

» **float | int | Vector2 width = 0.004** : Quelle est l'épaisseur de la ligne ?

» **float | int | Vector2 smooth = 5.0** : Quel est l'arrondissement des coins et casquettes ?

» **bool skinscale = false** : Devons-nous augmenter l'échelle d'agrandissement du teint chargé actuellement sur la ligne ?

» **Dictionary | Array actions** : Contient le(s) action(s) à éffectuée(s) au cours du déssin de la ligne. L'utilisation de ce paramètre est déjà décrite au niveau des bases du framework. Précisement le sujet portant sur l'utilisation de la propriété EventsBindings (la section des actions liées à un événement).

» **Dictionary oncolliding** : Appelé lorsque le raycast référé détecte un objet portant un collisionneur. N'utilisez cette clé que si vous donnez au niveau de la clé *points*, un raycast.

» **NodePath | String | PoolStringArray impact** : Quels est/sont le(s) chemin(s) de(s) objet(s) à mettre à la position du point d'impact donné par un raycast ? Vous avez également la possibilité de donner de(s) chemin(s) pointant vers de(s) objet(s) préfabriqué(s). N'utilisez cette clé qu'a l'intérieur de la clé *oncolliding*.

» **bool destroy = true** : Devons-nous détruire le(s) objet(s) préfabriqué(s) importé(s) lorsqu'on ne détecte aucun objet ?

Notez que vous avez la possibilité d'utiliser l'un des identifiants d'un objet (le nom, le groupe et le type de classe appartenant à l'objet détecté par un raycast) pour changer le comportement de la ligne.

» **Node object** : Quel noeud sera considéré pour éffectuer les différentes opérations ?
» **float live** : Quel est le temps de vie de la ligne après sa création.
» **float delay** : Quel est le temps mort avant le trâcement de la ligne ?

Code : GDScript

```
# Called every frame. "delta" is the elapsed time since the previous frame.
func _process(delta):
    # Lets make a laser.
    MegaAssets.draw_line_3d(Dictionary({
```

```
        # If nothing is detected.
        points = "./RayCast",
        parent = '.',
        destroy = true,
        mesh = Mesh.PRIMITIVE_TRIANGLES,
        freeze = MegaAssets.Axis.NONE,
        skin = Color.green,
        id = "RedLaser",
        visible = true,
        actions = {
            source = '.',
            action = "is_running",
            value = false
        }
        # If you detect any physic body or area.
        oncolliding = {
            smooth = "10.0",
            skin = self.get_node ("../ImmediateGeometry").material_override,
            actions = {
                source = '.',
                action = "is_running",
                value = true
            },
            impact = "res://prefabs/cube.tscn",
            width = 0.05,
            # If you found a physic body or area named "StaticBody6".
            StaticBody6 = {
                width = 0.004,
                skin = Color.blue
            }
        },
    }), self);
```

+ static void debug_line_3d (show, id, pts, parent, color, fze = 0, mesh = 3, delay = 0.0) :
Déssine une ligne à partir des différents points donnés.
» **bool show** : Devons-nous afficher la line ainsi formée ?
» **String id** : Contient le nom de la future ligne à trâcée.
» **Variant pts** : Quels sont les différents points à matérialisés ?
» **Spatial parent** : Où devons-nous trâcer la ligne ?
» **Color color** : Contient la couleur de la ligne à trâcée. Par défaut, celle-ci est noire.
» **int fze** : Quel axe bloqué au cours du déssin ? Les valeurs possibles sont :
-> **MegaAssets.Axis.NONE ou 0** : Aucun blockage.
-> **MegaAssets.Axis.X ou 1** : Blockage de l'axe des absisses.

-> MegaAssets.Axis.Y **ou** 2 : Blockage de l'axe des ordonnés.
-> MegaAssets.Axis.Z **ou** 3 : Bockage de l'axe des côtes.
-> MegaAssets.Axis.XY **ou** 7 : Bockage des axes x et y.
-> MegaAssets.Axis.XZ **ou** 8 : Bockage des axes x et z.
-> MegaAssets.Axis.YZ **ou** 9 : Bockage des axes y et z.
-> MegaAssets.Axis.XYZ **ou** 13 : Bockage des axes x, y et z.

