外部制御コマンド
リファレンス マニュアル

Ver 1.0.2

2026.10.2

・本書の著作権は、ヒビノプロビジョン株式会社が所有しています。
・本書の内容に関しては、将来予告無しに変更することがあります。
・本書は内容について万全を期して作成致しましたが、ご不審な点や誤り・記載もれなどお気付きの点がありましたら、
viz-support@hibino-provision.co.jp までご連絡ください。
・本書を運用した結果の影響については、いかなる件にも責任を負いかねますので予めご了承ください。

Vizrt、VizArtist、VizEngine は Vizrt 社の登録商標または商標です。

目次

◎はじめに ・・・・・・・・・・・・・・・・・・・・・・ 4

◎シーン関連のコマンド ・・・・・・・・・・・・・・・・・・・・・・ 5

◎ライト関連のコマンド ・・・・・・・・・・・・・・・・・・・・・・ 7

◎カメラ関連のコマンド ・・・・・・・・・・・・・・・・・・・・・・ 8

◎コンテナ関連のコマンド

★シーンツリー ・・・・・・・・・・・・・・・・・・・・・・ 11
★特定コンテナ ・・・・・・・・・・・・・・・・・・・・・・ 12
★テキスト (Classic) ・・・・・・・・・・・・・・・・・・・・・・ 13
★テキスト (VizRender) ・・・・・・・・・・・・・・・・・・・・・・ 14
★テクスチャ (Classic) ・・・・・・・・・・・・・・・・・・・・・・ 15
★マテリアル (Classic) ・・・・・・・・・・・・・・・・・・・・・・ 15
★MATERIAL_DEFINITION (VizRender) ・・・・・・・・・・・・・・・・・・・・・・ 15
★コンテナの検索 ・・・・・・・・・・・・・・・・・・・・・・ 16

◎Animation 関連のコマンド

★Stage 全体 ・・・・・・・・・・・・・・・・・・・・・・ 18
★全 Director ・・・・・・・・・・・・・・・・・・・・・・ 19
★特定の Director ・・・・・・・・・・・・・・・・・・・・・・ 20
★Animation - ポイント ・・・・・・・・・・・・・・・・・・・・・・ 22
★Animation-ACTION(Channel) ・・・・・・・・・・・・・・・・・・・・・・ 24
★Animation-ACTION(Keyframe) ・・・・・・・・・・・・・・・・・・・・・・ 24
★Animation-CHANNEL ・・・・・・・・・・・・・・・・・・・・・・ 27
★Animation-CHANNEL(Keyframe) ・・・・・・・・・・・・・・・・・・・・・・ 28

◎vizrt プラグイン関連のコマンド

★Scene Plugins ・・・・・・・・・・・・・・・・・・・・・・ 31
★Container Plugins ・・・・・・・・・・・・・・・・・・・・・・ 31
★Geometry Plugins ・・・・・・・・・・・・・・・・・・・・・・ 31
★BUILT_IN Plugins ・・・・・・・・・・・・・・・・・・・・・・ 32
★AUDIO Plugin ・・・・・・・・・・・・・・・・・・・・・・ 33

◎vizrt Script 関連のコマンド ・・・・・・・・・・・・・・・・・・・・・・ 35

◎Control Object 関連のコマンド ・・・・・・・・・・・・・・・・・・・・・・ 36

◎コントロールチャネル関連のコマンド ・・・・・・・・・・・・・・・・・・・・・・ 37

◎Asset 関連のコマンド ・・・・・・・・・・・・・・・・・・・・・・ 38

◎画像の取得関連のコマンド ・・・・・・・・・・・・・・・・・・・・・・ 38

◎システム関連のコマンド ・・・・・・・・・・・・・・・・・・・・・・ 39
◎選択範囲と Bounding box 関連のコマンド ・・・・・・・・・・・・・・・・・・・・ 41
◎ポストレンダリング関連のコマンド
★Post Render Templates の設定 ・・・・・・・・・・・・・・・・・・・・ 42
★Clip Plugin 項目の設定 ・・・・・・・・・・・・・・・・・・・・ 43
★Scene Setup 項目の設定 ・・・・・・・・・・・・・・・・・・・・ 45
◎アーカイブ関連のコマンド ・・・・・・・・・・・・・・・・・・・・ 47
◎動作ログ関連のコマンド ・・・・・・・・・・・・・・・・・・・・ 48
◎Config 関連のコマンド ・・・・・・・・・・・・・・・・・・・・ 49
◎Shared Memory (SHM)関連のコマンド ・・・・・・・・・・・・・・・・・・・・ 50

＜付録⓪＞一般的な vizrt の外部制御 ・・・・・・・・・・・・・・・・・・・・ 51
＜付録①＞VIZ コマンドの調べ方 ・・・・・・・・・・・・・・・・・・・・ 52
＜付録②＞VizCommand Console ・・・・・・・・・・・・・・・・・・・・ 53
＜付録③＞用語解説や補足 ・・・・・・・・・・・・・・・・・・・・ 54
＜付録④＞Shared Memory (SHM)について ・・・・・・・・・・・・・・・・・・・・ 58
＜付録⑤＞Control Object について ・・・・・・・・・・・・・・・・・・・・ 64
＜付録⑥＞Control Text 文字制御用タグ一覧 ・・・・・・・・・・・・・・・・・・・・ 69
＜付録⑦＞Vizrt 起動時のコマンドライン オプション ・・・・・・・・・・・・・・・・・・・・ 70
＜付録⑧＞その他 ・・・・・・・・・・・・・・・・・・・・ 72

はじめに

▲目次▲

- 本マニュアルは、VIZ コマンド構文の説明書です。
- VIZ コマンドシステム自体を知りたい方は、後述の「**付録⓪：一般的な vizrt の外部制御**」をご確認ください。
- 本文中の(クラス参照先)は、vizrt 付属コマンドリファレンスの Class 名です。
  本ドキュメント未記載の機能を調べる手掛かりにしてください。
- 後述の「**付録①：コマンドの調べ方**」も、ご確認ください。
- 制御コマンドは、ロケーション部分をオブジェクト ID に代えても動作します。
  但し、その際はオブジェクト ID がどのロケーションを指し示しているのかに注意する必要があります。
- 制御コマンドは、ロケーション部分をオブジェクト ID に代えても動作します。

本マニュアルで使用されている代替名規則

※下記以外の代替名もあります。必要に応じてコマンドに説明を追加しています。

<Asset パス名>	- フォルダパス	例: Hibino/Folder
<Asset フォルダ名>	- プロジェクト or フォルダ名	例: Hibino
<Asset シーン名>	- シーン名	例: SampleScene1
<ノード階層>	- ノード (階層)	例: allbase$kerne
	- コントロールチャネル名	例: @ControlChannelName
<シーン ID>	- シーン ID　(UUID)	例: <FA20E3EC-EFC2-A949-B8AADF9340B82E8F>
<シーン名>	- シーン名	例: Photron/SampleScene1
<シーン単体名>	- シーン単体名	例: SampleScene1
<#オブジェクト ID>	- オブジェクト ID	例: #407
<オブジェクト番号>	- オブジェクト ID の数字部分	例: 407
<$ディレクター名>	- $付きディレクター名	例: $Dir1
<ON/OFF 値>	- ONOFF 値	例: 0 (ON)、1 (OFF)
<ポイント名>	- STOP/TAG ポイント名	例: PointName
<キーフレーム名>	- キーフレーム名	例: Keyframe1、$Keyframe1
<キーフレーム番号>	- キーフレーム番号 (0～)	例: 0
<キーフレーム値>	- キーフレーム データ値	例: 10.0 20.0 30.0
<カメラ番号>	- 1～16 のカメラ番号	例: 1
<ライト番号>	- 1～8 のライト番号	例: 1
<Viz 画像パス>	- VizGH の画像名とパス	例: IMAGE*Hibino/image1
<外部画像パス>	- 外部の画像名とパス	例: z:¥share¥logos¥photron.png
<マテリアル名>	- マテリアル名	例: Photron/Material1
<プラグイン名>	- vizrt プラグイン名	例: TreeStatus
<プラグイン変数>	- vizrt プラグイン パラメータ名	例: command
<プラグインボタン名>	- vizrt プラグインのボタン名	例: initialize
<アクション名>	- アクション名	例: MyAction
<AUDIO CLIP パス>	- AUDIO CLIP パス	例: Hibino/audio_sample1
<AUDIO CLIP 名>	- AUDIO CLIP 名	例: audio_sample1
<Time 値>	- vizrt 時間 (秒)	例：30.3
	- vizrt 時間(フィールド)	例：f19、F19
<Time 値(秒)>	- vizrt 時間 (秒)のみ	例：30.3
[選択肢 1 | 選択肢 2]	- |で区切られた選択肢のどれか	例：[AAA | BBB]・・　AAA or BBB

◎シーン関連のコマンド

▲目次▲

- レンダラーにシーンを設定 （クラス参照先：RENDERER_BASE - Commands）

  送信コマンド　：REND SET_OBJECT SCENE*<シーン名>

  応答データ　：<シーン ID>

  - ✧ MAIN Layer にシーンをレンダリングします。
  - ✧ シーンを指定しない場合、各レンダリングエリアからシーンを削除します。但し、シーンはメモリにロードされている状態になります。

- レンダリングされているシーンのオブジェクト ID を取得 （クラス参照先：RENDERER_BASE - Commands）

  送信コマンド　：REND GET_OBJECT

  応答データ　：<#オブジェクト ID>

  - ✧ MAIN Layer 用です。

- レンダラーにシーンを設定 (Classic) （クラス参照先：RENDERER_BASE - Properties）

  BACK Layer　：REND*BACK_LAYER SET_OBJECT SCENE*<シーン名>

  Front Layer　：REND*FRONT_LAYER SET_OBJECT SCENE*<シーン名>

  応答データ　：<シーン ID>

  - ✧ BACK | FRONT Layer 用です。
  - ✧ MAIN Layer に Classic Render シーンがレンダリングされている時のみ使用できます。
  - ✧ シーンを指定しない場合、各レンダリングエリアからシーンを削除します。但し、シーンはメモリにロードされている状態になります。

- レンダリングされているシーンのオブジェクト ID を取得 (Classic) （クラス参照先：RENDERER_BASE - Properties）

  BACK Layer　：REND*BACK_LAYER GET_OBJECT

  Front Layer　：REND*FRONT_LAYER GET_OBJECT

  応答データ　：<#オブジェクト ID>

  - ✧ BACK | FRONT Layer 用です。
  - ✧ MAIN Layer に Classic Render シーンがレンダリングされている時のみ使用できます。

- シーンをメモリにロード （クラス参照先：Pool - Commands）

  送信コマンド　：SCENE*<シーン名> LOAD

  応答データ　：<#オブジェクト ID>

  - ✧ シーンを表示するには SET_OBJECT コマンドでレンダリングする必要があります。

- 指定フォルダのシーンをロード （クラス参照先：Pool - Commands）

  送信コマンド　：SCENE LOAD_ALL <シーンフォルダパス>

  応答データ　：無し

  - ✧ ロード済みのシーンは無視されます
  - ✧ シーンフォルダパス最後尾の '/' は、付けても付けなくても、どちらでも構いません。

- シーンを再読み込み(メモリにリロード) （クラス参照先：Pool - Commands）

  送信コマンド　：SCENE*<シーン名> RELOAD

  応答データ　：無し

  - ✧ メモリ上のシーンを VizGH からロードし直します。
  - ✧ レンダラーに設定されているシーンもリロードされます。
  - ✧ この再読み込みシーンをレンダリングするには、改めて SET_OBJECT を使用する必要があります。

- シーンの存在確認 （クラス参照先：Pool - Commands）

  送信コマンド　：SCENE*<シーン名> EXIST

  応答データ　：＜ON/OFF 値＞

- シーンの上書き保存　（クラス参照先：Pool - Commands）
  送信コマンド　：SCENE*<シーン名> SAVE
  応答データ　：無し
- シーンの別名保存　（クラス参照先：Pool - Commands）
  送信コマンド　：SCENE*<シーン名> SAVE_AS <シーン別名>
  応答データ　：<別名保存先のシーン ID>
- シーンを閉じる　（クラス参照先：Pool - Commands）
  送信コマンド　：SCENE CLOSE <シーン ID>
  応答データ　：無し
  - ✧ Image や Font を除いたシーン内で使用されている全てのものは、メモリから削除されます
  - ✧ レンダラーに設定されているシーンには使用できません
- Viz Artist エディタ上でシーンが変更されたかの判定　（クラス参照先：Pool - Commands）
  送信コマンド　：SCENE IS_CHANGED　<シーン名 or シーン ID>
  応答データ　：<変更有無値>
  - ✧ <変更有無値>：変更有：1、無し：0
  - ✧ 外部制御で変更した場合、応答データに変化はありません
- レンダリング中シーンのシーン名を取得　（クラス参照先：CPoolObj - Properties）
  送信コマンド　：REND*LOCATION_PATH GET
  応答データ　：<シーン名>
  - ✧ <シーン名>： 例) SCENE*Default/NewScene
- レンダリング中シーンのシーン名を取得　（クラス参照先：CPoolObj - Properties）
  送信コマンド　：REND*NAME GET
  応答データ　：<シーン単体名>
  - ✧ シーンパスは含みません。シーン単体の名前です。
- レンダリング中シーンのレンダラー種類を取得　（クラス参照先：SCENE – Properties）
  送信コマンド　：REND*RENDERENGINE_VERSION GET
  応答データ　：<レンダー種類>
  - ✧ <レンダー種類>：[V3 (Classic) | V4 (VizRender)]
- レンダリング中シーンの MediaAssets 一覧を取得　（クラス参照先：MediaLayerEditor – Properties）
  送信コマンド　：REND*MEDIA_ASSETS*OVERVIEW*DATA GET
  応答データ　：シーンで使用中の MediaAssets の構造データ
  - ✧ 例えば、GFX のオブジェクト ID、名称、種類(TEXTURE/DVE)等が含まれます
  - ✧ Live Channel、Clip 等も取得できます

◎ライト関連のコマンド

▲目次▲

- シーン LIGHT の色を設定 (Classic)　（クラス参照先：LIGHT - Properties）

  送信コマンド　：REND*LIGHT<ライト番号>*COLOR*COLOR SET <R 正規化値> <G 正規化値> <B 正規化値>

  応答データ　：無し

  - ✧ <正規価値>：0.0～1.0
  - ✧ COLOR は 2 つ並ぶ必要があります。最初の COLOR は MATERIAL の意味です。

- シーン LIGHT の ON/OFF を設定 (Classic)　（クラス参照先：LIGHT - Properties）

  送信コマンド　：REND*LIGHT<ライト番号>*ACTIVE SET <ON/OFF 値>

  応答データ　：無し

- シーン LIGHT の種類を設定 (Classic)　（クラス参照先：LIGHT - Properties）

  送信コマンド　：REND*LIGHT<ライト番号>*TYPE SET [LOCAL | SPOT | INFINITE]

  応答データ　：無し

- シーン LIGHT の位置を設定 (Classic)　（クラス参照先：LIGHT - Properties）

  送信コマンド　：REND*LIGHT<ライト番号>*POSITION SET <X 値> <Y 値> <Z 値>

  応答データ　：無し

◎カメラ関連のコマンド

▲目次▲

- 現在のカメラを取得（MAIN Layer）　（クラス参照先：COMMAND_INFO | RENDER_EDITOR - Properties）
  - 送信コマンド①　：REND*CURRENT_CAMERA GET
  - 送信コマンド②　：REND*CURRENT_CAMERA GET MAIN_LAYER
  - 応答データ　　：<カメラ番号>

- 現在のカメラを取得（FRONT/BACK Layer）　（クラス参照先：COMMAND_INFO | RENDER_EDITOR - Properties）
  - FRONT Layer　：REND*CURRENT_CAMERA GET FRONT_LAYER
  - BACK Layer　：REND*CURRENT_CAMERA GET BACK_LAYER
  - 応答データ　　：<カメラ番号>

- 現在のカメラを取得（SCENE）　（クラス参照先：SCENE - Properties）
  - 送信コマンド　：<シーン名>*CURRENT_CAMERA GET
  - 応答データ　　：<カメラ番号>
    - ✧　シーンに保存されているカメラ値、もしくは、ロード済みや使用中シーンの場合は、最後に設定されたカメラ値

- 現在のカメラを変更（MAIN Layer）　（クラス参照先：COMMAND_INFO | RENDER_EDITOR - Properties）
  - 送信コマンド①　：REND SET_CAMERA <カメラ番号>
  - 送信コマンド②　：REND*CURRENT_CAMERA SET <カメラ番号>
  - 送信コマンド③　：REND*MAIN_LAYER*CURRENT_CAMERA SET <カメラ番号>
  - 応答データ　　：無し

- 現在のカメラを変更（FRONT/BACK Layer）　（クラス参照先：COMMAND_INFO | RENDER_EDITOR - Properties）
  - FRONT Layer　：REND*FRONT_LAYER*CURRENT_CAMERA SET <カメラ番号>
  - BACK Layer　：REND*BACK_LAYER*CURRENT_CAMERA SET <カメラ番号>
  - 応答データ　　：無し

- 現在のカメラを変更（SCENE）　（クラス参照先：SCENE - Properties）
  - 送信コマンド　：<シーン名>*CURRENT_CAMERA SET
  - 応答データ　　：無し

- 現在のカメラを変更 (ACTION – Keyframe Command)　（クラス参照先：RENDER_EDITOR - Properties）
  - 送信コマンド①　：THIS_SCENE*SET_CAMERA <カメラ番号>
  - 送信コマンド②　：MAIN_SCENE*SET_CAMERA <カメラ番号>
  - 応答データ　　：無し

- カメラの Position を個別に取得　（クラス参照先：Rendercamera - Properties）
  - 送信コマンド　：REND*CAMERA<カメラ番号>*POSITION*[X | Y | Z] GET
  - 応答データ　　：<設定値>
    - ✧　※注) リファレンスに取得は無いが、Transform Position 値を個別に取得できる。

- カメラの Position を個別に変更(絶対値)　（クラス参照先：Rendercamera - Properties）
  - 送信コマンド　：REND*CAMERA<カメラ番号>*POSITION*[X | Y | Z] SET <絶対値>
  - 応答データ　　：無し
    - ✧　※注) リファレンスでは相対値設定だが、Viz5 では絶対値。

- カメラの Position を個別に変更(相対値) (クラス参照先：Rendercamera - Properties)
  送信コマンド　：REND*CAMERA<カメラ番号>*POSITION_RELATIVE *[X | Y | Z] SET <相対値>
  応答データ　　：無し

- カメラの Position を一括で取得 (クラス参照先：CVCamera - Properties)
  送信コマンド　：REND*CAMERA<カメラ番号>*POSITION GET
  応答データ　　：<X> <Y> <Z>

- カメラの Position を一括で変更 (クラス参照先：CVCamera - Properties)
  送信コマンド　：REND*CAMERA<カメラ番号>*POSITION SET <X> <Y> <Z>
  応答データ　　：無し

- カメラの Direction を個別に取得 (クラス参照先：Rendercamera - Properties)
  送信コマンド　：REND*CAMERA<カメラ番号>*[PAN | TILT | TWIST] GET
  応答データ　　：<設定値>

- カメラの Direction を個別に変更 (クラス参照先：Rendercamera - Properties)
  送信コマンド　：REND*CAMERA<カメラ番号>*[PAN | TILT | TWIST] SET <設定値>
  応答データ　　：無し

- カメラの Lens Angle を取得 (クラス参照先：Rendercamera - Properties)
  送信コマンド　：REND*CAMERA<カメラ番号>*ZOOM GET
  応答データ　　：<設定値>

- カメラの Lens Angle を変更 (クラス参照先：Rendercamera - Properties)
  送信コマンド　：REND*CAMERA<カメラ番号>*ZOOM SET <設定値>
  応答データ　　：無し
  - ZOOM 値は、Script での Camera.Fovy の値になります。
  - Camera.Fovx は自動的に設定されます。

- カメラの Center shift X を取得 (クラス参照先：Rendercamera - Properties)
  送信コマンド　：REND*CAMERA<カメラ番号>* CENTER_SHIFT_X GET
  応答データ　　：<設定値>

- カメラの Center shift X を変更 (クラス参照先：Rendercamera - Properties)
  送信コマンド　：REND*CAMERA<カメラ番号>* CENTER_SHIFT_X SET <設定値>
  応答データ　　：無し

- カメラの Center shift Y を取得 (クラス参照先：Rendercamera - Properties)
  送信コマンド　：REND*CAMERA<カメラ番号>* CENTER_SHIFT_Y GET
  応答データ　　：<設定値>

- カメラの Center shift Y を変更 (クラス参照先：Rendercamera - Properties)
  送信コマンド　：REND*CAMERA<カメラ番号>* CENTER_SHIFT_Y SET <設定値>
  応答データ　　：無し

- カメラのリモート ONOFF を取得　（クラス参照先：Rendercamera - Properties）
  送信コマンド　：REND*CAMERA<カメラ番号>*EXTERNAL GET
  応答データ　：<ON/OFF 値>

- カメラのリモート ONOFF を変更　（クラス参照先：Rendercamera - Properties）
  送信コマンド　：REND*CAMERA<カメラ番号>*EXTERNAL SET <ON/OFF 値>
  応答データ　：無し

- カメラの投影設定を取得　（クラス参照先：Rendercamera - Properties）
  送信コマンド　：REND*CAMERA<カメラ番号>*VIEW GET
  応答データ　：[PERSPECTIVE | ORTHOGONAL]（※透視投影 | ※並行投影）

- カメラの投影設定を変更　（クラス参照先：Rendercamera - Properties）
  送信コマンド　：REND*CAMERA<カメラ番号>*VIEW SET <投影モード>
  応答データ　：無し
  ✧　<投影モード>：[PERSPECTIVE | ORTHOGONAL]（※投資投影 | ※並行投影）

◎コンテナ関連のコマンド

▲目次▲

★シーンツリー

- シーンツリーのツリー情報を取得①　（クラス参照先：SCENETREE – Commands）
  - 送信コマンド　：REND*TREE GET
  - 応答データ　：{<階層番号> <オブジェクト番号> <コンテナ名>}{・・・・
    - ✧ <階層番号>： 例) 子コンテナ無し: 1(=同階層順)、 子コンテナあり 2 層: 1/2、 子コンテナ 3 層: 1/1/2

- シーンツリーのツリー情報を取得②　（クラス参照先：SCENETREE – Properties）
  - 送信コマンド　：REND*TREE*DATA GET
  - 応答データ　：{<階層番号> <オブジェクト番号> <コンテナ名>}{・・・・
    - ✧ <階層番号>： 例) 子コンテナ無し: 1(=同階層順)、 子コンテナあり 2 層: 1/2、 子コンテナ 3 層: 1/1/2

- 特定コンテナ以下のツリー情報を取得　（クラス参照先：CONTAINER - Commands）
  - 送信コマンド　：REND*TREE*<ノード階層> GET
  - 応答データ　：{<階層番号> <オブジェクト番号> <コンテナ名>}{・・・・
    - ✧ 応答データは、特定コンテナを含む同一階層の NEXT、及び、子コンテナが対象です。

- コンテナを追加① (シーンツリーの先頭①)　（クラス参照先：SCENETREE – Commands）
  - 送信コマンド　：REND*TREE CREATE TOP
  - 応答データ　：<#オブジェクト ID>
    - ✧ TOP キーワードは PREVIOUS でも可

- コンテナを追加② (シーンツリーの先頭②)　（クラス参照先：SCENETREE – Commands）
  - 送信コマンド　：REND*TREE ADD TOP
  - 応答データ　：<階層番号>
    - ✧ TOP キーワードは PREVIOUS でも可
    - ✧ このコマンドの場合、応答データは常に 1

- コンテナを追加③ (ノード階層を指定)　（クラス参照先：CONTAINER – Commands）
  - 送信コマンド　：REND*TREE*<ノード階層> ADD <追加位置>
  - 応答データ　：<階層番号>
    - ✧ <追加位置>：
      - ➢ 特定コンテナの後に作成　[NEXT | BOTTOM]
      - ➢ 特定コンテナの前に作成　[TOP | PREVIOUS]
      - ➢ 特定コンテナの子を作成　[DOWN | RIGHT]
      - ➢ 特定コンテナの親を作成　[UP | LEFT]
    - ✧ <階層番号>： 例) 子コンテナ無し: 1(=同階層順)、 子コンテナあり 2 層: 1/2、 子コンテナ 3 層: 1/1/2

- 特定コンテナの階層を移動　（クラス参照先：CONTAINER – Commands）
  - 送信コマンド　：REND*TREE*<ノード階層> MOVE REND*TREE*<移動先ノード階層> <移動位置>
  - 応答データ　：無し
    - ✧ <移動位置>：
      - ➢ 移動先コンテナの後に移動　[NEXT | BOTTOM]
      - ➢ 移動先コンテナの前に移動　[TOP | PREVIOUS]
      - ➢ 特定コンテナの子に移動　[DOWN | RIGHT]
    - ✧ [UP | LEFT]は使用出来ません。特定コンテナの親に移動せず、シーンツリーから消去された上に Undo が効きません。
    - ✧ 移動先コンテナが同一名で複数ある場合、ツリーの先頭から最初に見つかるコンテナが移動先になります。

★特定コンテナ

▲目次▲

- 特定コンテナに名前を設定① (オブジェクト ID を指定)　(クラス参照先：CONTAINER – Properties)
  - 送信コマンド　：<#オブジェクト ID>*NAME SET <コンテナ名>
  - 応答データ　：無し

