Table des matières

I – Définition

AnimatorFx est un module conçut pour éffectuer des animations sur la valeurs valeurs des propriétés de n'importe quel noeud du moteur. Cependant, on note des limites par rapport à certains types de données. Ce module se base sur le noeud *Tween* pour exécuter ses animations.
NB : Ce module est compatible à un jeu 2D, 3D et n'est pas sauvegardable.

II – Les propriétés disponibles

+ int Direction = 0 : Contient la direction de l'animation globale des éléments du module. Les valeurs possibles sont :
-> **MegaAssets.Orientation.NORMAL ou 0 :** Animation en sens normale.
-> **MegaAssets.Orientation.REVERSED ou 1 :** Animation en sens renversé.
-> **MegaAssets.Orientation.RANDOM ou 2 :** Animation aléatoire.
-> **MegaAssets.Orientation.ALTERNATE ou 3 :** Animation en sens alterné.
-> **MegaAssets.Orientation.ALTERNATE_REVERSE ou 4 :** Animation en sens alterné renversé.

+ bool Sync = false : Animation globale des éléments s'éffectuera t-elle de façon synchrônée ?

+ int Count = 1 : Combien de fois l'animation globale des éléments s'éffectuera ?

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

+ NodePath ListenModule : Contient l'instance d'un module du même trampe de ce module.
Exemple : *AnimatorFx, MenuFx, AudioTrackFx* etc... En d'autres termes, contient la référence d'un module possèdant également un curseur. Le but de cette fonctionnalité est de synchrôniser la valeur du curseur du module référé à celle de ce module.

+ Array Animations : Tableau de dictionnaires contenant toutes les différentes configurations sur chaque animation prise en charge par le développeur. Les dictionnaires issus de ce tableau supportent les clés suivantes :

» **float duration = 1.0 :** Quelle est la durée de l'animation à éffectuée ?

» **float speed = 1.0 :** Contrôle la vitesse de l'animation.

» **float seek = 0.0 :** Contrôle le temps de départ de l'animation.

» **float timeout = 1.0** : Quel est le délai avant le déclenchement de l'animation en question ?

» **int repeat = 1** : Combien de fois, l'animation sera répéter ? Cette option ne s'active que lorsque sa valeur est supérieur à **0**. Dans le cas contraire, l'animation n'est pas exécutée. Une valeur négative, entraînera une animation infinie.

» **int direction = 0** : Contrôle le sens d'exécution de l'animation. Les valeurs possibles sont :
-> **MegaAssets.Orientation.NORMAL ou 0** : Animation en sens normale.
-> **MegaAssets.Orientation.REVERSED ou 1** : Animation en sens renversé.
-> **MegaAssets.Orientation.ALTERNATE ou 3** : Animation en sens alterné.
-> **MegaAssets.Orientation.ALTERNATE_REVERSE ou 4** : Animation en sens alterné renversé.
-> **MegaAssets.Orientation.PING_PONG ou 5** : Animation booléenne.

» **int easing = 2** : Contrôle l'assouplissement de l'animation. Les valeurs possibles sont celles de Godot.

» **int type = 0** : Contrôle le type de transition à éffectuée. Les valeurs possibles sont celles de Godot.

» **String | NodePath target** : Quel est le chemin d'accès du noeud sera qui victime de l'animation en question ?

» **String | NodePath action** : Quel est le nom de la propriété ou méthode qui sera victime de l'animation tout en sachant que cette dernière est belle et bien définit au sein du noeud référé. Notez que si votre animation touche une méthode alors, les caractères *()* seront misent à la fin du nom donné à la méthode en question. Dans ce cas les valeurs possibles au sein de l'intervalle [*ivalue* ; *fvalue*]
seront passées en paramètre à la méthode ciblée.

» **Variant ivalue** : Quelle est la valeur initiale de la propriété ciblée ? Si vous donnez le chemin d'accès d'un noeud, la valeur initiale sera égale à celle définit dans le noeud référé tout en sachant que la propriété ciblée est également définie dans ce dernier avec le même type de donnée.
Notez que vous avez également la possibilité de donner un dictionaire supportant les clés : *source, action, value et params* dont la description se trouve dans les bases du framework au niveau de la clé *actions* du champ *EventBindings*.

» **Variant fvalue** : Quelle est la valeur finale de la propriété ciblée ? L'utilisation de cette clé est la même que la précédente.

» **bool forwards = false** : Devons-nous rénitialiser les valeurs de la ou des propriété(s) victime de l'animation après son exécution ?

» **Array | Dictionary started** : Signal déclenché à chaque fois qu'on démarre immédiatement cette animation. 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 l'animation 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 l'animation 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 animations définient par le développeur. Ce curseur n'est rien d'autre que l'index de position de chaque configuration d'animation. Par défaut, sa valeur est **0** lorsqu'il y a une ou plusieurs animation(s) configurée(s) sur le module et **-1** si aucune animation n'est disponible sur le module. Avant de continuer, sachez que nous ne tolérons pas les répétitions sur le nom des propriétés ou méthodes que vous utiliserez.

III – Les méthodes disponibles

+ **void play (id = null, config = {}, interval = 0.0) :** Joue une ou plusieurs animation(s). Si aucun identifiant n'a été référé, la valeur du curseur sera utiliser pour éffectuer le traitement demandé.
  » **int | PoolIntArray id** : Quel(s) est/ sont le(s) identifiant(s) de(s) animation(s) à exécutée(s) ?
  » **Dictionary | Array config** : Voulez-vous changer la valeur de certaines clés de(s) animation(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 animation ?