» **int mesh** : Quelle mesh utilisée pour trâcer la ligne ? Notez que les valeurs possibles sont :
-> Mesh.PRIMITIVE_LINES **ou** 1 : Rendre le tableau sous forme de lignes (tous les deux sommets, une ligne est créée).
-> Mesh.PRIMITIVE_LINE_STRIP **ou** 2 : Rendre le tableau sous forme de bande de ligne.
-> Mesh.PRIMITIVE_LINE_LOOP **ou** 3 : Rendre le tableau sous forme de boucle de ligne (comme une bande de ligne, mais fermée).

» **float delay** : Quel est le temps mort avant la matérialisation des points donnés ?

Code : GDScript

```
# Called every frame. "delta" is the elapsed time since the previous frame.
func _process(delta):
    # Let's debug an AI navigation path.
    MegaAssets.debug_line_3d(true, "NavigationPath",
        nav.get_simple_path(self.translation, target.translation),
        self.get_parent(), Color.cyan, MegaAssets.Axis.Y
    );
```

+ static void debug_ray_3d (show, id, start, end, parent, color, fze = 0, delay = 0.0) :
Déssine un rayon à partir des différents points donnés.

» **bool show** : Devons-nous afficher le rayon ainsi formé ?
» **String id** : Contient le nom du future rayon à trâcée.
» **Vector2 | Vector3 start** : Quelles sont les coordonnées du point de départ ?
» **Vector2 | Vector3 end** : Quelles sont les coordonnées du point d'arrivé ?
» **Spatial parent** : Où devons-nous trâcer le rayon ?
» **Color color** : Contient la couleur du rayon à trâcé. Par défaut, celle-ci est noire.
» **int fze** : Quel axe bloqué au cours du déssin ? Les valeurs possibles sont :
-> MegaAssets.Axis.NONE **ou** 0 : Aucun blockage.
-> MegaAssets.Axis.X **ou** 1 : Blockage de l'axe des absisses.
-> MegaAssets.Axis.Y **ou** 2 : Blockage de l'axe des ordonnés.
-> MegaAssets.Axis.Z **ou** 3 : Bockage de l'axe des côtes.
-> MegaAssets.Axis.XY **ou** 7 : Bockage des axes x et y.
-> MegaAssets.Axis.XZ **ou** 8 : Bockage des axes x et z.
-> MegaAssets.Axis.YZ **ou** 9 : Bockage des axes y et z.
-> MegaAssets.Axis.XYZ **ou** 13 : Bockage des axes x, y et z.

» **float delay** : Quel est le temps mort avant la matérialisation des points donnés ?

Code : GDScript

```
# Called every frame. "delta" is the elapsed time since the previous frame.
func _process(delta):
    # Let's debug a raycast path.
    MegaAssets.debug_ray_3d(true, "RayPath", Vector3.ZERO,
        self.get_node("RayCast").cast_to, self, Color.green, MegaAssets.Axis.NONE
    );
```

+ **static bool is_number (input) :** L'entrée précisée est- elle un nombre (entier ou réel) ?
  » **Variant input** : Contient une entrée.

Code : GDScript

```
# Called after game is ready.
func _ready():
    print(MegaAssets.is_number(5)); # Return true
    print(MegaAssets.is_number(-48.036)); # Return true
    print(MegaAssets.is_number("Kiss me !")); # Return false
    print(MegaAssets.is_number("03.896e-7")); # Return true
    print(MegaAssets.is_number("7")); # Return true
```

+ **static bool is_vector (input) :** L'entrée donnée est- elle un vecteur (2d ou 3d) ?
  » **Variant input** : Contient une entrée.

