HIBINO
ヒビノプロビジョン株式会社



VizScript 入門

Ver 1.0.1

2026.11.4

・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。

■ 本書の対象読者について

✧ 本書は、簡単なシーンを作成できる程度の Vizrt 操作を習得し Vizrt の基本的な仕組みを理解しているデザイナーやプログラマーを対象としています。

✧ もし Vizrt の基本操作に不安がある場合や初めて触れる方は、本書を読み進める前に、まずは Viz Artist 等での基本的な学習を行っていただくことをおすすめします。

目次

■ 第 1 章：VizScript の役割と標準エディタの使い方 ・・・・・・・・・・・・ p.2

✧ 1. VizScript の概要とリアルタイムグラフィックスにおける役割

✧ 2. スクリプトを適用する場所 (Container Script と Scene Script)

✧ 3. 標準スクリプトエディタの基本操作と動かす手順 (Compile and Run)

✧ 4. ⚠️現場での超重要 Tips：シーン保存時の挙動と未コンパイル保存の罠

✧ 5. デバッグの第一歩：Println と Viz Engine Console の活用

■ 第 2 章：基本文法とデータの扱い方 ・・・・・・・・・・・・ p.10

✧ 1. 変数宣言 (Dim、型指定、初期値ルール)

✧ 2. 標準的なデータ型とキャスト方法

✧ 3. CG 空間の座標を扱うオブジェクト型 (Vertex)

✧ 4. 色を扱うオブジェクト型 (Color)

✧ 5. 処理をコントロールする基本構文

■ 第 3 章：プラグインの操作とテキスト・コンテナプロパティの変更 ・・・・・・・・・・・・ p.17

✧ 1. コンテナプロパティや他のプラグインへのアクセス

✧ 2. Text ジオメトリの基本構造と文字列書き換えルール

✧ 3. コンテナスクリプトにおける this キーワードとツリーの探索

✧ 4. 超頻出コールバックプロシージャの基本と適用場所による制限

■ 第 4 章：パラメータパネル（UI）の作成とテキスト切り替えの実装 ・・・・・・・・・・・・ p.22

✧ 1. パラメータパネル（UI）へのパラメータの登録

✧ ⚠️重要な罠：第 1 引数（識別名）にユニークな名前を省略せずに設定すべき理由

✧ 2. 値の変化を検知するイベントコールバック (OnParameterChanged)

✧ 3.【実践例】：スクリプトの作成 (番号に応じたテキスト リアルタイム切替)

■ 第 5 章：外部コマンドによる制御とプロシージャ ・・・・・・・・・・・・ p.26

✧ 1. 処理をまとめるプロシージャ（Sub と Function）の基本

✧ 2. 値渡し ByVal と参照渡し ByRef の違い

✧ 3. 外部からコマンドで制御する仕組み

■ 第 6 章：スクリプト間・シーン間のデータ共有 (Shared Memory 入門) ・・・・・・・・・・・・ p.32

■ 第 1 章：VizScript の役割と標準エディタの使い方

▲目次▲

本章では、リアルタイム 3D 空間である Viz Engine 上における VizScript の基本的な役割を理解し、
標準エディタを使用した作成、ワークスペースの切り替え、ショートカットキーを活用した効率的な編集、
コンソールを活用したデバッグ、およびシーン保存時における不具合を防ぐ基本手順を習得します。

■ 1. VizScript の概要とリアルタイムグラフィックスにおける役割

グラフィックスの動的制御とシステム機能の拡張

- ✧ Viz Engine 上で、3D オブジェクトのパラメータやテキスト等を動的に操作します。
- ✧ 他にも、
  - ・ ネットワーク通信による外部システムへのデータ送信やコマンド実行
  - ・ 共有メモリを介したリアルタイムデータ同期
  - ・ 独自パラメータパネルの構築
  - ・ 各種イベントへの自動応答

  など、Viz Engine 全体の機能を高度に拡張および自動化するためのプログラム言語です。

Viz Engine のコマンドシステムとの連動

- ✧ Viz Engine 内の GUI 操作や外部からの制御は内部で VIZ コマンドとして処理されています。
- ✧ VizScript は、コマンドシステムと直接連動して各オブジェクトへ即座に命令を伝達します。

VizScript 構文の特徴

- ✧ VBScript ベースのシンプル構文

  Microsoft の VBScript に似た、直感的で学習しやすい文法です。
- ✧ 大文字・小文字の区別ルール

  言語の命令やオブジェクトのプロパティは大文字・小文字を区別しないため、記述の揺れに強い特徴があります。 ※ 例: this.position.x と this.Position.X は同じ扱い
- ✧ 大文字・小文字の注意点

  デザイナー自身が命名したコンテナ名や Director 名、スクリプトパラメーターの識別名などの固有名詞については、スクリプトからアクセスする際に大文字・小文字が厳密に区別されます。
- ✧ 単語（ワード）の区切り

  命令語、変数名、型名などは、半角スペースやタブなどのスペース文字で区切ることで、それぞれ個別の単語として認識されます。
- ✧ インデント（字下げ）のルール

  文法上は任意

  単語が空白で正しく区切られていれば、行頭のインデントが一切なくてもコンパイルエラーにはならず正常に動作します。

  実務上は必須

  文法上は任意ですが、Sub や If などのブロック内部を字下げすることは構造が視覚的にわかりやすくなり、End If などの閉じ忘れを防ぐため、推奨します。

オブジェクトの操作

- ✧ プロパティやプロシージャへのアクセスは、オブジェクト名をドットで接続して操作します。

  ※例：自分自身のコンテナの Text ジオメトリの文字列を変更する

```
this.Geometry.Text = "表示するテキスト"
```

■ 2. スクリプトを適用する場所 (Container Script と Scene Script)

VizScript では、スクリプトをどの階層にアタッチ(配置)するかによって、参照できるオブジェクトの範囲(this の意味)と利用できる機能が明確に分かれます。

Container Script（コンテナ スクリプト）

✧ 概念

特定のコンテナに直接適用するスクリプトです。

コード内の this は **Script プラグインが配置されたコンテナ自身（Container）**を指します。

✧ 用途

個別のコンテナ移動、テキスト変更、アニメ制御、マウスやタッチイベントの検知に使用します。

✧ 適用手順

Script プラグインを対象コンテナへドラッグ＆ドロップします。

Scene Script（シーン スクリプト）

✧ 概念

シーン全体に適用するグローバルスクリプトです。

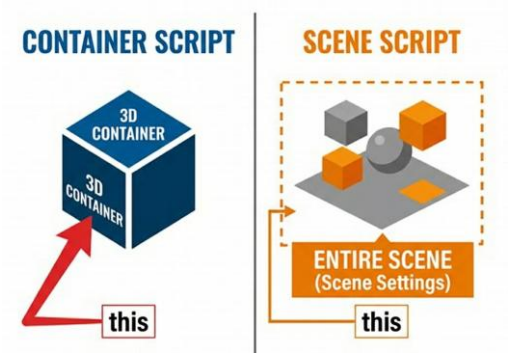
コード内の this は**ロードされているシーン全体（Scene）**を指します。

✧ 用途

特定のコンテナに依存せず、カメラの切り替え、ライティング、シーン全体のロジック管理、Shared Memory 同期などに使用します。

✧ 適用手順

Scripts ドック、または Scripting ワークスペースからエディタを起動します。



■ 3. 標準スクリプトエディタの基本操作と動かす手順 (Compile and Run)

外部の開発ツールを準備しなくても、Viz Artist 標準のユーザーインターフェースだけでスクリプトを記述し、即座にコンパイルして実行することができます。

スクリプトエディタは **Scene Script** と **Container Script** で共通のエディタ UI を使用します。

スクリプトの種別は、Scripts パネルのタブページで切り替えます。

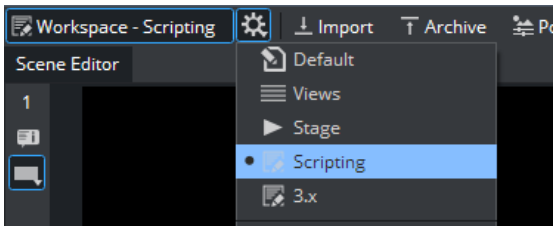
✧ (1) 画面環境の準備：Scripting ワークスペースへの切り替え（スクリプト共通）