- 特定コンテナに名前を設定② (階層番号を指定)　(クラス参照先：CONTAINER – Properties)
  - 送信コマンド　：REND*TREE*<階層番号>*NAME SET <コンテナ名>
  - 応答データ　：無し
    - ✧　<階層番号>：例) サブフォルダ無し: 1(=同階層順)、　サブフォルダあり 2 層: 1/2、　サブフォルダ 3 層: 1/1/2

- 特定コンテナのオブジェクト ID を取得① (ノード階層を指定)　(クラス参照先：OBJECTBASE - Properties)
  - 送信コマンド　：REND*TREE*<ノード階層>*OBJECT_ID GET
  - 応答データ　：<#オブジェクト ID>

- 特定コンテナのオブジェクト ID を取得② (階層番号を指定)　(クラス参照先：OBJECTBASE - Properties)
  - 送信コマンド　：REND*TREE*<階層番号>*OBJECT_ID GET
  - 応答データ　：<#オブジェクト ID>
    - ✧　<階層番号>：サブフォルダ無し: 1(=同階層順)、　サブフォルダあり 2 層: 1/2、　サブフォルダ 3 層: 1/1/2

- 特定コンテナのみを削除　(クラス参照先：CONTAINER - Commands)
  - 送信コマンド　：REND*TREE*<ノード階層> DELETE
  - 応答データ　：無し
    - ✧　子コンテナがあるコンテナを削除した場合、子コンテナの階層が一つ上がります。
    - ✧　同一名のコンテナは最初に見つかったコンテナが削除されます。

- 特定コンテナの子コンテナも含めて削除　(クラス参照先：CONTAINER - Commands)
  - 送信コマンド　：REND*TREE*<ノード階層> DELETE ALL
  - 応答データ　：無し
    - ✧　同一名のコンテナは最初に見つかったコンテナが削除されます。

- 特定コンテナの子コンテナのみ削除　(クラス参照先：CONTAINER - Commands)
  - 送信コマンド　：REND*TREE*<ノード階層> DELETE UNDER
  - 応答データ　：無し
    - ✧　同一名のコンテナは最初に見つかったコンテナが削除されます。

- 特定コンテナの表示/非表示の変更　(クラス参照先：CONTAINER - Properties)
  - 送信コマンド　：REND*TREE*<ノード階層>*ACTIVE SET <ON/OFF 値>
  - 応答データ　：無し

- 特定コンテナの RGB・Key の表示/非表示の変更　(クラス参照先：KEY - Properties)
  - RGB　：REND*TREE*<ノード階層>*KEY*DRAW_RGB SET <ON/OFF 値>
  - Key　：REND*TREE*<ノード階層>*KEY*DRAW_KEY SET <ON/OFF 値>
  - 応答データ　：無し
    - ✧　コンテナに Key-Function が必要です。

- 特定コンテナのプロパティ名の取得　(クラス参照先：CONTAINER - Properties)
  - 送信コマンド　：REND*TREE*<ノード階層>* PROPERTIES GET
  - 応答データ　：コンテナに適用中の(FUNCTION 名などの)プロパティ名

- 特定コンテナの[位置 | 回転 | スケール]パラメータの変更 （クラス参照先：TRANSFORMATION - Properties）
  送信コマンド　：REND*TREE*<ノード階層 >*TRANSFORMATION*<パラメータ種類> SET <x> <y> <z>
  応答データ　：無し
  ✧ <パラメータ種類> : POSITION / ROTATION / SCALING

- 特定コンテナの位置をカメラ中央に合わせる （クラス参照先：POSXYZ - Commands）
  送信コマンド　：REND*TREE*<ノード階層 >*TRANSFORMATION*POSITION CENTER_TO_CAMERA
  応答データ　：無し

- 特定コンテナの回転をカメラに向ける （クラス参照先：ROTATION - Commands）
  送信コマンド　：REND*TREE*<ノード階層 >*TRANSFORMATION*ROTATION FACE_CAMERA
  応答データ　：無し

- 特定コンテナの LOOK_AT を有効化 （クラス参照先：※COMMAND_INFO のみ）
  送信コマンド　：REND*TREE*<ノード階層>*LOOK_AT SET LOOK_AT*Look_At
  応答データ　：無し
  ✧ LOOK_AT Function が追加されます。DELETE コマンドで削除できます。

- 特定コンテナの LOOK_AT の有効/無効の確認 （クラス参照先：※COMMAND_INFO のみ）
  送信コマンド　：REND*TREE*<ノード階層>*LOOK_AT GET
  応答データ　：[有効時:<#オブジェクト ID>* LOOK_AT]、[無効時:無し]

- 特定コンテナの LOOK_AT の設定 （クラス参照先：CVLookAt - Properties）
  送信コマンド　：REND*TREE*<ノード階層 >*LOOK_AT*AUTO_ROTATION SET [BILLBOARD | NONE]
  応答データ　：無し
  ✧ NONE の代わりに FOLLOW_PATH でも可。

- 特定コンテナの LOOK_AT のビルボードの軸を変更 （クラス参照先：CVLookAt - Properties）
  送信コマンド　：REND*TREE*<ノード階層 >* LOOK_AT*BILLBOARD_AXIS SET [X | Y | XY]
  応答データ　：無し

★テキスト (Classic)

▲目次▲

- テキストの変更 （クラス参照先：GEOM_TEXT - Properties）
  送信コマンド　：REND*TREE*<ノード階層>*GEOM*TEXT SET <文字列>
  応答データ　：無し

- テキストを取得 （クラス参照先：GEOM_TEXT - Properties）
  送信コマンド　：REND*TREE*<ノード階層>*GEOM*TEXT GET
  応答データ　：文字列

- テキスト 文字色の変更 （クラス参照先：GEOM_TEXT - Properties）
  送信コマンド　：REND*TREE*<ノード階層>*GEOM*PROP*COLOR SET <POS 値> <r> <g> <b>
  応答データ　：無し
  ✧ <POS 値> ：文字の位置。 行位置（１～）と文字位置（０～）を.(ドット)で接続する。全てを指定する場合は、ALL。
  例) 1 行目の 2 文字目は、1.1
  ✧ <r>,<g>,<b> ：正規化値(0.0～1.0)
  ✧ 旧バージョン互換。MATERIAL の使用か、ControlText の使用を検討してください。

- テキスト 全体的な文字間隔の変更 （クラス参照先：GEOM_TEXT - Properties）

  送信コマンド　：REND*TREE*<ノード階層>*GEOM*KERNING SET <K 値>

  応答データ　：無し

  - <K 値> ：隙間値
  - Text ジオメトリの全体的な文字間隔を設定します。
  - Letter Spacing (=viz3:traking) に値が設定されます。
  - 下記の PROP*KERNING / PROP*EXT_KERNING(2 文字間用)と併用可能ですのでご注意ください。

- テキスト 全体的な文字間隔の取得 （クラス参照先：GEOM_TEXT - Properties）

  送信コマンド　：REND*TREE*<ノード階層>*GEOM*KERNING GET

  応答データ　：全体の隙間値

- テキスト 任意の 2 文字の間隔の変更 （クラス参照先：GEOM_TEXT - Properties）

  送信コマンド　：REND*TREE*<ノード階層>*GEOM*PROP*EXT_KERNING SET <POS 値> <K 値>

  応答データ　：無し

  - <POS 値> ： 2 文字間隔の位置。 行位置と文字位置を.(ドット)で接続する。全てを指定する場合は、ALL。
    例) 1 行目の 2 文字目と 3 文字目の間は、1.2。
  - <K 値>　：隙間値
  - PROP*KERNING コマンドでセットされるパラメータ値を置き換えます。
  - 上記の KERNING(全体用)と併用可能ですのでご注意ください。
  - 文字と設定値の関係が視覚的にわかりやすいので、ControlText の使用をご検討ください。

- テキスト 任意の 2 文字の間隔の取得 （クラス参照先：GEOM_TEXT - Properties）

  送信コマンド　：REND*TREE*<ノード階層>*GEOM*PROP*EXT_KERNING GET ALL

  応答データ　：“PROP*EXT_KERNING SET ALL”コマンドでセットした時の<K 値>

- テキスト 任意の 2 文字の間隔の変更 （クラス参照先：GEOM_TEXT - Properties）

  送信コマンド　：REND*TREE*<ノード階層>*GEOM*PROP*KERNING SET <POS 値> <K 値>

  応答データ　：無し

  - <POS 値> ： 2 文字間隔の位置。 行位置と文字位置を.(ドット)で接続する。
    例) 1 行目の 2 文字目と 3 文字目の間は、1.2。
  - <K 値>　：隙間値
  - SET コマンドですが、PROP*EXT_KERNING と異なり、ADD コマンドと同様に 2 文字間の隙間値を追加するので、ご注意ください。
  - 上記の KERNING(全体用)と併用可能ですのでご注意ください。
  - 文字と設定値の関係が視覚的にわかりやすいので、ControlText の使用をご検討ください。

- テキスト 任意の 2 文字の間隔の取得 （クラス参照先：GEOM_TEXT - Properties）

  送信コマンド　：REND*TREE*<ノード階層>*GEOM*PROP*KERNING GET

  応答データ　：任意の 2 文字の隙間値

★テキスト (VizRender)

▲目次▲

- テキストの変更 （クラス参照先：viz::engine::FusionText - Properties）

  送信コマンド　：REND*TREE*<ノード階層>*GEOM*UTF8_TEXT SET <文字列>

  応答データ　：無し

- テキストを取得 （クラス参照先：viz::engine::FusionText - Properties）

  送信コマンド　：REND*TREE*<ノード階層>*GEOM*UTF8_TEXT GET

  応答データ　：文字列

★テクスチャ (Classic)

▲目次▲

- 画像を設定① (クラス参照先：TEXTURE - Properties)
  - 送信コマンド　：REND*TREE*<ノード階層>*TEXTURE*IMAGE SET [<Viz 画像パス> | <外部画像パス>]
  - 応答データ　：無し
- 画像を設定② (クラス参照先：CONTAINER - Properties)
  - 送信コマンド　：REND*TREE*<ノード階層>*IMAGE SET [<Viz 画像パス> | <外部画像パス>]
  - 応答データ　：無し
    - ✧　設定のみ TEXTURE キーワードが無くても構いません。
- 画像名を取得 (クラス参照先：TEXTURE - Properties)
  - 送信コマンド　：REND*TREE*<ノード階層>*TEXTURE*IMAGE GET
  - 応答データ　：[<Viz 画像パス> | <外部画像#オブジェクト ID>]
    - ✧　<外部画像#オブジェクト ID>：設定されている画像が外部画像の場合、IMAGE*<#オブジェクト ID>が返ります。
    - ✧　取得する際は、TEXTURE*IMAGE キーワードです。
- 画像を削除 (クラス参照先：※COMMAND_INFO のみ)
  - 送信コマンド　：REND*TREE*<ノード階層>*TEXTURE DELETE
  - 応答データ　：無し
    - ✧　削除する際は、TEXTURE キーワードです。

★マテリアル (Classic)

- マテリアルを変更 (クラス参照先：CONTAINER - Properties)
  - 送信コマンド　：REND*TREE*<ノード階層>*MATERIAL SET MATERIAL*<マテリアル名>
  - 応答データ　：無し
- マテリアルの RGB 値を変更 (クラス参照先：Material - Properties)
  - 送信コマンド　：REND*TREE*<ノード階層>*MATERIAL*<マテリアルパラメータ種類> SET <r> <g> <b>
  - 応答データ　：無し
    - ✧　<マテリアルパラメータ種類> : [AMBIENT | SPECULAR | DIFFUSE | EMISSION | COLOR]
    - ✧　<r>,<g>,<b> ：正規化値(0.0～1.0)
    - ✧　マテリアルパラメータ種類 [COLOR] ：※ライトに影響されないベースカラー
- マテリアルのアルファ値を変更 (クラス参照先：Material - Properties)
  - 送信コマンド　：REND*TREE*<ノード階層>*MATERIAL*ALPHA SET <アルファ値>
  - 応答データ　：無し
    - ✧　<アルファ値> : 0～100 (%)

★MATERIAL_DEFINITION (VizRender)

- MD 用に画像をメモリにロード (クラス参照先：Pool - Commands)
  - 送信コマンド　：TEXTURE_IMAGE LOAD <Viz 画像パス>
  - 応答データ　：無し
- MD のテクスチャを変更 (クラス参照先：RenderSystemMaterial - Derived classes)
  - 送信コマンド　：REND*TREE*<ノード階層>*MATERIAL_DEFINITION*COLOR SET [<Viz 画像パス> | <外部画像パス>]
  - 応答データ　：無し

★コンテナの検索

▲目次▲

- コンテナ名での検索①　(クラス参照先：SCENETREE – Commands)

  送信コマンド　：REND*TREE SEARCH <取得範囲> <検索ワード>

  応答データ　：{<#オブジェクト ID>}

  <取得範囲>：作成した検索結果リストから、応答データで返すオブジェクト ID の範囲を指定します。

  ALL　：全てのコンテナ

  FIRST　：最初のコンテナ　　LAST　：最後のコンテナ

  NEXT　：次のコンテナ　　PREVIOUS：一つ前のコンテナ

  <検索ワード>：検索対象のワード。*(アスタリスク)で、あいまい検索が可能です。

  ✧ 検索ワード 例)　group　：コンテナ名が group の、すべてのコンテナを検索

  group　：名前のどこかに group が含まれる、すべてのコンテナを検索

  *group　：group で終わる、すべてのコンテナを検索

  group*　：group で始まる、すべてのコンテナを検索

  gr*oup　：gr で始まり、oup で終わる、すべてのコンテナを検索

  ✧ どの<検索対象>で検索を行っても、すべて含まれる検索結果リストを作成します。応答で返すオブジェクト ID の範囲が異なります。

  ✧ 作成された検索結果リストは、SEARCH_SELECT_ALL でも取得できます。

- プラグイン名での検索①　(クラス参照先：SCENETREE – Commands)

  送信コマンド　：REND*TREE SEARCH_PROPERTY <取得範囲> BUILT_IN*<プラグイン種類>
  BUILT_IN*<プラグイン種類>*<プラグイン名>

  応答データ　：{<#オブジェクト ID>}

  <取得範囲>：作成した検索結果リストから、応答データで返すオブジェクト ID の範囲を指定します。

  ALL　：全てのコンテナ

  FIRST　：最初のコンテナ　　LAST　：最後のコンテナ

  NEXT　：次のコンテナ　　PREVIOUS：一つ前のコンテナ

  <プラグイン種類>：　**FUNCTION**　：コンテナプラグイン

  GEOM　：ジオメトリ

  SHADER　：シェーダー

  ✧ 例)　ControlActionTable プラグインが適用されているコンテナを検索
  REND*TREE SEARCH_PROPERTY ALL BUILT_IN*FUNCTION BUILT_IN*FUNCTION*ControlActionTable

  ✧ 例)　Circle ジオメトリが適用されているコンテナを検索
  REND*TREE SEARCH_PROPERTY ALL BUILT_IN*GEOM BUILT_IN*GEOM*Circle

  ✧ どの<取得範囲>で検索を行っても、すべて含まれる検索結果リストを作成します。応答で返すオブジェクト ID の範囲が異なります。

  ✧ 作成された検索結果リストは、SEARCH_SELECT_ALL でも取得できます。

- 検索で一致したコンテナリストの取得　(クラス参照先：SCENETREE – Commands)

  送信コマンド　：REND*TREE SEARCH_SELECT_ALL

  応答データ　：{<#オブジェクト ID>}

  ✧ 最後の検索コマンドで取得した検索結果リストを再取得します。

  ✧ 検索コマンドの＜取得範囲＞に関わらず、<取得範囲>=ALL の検索結果と同じオブジェクト ID を取得します。

- 検索で一致した個数の取得　(クラス参照先：SCENETREE – Commands)

  送信コマンド　：REND*TREE GET_SEARCH_RESULT

  応答データ　：※最後の検索コマンドで一致した個数

- 検索結果のクリア　(クラス参照先：SCENETREE – Commands)

  送信コマンド　：REND*TREE RESET_RESULTS

  応答データ　：無し

- 「コンテナ名での検索①」で作成された検索結果リスト内から取得　（クラス参照先：SCENETREE – Commands）

  送信コマンド　：REND*TREE SEARCH_ONE_BY_PARAMETER <取得範囲> ""

  応答データ　：{<#オブジェクト ID>}

  <取得範囲>：以下のキーワードで応答するオブジェクト ID の範囲を指定します。

  ALL　：全てのコンテナ

  FIRST　：最初のコンテナ　　LAST　：最後のコンテナ

  NEXT　：次のコンテナ　　PREVIOUS：一つ前のコンテナ

  ✧ 「コンテナ名での検索①」の後で使用します。検索結果リストを作成せずにオブジェクト ID を取得します。

  ✧ 最後の""は必須です。

- コンテナ名での検索②　（クラス参照先：SCENETREE – Commands）

  送信コマンド　：REND*TREE SEARCH_ALL_BY_PARAMETER <検索ワード>

  応答データ　：<#オブジェクト ID>

  <検索ワード>：検索対象のワード。*(アスタリスク)で、あいまい検索が可能です。

  ✧ 検索ワード 例）　group　：コンテナ名が group の、すべてのコンテナを検索

  group　：名前のどこかに group が含まれる、すべてのコンテナを検索

  *group　：group で終わる、すべてのコンテナを検索

  group*　：group で始まる、すべてのコンテナを検索

  gr*oup　：gr で始まり、oup で終わる、すべてのコンテナを検索

  ✧ 「コンテナ名での検索①」の<検索対象>=ALL と同じ結果を取得できます。但し、応答データに{}は含まれません。

  ✧ 検索結果リストはリセットされるので、検索結果を SEARCH_SELECT_ALL や GET_SEARCH_RESULT で再取得できません。

- プラグイン名での検索②　（クラス参照先：SCENETREE – Commands）

  送信コマンド　：REND*TREE SEARCH_FOR_PROPERTY BUILT_IN*<プラグイン種類>

  BUILT_IN*<プラグイン種類>*<プラグイン名>

  応答データ　：<#オブジェクト ID>

  <プラグイン種類>：　**FUNCTION**　：コンテナプラグイン

  GEOM　：ジオメトリ

  SHADER　：シェーダー

  ✧ 例）　ControlActionTable プラグインが適用されているコンテナを検索

  REND*TREE SEARCH_FOR_PROPERTY BUILT_IN*FUNCTION BUILT_IN*FUNCTION*ControlActionTable

  ✧ 例）　Circle ジオメトリが適用されているコンテナを検索

  REND*TREE SEARCH_FOR_PROPERTY BUILT_IN*GEOM BUILT_IN*GEOM*Circle

  ✧ 「プラグイン名の検索①」の<取得範囲>=ALL と同じ結果を取得できます。但し、応答データに{}は含まれません。

  ✧ 検索結果リストはリセットされるので、検索結果を SEARCH_SELECT_ALL や GET_SEARCH_RESULT で再取得できません。

  ✧ 現在はマージオブジェクト内も検索する為、SEARCH_FOR_CONTAINER_WITH_PROPERTY と同じです。

- 非表示のコンテナの検索　（クラス参照先：SCENETREE – Commands）

  送信コマンド　：REND*TREE SEARCH_FOR_PROPERTY INACTIVE ""

  応答データ　：<#オブジェクト ID>

  ✧ 最後の""は省略可能です。

- Lock されたコンテナの検索　（クラス参照先：SCENETREE – Commands）

  送信コマンド　：REND*TREE SEARCH_FOR_PROPERTY LOCKED ""

  応答データ　：<#オブジェクト ID>

  ✧ 最後の""は省略可能です。

◎Animation 関連のコマンド

▲目次▲

★Stage 全体

- アニメーション表示の ON/OFF①を設定 （クラス参照先：RENDERER_BASE - Properties）

  送信コマンド　：REND*CONTENTS_VISIBLE SET <ON/OFF 値>

  応答データ　：無し

  - OFF(0)時、**アニメがバックグラウンドで実行されますが描画されず(Action も実行されず)、DirectorStatus も更新されません。**
  - OFF(0)時でも、コンテナ値を直接操作するコマンドは即時反映されますが、OFF にする前のコンテナ状態に対しての操作になります。
  - ON(1)にしたタイミングで、アニメ後の結果が描画され、DirectorStatus が更新され、Action コマンドが実行開始されます。
  - シーンに対してではなくレンダラーへの設定です。

- アニメーション表示の ON/OFF①を取得 （クラス参照先：RENDERER_BASE - Properties）

  送信コマンド　：REND*CONTENTS_VISIBLE GET

  応答データ　：＜ON/OFF 値＞

  - デフォルト値：1

- アニメーション表示の ON/OFF②を設定 （クラス参照先：RENDERER_BASE - Properties）

  送信コマンド　：REND*UPDATE SET <ON/OFF 値>

  応答データ　：無し

  - OFF(0)時、**アニメーションコマンドを送っても実行されません。**
  - OFF(0)時でも、コンテナ値を直接操作するコマンドは即時反映されますが、OFF にする前のコンテナ状態に対しての操作になります。
  - ON(1) にしたタイミングで、OFF 中に送ったコマンドが同時に実行を開始します。同じディレクターへのコマンドは、後に送った方が優先されます。
  - マニュアル記載は、「RENDERER の自動初期化設定」という機能です。
  - シーンに対してではなくレンダラーへの設定です。

- アニメーション表示の ON/OFF②を取得 （クラス参照先：RENDERER_BASE - Properties）

  送信コマンド　：REND*UPDATE GET

  応答データ　：＜ON/OFF 値＞

  - デフォルト値：1

- Stage の全データを取得 （クラス参照先：CStage – Properties）

  送信コマンド　：REND*STAGE*ANIMATION GET_DIRECTOR_DATA

  応答データ　：{Stage 構造データ}

  - ディレクター名・コンテナアニメ名・アクション名・オブジェクト ID・Time 値 Stage など全構造データを取得します。
  - **ACTION 内の KeyFrame 一覧(オブジェクト ID や個数)を取得するにはこれしかありません。**

- StageEditor の階層データを取得 （クラス参照先：CStage – Commands）

  送信コマンド　：REND*STAGE GET

  応答データ　：{StageEditor で表示される階層データ}

  - ディレクターとチャンネルの階層情報のみを取得します。
  - 全ディレクター・アクション・コンテナアニメーションの、階層・名前、オブジェクト ID、クラス名、いくつかのパラメータ
  - 応答データは StageEditor での階層の開き方の影響を受けます。

- StageEditor の並び順から Director のオブジェクト番号を取得 （クラス参照先：CStage – Properties）

  送信コマンド　：REND*STAGE*TREE*DIRECTOR GET <並び順>

  応答データ　：所属するディレクターの＜オブジェクト番号＞

  - Stage-Editor の並び順は 0 始まりです。
  - 応答データは StageEditor での階層の開き方の影響を受けます。

- StageEditor の並び順からコンテナのオブジェクト番号を取得　（クラス参照先：CStage – Properties）
  送信コマンド　：REND*STAGE*TREE*CONTAINER GET <並び順>
  応答データ　：所属するコンテナの＜オブジェクト番号＞
  - Stage-Editor の並び順は 0 始まりです。
  - 指定した並び順のアニメーションががコンテナアニメでは無い時は、-1 を返します。
  - 応答データは StageEditor での階層の開き方の影響を受けます。

- StageEditor 階層を全て開く　（クラス参照先：CStage – Commands）
  送信コマンド　：REND*STAGE OPEN_TREE
  応答データ　：無し
  - StageEditor TREE 操作系コマンドの前に実行します。

- StageEditor 階層を全て閉じる　（クラス参照先：CStage – Commands）
  送信コマンド　：REND*STAGE CLOSE_TREE
  応答データ　：無し

- Director の並びを移動　（クラス参照先：※COMMAND_INFO のみ）
  送信コマンド　：REND*STAGE*TREE MOVE_ENTRIES <移動元番号> <移動先番号> [BEFORE | AFTER]
  応答データ　：無し
  - <移動先番号> <移動元番号> : Stage 並び順（0～）
  - [BEFORE | AFTER]は、移動先の上下方向を指定します。省略時は移動先の子になります。
  - Editor モードでは使用できません。OA モードに切り替えてから使用します。
  - StageEditor での階層の開き方の影響を受けますので、使用には注意が必要です。
  - TREE 表示の変更は OPEN_TREE/CLOSE_TREE では変更されません。Stage Editor 上にて手動で行う必要があります。

★全 Director 用操作

▲目次▲

- 全 Director 名を取得　（クラス参照先：※COMMAND_INFO のみ）
  送信コマンド　：REND*STAGE*DIRECTOR GET
  応答データ　：最上位階層のディレクター名一覧
  - 応答データのディレクター名は、半角区切りです。

- 全 Director を開始　（クラス参照先：CStage – Commands）
  送信コマンド　：REND*STAGE START
  応答データ　：無し

- 全 Director を停止　（クラス参照先：CStage – Commands）
  送信コマンド　：REND*STAGE STOP
  応答データ　：無し
  - REND の代わりにロード済みシーンの絶対パスでバックグラウンド実行させた場合、STOP では無くリセットされます(SHOW 0 と同等)

- 全 Director を続行　（クラス参照先：CStage – Commands）
  送信コマンド　：REND*STAGE CONTINUE
  応答データ　：無し

- 全 Director を反転して続行　（クラス参照先：CStage – Commands）
  送信コマンド　：REND*STAGE CONTINUE REVERSE
  応答データ　：無し

- 全 Director を STOP ポイントで停止せずに開始 (クラス参照先：CStage – Commands)
  送信コマンド　：REND*STAGE START_NOSTOP
  応答データ　　：無し

- 全 Director を STOP ポイントで停止せずに続行 (クラス参照先：CStage – Commands)
  送信コマンド　：REND*STAGE CONTINUE_NOSTOP
  応答データ　　：無し

