Table des matières

I – Définition

VideoRecorderFx est un module conçut pour la gestion des enregistrements des captures d'écran dans un jeu vidéo.
NB : Ce module est de nature indestructible, est compatible à un jeu 2D, 3D et est sauvegardable.

II – Les propriétés disponibles

+ int Mode = 0 : Contient les différents modes possibles que supporte ce module. Les valeurs possibles sont :
-> **VideoRecorderFx.Model.RECORDER ou 0 :** Mode diasporama.
-> **VideoRecorderFx.Model.READER ou 1 :** Mode lecture de séquence.

+ bool BufferOptimization = false : Souhaitez-vous supprimer les données de toutes les séquences précédements lues avant la lecture d'une nouvelle séquence ? N'utilisez cette propriété que si le module est utilisé en tant que lecteur de séquence(s).

+ bool Sync = false : La lecture des séquences s'éffectuera t-elle de façon synchrône ? N'utilisez cette propriété que si le module est utilisé comme un lecteur.

+ int Count = 1 : Combien de fois la liste des séquences sera lue ? N'utilisez cette propriété que si le module est utilisé comme un lecteur.

+ int Direction = 0 : Contient les différents sens possibles de lecture ou d'écriture que supporte ce module. Les valeurs possibles sont :
-> **MegaAssets.Orientation.NORMAL ou 0 :** Sens normal de lecture ou d'enregistrement des séquences.
-> **MegaAssets.Orientation.REVERSED ou 1 :** Sens inverse de lecture ou d'enregistrement des séquences.
-> **MegaAssets.Orientation.RANDOM ou 2 :** Choix aléatoire de lecture ou d'enregistrement.

+ int Action = 0 : Contient les différentes actions possibles qu'on peut éffectuées sur les séquences d'images. Les valeurs possibles sont :
-> **MegaAssets.MediaState.NONE ou 0 :** Aucune action ne sera éffectuée.
-> **MegaAssets.MediaState.PLAY ou 1 :** Joue toutes les séquences disponibles sur le module.
-> **MegaAssets.MediaState.PAUSE ou 2 :** Suspend toutes les séquences en cours de lecture.
-> **MegaAssets.MediaState.STOP ou 3 :** Stop toutes les séquences en cours de lecture.

+ Array Sequences : Tableau de dictionnaires contenant les différentes configurations de chaque séquence d'images prises en charge par le développeur. Les clés que supportent les dictionnaires sont :
» **int path = 0 :** Contient les différents chemins que prend en charge ce module. Ces chemins représentent les endroits possibles où l'on peut déposé le(s) séquence(s) d'images du jeu. 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.

» **String source** : Contient le chemin pointant vers un fichier ou un dossier se trouvant sur le disque dure. Evitez les répétitions, car cela n'est pas tolérées. Notez que si vous cryptez le(s) séquence(s), vous devez préciser le(s) fichier(s) qui contiendra/ont les différentes données enregistrez au cours des séances de capture d'écran. Dans le cas contraire, on considéra que la source référée pointe toujours vers un dossier. L'utilisation de cette clé est obligatoire.

» **int timeout = 0.0** : Quel est le temps mort avant le démarrage de la séquence en question ?

» **Vector2 resolution = Vector2 (-1, -1)** : Contrôle la taille des différentes images de la séquence en question.

» **int quality = 2** : Contrôle la qualité des différentes images de la séquence en question. Les valeurs possibles sont celles définient au sein de la classe *Image* de Godot.

» **bool encrypted = false** : Devons nous crypter la séquence ? N'utilisez cette option que si vous voulez éffectuer un enregistrement.

» **bool audio = false** : Devons nous enregister tous les sons à notre portés. Dans ce cas, la séquence sera enregistrer avec son propre fichier audio. Assurez-vous de la présence du module *AudioRecorderFx* avant d'activer cette option. Notez que si le module est utilisé comme un lecteur, cette option aura pour éffet d'activer et de désactiver tout simplement le son en cours de lecture.

» **int | float fps = 60** : Combien d'images seront enregistrées ou lue en une second ?

» **int repeat = 1** : Combien de fois la même séquence sera lue ? N'utilisez cette clé que si le module utilisé comme un lecteur.

» **float duration = 0.0** : Quelle est la durée de l'enregistrement ou de la lecture de la séquence ?