より広い画面で快適にスクリプトを編集するため、Viz Artist には専用ワークスペースレイアウトが用意されています。

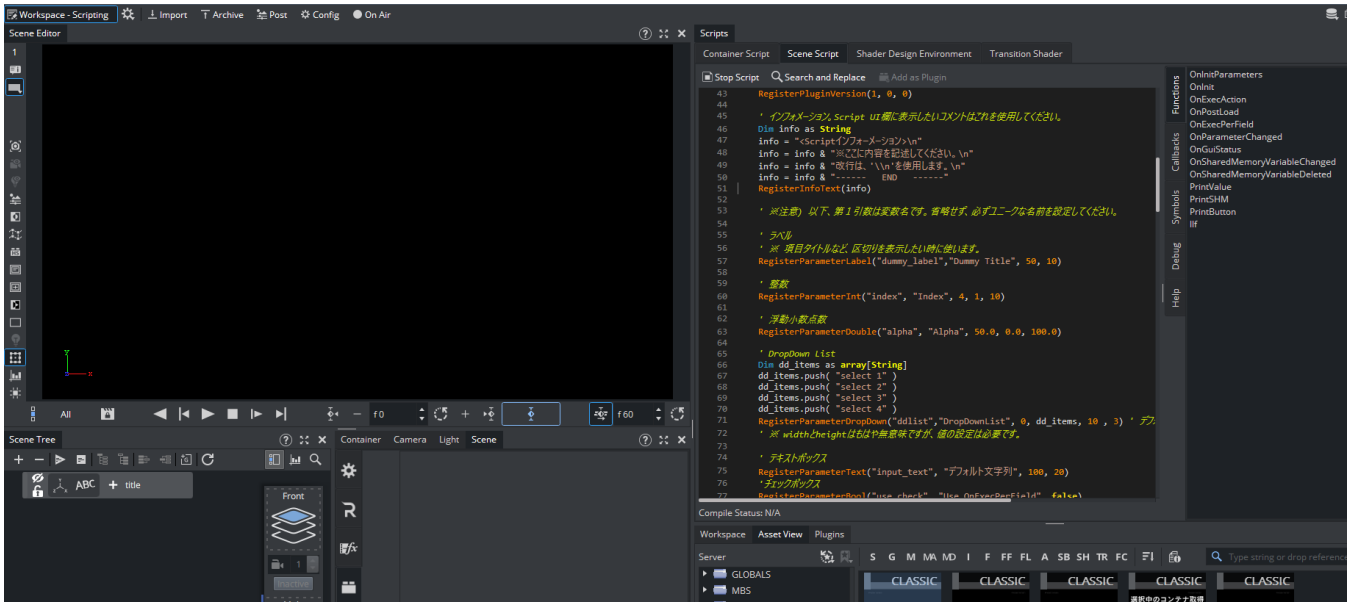
切り替え手順

✧ 画面上部にあるワークスペースメニューをクリックし、メニュー内から **Scripting** を選択します。

✧ ショートカットキー：**Alt + F10**



参考）Scripting ワークスペースレイアウト

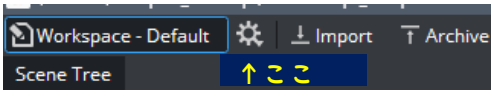


✧ (2) Scripting ワークスペース以外でのエディタ起動手順（スクリプト共通）

Scripting 以外のワークスペースレイアウトでは、Scripts パネルを表示する必要があります。

Scripts ドックの有効化

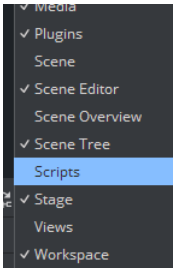
① メイン UI 上の **Workspace** アイコンをクリックします。



② メニュー内の **List of available Docks** まで下へスクロールします。

③ **Scripts** の項目にチェックを入れます。

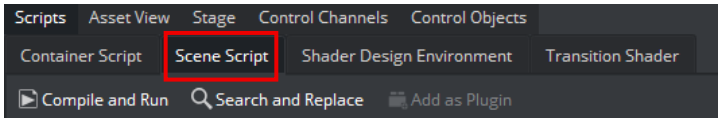
画面上に独立した Scene Script パネルがドックとして配置され、
いつでも編集可能になります。



✧ (3) Scripts ドック表示後の、スクリプト別エディタ起動手順

Scene Script エディタ起動

Scene Script タブをクリックすると、Scene Script エディタが開きます。



Container Script エディタ起動

① Script プラグインの適用

Plugins プール内の Global フォルダなどから **Script** プラグインを検索し、
対象のコンテナへドラッグ＆ドロップします。

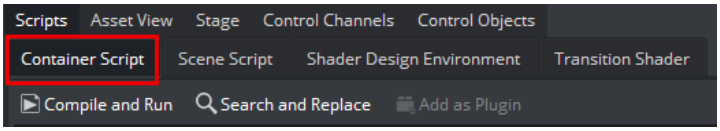


② パラメータパネルの表示

コンテナ上の **Script** アイコンをクリックして Scripts パネルを開きます。

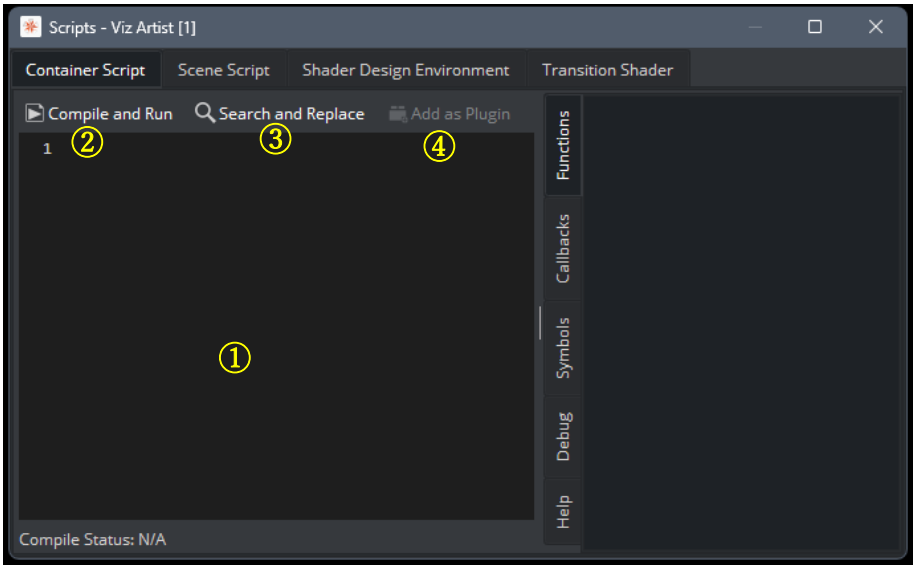
③ エディタの起動

通常、Scripts パネルを開くと、そのコンテナ用の Container Script エディタが開きます。
開いていなければ Container Script タブをクリックすると Container Script エディタが開きます。



✧ (4) 標準スクリプトエディタの画面構成と主要機能

Scripts エディタには、以下の機能ボタンと表示欄が配置されています。



① コード編集エリア

- 行番号表示および構文のカラーハイライト機能を備えたテキストエディタです。
- VizScript 構文を直接記述・編集します。

② **Compile and Run** ボタン (Ctrl + R)

- 記述したスクリプトのコンパイルを行い、エラーが無ければ即座にスクリプトの実行を開始します。
- コンパイルが成功すれば、画面下部のステータスバーが「Compile Status: OK」(緑)と表示され、ボタンが **Stop Script** に変わります。
- コンパイルエラーが存在する場合は、画面下部のステータスバーや Viz Engine Console にエラー行と内容が赤字で出力されます。

③ Search and Replace ボタン（検索と置換）

- エディタ上部のツールバーに配置されている検索・置換機能です。
- コード内の任意の文字列を検索し、新しい文字列へ一括または個別に置換することができます。
- 変数名の一括変更やコード修正時に活用します。

④ Add as Plugin ボタン

- [Compile and Run]が成功して正常動作が確認された際のみクリック可能になります。
- このボタンを押すと、作成したスクリプトを .vsl ファイルとしてローカル保存先に保存し、Plugin プールの「ScriptPlugins」フォルダに登録します。

  ※ ローカル保存先例：C:¥ProgramData¥vizrt¥VizEngine¥ScriptPlugins

  他のコンテナ(Scene Script の場合は他のシーン)で再利用できます。
- ボタンが無効なときは、一度 Stop Script → Compile and Run を実行すると有効になります。