- 全 Director の停止位置を移動 (クラス参照先：CStage – Commands)
  送信コマンド　：REND*STAGE SHOW <Time 値>
  応答データ　　：無し

- 全 Director の停止位置を先頭に移動 (クラス参照先：CStage – Commands)
  送信コマンド②　：REND*STAGE SHOW f0
  応答データ　　：無し

★特定の Director 用操作 ▲目次▲

- Director を追加 (クラス参照先：CStage – Commands)
  送信コマンド　：REND*STAGE ADD <ディレクター名>
  応答データ　　：無し
  ✧　成功時、応答はありませんが、既に同じ名前の Director があれば、エラーが返ります。

- Director を開始 (クラス参照先：CDirector – Commands)
  送信コマンド　：REND*STAGE*$<ディレクター名> START
  応答データ　　：無し

- Director を停止 (クラス参照先：CDirector – Commands)
  送信コマンド　：REND*STAGE*$<ディレクター名> STOP
  応答データ　　：無し
  ✧　REND の代わりにロード済みシーンの絶対パスでバックグラウンド実行させた場合、STOP では無くリセットされます(SHOW 0 と同等)

- Director を続行 (クラス参照先：CDirector – Commands)
  送信コマンド　：REND*STAGE*$<ディレクター名> CONTINUE
  応答データ　　：無し

- Director を反転して続行 (クラス参照先：CDirector – Commands)
  送信コマンド　：REND*STAGE*$<ディレクター名> CONTINUE REVERSE
  応答データ　　：無し

- Director を STOP ポイントで停止せずに開始 (クラス参照先：CDirector – Commands)
  送信コマンド　：REND*STAGE*$<ディレクター名> START_NOSTOP
  応答データ　　：無し
- Director を STOP ポイントで停止せずに続行 (クラス参照先：CDirector – Commands)
  送信コマンド　：REND*STAGE*$<ディレクター名> CONTINUE_NOSTOP
  応答データ　　：無し

- Director の 2 つの STOP/TAG ポイント間を実行　（クラス参照先：CDirector – Commands）
  送信コマンド　：REND*STAGE*$<ディレクター名> GOTO $<開始名> $<終了名>
  応答データ　：無し
  ✧　ポイント名には $ が必要です。
- Director の現在位置(停止)を移動　（クラス参照先：CDirector – Commands）
  送信コマンド　：REND*STAGE*$<ディレクター名> SHOW <Time 値>
  応答データ　：無し
- Director の現在位置(停止)を先頭に移動　（クラス参照先：CDirector – Commands）
  送信コマンド　：REND*STAGE*$<ディレクター名> SHOW f0
  応答データ　：無し
- Director の現在位置(停止)を任意の STOP/TAG ポイントに移動　（クラス参照先：CDirector – Commands）
  送信コマンド　：REND*STAGE*$<ディレクター名> SHOW $<ポイント名>
  応答データ　：無し
  ✧　ポイント名には $ が必要です。
- Director にオフセット時間を設定　（クラス参照先：CDirector – Properties）
  送信コマンド　：REND*STAGE*$<ディレクター名>*OFFSET SET <Time 値>
  応答データ　：無し
- Director のデータを取得　（クラス参照先：CDirector – Properties）
  送信コマンド　：REND*STAGE*$<ディレクター名>*DATA GET
  応答データ　：※Director Editor Properties の設定値
  ✧　Director 自体のパラメータ情報を取得します。STOP ポイントや ACTION は含まれません。
- Director の Animation 終了時間を取得　（クラス参照先：CDirector – Properties）
  送信コマンド　：REND*STAGE*$<ディレクター名>*END_TIME GET
  応答データ　：※Director 内のアニメーション終了秒
- Director の Animation 開始時間を取得　（クラス参照先：CDirector – Properties）
  送信コマンド　：REND*STAGE*$<ディレクター名>*START_TIME GET
  応答データ　：<Time 値(秒)>
- Director ステータスデータを取得　（クラス参照先：CDirector – Commands）
  送信コマンド　：REND*STAGE*$<ディレクター名> GET_STATUS
  応答データ　：[RUNNING | STOP] <値 1>　<値 2>
  ✧　<値 1>：　NEXT が STOP の場合は STOP キーフレームの時間、NEXT に STOP が無い場合は 0
  ✧　<値 2>：　TIMELINE 値(経過時間(秒))
  応答データ 例)　動作中：RUNNING 0 2.285618952285619
  停止中：　STOP 0 8.058058058058059
- Director の現在位置を取得　（クラス参照先：CDirector – Commands）
  送信コマンド　：REND*STAGE*$<ディレクター名> GET_TIMELINE
  応答データ　：<Time 値(秒)>
- Director を削除　（クラス参照先：CStage – Commands）
  送信コマンド　：REND*STAGE DELETE_DIRECTOR REND*STAGE*$<ディレクター名>
  応答データ　：無し

★Animation - EVENT [STOP |TAG | PAUSE]ポイント

▲目次▲

- ポイントのデータを取得　（クラス参照先：CDirector – Properties）

  送信コマンド　：REND*STAGE*$<ディレクター名>*EVENT GET

  応答データ　：<オブジェクト番号> <Time 値(秒)> <種類> ・・・

  - <種類>：0:Stop、1:Tag、3:Pause、4:Next Scene
  - Local Stop は Stop が返ります。
  - オブジェクト番号には#は含まれません。
  - ポイント名の取得は、このコマンドで取得したオブジェクト番号を元に「オブジェクト ID からポイント名を取得」コマンドを使用します。

- STOP ポイントの個数を取得　（クラス参照先：CChannelEvent – Properties）

  送信コマンド　：REND*STAGE*$<ディレクター名>*EVENT_CHANNEL*STOPS GET_COUNT

  応答データ　：STOP ポイントの個数

- STOP ポイントのオブジェクト ID を取得　（クラス参照先：CChannelEvent – Properties）

  送信コマンド　：REND*STAGE*$<ディレクター名>*EVENT_CHANNEL*STOPS GET_ALL

  応答データ　：<#オブジェクト ID> ・・・

  - そのディレクターの全 STOP ポイントのオブジェクト ID が取得できます。

- オブジェクト ID からポイント名を取得　（クラス参照先：CKeyframeBase – Properties）

  送信コマンド　：<#オブジェクト ID>*NAME GET

  応答データ　：<ポイント名>

  - 取得したポイント名に、$ は含まれません。

- ポイント名からオブジェクト ID を取得　（クラス参照先：OBJECTBASE – Properties）

  送信コマンド　：REND*STAGE*$<ディレクター名>*KEY*$<ポイント名>*OBJECT_ID GET

  応答データ　：<#オブジェクト ID>

  - ポイント名には $ が必要です。

- ポイント名の変更　（クラス参照先：CKeyframeBase – Properties）

  送信コマンド①　：REND*STAGE*$<ディレクター名>*KEY*$<ポイント名>*NAME SET <ポイント名>

  送信コマンド②　：<#オブジェクト ID>*NAME SET <ポイント名>

  応答データ　：無し

  - 「STOP ポイントの追加」と合わせて使う場合は、送信コマンド②を使う方がコマンドが短くて済みます。
  - 変更前のポイント名には $ が必要ですが、SET する変更後のポイント名には$をつけてはいけません。

- ポイントの追加　（クラス参照先：CDirector – Commands）

  送信コマンド　：REND*STAGE*$<ディレクター名> ADD < Time 値> <種類>

  応答データ　：<#オブジェクト ID>

  - <種類>：[STOP | TAG | LOCAL_STOP]
  - ポイントの ADD コマンドでは、上記 3 種類しか対応していません。
    [PAUSE | NEXT_SCENE]を利用したい時は、応答で返るオブジェクト ID に対して、「ポイント種類の変更」コマンドを送信してください。
  - <種類>を省略した時は、STOP が追加されます。
  - 注) 上記 ADD コマンドの引数並びは公式ドキュメントと異なりますが、この並びでご使用ください。

- ポイントの移動　（クラス参照先：CKeyframeBase – Properties）
  送信コマンド①　：REND*STAGE*$<ディレクター名>*KEY*$<ポイント名>*TIME SET <Time 値>
  送信コマンド②　：<#オブジェクト ID>*TIME SET <Time 値>
  応答データ　　：無し
  - ポイント名には $ が必要です。

- ポイントの時間値を取得　（クラス参照先：CKeyframeBase – Properties）
  送信コマンド①　：REND*STAGE*$<ディレクター名>*KEY*$<ポイント名>*TIME GET
  送信コマンド②　：<#オブジェクト ID>*TIME GET
  応答データ　　：<Time 値(秒)>
  - ポイント名には $ が必要です。

- ポイントの種類の変更　（クラス参照先：CKeyframeEvent – Properties）
  送信コマンド①　：REND*STAGE*$<ディレクター名>*KEY*$<ポイント名>*VALUE SET ＜種類＞
  送信コマンド②　：<#オブジェクト ID>*VALUE SET ＜種類＞
  応答データ　　：無し
  - <種類>：[STOP | TAG | LOCAL_STOP | PAUSE | NEXT_SCENE]
  - ポイント名には $ が必要です。

- ポイントの種類の取得　（クラス参照先：CKeyframeEvent – Properties）
  送信コマンド①　：REND*STAGE*$<ディレクター名>*KEY*$<ポイント名>*VALUE GET
  送信コマンド②　：<#オブジェクト ID>*VALUE GET
  応答データ　　：＜種類＞
  - <種類>：[STOP | TAG | PAUSE | NEXT_SCENE]
  - LOCAL_STOP は STOP が返ります。
  - ポイント名には $ が必要です。

- PAUSE ポイントの長さを変更　（クラス参照先：CKeyframeEvent – Properties）
  送信コマンド①　：REND*STAGE*$<ディレクター名>*KEY*$<ポイント名>*PAUSE_LENGTH SET <Time 値>
  送信コマンド②　：<#オブジェクト ID>*PAUSE_LENGTH SET <Time 値>
  応答データ　　：<Time 値(秒)>
  - ポイント名には $ が必要です。

★Animation-ACTION (アクション-Channel コマンド)

▲目次▲

- Director の Action 名を取得 (クラス参照先：※COMMAND_INFO のみ)
  - 送信コマンド　：REND*STAGE*$<ディレクター名>*ACTION GET
  - 応答データ　：<ACTION 名>
- Director に Action を追加 (クラス参照先：※COMMAND_INFO のみ)
  - 送信コマンド　：REND*STAGE*$<ディレクター名>*ACTION ADD <アクション名>
  - 応答データ　：無し
- Director から Action を削除 (クラス参照先：※COMMAND_INFO のみ)
  - 送信コマンド　：REND*STAGE*$<ディレクター名>*ACTION DELETE <アクション名>
  - 応答データ　：無し
    - ✧ Action 自体を削除します。
- Action 名からオブジェクト ID を取得 (クラス参照先：OBJECTBASE – Properties)
  - 送信コマンド　：REND*STAGE*$<ディレクター名>*ACTION*<アクション名>*OBJECT_ID GET
  - 応答データ　：<#オブジェクト ID>
- Action のデータを取得 (クラス参照先：CChannelAction – Properties)
  - 送信コマンド　：REND*STAGE*$<ディレクター名>*ACTION*<アクション名>*DATA GET
  - 応答データ　：何かのデータ？
    - ✧ データ未解析ですが、Action 自体のパラメータ情報を取得します。Keyframe 情報は含まれません。

★Animation-ACTION① (アクション-Keyframe コマンド)

▲目次▲

- ➢ <キーフレーム指定>：キーフレームの指定方法は 2 通りあり
  - ● KEY*<キーフレーム名>　：キーフレーム名の $ は付けても付けなくても、どちらでも構いません
  - ● KEYN*<キーフレーム番号>　：0～。番号がキーフレームの範囲外や、キーフレームが無い時は、コマンドエラー

- Acttion-Keyframe を追加 (クラス参照先：CChannelBase – Commands)
  - 送信コマンド①　：REND*STAGE*$<ディレクター名>*ACTION*<アクション名> ADD <Time 値>
  - 送信コマンド②　：<アクションの#オブジェクト ID> ADD <Time 値>
  - 応答データ　：<キーフレームの#オブジェクト ID>
- 現在時間とキーフレーム値で Action-Keyframe を追加 (クラス参照先：CChannelBase – Commands)
  - 送信コマンド①　：REND*STAGE*$<ディレクター名>*ACTION*<アクション名> ADD_VALUE <Time 値> <キーフレーム値>
  - 送信コマンド②　：<アクションの#オブジェクト ID> ADD_VALUE <Time 値> <キーフレーム値>
  - 応答データ　：<キーフレームの#オブジェクト ID>
    - ✧ <Time 値>はなんでも良いです。例えば、0。何の値を指定しても現在の TimeLine の場所に Keyframe が追加されます。
- 全 Action-Keyframe を削除 (クラス参照先：CChannelBase – Properties)
  - 送信コマンド　：REND*STAGE*$<ディレクター名>*ACTION*<アクション名> DELETE
  - 応答データ　：<#オブジェクト ID>
    - ✧ Action 内の全 Action-Keyframe は削除されますが、Action 自体は残ります。
- Action-Keyframe 名で Action-Keyframe の名前を変更 (クラス参照先：CKeyframeBase – Properties)
  - 送信コマンド　：REND*STAGE*$<ディレクター名>*ACTION*<アクション名>*<キーフレーム指定>*NAME SET <キーフレーム名>
  - 応答データ　：無し
    - ✧ キーフレーム指定(キーフレーム名)の $ の有無は問いませんが、SET する変更後のキーフレーム名には$をつけてはいけません。

★Animation-ACTION② (アクション-Keyframe コマンド)

▲目次▲

- <キーフレーム指定>：キーフレームの指定方法は 2 通りあり
  - KEY*<キーフレーム名>　　：キーフレーム名の $ は付けても付けなくても、どちらでも構いません
  - KEYN*<キーフレーム番号>　　：0～。番号がキーフレームの範囲外や、キーフレームが無い時は、コマンドエラー

- オブジェクト ID で Action-Keyframe の名前を変更　(クラス参照先：CKeyframeBase – Properties)
  - 送信コマンド　：<#オブジェクト ID>*NAME SET <キーフレーム名>
  - 応答データ　：無し
  - ✧　「キーフレームを追加した後にキーフレームに名前を付ける」 などに使用します。

- Keyframe 番号で Action-Keyframe の名前を変更　(クラス参照先：CKeyframeBase – Properties)
  - 送信コマンド　：REND*STAGE*$<ディレクター名>*ACTION*<アクション名>*<キーフレーム指定>*NAME SET <キーフレーム名>
  - 応答データ　：無し

- Action-Keyframe のオブジェクト ID を取得　(クラス参照先：OBJECTBASE – Properties)
  - 送信コマンド　：REND*STAGE*$<ディレクター名>*ACTION*<アクション名>*<キーフレーム指定>*OBJECT_ID GET
  - 応答データ　：<#オブジェクト ID>

- オブジェクト ID から Action-Keyframe 名を取得　(クラス参照先：CKeyframeBase – Properties)
  - 送信コマンド　：<#オブジェクト ID>*NAME GET
  - 応答データ　：<キーフレーム名>

- Action-Keyframe を削除　(クラス参照先：CKeyframeBase – Commands)
  - 送信コマンド①　：REND*STAGE*$<ディレクター名>*ACTION*<アクション名>*<キーフレーム指定> DELETE
  - 送信コマンド②　：<#オブジェクト ID> DELETE
  - 応答データ　：無し

- Action-Keyframe の時間を変更　(クラス参照先：CKeyframeBase – Properties)
  - 送信コマンド①　：REND*STAGE*$<ディレクター名>*ACTION*<アクション名>*<キーフレーム指定>*TIME SET <Time 値>
  - 送信コマンド②　：<#オブジェクト ID>*TIME SET <Time 値>
  - 応答データ　：無し
  - ✧　キーフレーム名の $ は付けても付けなくても、どちらでも構いません。

- Action-Keyframe の時間を取得　(クラス参照先：CKeyframeBase – Properties)
  - 送信コマンド①　：REND*STAGE*$<ディレクター名>*ACTION*<アクション名>*<キーフレーム指定>*TIME GET
  - 送信コマンド②　：<#オブジェクト ID>*TIME GET
  - 応答データ　：<Time 値(秒)>
  - ✧　キーフレーム名の $ は付けても付けなくても、どちらでも構いません。

- Action-Keyframe の値(コマンド)を変更　(クラス参照先：CKeyframeAction – Properties)
  - 送信コマンド①　：REND*STAGE*$<ディレクター名>*ACTION*<アクション名>*<キーフレーム指定>*VALUE SET <キーフレーム値>
  - 送信コマンド②　：<#オブジェクト ID>*VALUE SET <キーフレーム値>
  - 応答データ　：無し
  - ✧　キーフレーム名の $ は付けても付けなくても、どちらでも構いません。
  - ✧　<キーフレーム値>にコマンドを設定すると、指定した時間でコマンド(もしくはタスク)が実行されます。
  - ✧　コマンドを入力する際は、コマンドの REND の部分を THIS_SCENE、もしくは、MAIN_SCENE にしてください。

★Animation-ACTION③ (アクション-Keyframe コマンド)

▲目次▲

- <キーフレーム指定>：キーフレームの指定方法は 2 通り
  - KEY*<キーフレーム名>　　：キーフレーム名の $ は付けても付けなくても、どちらでも構いません
  - KEYN*<キーフレーム番号>　　：0～。番号がキーフレームの範囲外や、キーフレームが無い時は、コマンドエラー

- Action-Keyframe の値(コマンド)を取得　(クラス参照先：CKeyframeAction – Properties)

  送信コマンド①　：REND*STAGE*$<ディレクター名>*ACTION*<アクション名>*<キーフレーム指定>*VALUE GET
  送信コマンド②　：<#オブジェクト ID>*VALUE GET
  応答データ　　：<キーフレーム値>

- Action-Keyframe の進行方向を変更　(クラス参照先：CKeyframeAction – Properties)

  送信コマンド①　：REND*STAGE*$<ディレクター名>*ACTION*<アクション名>*<キーフレーム指定>*DIRECTION SET <方向値>
  送信コマンド②　：<#オブジェクト ID>*DIRECTION SET <方向値>
  応答データ　　：無し
  - ✧　<方向値>　：[NORMAL | REVERSE | BOTH]

- Action-KeyFrame の進行方向を取得　(クラス参照先：CKeyframeAction – Properties)

  送信コマンド①　：REND*STAGE*$<ディレクター名>*ACTION*<アクション名>*<キーフレーム指定>*DIRECTION GET
  送信コマンド②　：<#オブジェクト ID>*DIRECTION GET
  応答データ　　：[NORMAL | REVERSE | BOTH]
  - ✧　ACTION がこの DIRECTON 方向に実行されている時のみ、KeyFrame 内のコマンドが実行されます。
  - ✧　デフォルト値：NORMAL

- Action-KeyFrame の動作モードを変更　(クラス参照先：CKeyframeAction – Properties)

  送信コマンド①　：REND*STAGE*$<ディレクター名>*ACTION*<アクション名>*<キーフレーム指定>*MODE SET <MODE 値>
  送信コマンド②　：<#オブジェクト ID>*MODE SET <MODE 値>
  応答データ　　：無し
  - ✧　<MODE 値> : [COMMAND | TASK]
  - ✧　COMMAND　: vizrt 外部制御コマンド [Default]
  - ✧　TASK　　: Windows コマンド

- Action-KeyFrame の動作モードを取得　(クラス参照先：CKeyframeAction – Properties)

  送信コマンド①　：REND*STAGE*$<ディレクター名>*ACTION*<アクション名>*<キーフレーム指定>*MODE GET
  送信コマンド②　：<#オブジェクト ID>*MODE GET
  応答データ　　：[COMMAND | TASK]

★Animation-CHANNEL (オブジェクトアニメコマンド)

▲目次▲

- 特定コンテナ アニメを特定のディレクターに集約・移動① (クラス参照先：CDirector – Properties)
  - 送信コマンド① ：REND*STAGE*$<ディレクター名>*OBJECT SET REND*TREE*<ノード階層>
  - 送信コマンド② ：REND*STAGE*$<ディレクター名>*OBJECT SET <ノード階層・オブジェクト ID>
  - 応答データ ：無し
    - ✧ <ノード階層>のコンテナアニメが<ディレクター名>ディレクターに移動します。
    - ✧ もし複数のディレクターに分散している同一<ノード階層>のコンテナアニメが、全て<ディレクター名>ディレクターに移動されます。
- 特定コンテナ アニメを特定のディレクターに集約・移動② (クラス参照先：CONTAINER – Properties)
  - 送信コマンド① ：REND*TREE*<ノード階層>*ANIM_CHANNELS*ALL MOVE_TO_DIRECTOR REND*STAGE*$<ディレクター名>
  - 送信コマンド② ：REND*TREE*<ノード階層>*ANIM_CHANNELS*ALL MOVE_TO_DIRECTOR <ディレクター・オブジェクト ID>
  - 応答データ ：無し
    - ✧ <ノード階層>のコンテナアニメが<ディレクター名>ディレクターに移動します。
    - ✧ もし複数のディレクターに分散している同一<ノード階層>のコンテナアニメが、全て<ディレクター名>ディレクターに移動されます。
- 特定コンテナの特定チャンネルアニメを削除 (クラス参照先：CChannelBase – Commands)
  - 送信コマンド① ：REND*TREE*<ノード階層>*ANIMATION*<チャンネル名> DELETE
  - 送信コマンド② ：<#オブジェクト ID> DELETE
  - 応答データ ：無し
    - ✧ 特定コンテナの特定チャンネルにある KeyFrame は全て消えますが、チャンネル自体は残ります。
- 特定コンテナの全アニメを削除 (クラス参照先：※COMMAND_INFO のみ)
  - 送信コマンド① ：REND*TREE*<ノード階層>*ANIMATION DELETE
  - 送信コマンド② ：<#オブジェクト ID>*ANIMATION DELETE
  - 応答データ ：無し
    - ✧ 特定コンテナのチャンネルを含む全てのアニメが削除されます。
- 特定コンテナのアニメオフセットを変更 (クラス参照先：CAnimContainer – Properties)
  - 送信コマンド ：REND*TREE*<ノード階層>*ANIMATION*OFFSET SET <TIME 値>
  - 応答データ ：無し
- 特定コンテナ アニメのループを設定 (クラス参照先：CChannelBase – Properties)
  - 送信コマンド① ：REND*TREE*<ノード階層>*ANIMATION*<チャンネル名>*POST_LOOP SET <ON/OFF 値>
  - 送信コマンド② ：REND*TREE*<ノード階層>*ANIMATION*<チャンネル名>*LOOP SET <ON/OFF 値>
  - 応答データ ：無し
- 特定コンテナ アニメのスイングを設定 (クラス参照先：CChannelBase – Properties)
  - 送信コマンド① ：REND*TREE*<ノード階層>*ANIMATION*<チャンネル名>*POST_SWING SET <ON/OFF 値>
  - 送信コマンド② ：REND*TREE*<ノード階層>*ANIMATION*<チャンネル名>* SWING SET <ON/OFF 値>
  - 応答データ ：無し
- 特定コンテナ アニメのループ回数を設定 (クラス参照先：CChannelBase – Properties)
  - 送信コマンド① ：REND*TREE*<ノード階層>*ANIMATION*<チャンネル名>*POST_LOOP_COUNTER SET <ループ回数>
  - 送信コマンド② ：REND*TREE*<ノード階層>*ANIMATION*<チャンネル名>*LOOP_COUNTER SET <ループ回数>
  - 応答データ ：無し

- 特定コンテナ アニメの無限ループを設定　（クラス参照先：CChannelBase – Properties）

  送信コマンド①　：REND*TREE*<ノード階層>*ANIMATION*<チャンネル名>*POST_LOOP_INFINITE SET <ON/OFF 値>

  応答データ　　：無し

★Animation-CHANNEL (オブジェクトアニメ-Keyframe コマンド)

▲目次▲

- <キーフレーム指定>：キーフレームの指定方法は 2 通り
  - KEY*<キーフレーム名>　　：キーフレーム名の $ は付けても付けなくても、どちらでも構いません
  - KEYN*<キーフレーム番号>　　：0～。番号がキーフレームの範囲外や、キーフレームが無い時は、コマンドエラー

- Keyframe 名からオブジェクト ID を取得　（クラス参照先：OBJECTBASE – Properties）

  送信コマンド　：REND*TREE*<ノード階層>*ANIMATION*<キーフレーム指定>*OBJECT_ID GET

  応答データ　　：<#オブジェクト ID>

- Keyframe 名を変更　（クラス参照先：CKeyframeBase – Properties）

  送信コマンド①　：REND*TREE*<ノード階層>*ANIMATION*<チャンネル名>*<キーフレーム指定>*NAME　SET <新キーフレーム名>

  送信コマンド②　：<#オブジェクト ID>*NAME SET <新キーフレーム名>

  応答データ　　：無し

  - <チャンネル名>：Stage TREE に表示されるアニメーション種類名。(Position | Rotation | Scaling | Active | Alpha など]