Code : GDScript

```
# Called after game is ready.
func _ready():
    print(MegaAssets.is_vector(Vector2.DOWN)); # Return true
    print(MegaAssets.is_vector(Vector3.UP)); # Return true
    print(MegaAssets.is_vector(56.01255)); # Return false
    print(MegaAssets.is_vector("(03.896e+44.25, -32.14)")); # Return true
    print(MegaAssets.is_vector("(7, -1, 0.25)")); # Return true
    print(MegaAssets.is_vector([2.25, -4.256])); # Return true
    print(MegaAssets.is_vector([-2.5, 9.42e-2, 0])); # Return true
    print(MegaAssets.is_vector([-7.75, 10.5, 0.0325, -1.014])); # Return false
```

```
    print (MegaAssets.is_vector ("[4.69, 8.512]")); # Return true
    print (MegaAssets.is_vector ("[1, 23.93, -0.289]")); # Return true
```

+ **static Variant any_to_vector2 (input)** : Converti une entrée donnée en un vecteur deux dimension. Notez que si l'entrée précisée ne respecte pas les exigences d'un vecteur 2d, vous aurez une valeur nulle.
  » **Variant input** : Contient une entrée.

Code : GDScript

```
    # Called after game is ready.
    func _ready ():
        print (MegaAssets.any_to_vector2 (Vector2.DOWN)); # Return (0, 1)
        print (MegaAssets.any_to_vector2 (Vector3.UP)); # Return (0, 1)
        print (MegaAssets.any_to_vector2 (56.01255)); # Return Null
        print (MegaAssets.any_to_vector2 ("(03.896e+44.25, -32)")); # Return (inf, -32)
        print (MegaAssets.any_to_vector2 ("(7, -1, 0.25)")); # Return (7, -1)
        print (MegaAssets.any_to_vector2 ([2.25, -4.256])); # Return (2.25, -4.256)
        print (MegaAssets.any_to_vector2 ([-2.5, 9.42e-2, 0])); # Return (-2.5, 0.0942)
        print (MegaAssets.any_to_vector2 ([-7.75, 10.5, 0.0325, -1.014])); # Return Null
        print (MegaAssets.any_to_vector2 ("[4.69, 8.512]")); # Return (4.69, 8.512)
        print (MegaAssets.any_to_vector2 ("[1, 23.93, -0.289]")); # Return (1, 23.93)
```

+ **static Variant any_to_vector3 (input)** : Converti une entrée donnée en un vecteur trois dimension. Notez que si l'entrée précisée ne respecte pas les exigences d'un vecteur 3d, vous aurez une valeur nulle.
  » **Variant input** : Contient une entrée.

Code : GDScript

```
    # Called after game is ready.
    func _ready ():
        print (MegaAssets.any_to_vector3 (Vector2.DOWN)); # Return (0, 1, 0)
        print (MegaAssets.any_to_vector3 (Vector3.UP)); # Return (0, 1, 0)
        print (MegaAssets.any_to_vector3 (56.01255)); # Return Null
        print (MegaAssets.any_to_vector3 ("(03.896e+44.25, -32)")); # Return (inf, -32, 0)
        print (MegaAssets.any_to_vector3 ("(7, -1, 0.25)")); # Return (7, -1, 0.25)
        print (MegaAssets.any_to_vector3 ([2.25, -4.256])); # Return (2.25, -4.256, 0)
        print (MegaAssets.any_to_vector3 ([-2.5, 9.42e-2, 0])); # Return (-2.5, 0.0942, 0)
        print (MegaAssets.any_to_vector3 ([-7.75, 10.5, 0.0325, -1.014])); # Return Null
```

```
    print (MegaAssets.any_to_vector3 ("[4.69, 8.512]")); # Return (4.69, 8.512, 0)
    print (MegaAssets.any_to_vector3 ("[1, 23.93, −0.289]")); # Return (1, 23.93, 0.289)
```

+ **static Variant split_vector3_path (points, count)** : Subdivise un chemin représenté par plusieurs points de l'espace en de sous petit points de l'espace dont chacun appartient à la trajectoire représentée par les points originaux.
  » **Array | PoolVector2Array | PoolVector3Array points** : Contient l'ensemble des points représentant un chemin quelconque.
  » **int count** : Les différents segments formés les points originaux seront subdivisés en combien de sous segment ?