✧ (5) 編集作業を効率化する便利ショートカットキー一覧

F5	：選択したコード行をワンタッチでコメントアウト（行頭に ' を付与）
Ctrl + R	：スクリプトのコンパイルと実行（Compile and Run）
F4	：選択している文字列をコード内から検索 (Selection Search)
F3	：カーソル位置にある単語をコード内から下方向へ検索 (Word Search)
F6	：選択したコード行のコメントアウトを解除
Ctrl + F1	：スクリプトエディタの全ショートカットキーのヘルプ一覧をポップアップ表示

■ 4. ⚠現場での超重要 Tips：シーン保存時の挙動と未コンパイル保存の罠

本番送出やオンエアの現場で予期せぬトラブルを防ぐために、CG デザイナーが必ず知っておくべき極めて重要な保存時の挙動です。

✧ シーン保存時の動作仕様

保存されるタイミングとコード

シーンを保存したその瞬間に、スクリプトエディタに記述されている現在のコードがそのままシーンファイルに書き込まれて保存されます。

Compile and Run と Stop Script の状態保存

シーンを保存する際、エディタ下部で最後に Compile and Run が押されていたか、それとも Stop Script が押されていたかという動作の状態も一緒に保存されます。

次回ロード時の挙動

保存時に Compile and Run が有効だったスクリプトは、次回そのシーンがエンジンに読み込まれた際に、自動的にバックグラウンドでコンパイルが走り、即座に実行が開始されます。

✧ ⚠重大な落とし穴（未コンパイルコード保存による不具合の罠）

発生する不具合

・スクリプトを書き換えて、Compile and Run を押して正しい動作を確認したとします。
・その後、エディタを閉じる前に誤ってキーボードに触れ、無駄な 1 文字を入力してしまい、文法的に間違ったコードに修正してしまったとします。

落とし穴 (不具合の罠)

その[Compile and Run]を実行していない状態のままシーンを上書き保存してしまうと、次回シーンをロードした際に、その間違ったコードで自動コンパイルが実行されてしまいます。

想定される結果

ロードした瞬間にコンパイルエラーが出力され、現場でスクリプトが一切機能しなくなるという、極めて危険なトラブルを引き起こします。

推奨される安全なワークフロー

コードを 1 文字でも書き換えた後は、必ず Compile and Run をもう一度実行してください。
エラーが出ないことをコンソールで確認し、シーンの保存操作を行うことを徹底してください。

■ 5. デバッグの第一歩：Println と Viz Engine Console の活用

プログラムが狙い通りに動いているかを簡単に確認するための、最も標準的なデバッグ方法です。

✧ Println による出力

基本構文

✧ 説明

- 引数に指定した文字列や変数の内容を出力します。
- 文字と変数を繋ぐにはアンパサンド演算子（&）を使用します。
- Viz Engine Console に出力するメッセージは半角英数字を使用してください。
  日本語は文字化けします。

✧ 活用例) 変数の値を含めて結合して出力

```
Dim current_text As String = "BREAKING NEWS"
Println("Current Text " & current_text)
```

カラーインデックス指定出力

✧ 説明

- Println の第一引数に色番号を指定することで、コンソールに出力されるテキストに色がつきます。
- エラーや特定の警告などを強調させたい場合に非常に有効です。

✧ 色番号の仕様

- 色番号は 1（青）・2（緑）・4（赤）の足し合わせで指定します。
  ※ 3=シアン、5=マゼンタ、6=イエロー、7=白
- 8 は「明るく表示」のフラグで、単独では明るいグレー、他の値に足すと明るい色になります。

✧ 活用例) コンソールに赤字でエラーを強調して出力する

```
Println(4, "<ERROR> The specified text container does not exist")
```

✧ Viz Engine Console の確認方法と手動コマンド送信

出力先と表示切り替え

✧ 説明

- Println で出力した文字は描画画面上には出力されず、Viz Engine Console 、または Viz Artist 画面下部のコンソールエリアに出力されます。
- キーボードの Alt + C で表示切り替えが可能です。

※ 参考) Viz Engine Console (Command Console)



手動コマンド送信による動作検証（send）

✧ 説明

- Viz Engine Console ではログを確認するだけでなく、画面下部のコマンド入力行から **send** キーワードを使って Viz Engine へ直接コマンドを発行できます。
- 例えば、スクリプト内のプロシージャが正しく動作するかテストしたい場合など、コンソールからコマンドを送信して即座に挙動を確認できます。

✧ 活用例) コンテナ「ScriptContainer」内の SetText プロシージャを手動で実行

send -1 MAIN_SCENE*TREE*$ScriptContainer*SCRIPT INVOKE SetText 3 aiueo

✧ 注意点

- Viz Artist のエディターモードでコマンド検証する場合は、**MAIN_SCENE** を推奨します。
- **REND** コマンドでは、エディターモードで実行されない場合があります。

※もちろん、On Air モード時は **REND** を指定して問題ありません。

✧ ⚠️現場でのトラブル回避 Tips：コンソールの「簡易編集モード」によるフリーズ

発生する問題

✧ 状況

Viz Engine Console のウィンドウ内をマウスで誤ってクリックすると、タイトルバーに「選択」や「簡易編集モード」と表示される状態に入ります。

✧ 影響

この状態になると、Windows の保護機能によりコンソールへのログ出力が一時ストップし、連動して Viz Engine 全体のレンダリング処理も停止するため、画面が**見かけ上フリーズした状態**になります。

解除方法

✧ 対策

フリーズしたように見えた場合は、コンソールウィンドウ上で **右クリック**するか、**Enter キー** を押すことで即座に簡易編集モードが解除され、正常な描画が再開されます。

■ 第 2 章：基本文法とデータの扱い方

▲目次▲

本章では、VizScript の基本的な変数の宣言方法、主要なデータ型、CG 空間の座標やトランスフォームを扱う Vertex 型、および処理の流れを制御する各種構文の書き方を習得します。

■ 1. 変数宣言 (Dim、型指定、初期値ルール)

プログラム内で数値や文字列などのデータを一時的に保持しておくための領域を変数と呼びます。

◆ スコープ（有効範囲）の種類

グローバル変数

- プロシージャの外側かつコードの先頭付近で宣言する変数です。
- スクリプト内のすべてのプロシージャから値を参照および変更できます。

ローカル変数

- Sub や Function などのプロシージャの内側で宣言する変数です。
- 宣言したプロシージャの中でのみ有効であり、処理が終了すると値は破棄されます。

◆ Dim ステートメントと型指定・初期化ルール

✧ 基本構文

```
Dim 変数名 As データ型
Dim 変数名 As データ型 = 初期値
```

✧ 初期値を指定せずに宣言した場合のデフォルト初期化ルール

数値型（Integer, Double）　：0 または 0.0
ブーリアン型（Boolean）　：False
文字列型（String）　：空文字 ""

✧ 初期値指定時の型省略

```
Dim 変数名 = 初期値
```

- 初期値を同時に指定して宣言する場合、"As データ型"の記述を省略しても右辺の値から型が自動推論されます。

✧ 使用例）

・ グローバル変数の宣言例（コード先頭に記述）

```
Dim global_count As Integer = 0
Dim is_running As Boolean = False
```

・ ローカル変数の宣言例（プロシージャ内）

```
Sub ExampleProcedure()
    Dim local_val As Double = 100.5
    Dim message = "処理開始"   ' String 型として自動判別される
End Sub
```

■ 2. 標準的なデータ型とキャスト方法

VizScript で最も頻繁に使用される基本データ型と、型同士を変換するキャスト方法です。

◆ 標準データ型（Integer, Double, String, Boolean）

Integer（整数型）

✧ 説明

- 32 ビット符号付き整数です。
- カウント処理やインデックス番号の指定など、小数点を持たない数値に使用します。

✧ 使用例）

```
Dim count As Integer = 5
count += 1        'インクリメント処理
```

Double（浮動小数点数型）

✧ 説明

- 64 ビット浮動小数点数です。
- 位置座標、拡大率、回転角度などの小数を扱う際に使用します。
- 数値に小数点が含まれている場合、システムは自動的に Double 型として認識します。