- KeyFrame 名で KeyFrame の時間を変更　（クラス参照先：CKeyframeBase – Properties）

  送信コマンド①　：REND*TREE*<ノード階層>*ANIMATION*<チャンネル名>*<キーフレーム指定>*TIME SET <Time 値>

  送信コマンド②　：<#オブジェクト ID>*TIME SET <Time 値>

  応答データ　　：無し

- KeyFrame 名で KeyFrame の値を変更　（クラス参照先：CKeyframeEvent – Properties）

  送信コマンド①　：REND*TREE*<ノード階層>*ANIMATION*<チャンネル名>*<キーフレーム指定>*VALUE SET <キーフレーム値>

  送信コマンド②　：<#オブジェクト ID>*VALUE SET <キーフレーム値>

  応答データ　　：無し

- KeyFrame 名で KeyFrame を削除　（クラス参照先：CKeyframeBase – Commands）

  送信コマンド①　：REND*TREE*<ノード階層>*ANIMATION*<チャンネル名>*<キーフレーム指定> DELETE

  送信コマンド②　：<#オブジェクト ID> DELETE

  応答データ　　：無し

- [コンテナ] アニメに Keyframe を時間指定で追加①　（クラス参照先：CChannelBase – Commands）

  送信コマンド　：REND*TREE*<ノード階層>*ANIMATION*<チャンネル名> ADD <Time 値>

  応答データ　　：<#オブジェクト ID>

  - <チャンネル名>：Stage TREE に表示されるアニメーション種類名。(Position | Rotation | Scaling | Active | Alpha など]
  - 新規に作成された場合は、Stage TREE の先頭ディレクターに作成されます。
  - Stage TREE が空の場合は、Default ディレクターが作成されます。
  - 値は、直近の値が引き継がれます。
    - 例)　Circle コンテナ Position アニメの TimeLine(f500)に KeyFrame を追加する

      REND*TREE*$Circle1*ANIMATION*Position ADD f500
    - 例)　Circle コンテナ Apha アニメの TimeLine(f500)に KeyFrame を追加する

      REND*TREE*$Circle2*ANIMATION*Alpha ADD f500

- [コンテナ] アニメに Keyframe を時間指定で追加②　（クラス参照先：※COMMAND_INFO のみ）

  送信コマンド　：REND*TREE*<ノード階層>*ANIMATION*KEY UPDATE <Time 値>

  応答データ　　：無し

  - ✧ コンテナの設定コマンドと共に、例えば、以下のように 2 つのコマンドを使用します。
    - ● 例） Circle コンテナ位置アニメの TimeLine(f600)に KeyFrame を追加する

      1. REND*TREE*$Circle1*TRANSFORMATION*POSITION SET 0 0 0
      2. REND*TREE*$Circle1*ANIMATION*KEY UPDATE f600

- [コンテナ] アニメに Keyframe を現在時間で追加　（クラス参照先：CChannelBase – Commands）

  送信コマンド　：REND*TREE*<ノード階層>*ANIMATION*<チャンネル名> ADD_KEYFRAME <キーフレーム名>

  応答データ　　：<#オブジェクト ID>

  - ✧ TimeLine の指定が現在時間になる以外は、上記「コンテナ アニメにキーフレームを時間指定で追加①」と同じです。
  - ✧ <キーフレーム名>は、無ければキーフレーム名がつかないだけで、無くても構いません。
    - ● 例） Circle コンテナ位置アニメの現在の TimeLine に KeyFrame を追加する

      REND*TREE*$Circle*ANIMATION*Position ADD_KEYFRAME

- [ジオメトリ｜ファンクション] アニメに Keyframe を時間指定で追加　（クラス参照先：CChannelBase – Commands）

  送信コマンド　：REND*TREE*<ノード階層>*ANIMATION*<パラメータ名> ADD <Time 値>

  応答データ　　：<#オブジェクト ID>

  - ✧ <パラメータ名>：各ジオメトリパラメータ名。パラメータによって大文字小文字があります。

    Properties パネルにあるパラメータタイトルをドラッグして表示される Tips で確認してください。

    もしくは、手動で Keyframe を追加し、Stage TREE で確認する方法もあります。
  - ✧ コンテナアニメーションと異なり、新規に作成された場合、つまり、事前にチャンネルが無い場合は、エラーになります。
  - ✧ 値は、直近の値が引き継がれます。
    - ● 例） Circle radius アニメの TimeLine(f600)に KeyFrame を追加する

      REND*TREE*$Circle*ANIMATION*radius ADD f600

- [ジオメトリ｜ファンクション] アニメに Keyframe を現在時間で追加　（クラス参照先：CChannelBase – Commands）

  送信コマンド　：REND*TREE*<ノード階層>*ANIMATION*< パラメータ名> ADD_KEYFRAME <キーフレーム名>

  応答データ　　：<#オブジェクト ID>

  - ✧ TimeLine の指定が現在時間になる以外は、上記「ジオメトリ|ファンクションアニメにキーフレームを時間指定で追加」と同じです。
  - ✧ <キーフレーム名>は、無ければキーフレーム名がつかないだけで、無くても構いません。
    - ● 例） Omo アニメの現在の TimeLine に KeyFrame を追加する

      REND*TREE*$group*ANIMATION*vis_con ADD_KEYFRAME

- [ジオメトリ] アニメに Keyframe を時間指定で追加　（クラス参照先：CChannelBase – Commands）

  送信コマンド　：REND*TREE*<ノード階層>*GEOM*ANIMATION*<パラメータ名> ADD <Time 値>

  応答データ　　：<#オブジェクト ID>

  - ✧ <パラメータ名>：各ジオメトリパラメータ名。パラメータによって大文字小文字があります。

    Properties パネルにあるパラメータタイトルをドラッグして表示される Tips で確認してください。

    もしくは、手動で Keyframe を追加し、Stage TREE で確認する方法もあります。
  - ✧ 新規に作成された場合は、Stage TREE の先頭ディレクターに作成されます。
  - ✧ Stage TREE が空の場合は、Default ディレクターが作成されます。
  - ✧ 値は、直近の値が引き継がれます。
    - ● 例） Circle radius アニメの TimeLine(f200)に KeyFrame を追加する

      REND*TREE*$Circle*GEOM*ANIMATION*radius ADD f200

- [ジオメトリ] アニメに Keyframe を現在時間で追加　(クラス参照先：CChannelBase – Commands)

  送信コマンド　：REND*TREE*<ノード階層>*GEOM*ANIMATION*< パラメータ名> ADD_KEYFRAME <キーフレーム名>

  応答データ　：<#オブジェクト ID>

  - ✧ TimeLine の指定が現在時間になる以外は、上記「ジオメトリ アニメに KeyFrame を「時間指定」追加」と同じです。
  - ✧ <キーフレーム名>は、無ければキーフレーム名がつかないだけで、無くても構いません。
    - ● 例)　Omo アニメの現在の TimeLine に KeyFrame を追加する

      REND*TREE*$Circle*GEOM*ANIMATION*radius ADD_KEYFRAME aiueo

- [ファンクション] アニメに Keyframe を「時間指定」追加　(クラス参照先：CChannelBase – Commands)

  送信コマンド　：REND*TREE*<ノード階層>*FUNCTION*<プラグイン名>*ANIMATION*<パラメータ名> ADD <Time 値>

  応答データ　：<#オブジェクト ID>

  - ✧ <パラメータ名>：各ジオメトリパラメータ名。パラメータによって大文字小文字があります。

    Properties パネルにあるパラメータタイトルをドラッグして表示される Tips で確認してください。

    もしくは、手動で Keyframe を追加し、Stage TREE で確認する方法もあります。
  - ✧ 新規に作成された場合は、Stage TREE の先頭ディレクターに作成されます。
  - ✧ Stage TREE が空の場合は、Default ディレクターが作成されます。
  - ✧ 値は、直近の値が引き継がれます。
    - ● 例)　Omo アニメの TimeLine(f200)に KeyFrame を追加する

      REND*TREE*$group*FUNCTION*Omo*ANIMATION*vis_con ADD f200

- [ファンクション] アニメに Keyframe を現在時間で追加　(クラス参照先：CChannelBase – Commands)

  送信コマンド：REND*TREE*<ノード階層>*FUNCTION*<プラグイン名>*ANIMATION*<パラメータ名> ADD_KEYFRAME <キーフレーム名>

  応答データ　：<#オブジェクト ID>

  - ✧ TimeLine の指定が現在時間になる以外は、上記「ファンクション アニメに KeyFrame を「時間指定」追加」と同じです。
  - ✧ <キーフレーム名>は、無ければキーフレーム名がつかないだけで、無くても構いません。
    - ● 例)　Omo アニメの現在の TimeLine に KeyFrame を追加する

      REND*TREE*$group*FUNCTION*Omo*ANIMATION*vis_con ADD_KEYFRAME aiueo

◎vizrt プラグイン関連のコマンド

▲目次▲

➢ 「付録③：用語解説や補足 – <プラグイン名>・<プラグイン変数>の調べ方」も参照してください。

★Scene Plugins

- プラグイン名の取得 （クラス参照先：SCENE - Properties）
  - 送信コマンド　：REND*FUNCTION GET
  - 応答データ　：FUNCTION*<プラグイン名>
    - ✧ 現在のシーンに適用されているプラグイン名を取得。”FUNCTION*”も付いてきます。
- プラグイン変数の取得 （クラス参照先：※COMMAND_INFO のみ）
  - 送信コマンド　：REND*FUNCTION*<プラグイン名>*<プラグイン変数> GET
  - 応答データ　：<設定値>
- プラグイン変数の変更 （クラス参照先：※COMMAND_INFO のみ）
  - 送信コマンド　：REND*FUNCTION*<プラグイン名>*<プラグイン変数> SET <設定値>
  - 応答データ　：無し
- ボタンの実行 （クラス参照先：※COMMAND_INFO のみ）
  - 送信コマンド　：REND*FUNCTION*<プラグイン名>*<プラグイン変数> INVOKE
  - 応答データ　：無し
- プラグイン構造の取得 （クラス参照先：CPluginInstance - Properties）
  - 送信コマンド　：REND*FUNCTION*<プラグイン名>*DYNAMIC_PARAMETER GET
  - 応答データ　：※プラグイン変数名、表示タイトル名、種類、など

★Container Plugins

- プラグイン名の取得 （クラス参照先：CONTAINER - Properties）
  - 送信コマンド　：REND*TREE*<ノード階層>*FUNCTION GET
  - 応答データ　：FUNCTION*<プラグイン名>
    - ✧ 現在のシーンに適用されているプラグイン名を取得。”FUNCTION*”も付いてきます。
- プラグイン変数の取得 （クラス参照先：※COMMAND_INFO のみ）
  - 送信コマンド　：REND*TREE*<ノード階層>*FUNCTION*<プラグイン名>*<プラグイン変数> GET
  - 応答データ　：<設定値>
- プラグイン変数の変更 （クラス参照先：※COMMAND_INFO のみ）
  - 送信コマンド　：REND*TREE*<ノード階層>*FUNCTION*<プラグイン名>*<プラグイン変数> SET <値>
  - 応答データ　：無し
- ボタンの実行 （クラス参照先：※COMMAND_INFO のみ）
  - 送信コマンド　：REND*TREE*<ノード階層>*FUNCTION*<プラグイン名>*<プラグイン変数> INVOKE
  - 応答データ　：無し
- プラグイン構造の取得 （クラス参照先：CPluginInstance - Properties）
  - 送信コマンド　：REND*TREE*<ノード階層>*FUNCTION*<プラグイン名>*DYNAMIC_PARAMETER GET
  - 応答データ　：※プラグイン変数名、表示タイトル名、種類、など

★Geometry Plugins

- プラグイン変数の取得　（クラス参照先：※COMMAND_INFO のみ）
  - 送信コマンド　：REND*TREE*<ノード階層>*GEOM*<プラグイン変数> GET
  - 応答データ　：<設定値>
    - ✧ Container には 1 つの Geometry しか配置できない為、Geometry Plugin 名は不要です。
- プラグイン変数の変更　（クラス参照先：※COMMAND_INFO のみ）
  - 送信コマンド　：REND*TREE*<ノード階層>*GEOM*<プラグイン変数> SET <設定値>
  - 応答データ　：無し
    - ✧ Container には 1 つの Geometry しか配置できない為、Geometry Plugin 名は不要です。

★BUILT_IN Plugins

▲目次▲

- Scene/Container Plugins プラグイン変数情報の取得　（クラス参照先：Plugin - Properties）
  - 送信コマンド　：BUILT_IN*FUNCTION*<プラグイン名>*PARAMETER GET
  - 応答データ　：※プラグインの各パラメータ情報
- Geometry Plugins プラグイン変数情報の取得　（クラス参照先：Plugin - Properties）
  - 送信コマンド　：BUILT_IN*GEOM*<プラグイン名>*PARAMETER GET
  - 応答データ　：※プラグインの各パラメータ情報
- Scene/Container Plugins プラグインの存在確認　（クラス参照先：CPluginManager - Commands）
  - 送信コマンド　：BUILT_IN PLUGIN_EXISTS BUILT_IN*FUNCTION*<プラグイン名>
  - 応答データ　：<ON/OFF 値>
- Geometry Plugins プラグインの存在確認　（クラス参照先：CPluginManager - Commands）
  - 送信コマンド　：BUILT_IN PLUGIN_EXISTS BUILT_IN*GEOM*<プラグイン名>
  - 応答データ　：<ON/OFF 値>
- BUILT IN プラグイン一覧情報の取得　（クラス参照先：CPluginManager - Commands）
  - 送信コマンド　：BUILT_IN GET_PLUGINS
  - 応答データ　：※BUILTIN プラグイン一覧

★AUDIO Plugin

▲目次▲

- <キーフレーム指定>：キーフレームの指定方法は２通りあります
  - KEY*<キーフレーム名>　　：キーフレーム名の＄は付けても付けなくても、どちらでも構いません
  - KEYN*<キーフレーム番号>　　：0～。番号がキーフレームの範囲外や、キーフレームが無い時は、コマンドエラー

- Audio Plugin 設定方法

  ① SceneTree に空のコンテナを追加し Audio Plugin を付与
  ② 音を走らせたい Director へ①のコンテナをドラッグ＆ドロップ
  ③ Director に追加されたコンテナ名を右クリックし、[Audio]-->[Clip]を選択
  ④ 出現したタイムラインを選択しプロパティエディタを表示する
  ⑤ Clip 枠へ AssetView からインポート済みの Audio ファイルをドラッグ＆ドロップ
  　※AssetView ウインドウをフローティングにする必要有
  　※「Browse...」ボタンで、ローカルの Audio ファイルを選択することも可能
  ⑥　音の開始や停止は、②で作成した Director を操作する

- Audio Plugin をコマンドで使用する際の注意点
  - 上記③の AudioClip チャンネルの追加しただけの場合、下記コマンドの<キーフレーム指定>は、**KEYN*0** を使用
    これは、上記④で出現したタイムラインの設定先全体がキーフレームであり、１つしか追加されない為
  - 上記③の AudioClip チャンネルの追加は複数行えて、ディレクターの開始・停止で複数の音源を一括して操作できる。
    但し、コマンドでは最初に追加した AudioClip しか操作できない。
    コマンド操作がある場合は、１つの AudioPlugin に１つの音源が良い。
  - 上記③の AudioClip チャンネルにキーフレームを追加することで複数の音源を直列に再生することができる。
    コマンドでの操作もキーフレーム番号を変更することで操作可能。
    しかし、管理が煩雑になる恐れもあり、異なる開始タイミングは Director・ACTION で分けることが望ましい。

- AUDIO CLIP の再生　（クラス参照先：CVKeyframeAudioClip - Properties）
  送信コマンド　：REND*TREE*<ノード階層>*ANIMATION*AudioClip*<キーフレーム指定>*CLIP PLAY
  応答データ　：無し
  - 通常は、ディレクターを操作してください。

- AUDIO CLIP の停止　（クラス参照先：CVKeyframeAudioClip - Properties）
  送信コマンド　：REND*TREE*<ノード階層>*ANIMATION*AudioClip*<キーフレーム指定>*CLIP STOP
  応答データ　：無し
  - 通常は、ディレクターを操作してください。

- AUDIO CLIP のロケーションパスの取得　（クラス参照先：CPoolObj - Properties）
  送信コマンド　：REND*TREE*<ノード階層>*ANIMATION*AudioClip*<キーフレーム指定>*LOCATION_PATH GET
  応答データ　：音声ファイルのロケーションパス（“AUDIO_CLIP*”付）
  - ローカルの Audio ファイルを選択した時の応答データは、AUDIO_CLIP*<#オブジェクト ID>

- AUDIO CLIP 名の取得　（クラス参照先：CPoolObj - Properties）
  送信コマンド　：REND*TREE*<ノード階層>*ANIMATION*AudioClip*<キーフレーム指定>*CLIP*NAME GET
  応答データ　：音声ファイル名
  - この応答データをファイル名として EXPORT で使用
  - ローカルの Audio ファイルを選択した時の応答データは、<#オブジェクト ID>

- AUDIO CLIP の EXPORT （クラス参照先：CVAudioClip - Commands）
  送信コマンド　：REND*TREE*<ノード階層>*ANIMATION*AudioClip*<キーフレーム指定>*CLIP EXPORT <出力先パス>
  応答データ　：無し
  - ✧ <出力先パス>：ローカルディレクトリとファイル名
  - ✧ 拡張子の設定は不要です。元データが wav 形式の場合、CLIP 名の wav ファイルが出力されます。
  - ✧ wav・ogg・mp3 に対応ですが、ogg はインポート時に wav に変換されます。
- AUDIO CLIP の開始時間の取得 （クラス参照先：CVKeyframeAudioClip - Properties）
  送信コマンド　：REND*TREE*<ノード階層>*ANIMATION*AudioClip*<キーフレーム指定>*START GET
  応答データ　：プロパティエディタの Clip In 時間
  - ✧ SET・ADD あり
- AUDIO CLIP の終了時間の取得 （クラス参照先：CVKeyframeAudioClip - Properties）
  送信コマンド　：REND*TREE*<ノード階層>*ANIMATION*AudioClip*<キーフレーム指定>*END GET
  応答データ　：プロパティエディタの Clip Out 時間
  - ✧ SET・ADD あり
- AUDIO CLIP 長の取得 （クラス参照先：CVKeyframeAudioClip - Properties）
  送信コマンド　：REND*TREE*<ノード階層>*ANIMATION*AudioClip*<キーフレーム指定>*DURATION GET
  応答データ　：プロパティエディタの Duration 時間
- [テスト用] AUDIO CLIP の再生 （クラス参照先：CVAudio - Commands）
  送信コマンド　：REND*TREE*<ノード階層>*AUDIO*TEST_CLIP PLAY
  応答データ　：無し
  - ✧ 将来、削除されるかもしれません。
- [テスト用] AUDIO CLIP のテスト停止 （クラス参照先：CVAudio - Commands）
  送信コマンド　：REND*TREE*<ノード階層>*AUDIO*TEST_CLIP STOP
  応答データ　：無し
  - ✧ 将来、削除されるかもしれません。
- [テスト用] AUDIO CLIP のテスト用設定 （クラス参照先：CVAudio - Commands）
  送信コマンド　：REND*TREE*<ノード階層>*AUDIO*TEST_CLIP SET AUDIO_CLIP*<AUDIO CLIP パス>
  応答データ　：無し
  - ✧ 将来、削除されるかもしれません。
- [AssetPool] AUDIO CLIP の IMPORT （クラス参照先：CVAudioClipPool - Commands）
  送信コマンド　：AUDIO_CLIP IMPORT <Asset パス名> <AUDIO CLIP 名> <音声データフルパス>
  応答データ　：無し
  - ✧ <Asset パス名> <AUDIO CLIP 名> <音声データフルパス>を半角スペースで区切ります。
- [AssetPool] AUDIO CLIP の EXPORT （クラス参照先：CVAudioClipPool - Commands）
  送信コマンド　：AUDIO_CLIP EXPORT AUDIO_CLIP* AUDIO_CLIP*<AUDIO CLIP パス> <出力フォルダパス>
  応答データ　：無し
  - ✧ <出力先パス>：ローカルディレクトリとファイル名
  - ✧ 拡張子の設定は不要です。元データが wav 形式の場合、CLIP 名の wav ファイルが出力されます。
  - ✧ wav・ogg・mp3 に対応ですが、ogg はインポート時に wav に変換されます。

◎vizrt Script 関連のコマンド

▲目次▲

➢ 「付録③：用語解説や補足 – Script とスクリプトベースプラグイン(VSL)」も参照してください。

★Scene Script

- スクリプトパラメータの取得 （クラス参照先：※COMMAND_INFO のみ）

  送信コマンド　：REND*SCRIPT*INSTANCE*<プラグイン変数名> GET

  応答データ　　：<設定値>

- スクリプトパラメータの変更 （クラス参照先：※COMMAND_INFO のみ）

  送信コマンド　：REND*SCRIPT*INSTANCE*<プラグイン変数名> SET <設定値>

  応答データ　　：無し

- ボタンの実行 （クラス参照先：※COMMAND_INFO のみ）

  送信コマンド　：REND*SCRIPT*INSTANCE*<プラグインボタン名> INVOKE

  応答データ　　：無し

- スクリプト構造の取得 （クラス参照先：CPluginInstance - Properties）

  送信コマンド　：REND*SCRIPT*INSTANCE*DYNAMIC_PARAMETER GET

  応答データ　　：※スクリプト変数名、表示タイトル名、種類、など

  ✧　スクリプト構造は、VizScript の UI パーツの各パラメータが取得できます。

  ✧　UI パーツの種類により、取得できる内容は変わります。

★Container Script

- スクリプトパラメータの取得 （クラス参照先：※COMMAND_INFO のみ）

  送信コマンド　：REND*TREE*<ノード階層>*SCRIPT*INSTANCE*<プラグイン変数名> GET

  応答データ　　：<設定値>

- スクリプトパラメータの変更 （クラス参照先：※COMMAND_INFO のみ）

  送信コマンド　：REND*TREE*<ノード階層>*SCRIPT*INSTANCE*<プラグイン変数名> SET <設定値>

  応答データ　　：無し

- ボタンの実行 （クラス参照先：※COMMAND_INFO のみ）

  送信コマンド　：REND*TREE*<ノード階層>*SCRIPT*INSTANCE*<プラグインボタン名> INVOKE

  応答データ　　：無し

- スクリプト構造の取得 （クラス参照先：CPluginInstance - Properties）

  送信コマンド　：REND*TREE*<ノード階層>*SCRIPT*INSTANCE*DYNAMIC_PARAMETER GET

  応答データ　　：※スクリプト変数名、表示タイトル名、種類、など

  ✧　スクリプト構造は、VizScript の UI パーツの各パラメータが取得できます。

  ✧　UI パーツの種類により、取得できる内容は変わります。

◎Control Object 関連のコマンド

▲目次▲

➢ 「付録⑤：Control Object について」も参照してください。

- コマンド構文

送信コマンド　：REND*TREE*<ノード階層>*FUNCTION*ControlObject*in SET ON <Field Identifier> SET <値>

応答データ　：無し

- ✧ <ノード階層>　：ControlObject プラグインが適用されているコンテナ階層。(対象の各 Control プラグインの方ではありません)
- ✧ <Field Identifier>　：各 Control プラグインの Field Identifier
- ✧ <値>　：各 Control プラグインに応じた設定値
- ✧ Control プラグインの種類を問わず、コマンド構文を統一
  例) REND*TREE*$group*FUNCTION*ControlObject*in SET ON 1 SET aiueo
- ✧ 但し、ControlMaterial の Field Identifier は、<ControlMaterial の Field Identifier>.material　となります。
  例) REND*TREE*$group*FUNCTION*ControlObject*in SET ON 1.material SET MATERIAL*Default/red
- ✧ [注意]：　どのような ControlPlugin を対象にしたとしても、ControlObject プラグインに対してのコマンドとなります。
  その為、ロケーションの間違いや SET が無い、などはエラーを返しますが、例えば、「ControlImage で画像が無い」などのように、各 ControlPlugin へのセット時の不具合に対してはエラーを取得できません。

- Control Object の一覧を取得①　(クラス参照先：SCENE - Properties)

送信コマンド　：REND*VDF*PAYLOAD GET

応答データ　：XML ペイロード

- ✧ コントロール オブジェクト名(FieldID)のみの一覧を取得できます
- ✧ 但し Viz5.3 以降は、ControlClip が含まれているとエラーになるため、「Control Object の一覧を取得②」を使用してください

- Control Object の一覧を取得②　(クラス参照先：SCENE - Properties)

送信コマンド　：REND*VDF*DATA GET <インデント種類>

応答データ　：XML 構造データ

- ✧ <インデント種類>　：[COMPACT | TAB_INDENT | SPACES_INDENT]
- ✧ インデント種類を省略時は、COMPACT になります。
- ✧ [Artist] – [The Control Objects panel]に表示される内容がそのまま取得されます
- ✧ 但し、ControlMap、ControlMultiHop、ControlSoftClip は含まれません

- GFX チャンネル経由でのコマンド構文　(クラス参照先：SCENE - Properties)

送信コマンド　：REND*GFX*<GFX チャンネル番号>*TREE*<ノード階層>*FUNCTION*ControlObject*in SET ON <Field Identifier> SET <値>

応答データ　：無し

- ✧ GFX チャンネルのシーンに対してコマンドを送るには、REND の後に"GFX*<GFX チャンネル番号>*"を差し込みます。
  ※「チャンネル番号の差し込み」は、ControlObject 専用では無く、GFX チャンネル制御の基本構文です。
  例) REND*GFX*16*TREE*$ALL*FUNCTION*ControlObject*in SET ON text SET XXXXX
- ✧ <ノード階層>　：GFX 先シーンの ControlObject プラグインが適用されているコンテナ階層
- ✧ <Field Identifier>　：GFX 先シーンの各 Control プラグインの Field Identifier
- ✧ <値>　：GFX 先シーンの各 Control プラグインに応じた設定値
- ✧ 通常、Artist 上で GFX 先シーンの ControlObject は見えません。
  しかし、[Docking and Workspaces (歯車メニュー)]-[Scene Overview]で、GFX 先へシーンを切り替えれば確認できます。
- ✧ 他にも、標準構文で ControlObject プラグインパラメータを設定すれば、prefix 文字列を付加することで制御可能です。
  詳細は、「付録⑤：Control Object について」をご確認ください。