» **int save = 0** : Souhaitez-vous définir un mode de sauvegarde au cours de l'enregistrement d'une séquence ? N'utilisez cette propriété que si le module est employé comme un enregistreur. Les valeurs possibles sont :
-> **VideoRecorderFx.SaveMode.NONE ou 0** : Aucune sauvegarde ne sera éffectuer (ni en mémoire, ni dans un fichier).
-> **VideoRecorderFx.SaveMode.MEMORY ou 1** : La séquence sera sauvegarder en mémoire uniquement.
-> **VideoRecorderFx.SaveMode.FILE ou 2** : La séquence sera sauvegarder dans un fichier ou ensemble de fichiers.

» **int load = 0** : Souhaitez-vous définir un mode de sauvegarde au cours de l'enregistrement d'une séquence ? N'utilisez cette propriété que si le module est employé comme un lecteur. Les valeurs possibles sont :
-> **VideoRecorderFx.SaveMode.NONE ou 0** : Aucun chargement ne sera éffectuer (ni dans la mémoire, ni à partir d'un fichier).
-> **VideoRecorderFx.SaveMode.MEMORY ou 1** : La séquence sera charger en mémoire avant d'être lue.
-> **VideoRecorderFx.SaveMode.FILE ou 2** : La séquence sera directement lue à partir d'un fichier ou ensemble de fichiers.

» **Array | Dictionary started** : Signal déclenché à chaque fois qu'on démarre immédiatement la séquence. Cette clé exécute les différentes actions données à son déclenchement. Pour soumettre les actions à exécutées référez vous à la méthode utilisée au niveau de la clé *actions* de la propriété *EventsBindings* dans les bases du framework.

» **Array | Dictionary finished** : Signal déclenché à chaque fois que la séquence a terminée son exécution. Cette clé exécute les différentes actions données à son déclenchement. Pour soumettre les actions à exécutées référez vous à la méthode utilisée au niveau de la clé *actions* de la propriété *EventsBindings* dans les bases du framework.

» **Array | Dictionary playing** : Signal déclenché pendant que la séquence est en cours d'exécution. Cette clé exécute les différentes actions données à son déclenchement. Pour soumettre les actions à exécutées référez vous à la méthode utilisée au niveau de la clé *actions* de la propriété *EventsBindings* dans les bases du framework.

NB : Gardez à l'esprit que ce module se sert d'un curseur pour sélectionner les séquences définient par le développeur. Ce curseur n'est rien d'autre que l'index de position de chaque configuration de séquence. Par défaut, sa valeur est **0** lorsqu'il y a une ou plusieurs séquence(s) configurée(s) sur le module et **-1** si aucune séquence n'est disponible sur ce dernier.

III – Les méthodes disponibles

+ **ImageTexture get_current_frame (id = null) :** Renvoie l'image de la frame actuelle de la séquence en cours d'enregistrement ou de lecture. Si aucun identifiant n'a été référé, la valeur du curseur sera utilisée pour éffectuer le traitement demandé.
  » **int | String id :** Quel est l'identifiant de la séquence à ciblée ?

+ **int get_current_frame_index (id = null) :** Renvoie la position de la frame actuelle de la séquence en cours d'enregistrement ou de lecture. Si aucun identifiant n'a été référé, la valeur du curseur sera utiliser pour éffectuer le traitement demandé.
  » **int | String id :** Quel est l'identifiant de la séquence à ciblée ?

+ **int get_sequence_progress (id = null) :** Renvoie la progression actuelle d'une séquence (lecture ou enregistrement). Si aucun identifiant n'a été référé, celle de l'ensemble des séquences sera renvoyer. La valeur de la progression est dans l'intervalle [0 ; 100].
  » **int | String id :** Quel est l'identifiant de la séquence à ciblée ?

+ **int get_normalized_sequence (id = null) :** Renvoie la version normalisée de la progression actuelle d'une séquence (lecture ou enregistrement). Si aucun identifiant n'a été référé, celle de l'ensemble des séquences sera renvoyer. La valeur de la progression est dans l'intervalle [0.0 ; 1.0].
  » **int | String id :** Quel est l'identifiant de la séquence à ciblée ?