✧ 使用例）

```
Dim pos_x As Double = 120.5
```

String（文字列型）

✧ 説明

- テキスト情報を扱う型です。
- 文字列はダブルクォーテーション "" で囲んで記述します。

✧ 文字列の結合

- アンパサンド & 演算子を使用して文字列同士、または文字列と数値などを自動変換して結合できます。

✧ 改行コードの挿入

- 文字列の中に ¥n を記述することで、改行を含めることができます。

✧ 使用例）

```
Dim title As String = "VizScript"
Dim label As String = title & "入門¥n2 行目のテキスト"
```

Boolean（ブーリアン型）

✧ 説明

- True（真）または False（偽）のいずれかの二値のみを保持する型です。

✧ 使用例）

```
Dim is_visible As Boolean = True
```

◆ キャスト（型変換）の方法

✧ 説明

- 型の異なるデータ同士で計算や代入を行う場合、型変換（キャスト）を行います。
- VizScript ではビルトイン関数による変換と、C 言語形式のキャストの両方をサポートしています。

✧ キャスト関数の利用

CInt(値) ： 整数型に変換（小数点以下切り捨て）
CDbl(値) ： 浮動小数点数型に変換
CStr(値) ： 文字列型に変換
CBool(値) ： ブーリアン型に変換

✧ 使用例） キャスト関数で整数に変換

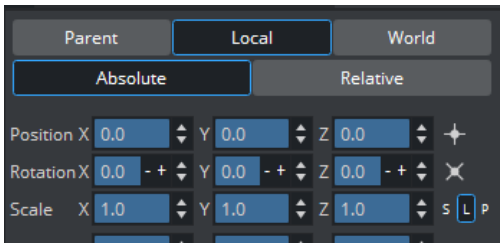
```
Dim raw_val As Double = 12.8
Dim int_val As Integer = CInt(raw_val)          '12 が代入される
```

✧ 使用例） C 言語形式のキャスト

```
Dim str_val As String = (String)int_val
Dim int_val2 As Integer = (Integer)raw_val      '12 が代入される
```

■ 3. CG 空間の座標を扱うオブジェクト型 (Vertex)

CG 空間での位置座標、拡大率、回転角度などを
スクリプトから操作するための特殊なオブジェクト型です。



◆ オブジェクト型（Vertex）

Vertex（頂点・ベクトル型）

✧ 構造と用途

- x、y、z という 3 つの Double 型要素から構成される構造体データ型です。
- オブジェクトの Position(位置)、Scaling(拡大率)、Rotation(回転角度)などを制御する際に使用します。

※ Position・Scaling・Rotation は専用の型で、.xyz プロパティで Vertex 値を一括取得・設定できます。

✧ UI 項目（Local Transform Panel）⇔ VizScript コード表現

Position (X, Y, Z) ⇔ this.Position.x, this.Position.y, this.Position.z
Scaling (X, Y, Z) ⇔ this.Scaling.x, this.Scaling.y, this.Scaling.z
Rotation (X, Y, Z) ⇔ this.Rotation.x, this.Rotation.y, this.Rotation.z

✧ 値のアクセスと生成方法

- ドット接続で各軸の値に個別にアクセスするほか、CVertex を使って一括生成することも可能です。
- 引数を 1 つだけ指定した場合は 3 軸均等に適用されます。

✧ 使用例）方式 1：個別に要素を設定する

✧

```
Dim pos As Vertex
pos.x = 100.0
pos.y = 50.0
pos.z = 0.0
this.Position.xyz = pos
```

✧ 使用例）方式 2：CVertex 関数で一括作成してコンテナの位置に適用する

```
this.Position.xyz = CVertex(100.0, 50.0, 0.0)
```

✧ 使用例）方式 3：3 軸均等スケール (CVertex(2.0, 2.0, 2.0)と同義)

```
this.Scaling.xyz = CVertex(2.0)
```

■ 4. 色を扱うオブジェクト型 (Color)

✧ 3D 空間上の位置を表す Vertex 型と並び、オブジェクトのマテリアルカラーやテキストの色をスクリプトから制御する際に使用するのが Color 型 です。

◆ オブジェクト型（Color）

Color（色型）

✧ 構造と用途

- Color 型は、以下の 4 つの要素(Double 型)で構成されています。

  Red
  赤成分（0.0 ～ 1.0）

  Green
  緑成分（0.0 ～ 1.0）

  Blue
  青成分（0.0 ～ 1.0）

  Alpha
  不透明度（0.0 :完全透明 ～ 1.0 :完全不透明）※デフォルト値は 1.0

- 各要素の値は 0.0（最小/暗）～ 1.0（最大/明） の範囲で指定します。

✧ CColor 関数による一括生成

- Color 型にも一括生成用の CColor 関数が用意されています。

  CColor(r, g, b)
  RGB 値を指定（Alpha は自動的に 1.0）

  CColor(r, g, b, a)
  RGB 値と Alpha 値をまとめて指定

✧ 使用例）赤色（不透明）を作成する例

```
Dim red_color As Color = CColor(1.0, 0.0, 0.0)
```

✧ 使用例）半透明の青色を作成する例

```
Dim blue_trans As Color = CColor(0.0, 0.0, 1.0, 0.5)
```

✧ 使用例）個別要素を変更する例

```
red_color.Green = 0.5     '上記例の赤に、緑成分を足してオレンジ寄りに変更
```

■ 5. 処理をコントロールする基本構文

条件に応じた処理の分岐や、繰り返し処理を実行するための基本構文です。

If ... Then による条件分岐

✧ 基本構造

- 条件式が True の場合に特定の処理を実行します。
- ElseIf や Else を組み合わせることで複数の条件判定に対応できます。

✧ 比較演算子

- 等しい比較　　：= または ==
- 等しくない比較　　：<>
- 小なり、および、以下　　：< および、<=
- 大なり、および、以上　　：> および、>=
- 論理結合には and、or、not を使用します。

✧ 使用例）

```
Dim current_x As Double = this.Position.x
If current_x > 100.0 Then
    Println("right side")
ElseIf current_x < -100.0 Then
    Println("left side")
Else
    Println("center")
End If
```

参考）自作三項演算子（IIf 関数）の活用

✧ 説明

- 標準の言語仕様に三項演算子はありませんが、条件に応じて 2 つの値から 1 つを即座に返すユーティリティ関数（IIf）をコード内に定義してよく利用します。

✧ コード例

```
Function IIf(ByVal result As Boolean, ByVal tvalue As Variant, ByVal fvalue As Variant) As Variant
    Dim answer As Variant
    If result Then   ' 真の時
        answer = tvalue
    Else  ' 偽の時
        answer = fvalue
    End If
        IIf = answer    ' 戻り値
End Function
```

✧ 使用例）条件が True なら 100.0、False なら 0.0 を返す

```
Dim is_active As Boolean = True
this.Position.x = IIf(is_active, 100.0, 0.0)
```

Select Case による多分岐

✧ 基本構造

- 特定の変数の値に応じて、多数の選択肢から実行するコードブロックを選びます。
- 長い **ElseIf** の列挙を避けてすっきりと記述できます。

✧ 複数の値や範囲の指定

- カンマ区切りで複数の値を並べるほか、To キーワードで数値の範囲指定、Is キーワードで比較演算による条件指定も可能です。
- 条件に一致した時点で Select Case を途中で抜ける場合は Exit Select を使用します。

✧ 使用例）

```
Dim mode_index As Integer = 2
Select Case mode_index
        Case 1
                Println("case 1")
                Exit Select
        Case 2, 3
                Println("case 2 or 3")
        Case 4 To 10
                Println("case 4 to 10")
        Case Is > 10
                Println("greater than 10")
        Case Else
                Println("no match")
End Select
```

For ... Next による指定回数ループ

✧ 基本構造

- カウンター変数を使用して、指定した回数だけ処理を繰り返します。
- カウンターの増減値を変更したい場合は Step を指定します。
- ループを途中で抜ける場合は Exit For を使用します。

✧ 使用例）1 から 5 まで 5 回繰り返す

```
Dim i As Integer
For i = 1 To 5
    Println("loop count: " & i)
Next
```

✧ 使用例）10 から 0 まで 2 ずつ減らしながら繰り返す

```
For i = 10 To 0 Step -2
    Println("count down: " & i)
Next
```

Do While / Do ... Loop による条件付きループ

✧ 基本構造

- 指定した条件が True である間、ループ処理を継続します。
- 判定の位置によって、処理の前に判定を行う構文と処理の後に判定を行う構文を選択できます。

✧ 使用例）処理の前に条件をチェックする

```
Dim count As Integer = 0
Do While count < 5
    Println("count: " & count)
    count += 1
Loop
```