◎コントロールチャネル関連のコマンド

▲目次▲

➢ 「付録③：用語解説や補足 – コントロールチャネル」も参照してください。

- コントロール チャネルの一覧を取得　（クラス参照先：SCENETREE - Properties）

  送信コマンド　：REND*TREE*CONTROL_CHANNEL GET

  応答データ　：{＜コントロールチャネル名＞<#オブジェクト ID>}・・・

- コントロールチャネルにコンテナを登録する　（クラス参照先：CVControlTree - Commands）

  送信コマンド　：REND*CONTROL ADD_CHANNEL ＜コントロールチャネル名＞ REND*TREE*<ノード階層＞ <作成先>

  応答データ　：<チャンネル階層番号>

  ✧ <作成先>　：Channels TREE のどこに作成するかを指定。以下の種類がある。
  - ➢ -1　：常に最後尾に追加される。
  - ➢ <既存名>　AFTER　：既存名のコントロールチャネルの後に追加される。
  - ➢ <既存名>　BEFORE　：既存名のコントロールチャネルの前に追加される。
  - ➢ <Group> UNDER　：Channel Group の下の先頭に追加される。

  ✧ <チャンネル階層番号>：※Channels TREE の上からの番号。最上位が 0。
  Channel Group がある場合は、1/0 等、/で区切られる。

  ✧ 登録例）　REND*CONTROL ADD_CHANNEL dmy_name REND*TREE*$group1 -1

  ✧ 使用例）　※Text ジオメトリに文字列をセット
  REND*TREE*@dmy_name*GEOM*TEXT SET aiueo

- コントロールチャネルにキーフレームを登録する　（クラス参照先：CVControlTree - Commands）

  送信コマンド　：REND*CONTROL ADD_CHANNEL ＜コントロールチャネル名＞ REND*STAGE*<$ディレクター名>*
  ACTION*<アクション名>*KEY*<キーフレーム名> <作成先>

  応答データ　：<チャンネル階層番号>

  ✧ <作成先>　：Channels TREE のどこに作成するかを指定。以下の種類がある。
  - ➢ -1　：常に最後尾に追加される。
  - ➢ <既存名>　AFTER　：既存名のコントロールチャネルの後に追加される。
  - ➢ <既存名>　BEFORE　：既存名のコントロールチャネルの前に追加される。
  - ➢ <Group> UNDER　：Channel Group の下の先頭に追加される。

  ✧ <チャンネル階層番号>：※Channels TREE の上からの番号で最上位が 0。Channel Group がある場合は/で区切られる(1/0 等)

  ✧ 登録例）　REND*CONTROL ADD_CHANNEL key1 REND*STAGE*$Default*ACTION*act*KEY*key1 -1

  ✧ 使用例）　※Action キーフレームに、Director START コマンドを設定
  REND*TREE*@key1*VALUE SET THIS_SCENE*STAGE*$Dir1 START

◎Asset 関連のコマンド

▲目次▲

- Asset (Scene Pool)の全フォルダ・プロジェクト名を取得 (クラス参照先：Pool – Commands)
  - 送信コマンド　：SCENE GET_ALL_GROUPS
  - 応答データ　：※Asset 内の全フォルダ・全プロジェクト名
    - ✧　取得出来る情報は、Asset に存在する階層情報(フォルダ名・プロジェクト名)のみです。
- 指定した Asset フォルダ内のツリー構造を取得 (クラス参照先：Pool – Commands)
  - 送信コマンド　：SCENE GET_GROUPS_DETAIL SCENE*<Asset フォルダ名>
  - 応答データ　：※フォルダ内のフォルダ名と階層情報
- 指定した Asset フォルダ内のツリー情報を取得 (クラス参照先：Pool – Commands)
  - 送信コマンド　：SCENE GET_GROUPS SCENE*<Asset フォルダ名>
  - 応答データ　：※指定したフォルダ内のフォルダ名のみ取得します。
- 指定した Asset フォルダ以下のツリー情報を取得 (クラス参照先：Pool – Commands)
  - 送信コマンド　：SCENE GET_GROUPS_DOWN SCENE*<Asset フォルダ名>
  - 応答データ　：※指定したフォルダ内の全てのサブフォルダを含むフォルダ名
- 指定した Asset フォルダ内のロケーション情報を取得 (クラス参照先：Pool – Commands)
  - 送信コマンド　：SCENE GET_DETAIL SCENE*<Asset フォルダ名>
  - 応答データ　：※Asset シーンパス　：SCENE*<Asset フォルダ名>/<Asset シーン名>
    - ✧　以下も、基本同様にそれぞれの内容が取得できる
      マテリアル　：MATERIAL GET_DETAIL MATERIAL*<Asset フォルダ名>
      マテリアルアドバンスド：MATERIAL_ADVANCED GET_DETAIL MATERIAL_ADVANCED*<Asset フォルダ名>
      画像　：IMAGE GET_DETAIL IMAGE*<Asset フォルダ名>
      フォント　：FONT GET_DETAIL FONT*<Asset フォルダ名>
- Asset にフォルダを作成 (クラス参照先：Pool – Commands)
  - 送信コマンド　：SCENE ADD_GROUP <Asset パス名>
  - 応答データ　：無し
    - ✧　Vizrt 2.x 時代の名残で、先頭は SCENE の他に IMAGE/FONT などでも同じ結果です。

◎画像の取得関連のコマンド

▲目次▲

- シーンのサムネイル画像(アイコン)を取得 (クラス参照先：CMain – Commands)
  - 送信コマンド　：peak_get_icon SCENE*<Asset フォルダ名>/<Asset シーン名>
  - 応答データ　：<横サイズ><縦サイズ><バイナリデータ>
    - ✧　<横・縦サイズ>　：サムネイル画像の横・縦サイズ。ビッグエンディアン、各 16bit。
    - ✧　<バイナリデータ>　：横サイズ×縦サイズ×3 バイト(RGB 順)のバイナリ形式
- スナップショット画像の作成 (クラス参照先：RENDERER_BASE – Commands)
  - 送信コマンド　：REND SNAPSHOT <ImagePath> [　RGBA | RGB　]
  - 応答データ　：<ImagePath>
    - ✧　例）送信コマンド　：1 REND SNAPSHOT d:¥snap.png RGBA
      応答データ　：d:¥snap.png

◎システム関連のコマンド

▲目次▲

- vizrt の起動　(参照先：[Viz Artist User Guide]-[Getting Started]-[Viz Command Line Options])

  実行コマンド　："<vizrt インストールフォルダ>¥VizEngine¥viz.exe" <コマンドラインオプション>

  - ✧ viz.exe にコマンドラインオプションを指定して実行します。
    例)　："%ProgramFiles%¥vizrt¥VizEngine¥viz.exe" -u1 -y -n
  - ✧ コマンドラインオプション
    オプションの詳細は、「付録⑦：Vizrt 起動時のコマンドライン オプション」を参照してください。
    参考) インストール時に自動作成されるショートカットのパラメータ
    Viz Artist　："<インストール先>¥VizEngine¥Viz.exe" -u1 -y
    Viz Engine　："<インストール先>¥VizEngine¥Viz.exe" -u1 -y -n
    Viz Engine – Config　："<インストール先>¥VizEngine¥Viz.exe" -u1 -y -c
  - ✧ Viz.exe が読み込む設定ファイルやプラグインは作業フォルダからの相対パスで管理されています。
    もしバッチファイルで起動する場合は、先に viz.exe のあるディレクトリに移動してから viz.exe を起動するようにしてください。

- vizrt の終了　(クラス参照先：CMain – Commands)

  送信コマンド　：EXIT
  応答データ　：無し

  - ✧ Viz エンジンを終了します。

- vizrt の再起動　(クラス参照先：CMain – Commands)

  送信コマンド　：RESTART <再起動モード>
  応答データ　：無し

  - ✧ <再起動モード>　引数無し　：起動中のモードと同じ
    CFG　：CONFIG モード
    ENG　：GUI 無し Engine モード
    ENG_GUI　：GUI あり Engine モード
    ART　：Airtist モード
  - ✧ Viz エンジンを終了し、指定されたモードで再起動します。
  - ✧ OA モードのみ使用可能ですので、VizGH への接続前や Config、編集モードでは使用できません。

- vizrt のバージョンを取得　(クラス参照先：CMain – Commands)

  送信コマンド①　：MAIN VERSION
  送信コマンド②　：VERSION
  応答データ　：<接頭辞番号> Version: <vizrt バージョン番号>
  例) 1 Version: 5.2.1.60000

  - ✧ MAIN は省略できます。
  - ✧ 送信側が複数コマンドの最後に VERSION を送る事で、一連の送信後処理の応答に使用します。

- vizrt artist の制御モード切り替え　(クラス参照先：CMain – Commands)

  OA モード　：SWITCH_EXTERNAL
  エディターモード　：SWITCH_EDIT
  ポストレンダーモード　：SWITCH_POST

  - ✧ OA モードとポストレンダーモード間の直接 GUI 切り替えは出来ません。一旦エディターモードを経由する必要があります。
  - ✧ Artist モードでの起動直後で VizGUI の起動完了前だと、SWITCH_EXTERNAL コマンドは失敗します。
    しかし、その後の SWITCH_EXTERNAL コマンドの再実行は無視されますので、一旦エディターモードを経由する必要があります。

- 外部制御モード (OA モード)の確認　（クラス参照先：CMain – Commands）
  送信コマンド　：IS_ON_AIR
  応答データ　：<ON/OFF 値>
  - ✧ SWITCH_EXTERNAL コマンドを実行した際に、正常に OA モードになったかを確認できます。

- OA ステータスを取得　（クラス参照先：CMain – Properties）
  送信コマンド①　：MAIN*ONAIR GET_INFO
  送信コマンド②　：ONAIR GET_INFO
  応答データ　：Hostname、IP Address、Port、GH-Server、Layer(Back/Middle/Front)、Uptime
  - ✧ MAIN は省略できます。

- パフォーマンスバーの表示切り替え　（クラス参照先：RENDERER_BASE – Commands）
  送信コマンド　：REND SET_PERFORMANCE <ON/OFF 値>
  応答データ　：無し

- メモリへのロード状況を取得　（クラス参照先：Pool – Commands）
  送信コマンド　：SCENE INFO
  応答データ　：メモリ上にある対象の VizID と付帯情報 (未解析)
  - ✧ 「レンダリングされているシーン以外がクリアされたか？」などの確認に使います。
  - ✧ SCENE 以外に、FONT / IMAGE も基本同様です。

- テクスチャメモリサイズと割り当て済みメモリ値を取得　（クラス参照先：CMain – Properties）
  送信コマンド　：TEXTURE*MEMORY GET
  応答データ　：TOTAL <メモリ値> PIXEL <メモリ値> ALLOCATED <割当済値> SIZE <テクスチャメモリ値>

- メモリに読み込まれた特定の画像をメモリから削除　（クラス参照先：Pool – Commands）
  送信コマンド　：<Viz 画像パス>　CLOSE
  応答データ　：無し

- メモリに読み込まれている未使用のデータをクリア　（クラス参照先：Pool – Commands）
  シーン用コマンド　：SCENE CLEANUP
  フォント用コマンド　：FONT CLEANUP
  画像用コマンド　：IMAGE CLEANUP
  MD 内画像用コマンド　：TEXTURE_IMAGE CLEANUP
  応答データ　：無し
  - ✧ レンダリング中のシーンや使用中のオブジェクトは除外されます。
  - ✧ 画像をクリアしてもコンソールウインドウのテクスチャ使用量はリセットされません。カウントアップされ続けます。
  - ✧ ”IMAGE INFO”コマンドを使用して画像がロードされていないことを確認してください。

- RENDERER の初期化　（クラス参照先：RENDERER_BASE – Commands）
  送信コマンド　：REND INITIALIZE
  応答データ　：無し
  - ✧ 強制的にレンダラーを初期化します。
    （プラグイン表示リスト作成、テキストオブジェクト構築、テクスチャ定義、テクスチャ用メモリ割り当てなど）
  - ✧ 安心・安全のために、本コマンドでの初期化では無く、vizrt の再起動をお勧めします。

- vizrt の再描画を強制的に即時実行　（クラス参照先：RENDERER_BASE – Commands）
  送信コマンド　：REND REDRAW
  応答データ　：無し

- 画面の特定位置をクリックして選択したオブジェクト ID を取得　（クラス参照先：RENDER_EDITOR – Commands）
  送信コマンド　：REND DROP <MouseX> <MouseY>
  応答データ　：<#オブジェクト ID>

- 疑似マウス入力　（クラス参照先：RENDERER_BASE – Properties）
  送信コマンド　：REND*SIMULATE_INPUT <疑似モード>
  応答データ　：無し
  - ✧ <疑似モード>　MOUSE_CLICK <X> <Y> <マウスボタン位置>
    MOUSE_DOWN <X> <Y> <マウスボタン位置>
    MOUSE_UP <X> <Y> <マウスボタン位置>
    MOUSE_MOVE <X> <Y>
  - ✧ <X>・<Y> ：疑似マウス位置
  - ✧ <マウスボタン位置>　：[LEFT | RIGHT | MIDDLE]
  - ✧ 例） 左マウスダウン (X=684,Y=264)
    REND*SIMULATE_INPUT MOUSE_DOWN 687 264 LEFT
  - ✧ 疑似モードはそれぞれ、以下の VizScript イベントが発生します。
    ※左ボタン (LEFT) の場合
    MOUSE_CLICK　：OnLButtonDown & OnLButtonUp
    MOUSE_DOWN　：OnLButtonDown
    MOUSE_MOVE　：OnMouseMove
    MOUSE_UP　：OnLButtonUp
  - ✧ 疑似モードは、他にも TOUCH 系があります。
    MOUSETOUCH_DOWN <X> <Y>
    [MOUSETOUCH_DOWN | MOUSETOUCH_MOVE | MOUSETOUCH_UP]
    [W7TOUCH_DOWN | W7TOUCH_MOVE | W7TOUCH_UP]

◎選択範囲と Bounding box 関連のコマンド

▲目次▲

- Bounding box の表示切り替え　（クラス参照先：RENDER_EDITOR – Commands）
  送信コマンド　：REND SHOW_BOUNDING_BOX <ON/OFF 値>
  応答データ　：　無し
  - ✧ Bounding box の設定する前に、Bounding box を表示する必要があります。

- Bounding box 対象コンテナを設定　（クラス参照先：SCENE – Properties）
  送信コマンド　：REND*EDITED_OBJECT SET REND*TREE*<ノード階層>
  応答データ　：　無し

- 対象コンテナの指定を解除　（クラス参照先：SCENE – Properties）
  送信コマンド　：REND*EDITED_OBJECT SET
  応答データ　：　無し
  - ✧ 特定ノードの指定を解除すれば、Bounding box は表示されません。

- 対象コンテナの座標と範囲の取得
  送信コマンド　：REND*TREE*<ノード階層>*BOUNDING_BOX GET
  応答データ　：　バウンディングボックスの範囲座標 (詳細は未確認)

◎ポストレンダリング関連のコマンド

▲目次▲

★Post Render Templates の設定

- このテンプレートは、出力形式と出力先の設定(Clip Plugin 項目)を保存しています。
- 外部コマンドでの設定では GUI 表示値は変化しませんが(Quality は Template のダブルクリックも変化無し）、値はセットされています。
- 但し、出力ファイル名は Template に含まれず、ロード時にクリアされる為、都度、設定する必要があります。
- Template はローカルフォルダでは無く、VizGH に保存されます。
- Template の存在確認コマンドを使用する場合、先に Template 名一覧の取得コマンドを実行し、VizGH から取得する必要があります。

- Post Render Templates の Template 名一覧の取得①　（クラス参照先：CVRenderToDiskManager – Properties）

  送信コマンド　：RENDER_TO_DISK*TEMPLATE*LIST GET

  応答データ　：　<Template 名一覧>

- Post Render Templates の Template 名一覧の取得②　（クラス参照先：※COMMAND_INFO のみ）

  送信コマンド　：RENDER_TO_DISK*TEMPLATE LIST

  応答データ　：　<Template 名一覧>

- Post Render Templates 全構造データの取得　（クラス参照先：CVRenderToDiskManager – Properties）

  送信コマンド　：RENDER_TO_DISK*DATA GET

  応答データ　：　<Templates の構造データ>

- Template の存在確認　（クラス参照先：※COMMAND_INFO のみ）

  送信コマンド　：RENDER_TO_DISK*TEMPLATE EXISTS <Template 名>

  応答データ　：　<ON/OFF 値>

- Template の読み込み　（クラス参照先：※COMMAND_INFO のみ）

  送信コマンド　：RENDER_TO_DISK*TEMPLATE LOAD <Template 名>

  応答データ　：　<結果>

  ✧　<結果：成功>：Loaded

  ✧　<結果：失敗>：コマンドエラー通知文

- Template の上書き保存　（クラス参照先：※COMMAND_INFO のみ）

  送信コマンド　：RENDER_TO_DISK*TEMPLATE SAVE <Template 名>

  応答データ　：　<結果>

  ✧　<結果：成功>：Saved

  ✧　<結果：失敗>：コマンドエラー通知文

- Template の別名保存　（クラス参照先：※COMMAND_INFO のみ）

  送信コマンド　：RENDER_TO_DISK*TEMPLATE SAVE_AS <Template 名>

  応答データ　：　<結果>

  ✧　<結果：成功>：Saved

  ✧　<結果：失敗>：コマンドエラー通知文

  ✧　同名 Template は上書きします。

- Template の削除　（クラス参照先：※COMMAND_INFO のみ）

  送信コマンド　：RENDER_TO_DISK*TEMPLATE DELETE <Template 名>

  応答データ　：　<結果>

  ✧　<結果：成功>：Deleted

  ✧　<結果：失敗>：コマンドエラー通知文

- Template 名の変更 （クラス参照先：※COMMAND_INFO のみ）
  送信コマンド　：RENDER_TO_DISK*TEMPLATE RENAME <元 Template 名> <新 Template 名>
  応答データ　：　<結果>
  - ✧　<結果：成功>：Renamed
  - ✧　<結果：失敗>：コマンドエラー通知文

- 現在の Template 名を取得 （クラス参照先：※COMMAND_INFO のみ）
  送信コマンド　：RENDER_TO_DISK*TEMPLATE GET_CURRENT
  応答データ　：　<現 Template 名>

★Clip Plugin 項目の設定

▲目次▲

- Post Render の出力形式と出力先の設定です。
- 以下、PNG 連番出力に関係する箇所を記載しています。
- 個別に設定するよりも、予め Post Render UI で Template に保存して利用する事を推奨します。

- レンダリング Plugin の取得 （クラス参照先：CVRenderToDiskManager – Properties）
  送信コマンド　：RENDER_TO_DISK*PLUGIN GET
  応答データ　：　<レンダーモード>
  - ✧　<レンダーモード>：[JpegRenderer| **PNGRenderer** | TiffRenderer|TgaRenderer|VFWRenderer|VbnRenderer|VideoRenderer]

- レンダリング Plugin の設定 （クラス参照先：CVRenderToDiskManager – Properties）
  送信コマンド　：RENDER_TO_DISK*PLUGIN SET <レンダーモード>
  応答データ　：無し
  - ✧　<レンダーモード>：[JpegRenderer| **PNGRenderer** | TiffRenderer|TgaRenderer|VFWRenderer|VbnRenderer|VideoRenderer]

- Quality の取得 （クラス参照先：※COMMAND_INFO のみ）
  送信コマンド　：RENDER_TO_DISK*PLUGIN_INSTANCE*quality GET
  応答データ　：<Quality 値>
  - ✧　デフォルト値：90。但し、Quality が高いと生成速度が遅い。低くても画質低下はあまりない為、10 程度が妥当

- Quality の設定 （クラス参照先：※COMMAND_INFO のみ）
  送信コマンド　：RENDER_TO_DISK*PLUGIN_INSTANCE*quality SET <Quality 値>
  応答データ　：無し

- Frame Format の取得 （クラス参照先：CVRenderToDiskManager – Properties）
  送信コマンド　：RENDER_TO_DISK*FRAME_FORMAT GET
  応答データ　：<Frame Format 値>
  - ✧　<Frame Format 値>：※Frame Format 選択肢の取得コマンド 参照

- Frame Format の設定 （クラス参照先：CVRenderToDiskManager – Properties）
  送信コマンド　：RENDER_TO_DISK*FRAME_FORMAT SET <Format 値>
  応答データ　：無し
  - ✧　<Frame Format 値>：※Frame Format 選択肢の取得コマンド 参照

- Pixel Format の取得 （クラス参照先：CVRenderToDiskManager – Properties）
  送信コマンド　：RENDER_TO_DISK*PIXEL_FORMAT GET
  応答データ　：<Pixel Format 値>
  - ✧　<Pixel Format 値>：※Pixel Format 選択肢の取得コマンド 参照

- Pixel Format の設定　（クラス参照先：CVRenderToDiskManager – Properties）
  送信コマンド　：RENDER_TO_DISK*PIXEL_FORMAT SET <Pixel Format 値>
  応答データ　：無し
  - ✧ <Pixel Format 値>：※Pixel Format 選択肢の取得コマンド 参照

- Resolution (Width Height)の取得　（クラス参照先：CVRenderToDiskManager – Properties）
  送信コマンド　：RENDER_TO_DISK*RESOLUTION SET <Width 値> <Height 値>
  応答データ　：無し

- Resolution (Width Height)の設定　（クラス参照先：CVRenderToDiskManager – Properties）
  送信コマンド　：RENDER_TO_DISK*RESOLUTION GET
  応答データ　：<Width 値> <Height 値>
  - ✧ ※[Config]-[Output Formats]値が初期値です。縮小は出来ますが、初期値を越えることは出来ません。

- Output Directory と Fille Name の取得　（クラス参照先：CVRenderToDiskManager – Properties）
  送信コマンド　：RENDER_TO_DISK*CLIP_NAME GET
  応答データ　：<ディレクトリパス>/<ファイル接頭辞>
  - ✧ <ディレクトリパス>と<ファイル接頭辞>は/で区切られている。
    つまり、文字列が/で終わると、全て<ディレクトリパス>となり、<ファイル接頭辞>は空欄になる。
  - ✧ <ディレクトリパス>は GUI の[Output Directory]、<ファイル接頭辞>は File name の値が取得される。
  - ✧ Template に<ファイル接頭辞>は保存されない。

- Output Directory と Fille Name の設定　（クラス参照先：CVRenderToDiskManager – Properties）
  送信コマンド　：RENDER_TO_DISK*CLIP_NAME SET <ディレクトリパス>/<ファイル接頭辞>
  応答データ　：無し
  - ✧ <ディレクトリパス>と<ファイル接頭辞>は、'¥'ではなく、'/'で区切る。
    連番ファイル名を 6 桁の数字のみにするには、文字列最後尾を'/'にして、<ディレクトリパス>のみ設定する。
  - ✧ Template に<ファイル接頭辞>は保存されない。**ファイル接頭辞をつけるには、RECORD コマンド実行前に、必ず実行する。**
  - ✧ 例：（ディレクトリ名：D:/Temp、　ファイル接頭辞：image ）
    - ➢ ディレクトリと接頭辞を共に設定　：RENDER_TO_DISK*CLIP_NAME SET d:/Temp/image
    - ➢ ディレクトリのみ設定、接頭辞未使用　：RENDER_TO_DISK*CLIP_NAME SET d:/Temp/

- Frame Format 選択肢の取得　（クラス参照先：CVRenderToDiskManager – Properties）
  送信コマンド　：RENDER_TO_DISK*FRAME_FORMAT GET_CAPS
  応答データ　：<Format 選択肢>
  - ✧ **FRAMES_FULLRATE**　：プログレッシブ・フルフレーム
  - ✧ FIELDS_TOP　：上位フィールドのみ・ハーフハイトフレーム
  - ✧ FIELDS_BOTTOM　：下位フィールドのみ・ハーフハイトフレーム
  - ✧ FULL_INTERLACED_TOP　：インターレース・上位フィールド優先フルフレーム
  - ✧ FULL_INTERLACED_BOTTOM　：インターレース・下位フィールド優先フルフレーム
  - ✧ FRAMES_DOUBLERATE　：1 つおき・フルフレーム (例:NTSC=30 frm/sec)

- Pixel Format 選択肢の取得　（クラス参照先：CVRenderToDiskManager – Properties）
  送信コマンド　：RENDER_TO_DISK*PIXEL_FORMAT GET_CAPS
  応答データ　：<PIXEL Format 選択肢>
  - ✧ RGB　：RGB 画像
  - ✧ **RGBA**　：アルファチャンネル付き RGB 画像。Key 映像がアルファチャンネルに格納される。
  - ✧ RGB_AAA　：Fill 映像と Key 映像を別々のファイルに保存。Key 側ファイル名は、<ファイル名>_key.png となる。

★Scene Setup 項目の設定

▲目次▲

- レンダリングするシーンの構成と、開始オフセットや STOP ポイントの設定です。
- Classic Render では、開始オフセットと STOP ポイントは FRONT/MAIN/BACK と各レイヤー毎に設定できます。
- STOP ポイントは PAUSE ポイントに置き換えられ、一時停止時間を秒値で設定できます。