Code : GDScript

```
# Called after game is ready.
func _ready ():
    print (MegaAssets.split_vector3_path ([Vector3.UP, Vector3.DOWN], 3));
        # Return [(0, 1, 0), (0, −0.333333, 0), (0, −1.666667, 0), (0, −1, 0)]
```

+ **static PoolByteArray rand_bytes (size)** : Génère un tableau d'octets en fonction d'une taille.
  » **int size** : Contient la taille du tableau à généré.

Code : GDScript

```
# Called after game is ready.
func _ready (): print (Array (MegaAssets.rand_bytes (4))); # Return [245, 235, 59, 232]
```

+ **static String str_replace (string, what, to, start = -1, end = -1)** : Remplace un ou plusieurs caractère(s) par un ou plusieurs autre(s) caractère(s).
  » **String string** : Contient une chaîne de caractères.
  » **String | PoolStringArray what** : Quel(s) est/sont le(s) caractère(s) qui sera/ront remplacé(s) ?
  » **String | PoolStringArray to** : Quel(s) est/sont le(s) caractère(s) qui prendera/ont la place de(s) caractère(s) désigné(s) ?
  » **int | String start** : A partir de quel caractère ou position les remplacements vont s'éffectués ?
  » **int | String end** : A quel caractère ou position les remplacements vont s'arrêtés ?

Code : GDScript

```
    # Called after game is ready.
    func _ready():
        # Multiple replacement test.
        print(MegaAssets.str_replace("Ooops ! Sorry", ['!', 'O', 'y'], ['?', 'o']));
            # Return "ooops ? Sorro"
```

+ **static String str_lrstrip (string, what_left = '', what_right = '')** : Supprime les caractères situés à gauche et à droite dans une chaîne de caractères.
  » **String string** : Contient une chaîne de caractères.
  » **String | PoolStringArray what_left** : Quel(s) est/sont le(s) caractère(s) à supprimé(s) à gauche de la chaîne de caractères donnée ?
  » **String | PoolStringArray what_right** : Quel(s) est/ sont le(s) caractère(s) à supprimé(s) à droite de la chaîne de caractères donnée ?

Code : GDScript

```
    # Called after game is ready.
    func _ready():
        # Mutliple removing test.
        print(MegaAssets.str_lrstrip("\n  Ooops ! Sorry  \t", ["\n", ' '], ["\t", ' ']));
            # Return "Ooop ! Sorry"
```

+ **static int get_real_image_format (option)** : Renvoie la position corresponde à l'option donnée au sein de l'énumération *Format* de la classe *Image* de Godot.
  » **int option** : Contient la position d'une option définit au sein de l'énumération MegaAssets.ImageFormat.

+ **static int get_real_image_compression (option)** : Renvoie la position corresponde à l'option donnée au sein de l'énumération *CompressMode* de la classe *Image* de Godot.
  » **int option** : Contient la position d'une option définit au sein de l'énumération MegaAssets.ImageCompression.

+ **static Array get_screen_shot_data (object, data = {}) :** Génère une capture de l'écran du jeu et renvoie ses données.
  » **Node object** : Quel noeud sera considérer pour éffectuer les différentes opérations ?
  » **Dictionary data** : Contient les différentes configurations à effectuées sur la capture, une fois générée. Pour avoir plus d'informations, consulter la documentation sur la fonction

get_screen_shot ().

+ **static Array load_csv_file (path, separator = ',') :** Charge le contenu d'un fichier d'extension (.csv).
  » **String path** : Contient le chemin pointant vers le fichier csv à chargé.
  » **String separator** : Quelle chaîne devons-nous utiliser pour distinguer les différentes données du fichier csv ?

+ **void open_doc (object, feature = '', delay = 0.0) :** Ouvre la documentation associée à la classe en question.
  » **Node object** : Quel noeud sera considéré pour éffectuer les différentes opérations ?
  » **String feature** : La documentation ciblera quelle fonctionnalité de la classe ?
  » **float delay** : Quel est le temps mort avant l'ouverture de la documentation ?