✧ 使用例）処理の後に条件をチェックする構文（少なくとも 1 回は実行される）

```
Dim val As Integer = 0
Do
    Println("val: " & val)
    val += 1
Loop While val < 5
```

For Each による配列要素の順次ループ

✧ 基本構造

- 配列（Array）のエレメントを先頭から順番に反復処理するためのループ構文です。
- インデックス番号によるカウンター管理が不要で、すっきりと記述できます。

✧ 使用例）配列内の数値を順番にコンソール出力する

```
Dim numbers As Array[Integer]
numbers.Push(10)
numbers.Push(20)
numbers.Push(30)

For Each num In numbers
    Println("value: " & num)
Next
```

■ 第 3 章：プラグインの操作とテキスト・コンテナプロパティの変更

▲目次▲

本章では、スクリプトからコンテナの各種プロパティや他のプラグインへアクセスする方法、this キーワードを使用したシーンツリーの探索、および、システムから自動実行されるコールバックプロシージャの仕様と操作を習得します。

■ 1. コンテナプロパティや他のプラグインへのアクセス

VizScript では、スクリプトが追加されているコンテナ自身や、そのコンテナに割り当てられている各種プラグインのパラメータを動的に変更できます。

◆ コンテナの基本プロパティへのアクセス

コンテナが標準で保持している名前、有効状態、トランスフォーム情報などには、プロパティ名をドットで接続して直接アクセスします。

Name プロパティ

- コンテナの名前を取得または変更します。

Active プロパティ

- コンテナの表示状態をブーリアン値で切り替えます。
- False に設定すると、コンテナとその子階層全体が非表示になります。

Position / Scaling / Rotation プロパティ

- 位置、拡大率、回転角度は、それぞれ.xyz プロパティ(Vertex 型)で一括して取得・設定できます。

✧ 使用例）コンテナを非表示にする

```
this.Active = False
```

✧ 使用例）位置と拡大率を変更する

```
this.Position.xyz = CVertex(50.0, 0.0, 0.0)
this.Scaling.xyz = CVertex(1.5, 1.5, 1.5)
```

◆ 他のプラグイン（Material, Alpha 等）へのアクセス

コンテナに追加されている他のプラグインやマテリアルのプロパティも、ドット接続で操作できます。

✧ 使用例）アルファプラグインの透明度を変更する

```
If this.Alpha <> Null Then
        this.Alpha.Value = 50.0
End If
```

■ 2. Text ジオメトリの基本構造と文字列書き換えルール (Classic Render)

テキストコンテナの文字情報は Geometry オブジェクトの Text プロパティとして保持されています。

✧ Text プロパティへの代入

- this.Geometry.Text に対して String 型の文字列を代入することで、画面上の表示テキストを一瞬で更新できます。

✧ 安全なアクセス

- Text ジオメトリが割り当てられていないコンテナで Text プロパティを操作しようとするとエラーになるため、事前に **this.Geometry <> Null** であることを確認します。

✧ 使用例）テキストが存在することを確認して文字を書き換える

```
If this.Geometry <> Null Then
        this.Geometry.Text = "表示するテキスト"
Else
        Println(4, "Error: No Text geometry exists in this container. ")
End If
```

■ 3. コンテナスクリプトにおける this キーワードとツリーの探索

Container Script 内において、this はスクリプトが直接貼り付けられているコンテナ自身を参照するための特別なキーワードです。
この this を起点として、親子関係や兄弟関係にある他のコンテナを探し出して操作することができます。

✧ 親子・兄弟関係をたどるプロパティ

FirstChildContainer	：階層下にある最初の直系子コンテナを取得します。
ParentContainer	：1 つ上の階層にある親コンテナを取得します。
NextContainer	：同じ階層にある次の兄弟コンテナを取得します。
PreviousContainer	：同じ階層にある前の兄弟コンテナを取得します。

✧ 使用例）直系の子コンテナを取得して非表示にする

```
Dim child_node As Container = this.FirstChildContainer
If child_node <> Null Then
        child_node.Active = False
End If
```

✧ 名前によるコンテナ検索（Scene.FindContainer）

- シーンツリー全体のどこかにある特定のコンテナを名前で直接探すには、Scene.FindContainer("コンテナ名")を使用します。
- ⚠️ 超重要ルール
  デザイナー自身が命名したコンテナ名は大文字・小文字が厳密に区別されます。
  ※ 例: "Cube" と "cube" は別物
- 使用例）"Header_Text"という名前のコンテナを探してテキストを更新する

```
Dim target_container As Container = Scene.FindContainer("Header_Text")
If target_container <> Null And target_container.Geometry <> Null Then
    target_container.Geometry.Text = "検索したコンテナのテキストを更新"
Else
    Println(4, "Error: Target container or Text geometry not found.")
End If
```

■ 4. 超頻出コールバックプロシージャの基本と適用場所による制限

VizScript は特定のイベント発生時にシステムから自動的に呼び出される「コールバックプロシージャ」によって動作します。
本節では、特に多用される超頻出コールバックと適用場所による利用制限の公式仕様を解説します。

◆ コールバックの仕組みと適用場所による決定的な違い

✧ **Container Script（コンテナスクリプト）**
すべてのコールバック（マウス・キーボード・タッチ・毎フィールド等）がサポートされ、制限なく利用可能です。

✧ **Scene Script（シーンスクリプト）**
公式にサポートされるコールバックは以下の **6 種類**です。

①Sub OnInitParameters()	：UI パラメータの登録
②Sub OnInit()	：インスタンス生成直後の初期化
③Sub OnExecPerField()	：毎フィールドの周期実行
④Sub OnExecAction(buttonId As Integer)	：UI ボタン押下時の処理
⑤Sub OnParameterChanged(parameterName As String)	：UI パラメータ変更時の処理
⑥Sub OnRequestStatusUpdate(...)	：非同期リクエスト完了時の処理

- **注意点**
  - 公式マニュアルには Scene で有意なのを 6 種としていますが、Scene 向けの OnPostLoad や OnGuiStatus も動作します。
  - OnLButtonDown などのマウスイベントや OnKeyDown などのキーボードイベントは **Container Script 専用**であり、Scene Script に記述しても呼び出されません。

◆ 主要コールバックの解説とコード例

初期化処理（Sub OnInit）

✧ 実行タイミング

- Compile and Run ボタンを押した直後や、スクリプトを含むシーンが Viz Engine にロードされた瞬間に、1 度だけ自動実行されます。

✧ 用途

・ 変数の初期値セット
・ 共有メモリの監視キーワードの事前登録(RegisterChangedCallback)
・ 起動時の初期表示設定

などに使用します。

✧ 使用例）スクリプト生成・ロード時に変数を初期化する

```
Dim counter As Integer
Sub OnInit()
    counter = 0
    Println("Script initialization complete. ")
End Sub
```

毎フィールド実行処理（Sub OnExecPerField）

✧ 実行タイミング

- Viz Engine が画面を描画するフィールドごとに連続して繰り返し自動的に呼び出されます。

✧ 用途

- 時間経過に伴うオブジェクトの移動・拡大アニメーション制御、センサーや外部数値の常時監視、計算結果の毎フレームリアルタイム反映などに使用します。

✧ 使用例）毎フィールドごとに X 座標を少しずつ移動させるアニメーション

```
Sub OnExecPerField()
    If this.Position.x < 200.0 Then
        this.Position.x += 1.0
    End If
End Sub
```

マウスクリック判定処理（Sub OnLButtonDown）【Container Script 専用】

✧ 実行タイミング

- 出力ウィンドウ上で、スクリプトがアタッチされているコンテナの領域をマウスで左クリックした瞬間に実行されます。

✧ 用途

- 画面上のボタンクリックに応じたアニメーションの再生開始、テキストの切替、フラグのトグル処理などのインタラクティブなイベント制御に使用します。

✧ 使用例）コンテナがクリックされたときにテキストを変更する

```
Sub OnLButtonDown()
    If this.Geometry <> Null Then
        this.Geometry.Text = "クリックされました！"
    End If
End Sub
```

マウスホバー検出（Sub OnEnter / Sub OnLeave）【Container Script 専用】

✧ 実行タイミング

OnEnter

出力ウィンドウ上で、スクリプトがアタッチされているコンテナの領域に
マウスカーソルが入った瞬間に実行されます。

OnLeave

マウスカーソルがコンテナの領域外へ出た瞬間に実行されます。

✧ 用途

- 画面上のボタンコンテナにマウスが乗った時に拡大・ハイライト表示し、離れた時に元の状態へ戻すようなインタラクティブな UI 演出を作成する際にペアで使用します。