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

+ **void stop (id = null, interval = 0.0) :** Arrète l'exécution d'une ou de plusieurs animation(s) à partir de sa/leur position. Si aucun identifiant n'a été référé, la valeur du curseur sera utiliser pour éffectuer le traitement demandé.
  » **int | PoolIntArray id** : Quel(s) est/ sont le(s) identifiant(s) de(s) animation(s) à arrètée(s) ?

» **float interval** : Quel est le temps mort avant l'arrêt de chaque animation ?

+ **void preview (config = {}, delay = 0.0) :** Exécute l'animation immédiatement précédente de celle indexée par le curseur du module.
» **Dictionary config** : Voulez-vous changer la valeur de certaines clés de l'animation avant son exécution ?
» **float delay** : Quel est le temps mort avant l'exécution de l'animation en question ?

+ **void next (config = {}, delay = 0.0) :** Exécute l'animation suivant celle indexée par le curseur du module.
» **Dictionary config** : Voulez-vous changer la valeur de certaines clés de l'animation avant son exécution ?
» **float delay** : Quel est le temps mort avant l'exécution de l'animation en question ?

+ **int get_progress (id = null) :** Retourne la progression actuelle d'une animation en cours d'exécution. Si aucun identifiant n'a été précisé, celle de l'ensemble des animations sera renvoyer. La valeur de la progression est dans l'intervalle [0 ; 100].
» **int id** : Quel est l'identifiant de l'animation dont-on souhaite récupérée la progression ?

+ **float get_normalized_progress (id = null) :** Retourne la valeur normalisée de la progression actuelle d'une animation en cours d'exécution. Si aucun identifiant n'a été précisé, celle de l'ensemble des animations sera renvoyer. La valeur de la progression est dans l'intervalle [0.0 ; 1.0].
» **int id** : Quel est l'identifiant de l'animation dont-on souhaite récupérée la progression ?

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

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

+ **int get_current_repeatition (id = null) :** Renvoie le nombre actuelle de répétitions éffectuées auprès d'une animation donnée. Si aucun identifiant n'a été renseigné, celui de l'ensemble des animations sera renvoyer.
  » **int id** : Quel est l'identifiant de l'animation 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_anim_index () :** Renvoie le(s) position(s) de la ou des animation(s) en cours d'exécution.

+ **int get_prev_anim_index () :** Renvoie la position de l'animation précédement exécutée ou générée.

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

+ **int get_big_anim_index () :** Renvoie la position de l'animation ayant la plus grande durée parmit celles définient par le développeur.

+ **int get_small_anim_index () :** Renvoie la position de l'animation ayant la plus petite durée parmit celles définient par le développeur.

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

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

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

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

IV – Les événements disponibles

+ **animation_changed (data) :** Signal déclenché lorsque l'animation en cours d'exécution a changée. Notez que cet événement ne s'appel que lorsque le mode de lecture des animations du module est synchrône. Il renvoie un dictionaire contenant les clés suivantes :
  » **Node node** : Contient le noeud où cet signal a été émit.
  » **int preview** : Contient la position de l'animation précédement exécutée.
  » **int current** : Contient l'index de position de l'animation actuellement en cours d'exécution.

+ **animation_started (data) :** Signal déclenché à chaque fois qu'on démarre immédiatement une animation. Cet événement renvoie un dictionaire contenant les clés suivantes :
  » **Node node** : Contient le noeud où cet signal a été émit.
  » **int index** : Contient la position de l'animation en question.

+ **animation_finished (data) :** Signal déclenché à chaque fois qu'une animation a terminée son exécution. Cet événement renvoie un dictionaire contenant les clés suivantes :
  » **Node node** : Contient le noeud où cet signal a été émit.
  » **int index** : Contient la position de l'animation en question.

+ **animation_playing (data) :** Signal déclenché pendant qu'une animation est en cours d'exécution. Cet événement renvoie un dictionaire contenant les clés suivantes :
  » **Node node** : Contient le noeud où cet signal a été émit.
  » **float time** : Contient le temps écoulé depuis l'exécution de l'animation.
  » **int progress** : Contient la progression actuelle de l'animation.
  » **float normalized** : Contient la progression normalisée de l'animation.
  » **int count** : Contient le nombre total de répétitions éffectuées depuis la première exécution. Cependant, si l'animation est exécutée à l'infinie, vous aurez une valeur négative.
  » **int index** : Contient la position de l'animation en question.

+ **list_started (node) :** Signal déclenché à chaque fois qu'une animation globale démarre son exécution.
  » **Node node** : Contient le noeud où cet signal a été émit.

+ **list_finished (node) :** Signal déclenché à chaque fois qu'une animation globale termine son exécution.
  » **Node node** : Contient le noeud où cet signal a été émit.

+ **list_playing (data) :** Signal déclenché pendant qu'une animation globale est en cours d'exécution. Cet événement renvoie un dictionaire contenant les clés suivantes :
  » **Node node** : Contient le noeud où cet signal a été émit.
  » **float time** : Contient le temps écoulé depuis l'exécution de l'ensemble des animations du module.

» **int progress** : Contient la progression actuelle de l'ensemble des animations du module.
» **float normalized** : Contient la progression normalisée de l'ensemble des animations du module.
» **int count** : Contient le nombre total de répétitions éffectuées depuis la première exécution. Cependant, si l'animation générale 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 animation. Cet événement renvoie un dictionaire contenant les clés suivantes :
» **Node node** : Contient le noeud où cet signal a été émit.
» **int preview** : Contient la position de l'animation précédente.
» **int current** : Contient la position de l'animation future.