+ **void play (id = null, config = {}, interval = 0.0) :** Exécute les configurations d'une ou de plusieurs séquence(s). Si aucun identifiant n'a été référé, la valeur du curseur sera utiliser pour éffectuer le traitement demandé.
  » **int | PoolIntArray | String | PoolStringArray id :** Quel(s) est/sont le(s) identifiant(s) de(s) séquence(s) à enregistrée(s) ou à lire ?
  » **Dictionary | Array config :** Voulez-vous changer la valeur de certaines clés de(s) sequence(s) avant son/leur exécution ? Si vous donnez un tableau, alors il ne devra que contenir des dictionaires.
  » **float interval :** Quel est le temps mort avant l'exécution de chaque séquence ?

+ **void pause (id = null, interval = 0.0) :** Suspend l'exécution d'une ou de plusieurs séquence(s). Si aucun identifiant n'a été référé, la valeur du curseur sera utiliser pour éffectuer le traitement demandé.
  » **int | PoolIntArray | String | PoolStringArray id :** Quel(s) est/sont le(s) identifiant(s) de(s) séquence(s) à suspendre ?
  » **float interval :** Quel est le temps mort avant la suspension de chaque séquence ?

+ **void stop (id = null, interval = 0.0) :** Arrète l'exécution d'une ou de plusieurs séquence(s). Si aucun identifiant n'a été référé, la valeur du curseur sera utiliser pour cibler la séquence à arrètée.
  » **int | PoolIntArray | String | PoolStringArray id** : Quel(s) est/sont le(s) identifiant(s) de(s) séquence(s) à arrètée(s) ?
  » **float interval** : Quel est le temps mort avant l'arrèt de chaque séquence ?

+ **float get_eleapsed_time (id = null) :** Retourne le temps écoulé depuis l'enregistrement ou la lecture d'une séquence. Si aucun identifiant n'a été précisé, celui de l'ensemble des séquences sera renvoyer.
  » **int | String id** : Quel est l'identifiant de l'animation à ciblée ?

+ **int get_state (id = null) :** Renvoie l'état d'utilisation d'une séquence donnée. Si aucun identifiant n'a été précisé, le traitement sera éffectué sur l'ensemble des séquences du module. Les valeurs possibles de retour sont :
  -> **MegaAssets.MediaState.NONE ou 0** : Aucun traitement ou séquence arrètée.
  -> **MegaAssets.MediaState.PLAY ou 1** : Séquence en cours d'exécution.
  -> **MegaAssets.MediaState.PAUSE ou 2** : Séquence suspendu.
  -> **MegaAssets.MediaState.LOOP ou 4** : Séquence en exécution infinie.
  » **int | String id** : Quel est l'identifiant de la séquence à ciblée ?

+ **int get_current_repeatition (id = null) :** Renvoie le nombre actuelle de répétitions éffectuées auprès d'une séquence donnée. Si aucun identifiant n'a été renseigné, celui de l'ensemble des séquences sera renvoyer.
  » **int id** : Quel est l'identifiant de la séquence à ciblée ?

+ **int get_cursor () :** Renvoie la valeur actuelle du curseur du module.

+ **void set_cursor (new_value) :** Change la valeur actuelle du curseur du module.
  » **int new_value** : Quel est la nouvelle valeur du curseur ?

+ **int | PoolIntArray get_active_seq_index () :** Renvoie le(s) position(s) de la ou des séquence(s) en cours d'exécution.

+ **int get_prev_seq_index () :** Renvoie la position de la séquence précédement exécutée ou générée.

+ **int get_next_seq_index () :** Renvoie la position du future séquence à exécutée. N'appelée cette méthode que si la valeur du champ *Direction* est sur *RANDOM*.

+ **int get_big_seq_index () :** Renvoie la position de la séquence ayant la plus grande durée parmit celles définient par le développeur.

+ **int get_small_seq_index () :** Renvoie la position de la séquence ayant la plus petite durée parmit celles définient par le développeur.

+ **float get_total_duration () :** Renvoie le temps total des séquences disponibles sur le module (sans les délai).

+ **float get_total_timeout () :** Renvoie la somme des délai définient sur les séquences du module.

+ **float get_total_time () :** Renvoie le temps total des séquences disponibles sur le module (avec les délai).

+ **void save_sequences (id = null) :** Sauvegarde dans un fichier ou ensemble de fichiers, une ou plusieurs séquence(s) déjà enregistrée(s) en mémoire. N'utilisez cette méthode que si le module est utilisé comme en enregistreur.
  » **int | String | String | PoolStringArray id :** Quel est l'identifiant de(s) séquence(s) à ciblée(s) ?