✧ 使用例）マウスが乗った時にコンテナを不透明（半透明解除）にする

```
' マウスカーソルがコンテナに乗った時
Sub OnEnter()
	If this.Alpha <> Null Then
		this.Alpha.Value = 100.0  ' アルファを 100%にする
	End If
End Sub
```

✧ 使用例）マウスが離れたら半透明に戻す

```
' マウスカーソルがコンテナから離れた時
Sub OnLeave()
	If this.Alpha <> Null Then
		this.Alpha.Value = 50.0   ' アルファを 50%に戻す
	End If
End Sub
```

参考）コールバックの実行順序

1. OnInitParameters
2. 保存値の読み込み
3. OnInit
4. OnParameterChanged（デフォルト値・保存値セット時）
5. OnPostLoad（Scene Script のみ）
6. OnExecPerField 開始

参考）キーボード操作の検出（Sub OnKeyDown / Sub OnKeyUp）

✧ コンテナスクリプトでは、マウスクリックだけでなくキーボードのキー押し下げ・離しイベントも検出可能です。

Sub **OnKeyDown**(keyCode As Integer)　　：キーが押された瞬間に実行されます。
Sub **OnKeyUp**(keyCode As Integer)　　：キーが離された瞬間に実行されます。

- 引数 keyCode には押されたキーに応じたシステム定数（KEY_A ～ KEY_Z、KEY_0 ～ KEY_9、KEY_RETURN、KEY_SPACE、KEY_F1 ～ KEY_F12 など）が渡されます。

✧ 使用例）スペースキーが押されたらアニメーションを開始する

```
Sub OnKeyDown(keyCode As Integer)
    If keyCode == KEY_SPACE Then
        Dim dir As Director = Stage.FindDirector("Default")
        If dir <> Null Then
            dir.StartAnimation()
        End If
    End If
End Sub
```

■ 第 4 章：パラメータパネル（UI）の作成とテキスト切り替えの実装

▲目次▲

本章では、VizScript の強力な機能である「パラメータパネル（UI）の動的構築」を習得します。

スクリプト内に UI 登録関数を記述して専用のパラメータ入力欄を作成し、ユーザーがパラメータパネルの番号(1〜5)や文字列を変更した際に、リアルタイムでコンテナの表示テキストが切り替わるスクリプトを構築します。

■ 1. パラメータパネル（UI）へのパラメータの登録

VizScript では、スクリプトのパラメータパネル上に独自のスライダー、テキストボックス、ドロップダウンメニューなどの UI パーツを動的に生成できます。

これにより、スクリプトのコードを直接書き換えることなく Viz Artist のパラメータパネルから安全に設定値を変更できるようになります。

◆ UI 登録専用プロシージャ（Sub OnInitParameters）

- ✧ パラメータパネルの UI パーツを定義するには、OnInitParameters() コールバックを使用します。
- ✧ このプロシージャの中にパラメータ登録関数(Register...)を記述することで、スクリプトがコンパイルされた際、パネル上に GUI 要素が自動的に配置されます。

◆ 数値選択欄と 1 行テキストボックスの登録関数

- ✧ 本章のゴールである番号選択欄(1〜5)と、5 つの 1 行テキストボックスを作成するために、以下の 2 つの関数を使用します。

RegisterParameterInt（整数入力欄の登録）

- ✧ 基本構文

```
RegisterParameterInt("識別名", "表示ラベル", 初期値, 最小値, 最大値)
```

- ✧ 役割

  パラメータパネル上に数値入力ボックスを生成します。

- ✧ 使用例）選択番号(1〜5)を指定する整数パラメータを登録する

```
RegisterParameterInt("select_num", "Select No (1-5)", 1, 1, 5)
```

RegisterParameterString（1 行テキストボックスの登録）

- ✧ 基本構文

```
RegisterParameterString("識別名", "表示ラベル", "初期文字列", 表示幅, 最大文字数, "文字フィルタ")
```

- ✧ 役割

  パラメータパネル上に文字を入力するための 1 行テキストボックスを生成します。

- ✧ 使用例）5 つの 1 行テキストボックスを登録する

```
RegisterParameterString("text_1","Text 1","ニュース速報",100,100,"")
RegisterParameterString("text_2","Text 2","天気予報",100,100,"")
RegisterParameterString("text_3","Text 3","交通情報",100,100,"")
RegisterParameterString("text_4","Text 4","スポーツ情報",100,100,"")
RegisterParameterString("text_5","Text 5","株価情報",100,100,"")
```

◆ ⚠Viz 5 以降における UI サイズ指定の注意点

- ✧ Viz Artist 5 以降、パラメータパネルのレイアウトが自動調整される仕様に変更されたため、widgetSize や width(表示幅)、height(表示高さ)といったサイズ指定引数は実際の画面表示には反映されなくなりました。
- ✧ ただし、コンパイルエラー(構文エラー)を防ぐため引数を省略することはできません。

  サンプルコードのように 100 や 1 などのダミー数値を必ず指定してください。

◆ ⚠️重要な罠：第 1 引数（識別名）にユニークな名前を省略せずに設定すべき理由

✧ パラメータ登録関数を呼び出す際、必ず知っておくべき極めて重要な注意点があります。

識別名の役割

- 各登録関数の第 1 引数は UI 上のラベル表示ではなく、**スクリプト内部でそのパラメータを特定するためのシステム識別名**です。
- 外部コマンドでアクセスする際などでも、大文字・小文字が厳密に区別されます。

発生するトラブル

- 第 1 引数を省略したり、空文字 "" にしてしまうと、スクリプト内部からその UI の値を取得できなくなります。
- 複数の UI パーツに対して同じ第 1 引数を指定(重複)してしまうと、システムが値を区別できなくなり、意図しないデータが上書きされる原因になります。

対策

第 1 引数には、半角英数字で必ず他と重複しないユニークな名前（変数名に似た識別名）を省略せずに設定してください。

■ 2. 値の変化を検知するイベントコールバック

パラメータパネル上でユーザーが数字を変更したり、テキストボックスの文字を書き換えたりした瞬間に処理を発生させるための仕組みです。

◆ 値変更時のコールバック（Sub OnParameterChanged）

✧ 実行タイミング

デザイナーがパラメータパネル上の UI(スライダーやテキストボックスなど)の値を変更した瞬間に、Viz Engine から自動的に呼び出されます。

✧ 引数 parameterName の役割

- 変更された UI パーツの第 1 引数(識別名)が文字列として parameterName に渡されます。
- この名前を判定することで、「どの UI が変更されたか」を特定できます。

✧ 使用例）

```
Sub OnParameterChanged(parameterName As String)
    Println("changed parameters : " & parameterName)
    UpdateTextDisplay()      ' 値が変更されたので、画面上のテキスト更新処理を呼び出す
End Sub
```

■ 3.【実践例】：スクリプトの作成（番号に応じたテキスト リアルタイム切替）

パラメータパネルの番号（1〜5）を変更すると、対応する文字列スロットの内容を読み出してコンテナの Geometry.Text をリアルタイムに書き換える完全なプログラムを構築します。

※ 実践例のスクリプトコードは、

・Script プラグインが適用されているコンテナを操作している

・Geometry.Text を対象としている

ため、Classic Render 用 Container Script となります。

◆ パラメータパネルの番号指定により表示テキストを動的に切り替えるスクリプト

✧ 完全コード例）

※次ページにコード全文を記載しています。

✧ コードの動きと解説

OnInitParameters()での UI 登録

✧ RegisterParameterInt("select_num",…)

選択番号用の入力 UI を生成します。

✧ RegisterParameterString("text_1" 〜 "text_5",…)

5 つの文字列入力 UI を生成します。

✧ **GetParameterInt と GetParameterString による値の読み出し**

✧ GetParameterInt("select_num")

現在パラメータパネル上で指定されている番号を取得します。

✧ GetParameterString("text_1" 〜 "text_5")

該当する番号のテキストボックスに格納されている文字列を取得します。

✧ **OnParameterChanged による即時反応**

ユーザーが番号を変えたり、テキストボックスの文字を打ち替えたりした瞬間に、OnParameterChanged が反応し、内部の UpdateTextDisplay()を呼び出します。