- 開始オフセットの取得 （クラス参照先：CVRenderToDiskManager – Properties）

  送信コマンド　：RENDER_TO_DISK*<レイヤー>*OFFSET GET

  応答データ　：＜オフセット値＞

  - <レイヤー>：[FRONT | MAIN | BACK]
  - 収録開始オフセットフィールド数

- 開始オフセットの設定 （クラス参照先：CVRenderToDiskManager – Properties）

  送信コマンド　：RENDER_TO_DISK*<レイヤー>*OFFSET SET <値>

  応答データ　：無し

  - <レイヤー>：[FRONT | MAIN | BACK]
  - 収録開始オフセットフィールド数

- 一時停止時間の取得 （クラス参照先：CVRenderToDiskManager – Properties）

  送信コマンド　：RENDER_TO_DISK*<レイヤー>*STOPPAUSE GET

  応答データ　：<Time 値(秒)>

  - <レイヤー>：[FRONT | MAIN | BACK]
  - STOP ポイントを一定時間の PAUSE に置き換える為の停止秒数

- 一時停止時間の設定 （クラス参照先：CVRenderToDiskManager – Properties）

  送信コマンド　：RENDER_TO_DISK*<レイヤー>*STOPPAUSE SET <Time 値(秒)>

  応答データ　：無し

  - <レイヤー>：[FRONT | MAIN | BACK]
  - STOP ポイントを一定時間の PAUSE に置き換える為の停止秒数

- Duration の取得 (自動設定) （クラス参照先：CVRenderToDiskManager – Properties）

  送信コマンド　：RENDER_TO_DISK*DURATION GET

  応答データ　：<Duration 値>

  - Duration : 連番画像の場合、作成される画像枚数
  - このコマンドを実行すると、レンダリング中の Scene から Duration 値を取得し、適用される。
  - シーンからの取得値を使用する場合、**RECORD コマンド実行前に、必ず実行。**
    実行しないで収録を実行すると Duration 値が 999999 となり、 4 時間以上停止しない。

- Duration の設定 （クラス参照先：CVRenderToDiskManager – Properties）

  送信コマンド　：RENDER_TO_DISK*DURATION SET <Duration 値>

  応答データ　：無し

  - Duration : 連番画像の場合、作成される画像枚数
  - シーンからの取得値以外を使用する場合、RECORD コマンド実行前にセットする。
  - GET コマンドが SCENE からの Duration 値の自動抽出に使用されている為、セットした値を確認することは出来ない。

- 収録の開始　（クラス参照先：CVRenderToDiskManager – Commands）

  送信コマンド　：RENDER_TO_DISK RECORD

  応答データ　：“RENDER_TO_DISK CLIPNAMES VIDEO(, ,) AUDIO() VBI()”

  - ✧ 応答は返るが、利用目的は不明。
  - ✧ RECORD で開始されるアニメーションは、REND*STAGE START の実行結果と等しい。
  - ✧ 順序を指定してディレクターを実行したい場合は、制御用 Director を使用する。

    <使用手順>

    ① 制御用 Director を作成

    ② 制御用 Director 以外を一旦全停止

    - ✧ 制御用 Director - ACTION の Time 0 で、他の Director SHOW 0 コマンドを実行

    ③ 制御用 Director-ACTION 内で必要に応じ、順に他の Director を実行
  - ✧ 最後の Director が完了する Timeline 位置に ACTION-Keyframe、もしくは STOP ポイントが必要。

    それが無いと、Duration 値が最長 Director の長さになってしまい、Scene から正しく読み取れない。
  - ✧ リファレンスマニュアルには、いくつか引数があるが、挙動は全て上書き保存される為、もはや引数の効果は無い。

    [INTERACTIVE | INTERACTIVE_OVERWRITE | NO_OVERWRITE | OVERWRITE]
  - ✧ Post Render UI では、File Name が空欄だと Record ボタンが無効だが、外部制御コマンドは実行可能。
  - ✧ 作成されるファイル名：<Output Directory>nnnnnn.png　#n=0-9 の数字

- 収録の停止　（クラス参照先：CVRenderToDiskManager – Commands）

  送信コマンド　：RENDER_TO_DISK STOP

  応答データ　：無し

- 収録ステータスの取得①　（クラス参照先：CVRenderToDiskManager – Commands）

  送信コマンド　：RENDER_TO_DISK GET_STATUS

  応答データ　：<ステータス>

  - ✧ <ステータス：実行中>： Generating Field <現 Field 値> of <Duration 値>
  - ✧ <ステータス：停止中>： Idle

- 収録ステータスの取得②　（クラス参照先：RENDERER_BASE – Properties）

  送信コマンド　：REND*POST*STATUS GET

  応答データ　：<ステータス>

  - ✧ <ステータス：実行中>： Generating Field <現 Field 値> of <Duration 値>
  - ✧ <ステータス：停止中>： Post Rendering is not active

- ポストレンダリング GUI の有効化の取得　（クラス参照先：RENDERER_BASE – Properties）

  送信コマンド　：REND*POST_MODE GET

  応答データ　：　<ON/OFF 値>

- ポストレンダリング GUI の有効化の設定　（クラス参照先：RENDERER_BASE – Properties）

  送信コマンド　：REND*POST_MODE SET <ON/OFF 値>

  応答データ　：無し

  - ✧ GUI を”Post”に切り替えた時に実行されているが、影響不明

◎アーカイブ関連のコマンド

▲目次▲

- アーカイブのクリア　（クラス参照先：CVArchive – Commands）

  送信コマンド　：ARCHIVE CLEAR

  応答データ　：無し

  - ✧　現在のアーカイブを閉じます。

- アーカイブの作成 (追加)　（クラス参照先：CVArchive – Properties）

  送信コマンド　：ARCHIVE*SCENE ADD SCENE*<シーン名>

  応答データ　：無し

  - ✧　アーカイブファイルに追加したいシーンが複数ある場合は、このコマンドを複数実行します。
  - ✧　SCENE*<シーン名>で使用しているマテリアル、オブジェクト、イメージ、フォントは自動的にアーカイブに追加されます。
  - ✧　その他素材を追加するには、同様に”SCENE”を MATERIAL・IMAGE・GEOM 等に代えて追加できます。

- アーカイブから全てのフォントを削除　（クラス参照先：CVArchive – Properties）

  送信コマンド　：ARCHIVE*FONT REMOVE_ALL

  応答データ　：無し

- アーカイブをファイルに保存　（クラス参照先：CVArchive – Commands）

  送信コマンド　：ARCHIVE STORE <ファイルパス>

  応答データ　：無し

  - ✧　ファイルの拡張子は、自動的に”.via”が付加されます。
  - ✧　自動的にアーカイブは閉じられます。

- アーカイブファイルのインポート　（クラス参照先：CVArchive – Commands）

  送信コマンド　：ARCHIVE IMPORT <ファイルパス>

  応答データ　：無し

  - ✧　アーカイブファイルをインポートします。

  ＜※以下は非推奨＞

  - ✧　アーカイブファイルの上書きインポート　：~~ARCHIVE IMPORT_FORCE <ファイルパス>~~
    ※IMPORT との違いが不明
  - ✧　アーカイブファイルの新しいもののみ上書きインポート　：~~ARCHIVE IMPORT_NEWER <ファイルパス>~~
    ※IMPORT との違いが不明
  - ✧　アーカイブファイルの古いもののみ上書きインポート　：~~ARCHIVE IMPORT_OLDER <ファイルパス>~~
    ※IMPORT との違いが不明
  - ✧　アーカイブのオープン　：~~ARCHIVE OPEN <ファイルパス>~~
    インポートせずにアーカイブを開きます。
    ※シーンのみアーカイブに展開されません。
  - ✧　アーカイブを全てインポート　：~~ARCHIVE IMPORT_ALL <ファイルパス>~~
    アーカイブをインポートし強制的に上書きします。
    ※IMPORT との違いが不明

◎動作ログ関連のコマンド

▲目次▲

- コマンド コンソール ウインドウの表示 ON/OFF （クラス参照先：CMain – Commands）

  送信コマンド　：SHOW_COMMANDS [ON | OFF]

  応答データ　：無し

  - VizEngine 画面右上のアイコン(Show commands)の動作と同じです。
  - 現在のウインドウ表示状況を取得するコマンドはありません。

- コマンド コンソールのクリア （クラス参照先：CMain – Properties）

  送信コマンド　：CONSOLE CLEAR

  応答データ　：無し

  - コマンドコンソールの clear / cls コマンドに該当します。

- コマンド コンソールの内容を任意のログファイルに保存 （クラス参照先：CMain – Commands）

  送信コマンド　：CLOG <ファイル名>

  応答データ　：無し

  - <ファイル名> : ローカル PC 内の絶対パスでのファイル名
  - 必ず、先にコマンドコンソールを表示させてください。
    コマンドコンソールが表示されていないと、コマンドやイベントが記録されません。
  - [CONSOLE OUTPUT]行以降が、コマンドコンソールに表示された情報です。

- コマンド コンソールの内容をデフォルトのログファイルに保存 （クラス参照先：CMain – Commands）

  送信コマンド　：CLOG

  応答データ　：無し

  - コマンドコンソールの clog コマンドに該当します。
  - 必ず、先にコマンドコンソールを表示させてください。
    コマンドコンソールが表示されていないと、コマンドやイベントが記録されません。
  - [CONSOLE OUTPUT]行以降が、コマンドコンソールに表示された情報です。
  - ログファイルは、デフォルトパスとして VizEngine インストールフォルダに作成されます。
  - 出力されるログファイル名は、Console の次行に file ‘Engine_00_clog_xxxxxxxxxx.log’等と表示されます。
  - 次のコマンドも同じ動作ですが、古い為、あまりお勧めしません。 （クラス参照先：※COMMAND_INFO のみ）
    - 送信コマンド　：DEBUG_CONTROL*RENDERINFO STAT_STOP
    - 送信コマンド　：DEBUG_CONTROL*RENDERINFO STORE_COMMANDS

- コマンド コンソールの内容を取得 （クラス参照先：CDebugControl – Properties）

  送信コマンド　：DEBUG_CONTROL*RENDERINFO*CONSOLE GET

  応答データ　：コンソール表示内容

  - VizEngine 画面右上のアイコン(Show Commands)で表示される内容を取得します。

- ログ コンソールの内容を取得 （クラス参照先：CVLog – Commands）

  送信コマンド　：LOG GET

  応答データ　：コンソール表示内容

  - VizEngine 画面右上のアイコン(Show Log)で表示される内容を取得します。
  - 同じ内容がコマンドコンソールにも記録されますが、確認が容易です。
  - ログコンソールには送信コマンドでのエラーは含まれません。応答データやコマンドコンソールで確認してください。

- ログ コンソールのクリア （クラス参照先：CVLog – Commands）

  送信コマンド　：LOG CLEAR

  応答データ　：無し

◎Config 関連のコマンド

▲目次▲

- Config ファイルのロード　（クラス参照先：CCfgData – Commands）
  - 送信コマンド　：CONFIGURATION LOAD <設定ファイル名>
  - 応答データ　：無し
    - ✧ CONFIG 画面の LOAD ボタンに該当します。
- Config ファイルの保存　（クラス参照先：CCfgData – Commands）
  - 送信コマンド　：CONFIGURATION SAVE <設定ファイル名>
  - 応答データ　：無し
    - ✧ CONFIG 画面の SAVE ボタンに該当します。
    - ✧ <設定ファイル名>：省略時はデフォルトファイル(VizEngine-0.cfg)に保存されます。
- Config 設定のリセット　（クラス参照先：CCfgData – Commands）
  - 送信コマンド　：CONFIGURATION RESET
  - 応答データ　：無し
    - ✧ CONFIG 画面の RESET ボタンに該当します。
- 出力信号フォーマットの取得　（クラス参照先：※COMMAND_INFO のみ）
  - 送信コマンド　：CONFIGURATION*VIDEO*OUTPUT GET
  - 応答データ　：<Output Formats>
- 出力信号フォーマットの設定　（クラス参照先：※COMMAND_INFO のみ）
  - 送信コマンド　：CONFIGURATION*VIDEO*OUTPUT SET <Output Formats>
  - 応答データ　：無し
    - ✧ <Output Formats>　1080I_5994_SMPTE274　：HD
      2160P_5994_SMPTE2036_UHDTV1　：4K UHDTV
      480I_5994_SMPTE259_NTSC　：NTSC
    - ✧ [CONFIG]-[Output Formats]-[Specification]列値を設定
- Render Engine のデフォルト設定　（クラス参照先：CCfgData – Properties）
  - 送信コマンド　：CONFIGURATION*SCENE_DEFAULTS*RENDERENGINE_VERSION SET <Render 種類>
  - 応答データ　：無し
    - ✧ <Render 種類>　V3　：Classic
      V4　：Viz Engine Render
    - ✧ [CONFIG]-[Scene Default Values]-[Varios]-[Render Engine]の設定値

◎Shared Memory (SHM)関連のコマンド

▲目次▲

➢ 「付録④：Shared Memory (SHM)」も参照してください。

(クラス参照先：SCENE / CGlobal / CVVizCommunication – Properties)

- 標準送信コマンド

  特定シーン内　　：REND*MAP <データタイプ> <SHM Key 名> <値>
  ローカル vizrt 内共有　：GLOBAL*MAP <データタイプ> <SHM Key 名> <値>
  vizrt 間共有　　：VIZ_COMMUNICATION*MAP <データタイプ> <SHM Key 名> <値>

  ✧ MAP・<データタイプ>・<SHM Key 名>・<値>の間は、半角スペースで区切ります。

- Apply Shared Memory プラグイン用 送信コマンド

  ✧ 標準送信コマンドの<SHM Key 名>が、<MBK/ID>に変わります

  特定シーン内　　：REND*MAP <データタイプ> <MBK/ID> <値>
  ローカル vizrt 内共有　：GLOBAL*MAP <データタイプ> <MBK/ID> <値>
  vizrt 間共有　　：VIZ_COMMUNICATION*MAP <データタイプ> <MBK/ID> <値>

- 設定・取得用 SHM コマンド　(クラス参照先：CVSharedMemoryMap – Commands)

データタイプ		設定用	/	取得用
文字列型	：	SET_STRING_ELEMENT	/	GET_STRING_ELEMENT
Bool 型	：	SET_BOOLEAN_ELEMENT	/	GET_BOOLEAN_ELEMENT
整数型	：	SET_INTEGER_ELEMENT	/	GET_INTEGER_ELEMENT
浮動小数点型	：	SET_DOUBLE_ELEMENT	/	GET_DOUBLE_ELEMENT

例)
送信コマンド(シーン)　：-1 REND*MAP SET_INTEGER_ELEMENT pos1 100.0
送信コマンド(ローカル)　：-1 GLOBAL*MAP SET_STRING_ELEMENT name1 aiueo
Apply SHM①　：-1 REND*MAP SET_DOUBLE_ELEMENT /ALL/pos1 100.0
Apply SHM②　：-1 REND*MAP SET_STRING_ELEMENT /test/name abcdef

- SHM **UDP** ポートの取得　(クラス参照先：CCfgData – Properties)

  送信コマンド　：CONFIGURATION*COMMUNICATION*SMM_UDP_SERVICE GET
  応答データ　　：TCP ポート番号

- SHM **TCP** ポートの取得　(クラス参照先：CCfgData – Properties)

  送信コマンド　：CONFIGURATION*COMMUNICATION*SMM_TCP_SERVICE GET
  応答データ　　：UDP ポート番号

＜付録⓪＞一般的な vizrt の外部制御

▲目次▲

- vizrt は、GUI 操作も含め、全ての操作がコマンドによって実行されています。
- [Config]-[Communication]で定義された TCP ポート(Dft:6100)で、外部から制御コマンドを受け取れます。
- コマンドは vizrt Command コンソールで確認できます。このコンソールから送信も可能です。
- vizrt が応答データを返す時間はコマンドにより異なります。
  - ✧ シーンの読込みなどは応答までに時間が掛かりますので、タイムアウト処理を実装する場合はご注意ください。
- vizrt 制御コマンド形式

  送信コマンド　：<接頭辞番号>[20h]<ロケーション>[20h]<コマンド>[20h]<引数>[00h]

  応答データ　：<接頭辞番号>[20h]<応答データ>[00h]
  - ✧ [20h]　：半角スペース
  - ✧ [00h]　：終端文字(¥0)

  サムネイル画像　：<横サイズ><縦サイズ><バイナリデータ>
  - ✧ <横・縦サイズ>　：サムネイル画像の横・縦サイズ。ビッグエンディアン、各 16bit。
  - ✧ <バイナリデータ>　：横サイズ×縦サイズ×3 バイト(RGB 順)のバイナリ形式

- 送信コマンドとは、「<引数>を与えて<ロケーション>の<コマンド>の実行する」ことです。
- 送信コマンドと応答データの文字形式は UTF-8 です。
- 送信コマンドの最後のヌル終端文字(‘¥0’)が届くと、vizrt はコマンドの実行を開始します。
  - ✧ 送信コマンドの終端を'¥0'を付加する以外にも、接続ソケットが閉じる際に'¥0'を受信されて実行されることもあります。
- 複数の送信コマンドを**ヌル終端文字(‘¥0’)で連結してから送信**する方が**高速**に処理出来ます。
- 取得系コマンドを**ヌル終端文字で複数連携した場合、応答データもヌル終端文字で連結**されて返されます。
  - ✧ サムネイル画像には終端文字は付加されず、応答データもそのまま連結されますので連結せずに送る方が処理しやすいと思います。
- 大文字/小文字が区別されるのでご注意ください。
  - ✧ 例) DirName 、 dirName 、 dirname 、 DIRNAME　は、すべて異なる名前です。
- 接頭辞番号を忘れずにつけてください。
- すべての何かを取得するコマンドには、0(ゼロ)、または正の接頭辞番号が必要です。
- REND は RENDERER の短縮形です。

<接頭辞番号>

- 送信コマンドの先頭に 0 (ゼロ) 以上を付けると、応答が返されます。
- 応答データには、送信コマンドの接頭辞番号と同じ番号が先頭に設定されて返されます。
  これは、送信したコマンドと応答データの対応に使用します。
- 先頭に-1 以下を指定すると、応答はありません。
  これは、応答データが必要無く、連続してコマンドを送信する場合に使用します。

<ロケーション>

- vizrt レイヤー・シーン、オブジェクト・マテリアル・フォント・イメージ・コンテナ名等で指定されたプロパティ(クラス)です。
- ロケーションによって使用できるコマンドが異なります。
- ロケーション内の Asset パスは、”/”で区切ってフルパスで指定します。
- ロケーション内のプロパティは”*”で区切ります。

例) 全ディレクターをタイムラインの先頭に戻す

```
SCENE*Photron/Tutorial*STAGE   SHOW   0.0
                ↑               ↑      ↑
            ロケーション          コマンド  引数
                          ※CStage クラスの”SHOW”コマンドに、引数”0.0”を与えて実行
```

＜付録①＞**VIZ コマンドの調べ方**

▲目次▲

① コマンドコンソールで確認

- vizrt GUI 上で値を変化させたりしながら、コマンドコンソールから該当しそうなコマンドを読み取ります。

② COMMAND_INFO コマンドで確認

- 特定のロケーションで使用できる「コマンド・プロパティと引数」の説明文を取得します。
- vizrt 付属のコマンドリファレンスに載っていないコマンドも確認できます。
- コマンド構文

送信コマンド　：<ロケーション>[20h]COMMAND_INFO[20h]<CMD 名>[20h] <フィルター> [00h]

※[20h]　：半角スペース　　※[00h]　：終端文字(¥0)

✧ <CMD 名>：ロケーションの取得したいコマンド名

ワイルドカードとして、* も使えます。省略時は*となります。
実際には、ほとんど * での使用だと思われますので、フィルターを使用しない場合は省略できます。
フィルターを指定する場合は、省略できません。

✧ <フィルター>　:以下のいずれかを設定します (省略時は ALL)。

ALL	：特定ロケーションのコマンド・プロパティを全て取得	
COMMANDS	：特定ロケーションのコマンドを全て取得	
PROPERTIES	：特定ロケーションのプロパティを全て取得	
ONE_LEVEL	：特定ロケーションのコマンド・プロパティを取得	(派生クラスのみ)
ONE_LEVEL_COMMANDS	：特定ロケーションのコマンドを取得	(派生クラスのみ)
ONE_LEVEL_PROPERTIES	：特定ロケーションのプロパティを取得	(派生クラスのみ)

✧ vizrt 付属のコマンドリファレンスの継承分を[含む・除外]を指定して、コマンド・プロパティと引数がの説明が取得できます。

例)

CStage クラスの コマンド・プロパティ　：REND*STAGE COMMAND_INFO * ALL
CStage クラスの コマンドのみ　：REND*STAGE COMMAND_INFO * COMMANDS
CStage クラスの プロパティのみ　：REND*STAGE COMMAND_INFO * PROPERTIES

③ vizrt 付属のコマンドリファレンスで確認

- ドキュメント格納場所

<VizEngine インストールパス>¥Documentation¥viz-cmd-doc-webhelp¥index.html

- 各クラスには 2 種類のリンクからアクセスできます。

●Top Locations　：コマンドロケーションの最上位からリンク

✧ 最上位は MAIN ロケーション(CMain クラス)

●Class Index　：クラス別

✧ 本ドキュメントの(クラス参照先)を元に調べるには、この Class Index からアクセスしてください。
✧ 各クラスでは、そのページの内容に加えて、継承元(Inherits)クラスのコマンドとプロパティが使用できます。
✧ 各 Commands の Arguments 列内は、引数が複数行にわたって、指定順に並んでいます。
※ 但し、一部順序を間違えて記載している物があります。エラーが出る場合は順序を変えてください。
（間違いがあるのは、使用頻度が低そうなコマンド。例えば、疑似マウス入力。他にもあるかもしれません）
✧ また、Arguments 列の左側に引数の型、右側に引数の意味やデフォルト値が記載されています。
※ 引数の意味の記載は、「書いてあるものもある」という程度です。
✧ **ñ** : Arguments 列や Info 列にあり、「無い」という意味です。

＜付録②＞VizCommand Console（コマンドコンソール）

▲目次▲

※VizCommand Console の手動での表示方法は、vizrt artist マニュアルをご覧ください。

コンソール用送信コマンド

- コマンドコンソールから vizrt にコマンドを送信するには、次のように接頭辞”send”を使用します

  コンソール送信コマンド　　: send[20h]<コマンド>[CRLF]

  ✧　[20h]: 半角スペース　　[CRLF]: 改行コード

  ➢ 例) バージョン番号の表示

  send 1 VERSION [+ENTER キー押下]

  応答データ　　: 1 Version: 5.2.1.60000

  コンソール表示　: CONSOLE: answer < Version: 5.2.1.60000 >

  ✧　コンソールには、接頭辞番号が外され、応答データのみが表示されます

その他のコンソール用コマンド

- コンソールクリア　　: clear　[+ENTER キー押下]
- コンソールクリア　　: cls　[+ENTER キー押下]
- ログの作成　　: clog　[+ENTER キー押下]

  ✧　ログファイル名はコマンドコンソールの次行に file 'xxxxxx.log' written to disk として表示
- コマンドコンソールヘルプ　　: ?　[+ENTER キー押下]

  ✧　? (半角のクエスチョン記号)

コマンドコンソールの簡易編集モードの解除手順

- コマンドコンソール上でマウスをクリックしたら、簡易編集モードになります。
- 簡易編集モードになると、Enter でモードを解除するまで VizEngine はコンソールにメッセージを表示できなくなります。
- その影響でレンダリングも止まるので見かけ上**フリーズしたように**なります。
- **右クリックで解除**できます。

- 「簡易編集モードにさせない」方法もあります。

  ➢ VizEngine コンソールが簡易編集モードにならないようにする手順

  (1)　該当 VizEngine コンソールのタイトルバー上でマウス右ボタンクリック

  (2)　コンテキストメニューでプロパティを開く

  (3)　簡易編集モードにチェックを入れて、OK ボタンをクリック

  ※ON→OFF にした場合は、再起動もしくは、Console に何か追記された以降、適用されます。

  ※OFF→ON にした場合は、即時適用されます。

＜付録③＞**用語解説や補足**

▲目次▲

RENDERER

（クラス参照先：RENDERER_BASE ・ RENDER_EDITOR ・ SCENE ・ SCENETREE）

- RENDERER は **REND** と省略できます。**SCENE_RENDERER** とも書けます。
  - ✧ RENDERER / REND / SCENE_RENDERER ：RENDER_EDITOR
  - ✧ SCENE_RENDERER でも RENDER_EDITOR メンバにアクセスが可能です。
- Action-Keyframe や VizScript から内部コマンドを実行するには、REND を **THIS_SCENE** に置きかえてください。
- コマンドコンソールの send コマンドを使用する場合、REND を **MAIN_SCENE** に置き換えればエディタモードでも実行可能です。
- GFX チャンネル経由でのコマンド構文
  - ✧ GFX チャンネルのシーンに対してコマンドを送るには、REND の後に"GFX*<GFX チャンネル番号>*"を差し込みます。

    例) REND*GFX*16*TREE*$ALL*FUNCTION*ControlObject*in SET ON text SET XXXXX

レイヤー

（クラス参照先：RENDERER_BASE）

- vizrt には、Back layer、Main layer(Middle layer)、Front layer の 3 つのレイヤーがあります。
- 但し、Back layer ・ Front layer は、vizrt5.2 までは Classic Renderer モードでのみ使用できます。
  Viz Engine Renderer モードでは vizrt5.3 以降での対応となります。以下は、5.2 までの場合です。
  - ✧ Layer Manager は Classic レンダラーしか機能しません。
  - ✧ [Config] -[Scene Dafault Values] - [Render Engine]が"Viz Engine Renderer"になっていると、何もレンダリングされてない状態が Viz Engine レンダラーになり、Classic 用シーンをレンダリングするまでレイヤーが使えません。
  - ✧ Viz Engine Renderer を使う機会が少ない場合は、Config 設定で Classic Renderer にしておいた方が良いかもしれません。
- Classic レンダーモードの時は、シーンを layer ごとに操作できます。