+ **void load_sequences (id = null) :** Charge une ou plusieurs séquence(s) déjà enregistrée(s) sur le disque dure. N'utilisez cette méthode que si le module est utilisé comme en lecteur.
  » **int | String | String | PoolStringArray id :** Quel est l'identifiant de(s) séquence(s) à ciblée(s) ?

+ **Dictionary get_sequences_data (json = false) :** Renvoie toutes les données sur les séquences disponibles au sein du module.
  » **bool json :** Voulez-vous renvoyer les données au format json ?

IV – Les événements disponibles

+ **sequence_changed (data) :** Signal déclenché lorsque la séquence en cours d'enregistrement ou de lecture a changée. Notez que cet événement ne s'appel que lorsque le mode de lecture des séquences du module est synchrône. Il renvoie un dictionaire contenant les clés suivantes :
  » **int preview :** Contient la position de la séquence précédement exécutée.
  » **int current :** Contient l'index de position de la séquence actuellement en cours d'exécution.

+ **sequence_started (data) :** Signal déclenché à chaque fois qu'on démarre immédiatement une séquence. Cet événement renvoie un dictionaire contenant les clés suivantes :
  » **int index :** Contient la position de la séquence en question.

+ **sequence_finished (data) :** Signal déclenché à chaque fois qu'une séquence a terminée son exécution (enregistrement ou lecture). Cet événement renvoie un dictionaire contenant les clés suivantes :
  » **int index** : Contient la position de la séquence en question.

+ **sequence_running (data) :** Signal déclenché pendant qu'une séquence est en cours d'enregistrement ou de lecture. Cet événement renvoie un dictionaire contenant les clés suivantes :
  » **float time** : Contient le temps écoulé depuis l'exécution de la séquence.
  » **int progress** : Contient la progression actuelle de la séquence.
  » **float normalized** : Contient la progression normalisée de la séquence.
  » **int count** : Contient le nombre total de répétitions éffectuées depuis la première exécution. Cependant, si la séquence est exécutée à l'infinie, vous aurez une valeur négative.
  » **int index** : Contient la position de la séquence en question.
  » **ImageTexture frame** : Contient la frame actuelle de la. séquence.
  » **int frame_index** : Contient la position de la frame de la séquence.

+ **list_started () :** Signal déclenché à chaque fois qu'une ou plusieurs séquence(s) sont en cours de traitement.

+ **list_finished () :** Signal déclenché à chaque fois qu'aucune séquence n'est en cours de traitement.

+ **list_running (data) :** Signal déclenché pendant qu'une ou plusieurs séquence(s) sont en cours de traitement. Cet événement renvoie un dictionaire contenant les clés suivantes :
  » **float time** : Contient le temps écoulé depuis l'exécution de l'ensemble des séquences du module.
  » **int progress** : Contient la progression actuelle de l'ensemble des séquences du module.
  » **float normalized** : Contient la progression normalisée de l'ensemble des séquences du module.
  » **int count** : Contient le nombre total de répétitions éffectuées depuis la première exécution. Cependant, si la liste des séquences est exécutée à l'infinie, vous aurez une valeur négative.

+ **timeout (data) :** Signal déclenché à chaque fois qu'un temps mort est requis avant exécuter une séquence. Cet événement renvoie un dictionaire contenant les clés suivantes :
  » **int preview** : Contient la position de la séquence précédente.
  » **int current** : Contient la position de la séquence future.

+ **sequence_loading (data) :** Signal déclenché pendant qu'une séquence est en cours de chargement. Cet événement renvoie un dictionaire contenant les clés suivantes :
  » **int index** : Contient la position de la séquence en cours de chargement.
  » **bool is_over** : Est-ce que toutes les données de la séquence ont-elles été complètement chargées ?
  » **int progress** : Contient le nombre total actuel de donnée(s) chargée(s) en mémoire.

+ **sequence_saving (data) :** Signal déclenché pendant qu'une séquence est en cours de sauvegarde. Cet événement renvoie un dictionaire contenant les clés suivantes :
  » **int index** : Contient la position de la séquence en question.
  » **int progress** : Contient la progression actuelle en pourcentage de la sauvegarde.

+ **sequence_error (index) :** Signal déclenché lorsque la séquence à chargée n'est pas définit ou que son chargement à échoué.
  » **int index** : Contient la position de la séquence endommagée.

+ **sequence_corrupted (index) :** Signal déclenché lorsque la séquence à chargée a été corromput de l'extérieur.
  » **int index** : Contient la position de la séquence corromput.