✧ **this.Geometry.Text への代入**

✧ 取得した文字列を自身(this)の Text ジオメトリへ代入します。

✧ これにより、画面上の文字が一瞬で切り替わります。

◆ 動作確認の手順

1. Text ジオメトリ(フォント）が追加されているコンテナに Script プラグインを適用します。
2. 次ページのコードを Container Script エディタに貼り付けます。
3. [Compile and Run]をクリックします。
4. 下部ステータス欄に「Compile Status:OK」と緑背景で表示されることを確認します。

※もし、エラーが出た時は、エラー内容に従って、修正してください。

5. パラメータパネルに選択番号欄と 5 つのテキストボックスが表示されたことを確認します。
6. 選択番号欄の値を 1 から 2、3 へと動かして、レンダリング画面上の文字がリアルタイムに「ニュース速報」→「天気予報」→「交通情報」へ切り替わることを確認します。

```
' パラメータパネルの番号指定により表示テキストを動的に切り替えるスクリプト
' ＜UI パラメータの初期化＞
Sub OnInitParameters()
    ' 1. 表示番号の選択欄 (1〜5)
    RegisterParameterInt("select_num", "Select No (1-5)", 1, 1, 5)

    ' 2. テキストスロット (1〜5) の登録
    RegisterParameterString("text_1", " Text 1", "ニュース速報", 100, 100, "")
    RegisterParameterString("text_2", " Text 2", "天気予報", 100, 100, "")
    RegisterParameterString("text_3", " Text 3", "交通情報", 100, 100, "")
    RegisterParameterString("text_4", " Text 4", "スポーツ情報", 100, 100, "")
    RegisterParameterString("text_5", " Text 5", "株価情報", 100, 100, "")
End Sub

' ＜スクリプト開始時の初期化＞
Sub OnInit()
    ' 起動時・コンパイル時に初期表示テキストをセットする
    UpdateTextDisplay()
End Sub

' ＜UI パラメータ値が変更された時＞
Sub OnParameterChanged(parameterName As String)
    ' いずれかの UI パラメータが変更されたら表示を更新する
    UpdateTextDisplay()
End Sub

' ＜テキスト更新プロシージャ（独自定義）＞
Sub UpdateTextDisplay()
    ' 1. 現在選択されている番号を取得する
    Dim selected_index As Integer = GetParameterInt("select_num")

    ' 2. 選択番号に対応する文字列を取得する
    Dim target_text As String = ""
    Select Case selected_index
        Case 1
            target_text = GetParameterString("text_1")
        Case 2
            target_text = GetParameterString("text_2")
        Case 3
            target_text = GetParameterString("text_3")
        Case 4
            target_text = GetParameterString("text_4")
        Case 5
            target_text = GetParameterString("text_5")
        Case Else
            target_text = "範囲外の番号です"
    End Select

    ' 3. コンテナの Text ジオメトリが存在することを確認して文字列を反映する
    If this.Geometry <> Null Then
        this.Geometry.Text = target_text
    Else
        Println(4, "Error: No Text geometry exists on the container to which the script is attached.")
    End If
End Sub
```

■ 第 5 章：外部コマンドによる制御とプロシージャ

▲目次▲

✧ 本章では、

・VizScript における処理の共通化と再利用を可能にするプロシージャの基本構造
・プロシージャの引数の受け渡し方法の違い
・外部制御コマンドによるスクリプトの遠隔制御

という、3 つの基本手法を習得します。

■ 1. 処理をまとめるプロシージャ（Sub と Function）の基本

複数の場所で繰り返し使用する処理や、特定の機能ブロックをまとめて整理するために定義するコードの単位を**プロシージャ**と呼びます。

VizScript には、戻り値を持たない **Sub プロシージャ**と処理結果の戻り値を返す **Function プロシージャ**の 2 種類が存在します。

◆ Sub（サブルーチン）プロシージャ

✧ 基本構造

一連の処理命令を実行しますが、呼び出し元へ値を返しません。

✧ 構文

Sub プロシージャ名 (引数リスト) ... **End Sub**

✧ 途中で脱出する処理

処理の途中でプロシージャを終了して抜け出す場合は **Exit Sub** ステートメントを使用します。

✧ 使用例）文字列をコンソールに出力する Sub プロシージャ

```
Sub DisplayLog(message As String)
    If message == "" Then
        Exit Sub        ' 空文字の場合は処理を中断して脱出する
    End If
    Println("LOG: " & message)
End Sub
```

✧ 使用例）Sub プロシージャの呼び出し例

```
DisplayLog("Processing has started.")
```

◆ **Function（関数）プロシージャ**

✧ 基本構造

計算結果や取得したオブジェクトなどの「戻り値」を呼び出し元へ返します。

✧ 構文

Function 関数名(引数リスト) **As** 戻り値の型 … 関数名 = 返す値 … **End Function**

✧ 戻り値の指定方法

VizScript では **関数名と同じ名前の変数に対して値を代入する**ことで戻り値を指定します。

✧ 途中で脱出する処理

途中で抜け出す場合は **Exit Function** を使用します。

✧ 使用例）2 つの数値を加算して結果を返す Function プロシージャ

```
Function AddValues(val1 As Integer, val2 As Integer) As Integer
    Dim result As Integer = val1 + val2
    AddValues = result                    ' 関数名に代入することで戻り値を指定する
End Function
```

✧ 使用例）Function プロシージャの呼び出し例（計算結果を変数で受け取る）

```
Dim total As Integer = AddValues(10, 20)
PrintIn("Total: " & total)          ' "Total: 30"が出力される
```

◆ **スクリプト間のプロシージャ呼び出し**

➢ コンテナ→別コンテナ

✧ Scene.FindContainer("コンテナ名").Script.プロシージャ名(引数)

➢ コンテナ→Scene Script

✧ Scene.プロシージャ名() または Scene.変数名

■ 2. 値渡し ByVal と参照渡し ByRef の違い

✧ プロシージャに引数を渡す際、データの渡し方として ByVal(バイ・バル)、ByRef(バイ・レフ)の 2 つのモードを選択できます。

◆ **ByVal（値渡し）**

✧ 仕組み

呼び出し元の変数の「コピー（値そのもの）」をプロシージャに渡します。

✧ 影響

- プロシージャ内で引数の値を書き換えても、呼び出し側にある元の変数の値には、**一切影響を与えません** 。
- 元の変数の保護・安全性を高めるため、特別な理由がない限り ByVal の使用を推奨します。

✧ 使用例）引数を ByVal で受け取るプロシージャ

```
Sub ResetCount(ByVal num As Integer)
    num = 0                          ' コピーされたローカル変数の値を 0 にする
End Sub
```

✧ 実行例）呼び出し後の引数値の確認

```
Dim count As Integer = 100
ResetCount(count)
Println("count: " & count)     ' 元の変数は変更されず、100 のまま
```

◆ **ByRef（参照渡し）**

✧ 仕組み

呼び出し元の変数そのものへの「参照（メモリアドレス）」を渡します。

※ 定数や式を渡した場合は、ByRef 指定でも常に値渡しになります。

✧ 影響

- プロシージャ内で引数の値を変更すると**呼び出し側にある元の変数の値も書き換わります** 。
- 複数の値を呼び出し元へ連動して戻したい場合などに使用します。

✧ 使用例）引数を ByRef で受け取るプロシージャ

```
Sub ForceResetCount(ByRef num As Integer)
    num = 0                          ' 参照先の元の変数を直接 0 にする
End Sub
```

✧ 実行例） 呼び出し後の引数値の確認

```
Dim count As Integer = 100
ForceResetCount(count)
Println("count: " & count)     ' 元の変数が書き換わり、0 になる
```

◆ ⚠ 重要な仕様（デフォルトの挙動）

✧ VizScript では、引数の定義時に ByVal や ByRef を省略（指定しない）した場合、自動的に **ByRef（参照渡し）** として処理されます。

✧ 呼び出し元の変数を意図せず書き換えてしまうリスクを防ぐため、**特別な理由がない限り明示的に ByVal を記述する**ことを推奨します。

■ 3. 外部からコマンドで制御する仕組み

Viz Engine は、外部システムやステージ上の Action キーフレームから文字列の VIZ コマンドを受信してリアルタイム動作する仕組みを持っています。
スクリプトに関しても、外部コマンドから操作・制御するための専用インターフェースがあります。

◆ 方法 1：外部からスクリプト内のプロシージャを直接実行する (SCRIPT INVOKE)