  例) シーン ロード

  Main layer : REND SET_OBJECT SCENE*<シーン名>

  Front layer : REND*FRONT_LAYER SET_OBJECT SCENE*<シーン名>

  Back layer : REND*BACK_LAYER SET_OBJECT SCENE*<シーン名>
  - ➢ Back layer ・ Front layer は RENDER_BASE クラスのプロパティであるため、Main layer の構文と異なります。

コンテナ

（クラス参照先：SCENETREE - Properties）

コンテナは以下の 4 通りの方法で指定できます。

- #オブジェクト ID ：vizrt オブジェクト管理番号
  - ✧ シーン起動毎に割り当てられます。
- [$コンテナ名]$コンテナ名 ：コンテナ階層パス
  - ✧ [$コンテナ名]は、コンテナ名がユニークであれば省略可能。
  - ✧ 階層を検索する為、時間がかかります。
- @コントロールチャネル名 ：※後述
  - ✧ 高速にアクセスできます。
- 番号/番号/番号 ：ツリー構造の番号によりアクセスします。
  - ✧ 例）2/2 ・・・ 「上から 2 番目」の「子グループの 2 番目」のコンテナ

コントロールチャネル

▲目次▲ ▲コントロールチャネル▲

- コントロールチャネルは、外部制御用にノード (コンテナなどのオブジェクト、プロパティなど)にリンクする方法です。
- 事前にシーン内で作成する必要があります。
- ノードを Control Channels Editor にドラッグ＆ドロップして名前を付けるととリンクが確立されます。
  以降、このコントロールチャネルを使用してノードにアクセスできるようになります。
- オブジェクト ID でのアクセスと異なり、シーンを再起動しても、リンクはそのまま残ります。
- コンテナ階層でのアクセスと異なり、後で階層を変更しても、リンクはそのまま残ります。
- コントロールチャネルの命名規則
  @コントロールチャネル名　　例) @NodeControl

- **キーフレームに VIZ コマンドでコントロールチャネルを付ける方法**

※キーフレームへの制御コマンドは構文が長くなりがちなので、コントロールチャネルだと短く出来ます。

➢ コントロールチャネル適用手順 [vizrt 5 以降]

✧ Viz5 では KeyFrame Editor からのドラッグ＆ドロップが出来なくなりました。

1．KEY フレームに対して、コントロールチャネルを追加するコマンドで作成
2．そのコントロールチャネル名に対してコマンドで値をセット
例）キーフレームの Position に xyz というコントロールチャネル名を割り当てる
使用　　: -1 REND*TREE*@xyz*VALUE SET 100 200 300
不使用　: -1 REND*TREE*$Cube*ANIMATION*Position*KEY*$xxx*VALUE SET 100 200 300

✧ X/Y/Z 個別にコントロールチャネルの割り当ても可能になりました。

➢ コントロールチャネル適用手順 [vizrt 3 まで]

1．ControlChannel エディタをフローティングで開く
2．キーフレームの設定を開き、各パラメータの名前の部分を ControlChannel エディタにドラッグ＆ドロップ
3．そのコントロールチャネル名に対してコマンドで値をセット
例）キーフレームの Position に xyz というコントロールチャネル名を割り当てる
使用　　: -1 REND*TREE*@xyz SET 100 200 300
不使用　: -1 REND*TREE*$Cube*ANIMATION*Position*KEY*$xxx*VALUE SET 100 200 300

✧ X/Y/Z 個別にコントロールチャネルの割り当ては出来ません。

ディレクター

(クラス参照先：CDirectorTree - Properties)

VIZ コマンド(ロケーション)でのディレクターは、以下の 6 通りの方法でアクセスできます。
※本ドキュメントは、基本、<$ディレクター名>のコマンドで記載しています。

例)　Default ディレクターの開始

- <ディレクター名フルパス>　　：REND*STAGE*DIRECTOR*Default START
- <$ディレクター名フルパス>　　：REND*STAGE*DIRECTOR*$Default START
- </ディレクター名フルパス>　　：REND*STAGE*DIRECTOR*/Default START
- <$ディレクター名>　　：REND*STAGE*$Default START
- </ディレクター名>　　：REND*STAGE*/Default START
- <#オブジェクト ID>　　：#7365 START　※Default ディレクターのオブジェクト ID=#7365 の場合)

ポイント(STOP/TAG)

（クラス参照先：CDirector - Properties）

VIZ コマンド(ロケーション)でのポイントは、以下の 2 通りの方法でアクセスできます。

※本ドキュメントは、基本、<＄ポイント名>のコマンドで記載しています。

例） Default ディレクターの Stop1 ポイントを f200 に移動

- <$ポイント名> ： REND*STAGE*$Default*KEY*$Stop1*TIME SET f200
  - ポイント名の＄は必須です。
- <#オブジェクト ID> ： #7372*TIME SET f200 ※Default ディレクターのオブジェクト ID=#7372 の場合

アクション

（クラス参照先：CDirector - Properties）

VIZ コマンド(ロケーション)でのアクションは、以下の 5 通りの方法でアクセスできます。

※本ドキュメントは、基本、<ACTION 名>のコマンドで記載しています。

例） Default ディレクターの act アクションに Keyframe を追加

- <ACTION 名> ： REND*STAGE*$Default*ACTION*act ADD f300
- <ACTION＄名> ： REND*STAGE*$Default*ACTION*$act ADD f300
  - ACTON の場合、アクション名の＄はあっても無くても、どちらでも構いません。
- <ACTION_CHANNEL＄名> ： REND*STAGE*$Default*ACTION_CHANNEL*$act ADD f300
  - ACTON_CHANNEL の場合、アクション名の＄は必須です。
- <CHANNEL$名> ： REND*STAGE*$Default*CHANNEL*$act ADD f300
  - CHANNEL の場合、アクション名の＄は必須です。
- <#オブジェクト ID> ： #4347 ADD f200 ※act アクションのオブジェクト ID=#4347 の場合

コンテナアニメーション（チャンネル）

（クラス参照先：CDirector - Properties）

VIZ コマンド(ロケーション)でのチャンネルは、以下の 5 通りの方法でアクセスできます。

※本ドキュメントは、基本、<TREE 階層>のコマンドで記載しています。

例） コンテナ表示 ONOFF ループアニメの ONOFF(@aaa：コントロールチャネル、 Default：ディレクター、 Active：チャンネル)

- <TREE 階層> ： REND*TREE*@aaa*ANIMATION*Active*LOOP SET 1
- < ANIMATION_CHANNEL＄名> ： REND*STAGE*$Default*ANIMATION_CHANNEL*$Active*LOOP SET 0
  - ANIMATION_CHANNEL の場合、チャンネル名の＄は必須です。
- <CHANNEL$名> ： REND*STAGE*$Default*CHANNEL*$Active*LOOP SET 0
  - CHANNEL の場合、チャンネル名の＄は必須です。
- <#オブジェクト ID> ： #4177*LOOP SET 0 ※Active チャンネルのオブジェクト ID=#4177 の場合

キーフレーム(ACTION)

（クラス参照先：CChannelBase - Properties）

VIZ コマンド(ロケーション)でのキーフレームは、以下の 4 通りの方法でアクセスできます。

※本ドキュメントは、基本、<KEY 名>のコマンドで記載しています。

例） キーフレーム時間の取得 (Default：ディレクター、 act：アクション、 key1：キーフレーム[=最初のキーフレーム])

- <KEY 名> ： REND*STAGE*$Default*ACTION*act*KEY*key1*TIME GET
- <KEY$名> ： REND*STAGE*$Default*ACTION*act*KEY*$key1*TIME GET
- <KEY 並び順番号> ： REND*STAGE*$Default*ACTION*act*KEYN*0*TIME GET
- <#オブジェクト ID> ： #7667 START （例: キーフレームのオブジェクト ID=#7667 とする）

プラグイン名/プラグイン変数名の調べ方

※外部制御コマンドで使用するプラグイン名やプラグイン変数名は、Built-Ins パネルや Properties パネルに表示されている名称とは異なります。

- 制御名称の調べ方は、以下の 5 通りあります。
  1．VizCommand Console で任意のパラメータを変化させ表示されるコマンドから、プラグイン名やプラグイン変数名を読み取る。
     ✧ 手数も多く、読み取りも面倒ですが、コピー＆ペーストが可能です。

  2．Properties パネルの表示名をドラッグし、薄っすらとグレーポップアップ表示された名称で確認する。
  <プラグイン名> : Properties パネルの「プラグインアイコン」をドラッグしてください。
  <プラグイン変数> : Properties パネルの各「パラメータタイトル」をドラッグしてください。
     ✧ 簡単ですが、コピー＆ペーストは出来ません。
     ✧ このポップアップは、マウスオーバーでのツールチップの事ではありません。ツールチップは白枠長方形です。
     ✧ ボタンは、ドラッグする場所が無い為、この方法は使用できません。

  3．COMMAND_INFO コマンドで取得する。
     ✧ 外部制御での取得コマンドで、プラグイン情報を全て取得します。難易度高めです。
     ✧ コマンドの構文は、「付録①：VIZ コマンドの調べ方」をご覧ください。

  4．「DYNAMIC_PARAMETER GET」コマンドで取得する。
     ✧ 外部制御での取得コマンドで、プラグイン情報を全て取得します。難易度高めです。
     ✧ コマンドの構文は、「◎vizrt プラグイン関連のコマンド」をご覧ください。

  5．「BUILT_IN*PARAMETER GET」コマンドで取得する。
     ✧ 外部制御での取得コマンドで、プラグイン情報を全て取得します。難易度高めです。
     ✧ コマンドの構文は、「◎vizrt プラグイン関連のコマンド」をご覧ください。

- 注意事項
  ➢ 制御用のプラグイン名とプラグイン変数名は、大文字・小文字が区別されます。

Script とスクリプトベースプラグイン(VSL)

- Container Script を他のコンテナでも利用したい、Scene Script を他のシーンでも使いたいなどの時に、スクリプトベースプラグイン(VSL プラグイン)を作成することができます。
- VSL 化すると BUILT IN プラグインのように使用できます。
  但し、外部制御コマンドを Script 用コマンドからプラグイン用コマンドに変更する必要があります。
- 作成手順
  ➢ Script Editor の「Add as Plugin」ボタンをクリックしてください。
  「Add as Plugin」が無効の場合は、一度「Stop Script」から「Compile and Run」を実行すると有効化されます。
  ➢ 名前入力ダイアログに PluginName を入力し OK ボタンをクリックすると、Plugin プールの[ScriptPlugins]フォルダに作成されます。
  ➢ コンテナスクリプトの場合、SceneTree の Script アイコンを Plugin プールに適当にドラッグしても作成できます。
  但し、Plugin プールのどこにドラッグしたとしても、[ScriptPlugins]フォルダに作成されます。
- 作成されたプラグインは、<viz data folder>¥Scriptplugins フォルダに"プラグイン名".vsl で保存されます。
  例）C:¥ProgramData¥vizrt¥VizEngine¥ScriptPlugins
- ゴミ箱にドラッグすれば、PluginPool から削除できます。
- VSL プラグインのソースコードは、プラグインを Plugin プールから Script Editor にドラッグすることで復元されます。
  但し、ドラッグしただけでは、vizrt が ScriptEditor の変更を感知できませんので以下の方法で vizrt に変更を知らせる必要があります。
  ➢ ① 展開されたソースコードを、一旦、コピー＆ペーストする。
  ➢ ② コメント部分に何か文字を入力する。文字はなんでも構わない。
  ※VSL をドラッグした直後、vizrt が変更を感知する前に Scene を保存すると、展開されたコードは保存されません。

＜付録④＞**Shared Memory (SHM)について**

▲目次▲

- シェアードメモリ(SHM) は、文字列によってインデックスされたユーザー定義の変数を持つマップです。
- この変数は Variant オブジェクトで、他の**どの型のオブジェクトでも保持**できます。
- SHM は格納位置により、アクセスできる範囲が 3 種類あります。
  - レンダリング中のシーン内でのみアクセス可

    名称　　: Scene
    VizScript 内名称　　: Scene.Map
    コマンド名称　　: SCENE / REND / RENDERER

  - vizrt 単体のメモリ内にロードされているシーン間で、マップを共有

    名称　　: Global
    VizScript 内名称　　: System.Map
    コマンド名称　　: GLOBAL

    ※VizScript とコマンドで異なるのでご注意ください。

  - 同じ GraphicHUB に接続する vizrt 間で、マップを共有

    名称　　: Distributed
    VizScript 内名称　　: VizCommunication.Map
    コマンド名称　　: VIZ_COMMUNICATION

- Shared Memory (SHM)の使い方
  - SHM を経由してシーン内のコンテナに値を適用する方法は 2 通りあります。
    - Viz Script を使う
    - Apply Shared Memory (Apply SHM)プラグインを使う
  - SHM の値を操作する方法は 4 通りあります。
    - Viz Script
    - 外部制御コマンド（コマンドシステム）
    - 外部制御コマンド（TCP/UDP ダイレクト送信）　※値の設定のみ。値の取得にはコマンドシステムを使用。
    - Rest Webservice の Web ページで確認や編集

＜付録④. A＞**VizScript で SHM を操作**

- VizScript では、SHM に書き込みと読み込みが出来ます。
- 「VizScript 内では SHM 値の読み込み、SHM への書き込みは外部制御」の場合が多いかもしれません。
- 書き込み手順
  - Script Editor で、書き込みたいタイミングのコールバックの場所に下記を記述してください。

    特定シーン内　　：Scene.Map[“<SHM Key 名>”] = <値>
    ローカル vizrt 内共有　　：System.Map[“<SHM Key 名>”] = <値>
    vizrt 間共有　　：VizCommunication.Map[“<SHM Key 名>”] = <値>

    ※ コールバック：OnMouseMove / OnButtonUp / OnMTHit / OnExecPerField / OnExecAction / etc…
    ※ 何らかの初期化が必要な場合は、OnInit を使います。

- 読み込み手順
  - まず OnInit で監視する変数名を、RegisterChangedCallback で登録します。
    - この記述を忘れると、OnSharedMemoryVariableChanged コールバックが呼ばれません。ご注意ください。
      ※コールバックやメソッドの詳細については VizScript マニュアルをご覧ください。

```
例） sub OnInit()
        Scene.Map.RegisterChangedCallback(“<SHM Key 名>”)                    ‘ 特定シーン用
        System.Map.RegisterChangedCallback(“<SHM Key 名>”)          ‘ ローカル vizrt 内共有用
        VizCommunication.Map.RegisterChangedCallback(“<SHM Key 名>”)          ‘ vizrt 間共有用
    end sub
```

  - 次に OnSharedMemoryVariableChanged コールバックで SHM を読み込みます。
    - これは、登録した SHM 変数の値が変化したら呼ばれます。
    - OnSharedMemoryVariableChanged には、SHM マップ参照と、値が変化した変数名が渡されています。

```
例）textA という SHM Key の SHM 値(文字列)を表示
    sub OnSharedMemoryVariableChanged(map As SharedMemory, mapKey As String)
        if mapKey = "textA" then
            dim text as Container
            text = Scene.findContainer(“<Container Name>”)
            text.Geometry.Text = map[mapKey]
        end if
    end sub
```

＜付録④. B＞**Apply Shared Memory(Apply SHM)プラグインで SHM を操作**

- Apply SHM プラグインが適用されたコンテナの子階層に存在する Control プラグインにのみ、値を適用します。
- 値は外部制御等で SHM に送信されます。
  ※'/'が入力できない為、「Viz Engine REST Interface」からは入力出来ません。
- 受け側として、事前に Apply SHM プラグインに設定を行ってください。
- ControlText を使用した作成手順例
  1．(親) Group コンテナに **Apply SHM プラグイン**を追加
  2．(子) その子コンテナに Text を追加し、**ControlText プラグイン**を追加
  3．下記 Apply SHM プラグインの設定を行う
  4．下記 ControlText プラグインの設定を行う
  5．外部から**コマンドシステムや TCP/UDP ダイレクト通信**で、SHM に値をセット

- Apply SHM プラグインの設定
  - **Shared Memory Source**
    - [Scene / Global / Distributed / Inactive]の中から、対象の SHM を選択します。
  - **Memory Base Key (MBK)**
    - 送信コマンド側では'/'で始まる SHM Key のルートにあたる、プレフィックスパスです。
      例）/stocks/nyse/ibm , /all
      ※ プラグイン側の設定では'/'で始まってなくても良いですが、送信コマンド側には必須です。統一して付ける方が望ましい。
  - Expose Base key
    - MBK プロパティを動的に変更したい時に使います。**一般的な使い方ではオフ**にしてください。
      ※MBK プロパティを動的に変更とは、外部制御で MBK 自体を変更するという事です。
    - 動的変更用のフィールド ID と説明を使用可能にします。
      ※例えば、Apply SHM プラグインの設定が、
      Shared Memory Source ： Scene
      MBK ： /old_mbk
      Expose Base key ： ON
      Field Identifer ： ex_name
      の時、MBK を"/old_mbk"から"/new_mbk"に変更したい場合、
      -1 REND*MAP SET_STRING_ELEMENT /old_mbk/ex_name /new_mbk
      というコマンドを送信して変更します。ApplySHM 自身も子階層の対象になるイメージです。
      ※一度変更した MBK を同じパス名には戻さないでください。vizrt が落ちる場合があります。

- ControlText プラグインの設定
  - Field Identifer ：フィールド ID
    - ※フィールド ID は、**'/'で始まると絶対パス**とみなされ、**無いと相対パス**とみなされます。
      - フィールド ID が絶対パス： MBK の値が何であれ、その絶対パスのみが SHM キーとなります。
        例） MBK(/hibino) + ID(/xyz/value) → SHM キー (/xyz/value)
      - **フィールド ID が相対パス**： MBK 基準となり、フィールド ID と合わせた結果が、SHM キーとなります。
        - **※一般的にはこちらを利用**します。
          例） MBK(/hibino) + ID(value) → SHM キー (/hibino/value)

      ※ 参考）コマンドコンソールからの SHM 値セットのコマンド例
      send -1 REND*MAP SET_STRING_ELEMENT /hibino/value aiueo

＜付録④. C＞**外部制御（コマンドシステム）で SHM を操作**

- デフォルト設定では、6100/TCP のコマンドシステムでアクセスします。
- コマンドシステムの拡張機能として、別のポートも追加可能です。
  [Config] - [Comunication] - [Global]タブ - [Additional Comunication] - [None / udp / udp&Multicast]
  - 但し、この拡張機能を使用するなら、TCP/UDP ダイレクト通信を検討してください。
  - Multicast は、「ネットワークが汚れる」という理由で、vizrt 非推奨です。
  - 遅延等の理由で「vizrt 間共有よりも各端末を直接制御」を推奨
- コマンドシステムの特徴
  - メリット　：デフォルト設定で気軽に使える、 通信安定性が高い
  - デメリット　：速度が遅い、 レンダリングに影響を与えてしまう

速度	（速）	SHM UDP	＞	SHM TCP	＞	**コマンド**	（遅）
通信安定度	（高）	SHM TCP	＝	**コマンド**	＞	SHM UDP	（低）
レンダリング影響度	（少）	SHM UDP	＝	SHM TCP	＞	**コマンド**	（多）

- コマンド構文は、前述「Shared Memory (SHM)関連のコマンド」を参照

＜付録④. D＞外部制御(TCP/UDP ダイレクト送信)で SHM を操作

- コマンドシステムとは別の手段として、TCP/UDP 通信で直接値を設定できます。
- 但し、取得は出来ません。取得にはコマンドシステムを使用してください。
- Pilot では¥0 を送信できない為、この方法は使えません。コマンドシステムでのみ操作可能です。
- 送信構文は、UDP / TCP 共通です。
- TCP/UDP ダイレクト通信の特徴
  - メリット　：速度が速い(UDP)、 通信安定性が高い(TCP)、 レンダリングに影響が少ない
  - デメリット　：有効化が必要、 コマンド型が特殊、 Pilot で使えない

速度	(速)	**SHM UDP**	＞	SHM TCP	＞	コマンド	(遅)
通信安定度	(高)	**SHM TCP**	＝	コマンド	＞	SHM UDP	(低)
レンダリング影響度	(少)	**SHM UDP**	＝	**SHM TCP**	＞	コマンド	(多)

- 一般送信構文

  特定シーン内　：SMM#<シーン ID>_SetValue|<SHM Key 名>|<値>¥0
  ローカル vizrt 内共有　：SMMSystem_SetValue|<SHM Key 名>|<値>¥0
  vizrt 間共有　：SharedMemoryMap_SetValue|<SHM Key 名>|<値>¥0

- Apply Shared Memory プラグイン専用 送信構文

  特定シーン内　：SMM#<シーン ID>_SetValue|<MBK/ID>|<値>¥0
  ローカル vizrt 内共有　：SMMSystem_SetValue|<MBK/ID>|<値>¥0
  vizrt 間共有　：SharedMemoryMap_SetValue|<MBK/ID>|<値>¥0

  ※一般送信コマンドの<SHM Key 名>の箇所が<MBK/ID>になります。

- ‘|’の前後に半角スペースは含めないでください。半角スペースが SHM Key や値に含まれてしまいます。
- 複数送る場合は、<SHM Key 名>|<値>だけではなく、送信構文全体を複数つなげてください。
- 複数つなげる場合は、’¥0’ (NULL 文字)を区切り文字として連結します。
- TCP/UDP ダイレクト通信はデフォルト無効ですので、有効化の設定と Vizrt の再起動が必要です。

  TCP/UDP ダイレクト通信　有効化手順

  1．[VizConfig] – [Comunication] – [Shared memory]タブを開きます。
  2．使用する通信プロトコルに応じて、ポート番号を入力してください。使用しない場合は”0”にしてください。

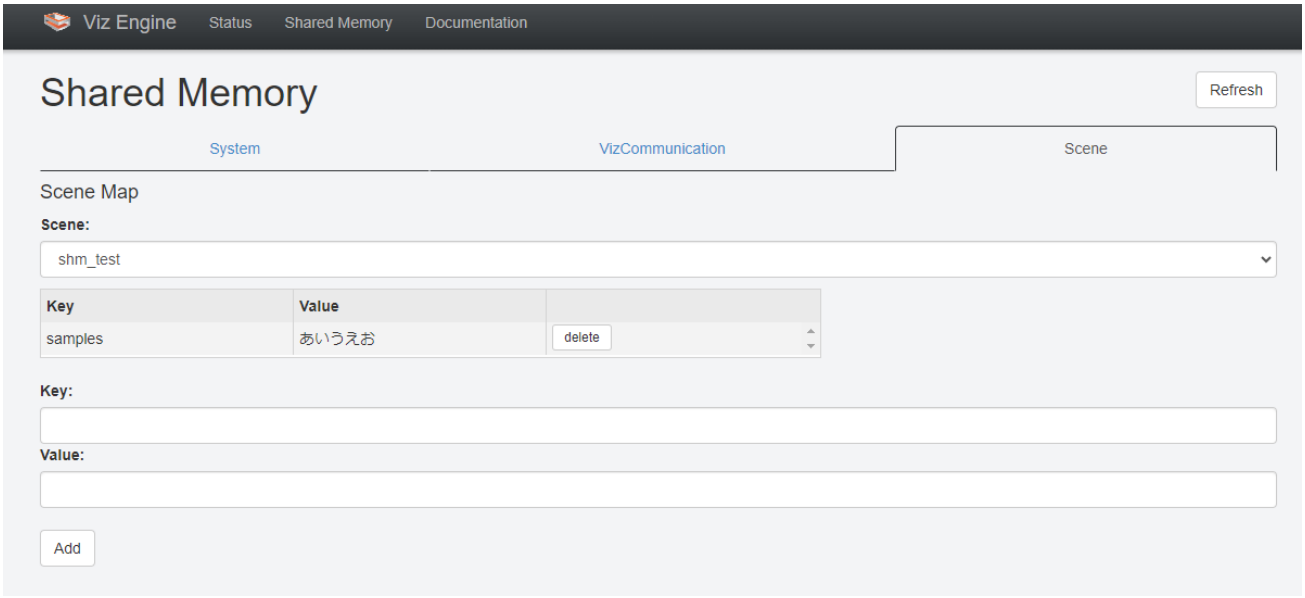
- TCP/UDP 外部制御で Debug する場合、vizrt コマンドコンソールに通信情報を表示できます。
  1．[VizConfig] – [Comunication] – [Shared memory]タブ – [Debug]のチェックを入れてください。

  例) Command Console　>> SHM(TCP): client '127.0.0.1' received 37 bytes

  ※　SHM(TCP)・・TCP or UDP、　client 'xxx'・・クライアントアドレス、　received xx byte・・受信バイト数

＜付録④. E＞**Shard Memory の格納データを確認する方法**

- Rest Webservice を使用すれば、ブラウザで現在の Shared Memory に格納されている Key と値を確認できます。
- Rest Webservice 上での SHM の追加や変更
  - ✧ VizScript で SHM をコンテナに適用している場合は、変化の挙動を確認できます。
  - ✧ Apply SHM プラグインで SHM をコンテナに適用している場合は、確認のみで、変更は出来ません。
- この機能は、デフォルトで無効です。
  使用する為には、事前に Config 設定で有効化する必要があります。
- 使用手順
  - ➢ [事前準備]
    1．[VizConfig]–[Comunication]–[Global]タブを開きます。
    2．下の方にある「REST Webservice」を確認します。　（※dafault port : 61000 ）
    3．[Install] ボタンをクリックしてください。
    4．[Save] ボタンをクリックして、Vizrt を再起動してください。
  - ➢ [確認手順]
    1．Web ブラウザで「localhost:61000」で検索します。
    2．画面上部メニューの[Shared Memory] をクリックしてください。
    3．[System | VizCommunication | Scene] から、目的の SHM を選択してください。
    ※Scene の場合、「ターゲットシーンの選択」を忘れないでください。異なるシーンを選んでいると表示されません。
    4．[Refresh] ボタンで最新の状態を確認できます。