スクリプト内部に独自で定義した Sub プロシージャを、外部コマンドから直接名指しで実行し、同時に引数データを引き渡す手法です。

✧ 使用用途

外部データ受信時の特殊なロジック処理、複合的なアニメーション制御のトリガーなど。

✧ 基本コマンド構文

コンテナ スクリプト

```
REND*TREE*<ノード指定>*SCRIPT INVOKE <プロシージャ名> <引数 1> <引数 2> ...
```

シーン スクリプト

```
REND*SCRIPT INVOKE <プロシージャ名> <引数 1> <引数 2> ...
```

✧ スクリプト記述例) 外部から直接実行されるパラメータ受け取り用プロシージャ

コンテナ スクリプト

```
Sub SetHeaderTitle(header_no As Integer, title_text As String)
    If this.Geometry <> Null Then
        this.Geometry.Text = "No." & header_no & " " & title_text
    End If
End Sub
```

シーン スクリプト

```
Sub SetHeaderTitle(header_no As Integer, title_text As String)
    Dim cont As Container = Scene.FindContainer("任意の対象コンテナ名")
    If cont <> Null and cont.Geometry <> Null Then
        cont.Geometry.Text = "No." & header_no & " " & title_text
    End If
End Sub
```

✧ 送信コマンド例)

コンテナ スクリプト

```
-1 REND*TREE*$Header_Container*SCRIPT INVOKE SetHeaderTitle 2 "aiueo"
```

※「Header_Container」コンテナの SetHeaderTitle を実行

シーン スクリプト

```
-1 REND*SCRIPT INVOKE SetHeaderTitle 3 "kakikukeko"
```

✧ ⚠プラグイン（VSL）化した場合の注意

スクリプトをプラグイン化(.vsl)した場合、外部からはプラグイン扱いになるため、
SCRIPT INVOKE コマンドによるプロシージャの直接呼び出しは使用できなくなります。

◆ 方法 2：外部からパラメータパネル（UI）の値を書き換える　　(SCRIPT INSTANCE 変数名 SET)

第 4 章で学んだパラメータパネル(UI)のパラメータ(RegisterParameter…で登録した識別名)に対して、外部コマンドから直接新しい値をセットする手法です。

✧ 使用用途

外部システムから設定値を書き換え、連動して OnParameterChanged を自動発火させて表示を更新する場合など。

✧ 基本コマンド構文

コンテナ スクリプト

REND*TREE*<ノード指定>*SCRIPT*INSTANCE*<識別名> SET <値>

シーン スクリプト

REND*SCRIPT*INSTANCE*<識別名> SET <値>

- 識別名は、RegisterParameter()で登録した文字列で、大文字・小文字が厳密に区別されます。

✧ スクリプト記述例)

- パラメータ識別名「select_num」を登録

```
Sub OnInitParameters()
        RegisterParameterInt("select_num", "Select No (1-5)", 1, 1, 5)
End Sub
```

- UI パラメータ更新時のコールバック

```
Sub OnParameterChanged(parameterName As String)
    If parameterName == "select_num" Then
        UpdateTextDisplay()          '例：テキスト更新プロシージャの呼び出し
    End If
End Sub
```

✧ 送信コマンド例)

コンテナ スクリプト

-1 REND*TREE*$Text_Container*SCRIPT*INSTANCE*select_num SET 3

※「Text_Container」の「select_num」の値を「3」に変更

※コマンドが実行されると UI の値が「3」に切り替わり、
同時に OnParameterChanged が自動呼び出しされて画面上のテキストが
「交通情報」へ即座に更新されます。

シーン スクリプト

-1 REND*SCRIPT*INSTANCE*index SET 3

※シーンスクリプトに index というパラメータがある場合

◆ 方法 3：外部から登録したボタンを実行する　（SCRIPT INSTANCE ボタン名 INVOKE）

スクリプト内で RegisterPushButton 関数を使ってパラメータパネル上に登録した UI ボタンを、外部コマンドから「押す(クリックする)」手法です。

✧ 使用用途

パラメータパネル上のボタンに割り当てた初期化処理や計算処理を、外部コマンドや Action キーフレームから実行する場合など。

✧ 基本コマンド構文

コンテナ スクリプト

REND*TREE*<ノード指定>*SCRIPT*INSTANCE*<ボタン識別名> INVOKE

シーン スクリプト

REND*SCRIPT*INSTANCE*<ボタン識別名> INVOKE

✧ スクリプト記述例)

● ボタン識別名「btn_reset」を登録（ID: 10）

```
Sub OnInitParameters()
    RegisterPushButton("btn_reset", "表示テキストリセット", 10)
End Sub
```

● ボタン押下時のコールバック

```
Sub OnExecAction(buttonId As Integer)
    If buttonId == 10 Then
        If this.Geometry <> Null Then
            this.Geometry.Text = "初期テキスト"
        End If
    End If
End Sub
```

✧ 送信コマンド例)

コンテナ スクリプト

REND*TREE*$Text_Container*SCRIPT*INSTANCE*btn_reset INVOKE

※「Text_Container」の「btn_reset」ボタンを外部から実行

シーン スクリプト

REND*SCRIPT*INSTANCE*btn_reset INVOKE

✧ ⚠ワンポイント注意点：外部制御と Viz Engine Console 手動検証の違い

スクリプトや外部コマンドの検証を行う際、実行環境によりターゲットの指定方法が異なります。

● 外部アプリケーション等からの外部制御（本番運用・TCP 制御）

Viz Artist を On Air モード(SWITCH_EXTERNAL)に切り替えた状態で、ターゲットに REND を指定して送信します。

例) REND*TREE*$Header_Container*SCRIPT INVOKE SetHeaderTitle 1 "ニュース"

● Viz Engine Console からの手動テスト（デザイン制作時の検証）

エディターモードのままコンソール画面から手動で動作確認を行う場合は、send コマンドを使用し、ターゲットを MAIN_SCENE に指定します。

例) send -1 MAIN_SCENE*TREE*$Header_Container*SCRIPT INVOKE SetHeaderTitle 1 "ニュース"

■ 第 6 章：スクリプト間・シーン間のデータ共有 (Shared Memory 入門)

▲目次▲

◆ 概要

これまで学んだ変数(Dim)は、そのスクリプト内でのみ有効でした。
しかし実際の運用では、
・スクリプト A で計算した結果をスクリプト B で参照したい
・別レイヤーで動いているシーンへデータを引き渡したい
という場面が多発します。
これを実現するのが Viz Engine の Shared Memory（共有メモリ / SHM） 機能です。

◆ 共有メモリの 3 つのスコープ (マップ)

VizScript では、用途に応じて 3 種類のメモリマップが用意されています。

Scene.Map	：同一シーン内の全スクリプト間でデータを共有
System.Map	：同一 Viz Engine にロードされている全シーン間でデータを共有
VizCommunication.Map	：同一 Graphic Hub 上の複数台の Viz Engine 間でデータを共有

◆ 共有メモリの基本操作（書き込み・監視・読み出し）

共有メモリを使う処理は、以下の 3 つのステップで構築します。

✧ ステップ 1：値の書き込み(セット)

- マップに対して任意のキー名(文字列)を指定して代入します。
- 外部制御コマンドやスクリプトなどで行います。
- スクリプト例)

```
Scene.Map["ScoreData"] = "100"  'シーンマップの "ScoreData" というキーに文字列を保存
```

✧ ステップ 2：監視キーワードの登録 (RegisterChangedCallback)

- 値を受け取る側のスクリプトの OnInit()内で変化を監視したいキー名を登録します。
- この登録を忘れると変更イベントが通知されません。
- スクリプト例)

```
Sub OnInit()
    ' "ScoreData" キーの値変化で、通知を受けるよう登録
    Scene.Map.RegisterChangedCallback("ScoreData")
End Sub
```

✧ ステップ 3：変更検知と読み出し (OnSharedMemoryVariableChanged)

- 登録したキーの値が書き換わると OnSharedMemoryVariableChanged が呼び出されます。
- スクリプト例)

```
Sub OnSharedMemoryVariableChanged(map As SharedMemory, mapKey As String)
    If mapKey == "ScoreData" Then
        'マップから値を取得（Variant 型のため String へキャスト）
        Dim new_score As String = (String)map[mapKey]
        If this.Geometry <> Null Then
            this.Geometry.Text = "得点: " & new_score
        End If
    End If
End Sub
```