＜付録⑤＞**Control Object について**

▲目次▲　▲Control Object▲

外部制御から送信された値は、この ControlObject を経由して各 Control プラグインに届きます。

- コマンド構文
  - ➢ 送信コマンド　：REND*TREE*<ノード階層>*FUNCTION*ControlObject*in SET ON <Field Identifier> SET <値>
  - ➢ 応答データ　：無し
    - ✧ Control プラグインの種類を問わず、コマンド構文を統一
      <ノード階層>　：ControlObject プラグインが適用されているコンテナ階層
      ※SET 対象の各 Control プラグインの方ではありません
      <Field Identifier>：各 Control プラグインの Field Identifier
      <値>　：各 Control プラグインに応じた設定値
    - ✧ 複数の「Field Identifier と値」を連結して、１コマンドでセットすることも出来ます。
      <Field Identifier> SET <値>を**文字列の￥0** で区切ってください。
      例） REND*TREE*$group*FUNCTION*ControlObject*in SET ON key1 SET abc¥0key2 SET xyz¥0key3=aiueo¥0
      ※key1=abc、key2=xyz、key3=aiueo をセット。(最後の(文字列)の¥0 は合っても無くても構いません)
      ※ヌル終端文字('¥0')ではありません。**'¥0'はコマンド区切り文字**なので２つ目の SET 以降がコマンドエラーになります。
- 適用範囲
  - ➢ ControlObject プラグインを適用されたコンテナの子階層に存在する Control プラグインにのみ、値を適用します。
- ControlObject プラグインの自動追加
  - ➢ Control プラグインの配置の際、階層に ControlObject が無ければ、最上位階層コンテナに ControlObject が自動追加されます。
    - ✧ 必ず、ControlObject プラグインは１つのシーンに１つにしてください。同一シーン内に複数あると、誤動作の可能性があります。
    - ✧ Apply SHM の場合も同様に、先に Apply SHM を追加してあれば、Control プラグインを追加しても自動追加されることはありません。
- 使用手順　※例）制御対象が Text で、ControlText プラグインを使用
  1．(親) の準備。 Group コンテナに[Control] – [ControlObject]プラグインを追加
  ※ トラブル防止で ControlText プラグインより先に、明示的に ControlObject プラグインを配置
  2．(子) の準備。 親 Group の子コンテナを作成し、Text を追加
  ※ [Control] – [ControlText]プラグインを追加
  3．ControlObject コマンドで文字列をセット

- Control Object と固有コマンドとの比較例

①制御対象を「Text に文字列をセット」とします。

A）直接 Text にセット
コントロールチャネル　：REND*TREE*@name1*GEOM*TEXT SET abc
コンテナ名　：REND*TREE*$name*GEOM*TEXT SET abc

B）ControlObject (ControlText プラグイン)
コントロールチャネル　：REND*TREE*@ALL*FUNCTION*ControlObject*in SET ON name2 SET abc
コンテナ名　：REND*TREE*$object*FUNCTION*ControlObject*in SET ON name2 SET abc

②制御対象を「コンテナに X 位置をセット」とします。

A）直接コンテナに X 位置をセット
コントロールチャネル　：REND*TREE*@name1*TRANSFORMATION*POSITION*X SET 100.0
コンテナ名　：REND*TREE*$name*TRANSFORMATION*POSITION*X SET 100.0

B）ControlOject (Control Parameter プラグイン)
コントロールチャネル　：REND*TREE*@ALL*FUNCTION*ControlObject*in SET ON pos1 SET 100.0
コンテナ名　：REND*TREE*$object*FUNCTION*ControlObject*in SET ON pos1 SET 100.0

※（参考） Apply SHM 版の 「X 位置にセット」コマンドの場合
Apply SHM　：REND*MAP SET_DOUBLE_ELEMENT /ALL/pos1 100.0

- GFX チャンネルの ControlObject に標準構文でコマンドを送信する

➢ MAIN シーン側の ControlObject を経由して GFX チャンネルの ControlObject にコマンドを送れます。

① MAIN シーンの ControlObject プラグイン プロパティを設定します。

[ControlObject]のプロパティ

◎ Show advanced ：ON

◎ Enable GFX subscene ：ON

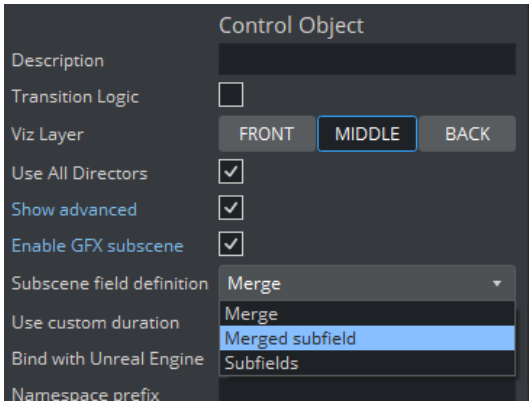
◎ Subscene field definition ：下記から選択

Merge： prefix を付けないで、MAIN と GFX での区別の無い<Field Identifier>を使います。
GFX 同士は同じ<Field Identifier>で構いませんが、MAIN と GFX で重なると GFX 側には適用されません。
※図.A) Merge①（MAIN と GFX 間で同じ<Field Identifier>を付けた場合）
※図.B) Merge②（MAIN と GFX 間で異なる<Field Identifier>を付けた場合）
※例) text, gfxtext など

Merged subfield： prefix 文字列を付加し、全ての GFX に共通の<Field Identifier>となります。
prefix の無いものは MAIN の<Field Identifier>となります。
※図.C) Merged subfield
※例) gfx.text など

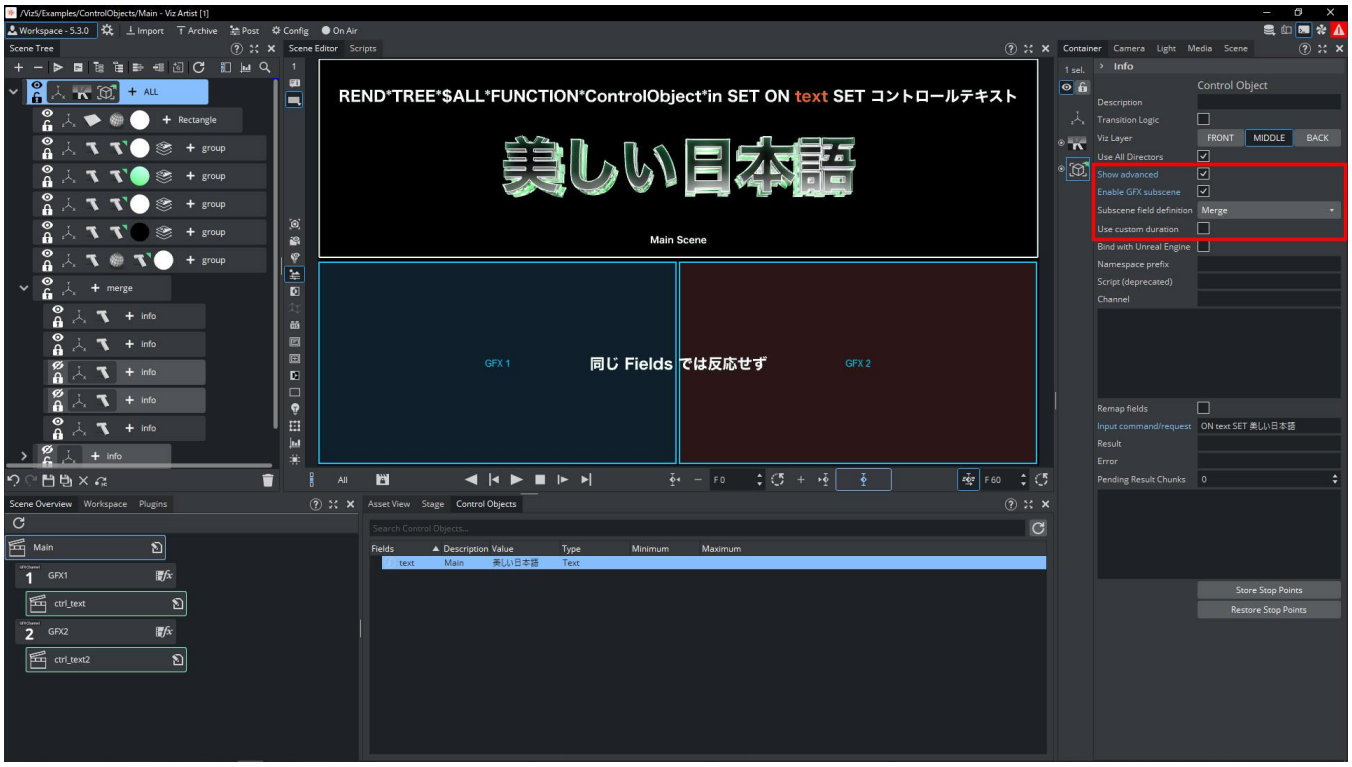
Subfields ： prefix として、prefix 文字列に GFX チャンネル番号が自動追加されたものを付加します。
GFX 別々の<Field Identifier>となります。
※図.D) Subfields
※例) gfx1.text、gfx3.text など

◎ Subscene field id/prefix ：付加する prefix 文字列を設定
※prefix を使用の際は、prefix と各<Field Identifier>を'.'(ドット)で連結します。



② MAIN シーンの ControlObject 一覧に GFX チャンネルの<Field Identifier>が追加で表示されます。
必要に応じて prefix も表示されます。階層で表示されるので、使用の際に**'.'(ドット)**で連結してください。
この表示されている Field Identifier を使い、MAIN シーンに対する標準構文コマンドで操作してください。

A）Merge①（MAIN と GFX 間で同じ<Field Identifier>を付けた場合）



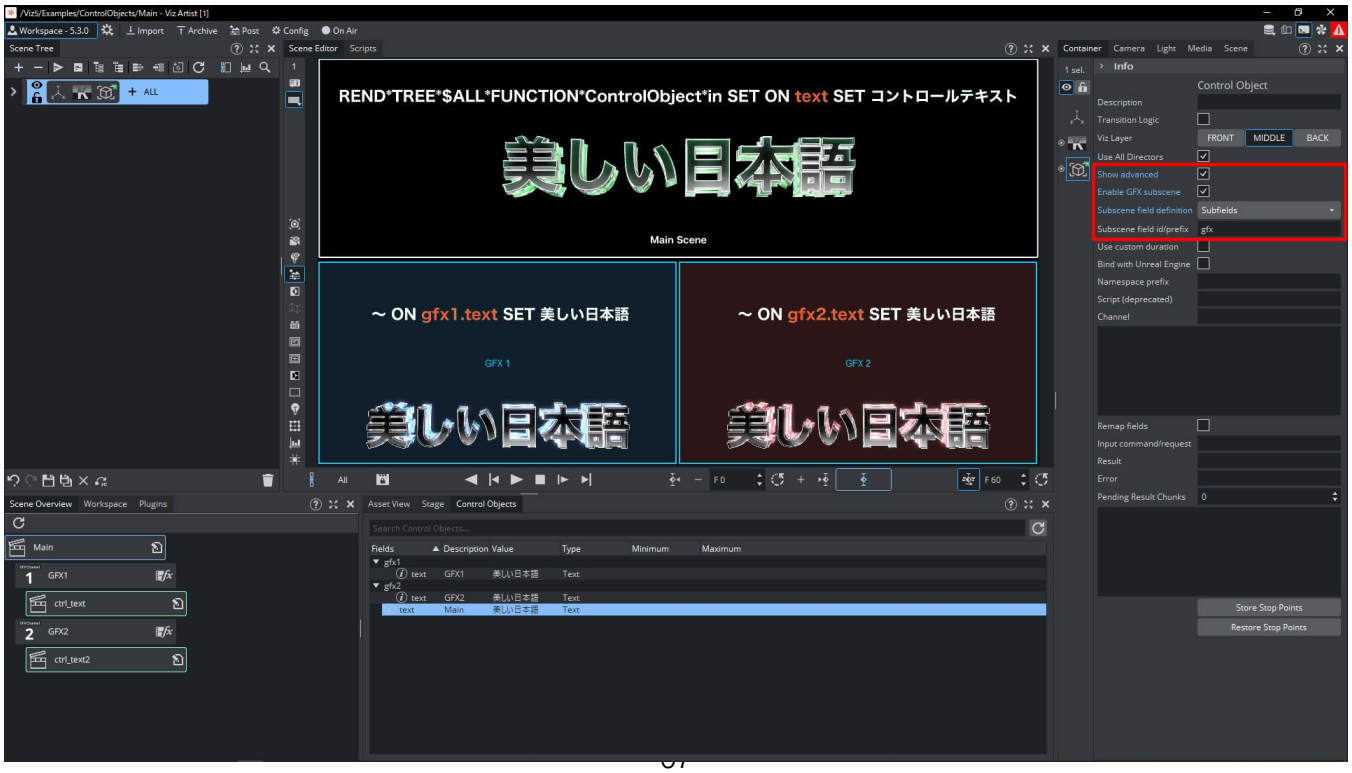
B）Merge②（MAIN と GFX 間で異なる<Field Identifier>を付けた場合）



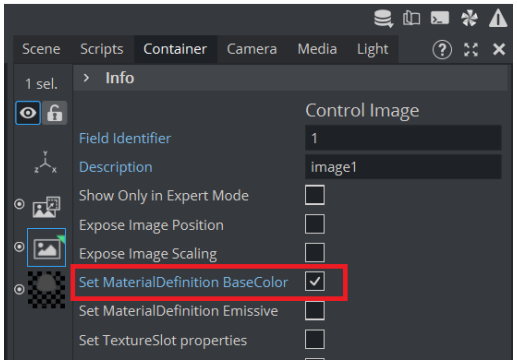
C) Merged subfield



D) Subfields



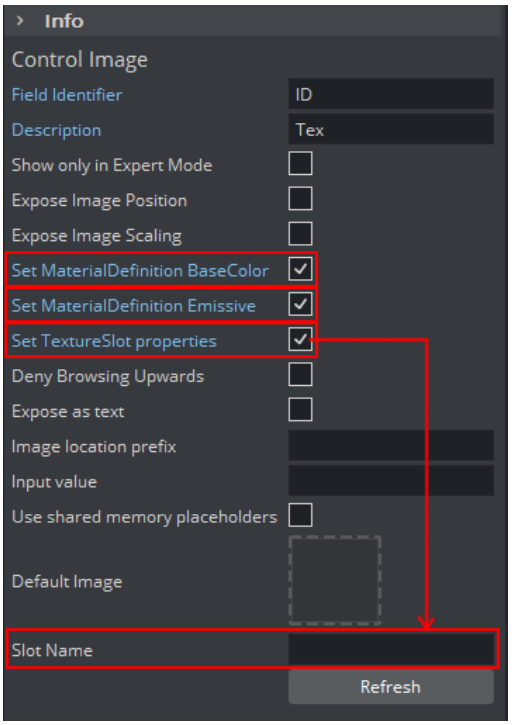
- Viz Engine Render での ControlImage の使用方法
  - [ControlImage] - [Set MaterialDefinition BaseColor]に
    チェックを入れると Classic Render と同様の動きになります



  - その他、マテリアル基テクスチャの貼り方にもよりますが、
    - ✧ BaseColor
    - ✧ Emissive（発光）

    に加え、
    別プラグインですが、TextureSlot というものがあります。

  - TextureSlot プラグインは、文字通り、プラグイン内でスロットを増やしてテクスチャーを重ねる事が出来ます。
    スロット毎に名前付けが可能で、下記のように TextureSlot のチェックを入れると下部に SlotName の項目が現れます。
    （TextureSlot プラグインについての詳細は、ここでは述べないので、別途、確認してください）



＜付録⑥＞Control Text 文字制御用タグ一覧

▲目次▲

No	項目	値
0	値を設定しない (前の文字の値のリセットにも使用)	</fo:wrapper>
1	文字間	<fo:wrapper letter-spacing="10 進数値">
2	文字 X スケール	<fo:wrapper scale-x="10 進数値">
3	文字 Y スケール	<fo:wrapper scale-y="10 進数値">
4	文字 X ポジション	<fo:wrapper pos-x="10 進数値">
5	文字 Y ポジション	<fo:wrapper pos-y="10 進数値">
6	文字 X ローテーション	<fo:wrapper rotate-x="10 進数値">
7	文字 Y ローテーション	<fo:wrapper rotate-y="10 進数値">
8	文字 Z ローテーション	<fo:wrapper rotate-z="10 進数値">
9	文字色(Classic)	<fo:wrapper color="#16 進数色コード">
10	透明度(Classic)	<fo:wrapper opacity="10 進数値">
11	文字色(VizRender)	<fo:wrapper color="#16 進数色コード(#ARGB)">
12	フォント	<fo:wrapper font="フォントパス">

- 各項目は順不同ですが、前の文字での記述順を参照しています。
- 前の文字に対してセットされた値を次の文字でリセットするには、</fo:wrapper>を挿入します。
  新しい値をセットする場合は、その後に続けて記述します。

※参考 (Control Text ： https://docs.vizrt.com/plugins-user-guide/5.3/Control_Text.html)

※使用例： 例えば、[ControlText - Input value] には、以下のように入力してください。

1. い<fo:wrapper scale-y="1.5"><fo:wrapper color="#FF0000">ろは
2. い<fo:wrapper scale-y="1.5"><fo:wrapper color="#FF0000">ろ</fo:wrapper>は
3. い<fo:wrapper scale-y="1.5"><fo:wrapper color="#FF0000">ろ</fo:wrapper></fo:wrapper>は
4. <fo:wrapper scale-x="0.5">い<fo:wrapper scale-y="1.5"><fo:wrapper color="#FF0000">ろ</fo:wrapper></fo:wrapper>は
5. <fo:wrapper scale-x="0.5">い<fo:wrapper scale-y="1.5"><fo:wrapper color="#FF0001">ろ</fo:wrapper></fo:wrapper></fo:wrapper>は
6. <fo:wrapper scale-y="2.5"><fo:wrapper color="#FFFF00">い</fo:wrapper></fo:wrapper><fo:wrapper color="#80FF00">ろ</fo:wrapper>は
7. <fo:wrapper color="#FF1C6ED2">いろは</fo:wrapper> ※VizRender Color
8. <fo:wrapper color="#1C6ED2" opacity="100">いろは</fo:wrapper> ※Classic Color
9. テスト<fo:wrapper font="Fonts/Segoe_UI_Emoji_Regular">😀</fo:wrapper>です ※フォント

＜付録⑦＞Vizrt 起動時のコマンドライン オプション

▲目次▲　▲システム▲

（参照先：[Viz Artist User Guide]-[Getting Started]-[Viz Command Line Options]）

-n Viz Engine モードで開始します。

-w レンダーウインドウを表示し Viz Engine モードで開始します。（ビデオウォールモード）

-y Viz Artist モードで開始します。
Viz Artist(vizgui.exe)は Viz Engine(viz.exe)の起動後に Viz Engine によって起動されます。
Engine モードのみの起動でも On Air interface を使用する為に必要です。

-c Viz Config モードで開始します。

-u1, -u2, ・・・,-u<n> Viz Engine インスタンス番号を設定します。　（<n> = 1 - 24）

参考) インストール時に自動作成されるショートカットのパラメータ

Viz Artist　："<インストール先>¥VizEngine¥Viz.exe" -u1 -y
Viz Engine　："<インストール先>¥VizEngine¥Viz.exe" -u1 -y -n
Viz Engine – Config　："<インストール先>¥VizEngine¥Viz.exe" -u1 -y -c

-T Viz コンソールを常に表示します。

-C Viz コンソール無しで開始します。

--db <user>:<pw>@<server>/<name-server>:<port>
VizGH を指定して起動します。パスワードのみ省略可能です

例) viz.exe --db Guest:@VizDbServer/localhost:19396

<user>　: Guest (PW 無)
<server>　: VizDbServer
<name-server>　: localhost
<port>　: 19396

-g <config file> Config file を指定して起動します。

-h , -? 使用可能なコマンドラインオプションを表示します。

-i テクスチャの事前初期化を有効にします。テクスチャはロード直後にグラフィックカード上に生成されます。

-l <title> デュアルチャンネル時に、VizEngine の区別用にコンソールタイトルを設定します。

-o <シーン> シーンを指定して起動します。メインレイヤーで起動します。

※シーンは、シーン ID、もしくはシーンパスでも指定できます。
例）viz.exe -u1 -y -o “SCENE*Default/1”
※Engine モードでの起動になりますが、On Air interface を使用する為に、-y も必要です。

-o <layer> <シーン> 指定されたシーンでオンエアモードで Viz Engine を起動します。

※シーンは、シーン ID、もしくはシーンパスでも指定できます。
バックレイヤー例）viz.exe -u1 -y -o “0 SCENE*Default/1”
メインレイヤー例）viz.exe -u1 -y -o “1 SCENE*Default/1”
フロントレイヤー例）viz.exe -u1 -y -o “2 SCENE*Default/1”
※Engine モードでの起動になりますが、On Air interface を使用する為に、-y も必要です。
※-o の後のシーンパスの””は必須です。

-W クラッシュ時の再起動を無効にします。

-X クラッシュ時に備えて、拡張ダンプファイル（フルメモリーダンプ）の書き込みを有効にします。

-M Trio 用。Viz Artist と Trio でコンテンツ共有を有効にします。

-B <path> Viz Engine の一時作業用フォルダを指定します。

-Y <path> Viz Engine がプログラムデータを保存するパスを指定します。

-P 自動マウスキャプチャを無効にします。

-t 非対話モードを有効にします。

-v <オプション引数>　Viz コンソールの表示詳細モードを有効にします。
オプション引数は、以下を加算して構成される数値です。

1	: 詳細情報をコンソールに表示
2	: タイムスタンプを追加
4	: OpenGL のログを記録
16	: 2D テクスチャメッセージをログに記録
32	: VizGH 関連のデバッグメッセージをログに記録
64	: 中・高レベルの OpenGL 警告をログに記録

-G1, -G2,・・・,-G<n>　GPU が複数ある場合、GPU を指定して起動します。
<n>は GPU インデックスで 1 から始まります。
-u フラグと組み合わせると、プライマリモニターの接続先に関係なく、専用 GPU で viz を起動できます。
-G パラメータは、主に特定のインスタンスを専用 GPU にバインドする為、または GPU が Viz Engine インスタンスによって占有されていないことを確認する為に使用されます。

例）viz.exe -u1 -y -n -G1
　　最初の GPU でレンダリングする最初のインスタンスとして Artist で開始
例）viz.exe -u2 -y -n -G2
　　2 番目の GPU でレンダリングする 2 番目のインスタンスとして Engine で開始
例）viz.exe -u2 -y -n -G1
　　最初の GPU でレンダリングする 2 番目のインスタンスとして Engine で開始

-u のみが指定されている場合、起動されたインスタンスに GPU を自動的に割り当てます。

＜付録⑧＞**その他**

▲目次▲

ON-AIR モードについて

- Viz5 から、マルチディスプレイの時は ON-AIR のプレビュー画面の表示スクリーンを選択できるようになりました。
  - 左上の Control Buttons-[Full-screen]右隣にあるプルダウンで表示先のスクリーンを選択できます。
  - プルダウンを変更した時には切り替わりません。次に[Preview]:ON、[Full-screen]:ON にした時に切り替わります。
- On Air interface ショートカット

(参照先：[Viz Artist User Guide]-[Keyboard and Mouse Shortcuts]-[On Air Shortcuts])

- [ESC] : Artist で Editor モードに戻る
- [SHIFT + BACKSPACE] : [Preview] チェックボックスの ONOFF
  [Full-screen]:ON なら、フルスクリーン表示の ON/OFF が切り替わる。
- [CTRL + F] : [Full-screen]チェックボックスの ONOFF
  [Preview]:ON なら、フルスクリーン表示の ON/OFF が切り替わる。

Scene Editor のショートカット

(参照先：[Viz Artist User Guide]-[Keyboard and Mouse Shortcuts]-[Scene Editor Shortcuts])

- 主なカメラコントロール ショートカット
  - [P + マウスクリック] : カメラを移動
    X 方向 (左ボタン)、 Y 方向 (中ボタン)、 Z 方向 (右ボタン)
  - [O + マウスクリック] : カメラを選択したオブジェクトを中心に回転
    X 方向 (左ボタン)、 Y 方向 (中ボタン)、 Z 方向 (右ボタン)
  - [I + マウスクリック] : カメラの向きを移動
    パン (左ボタン)、 チルト (中ボタン)、 ロール (右ボタン)
  - [U + 左マウスクリック] : ズーム
  - [T] : 選択したコンテナを中心に表示するようにカメラを移動
  - [R] : カメラ位置をリセット

VizRenderer 用フォントについて

- Classic レンダーの時のように、コンテナにフォントをドラッグ＆ドロップする手順では使用する事が出来ません。
- VizRenderer モードの場合は、まず Text Plugin を追加することで文字を扱う手順になります。
  [Plugins]タブ - [GeometryPlugins] - [Default] – Text
  - Scene の Render Engine を[Viz Engine Renderer]にしないと Plugins Pool に表示されません。
- フォントファイル自体も Import しただけのフォントは使えません。
  Font Management を使います。詳しくは参照先をご覧ください。
  (参照先：[Viz Artist User Guide]-[Manage Items and Plug-ins]-[Working with Items] –
  [Working with Fonts]-[Font Management])