紙芝居アプリ内部仕様書

目次

紙芝居アプリ内部仕様書

この文書は、TMPose紙芝居の汎用アプリSB3について、target、変数、event、 custom block、呼出し関係、状態遷移の内部仕様を現在の実装に対応させて記録します。 成果物プロファイル、SB3・台本変換ビルダーの外部契約、開発・検証・公開手順は ソフトウェアメンテナンスガイドを参照してください。台本の外部仕様は台本DSLマニュアルと コマンドリファレンスを参照してください。

本書は「アプリが内部でどのように動くか」を扱い、「リポジトリをどう変更・公開するか」 や「ビルダーをどう利用するか」は扱いません。

対象アプリ／DSL: `kamishibai=3.1`

過去のバージョンからの変更は`history.md`を参照してください。

この文書で使う用語

本書ではScratch／TurboWarpの用語と、このアプリ固有の用語を次の意味で使います。 表中の等幅書体（`Stage`、 `script`、 `action=`など）は、target名、変数名、DSL記法として 実装に現れる正確な綴りを表します。通常書体の用語は、概念または分類名を表します。 詳細なデータ構造や処理は、表に示した後続の章で説明します。

Scratch／TurboWarpのproject構造

用語	この文書での意味
project	Stage、sprite、block、変数、画像、音声などをまとめたScratch／TurboWarp作品全体
SB3	projectを1ファイルに格納するScratch 3形式。このリポジトリではbuildによって生成する成果物
target	project内でblock、変数、costumeなどを所有する実行単位。1つのStage targetと、0個以上のsprite targetがある
`Stage`	projectに1つだけある舞台のtarget。このアプリでは背景表示に加え、台本解析と実行状態の統括を担う。詳細は「SB3の構成」（第2章）を参照
sprite	舞台上に表示・移動でき、costume、sound、block、変数を持てるtarget

clone と Actor

用語	この文書での意味
clone	実行中に sprite から作る複製。同じ block を共有する一方、スプライトローカル変数には clone ごとの値を持てる
`Actor` target／アクタースプライト	アクターを clone として生成するための sprite 雛形。この project では target 名が `Actor`
アクター	`Actor` target から作られ、`actorName` で区別される個々の clone。物語上の登場人物として action を実行する

紙芝居 DSL と実行

用語	この文書での意味
asset	Asset Manager へ名前付きで登録する画像、音声、text。SB3 内の costume などと外部 URL の resource を同じ方法で参照できる
紙芝居 DSL／台本ファイル	asset、アクター、scene、action などをテキストで定義する言語と、その言語で記述したファイル
`script`	読み込んだ台本を保持する runtime variable。外部ファイルと組み込み台本は、ここから同じ処理経路へ入る
scene	台本を `---` で区切った実行単位。内部では最初の区切りより前を scene 0 として扱う
command	台本の `key=value` 形式の 1 行。`exec command %s %s` が key に応じて設定、定義、action 登録などを行う
action	scene 内で順番に実行する演出命令。Stage が実行する Stage action と、アクターへ送る Actor action がある
action envelope	Actor action の宛先、command、引数を入れる `actionTarget`、`actionCommand`、`actionParam`、`actionParam2` のまとまり
message／broadcast	Scratch／TurboWarp の event 配送機構。broadcast すると、その message に対応する hat block がある target や clone で処理が始まる

変数とblock

用語	この文書での意味
Scratch変数／list	SB3に保存され、Stageまたはspriteが所有する値と配列
runtime variable	Runtime Variables機能拡張が保持し、project全体から参照する共有値
thread variable	Thread Variables機能拡張が、カスタムブロック呼出し単位で保持する一時値
event hat／hat block	green flag、broadcast受信、clickなどのeventを受けて処理を開始する、stack最上部のblock
カスタムブロック	project内で定義し、名前と引数によって呼び出す処理。一般的なプログラミング言語のprocedureに相当する
block ID	SB3の`targets[].blocks`内でblockを識別するkey。実装スナップショット内の調査にだけ使い、版をまたぐ永続IDとしては扱わない。詳細は「event、カスタムブロック、呼出し関係」（第4章）を参照

1. 文書の範囲と実装基準

アプリ内部構造の正本は`app/project.source.json`です。本書のtarget、変数、list、 broadcast、hat、カスタムブロック定義は、このファイルから抽出した現在の構造を 記載しています。配布用`kamishibai.sb3`は`app/`から生成する成果物であり、本書の 調査元にはしません。

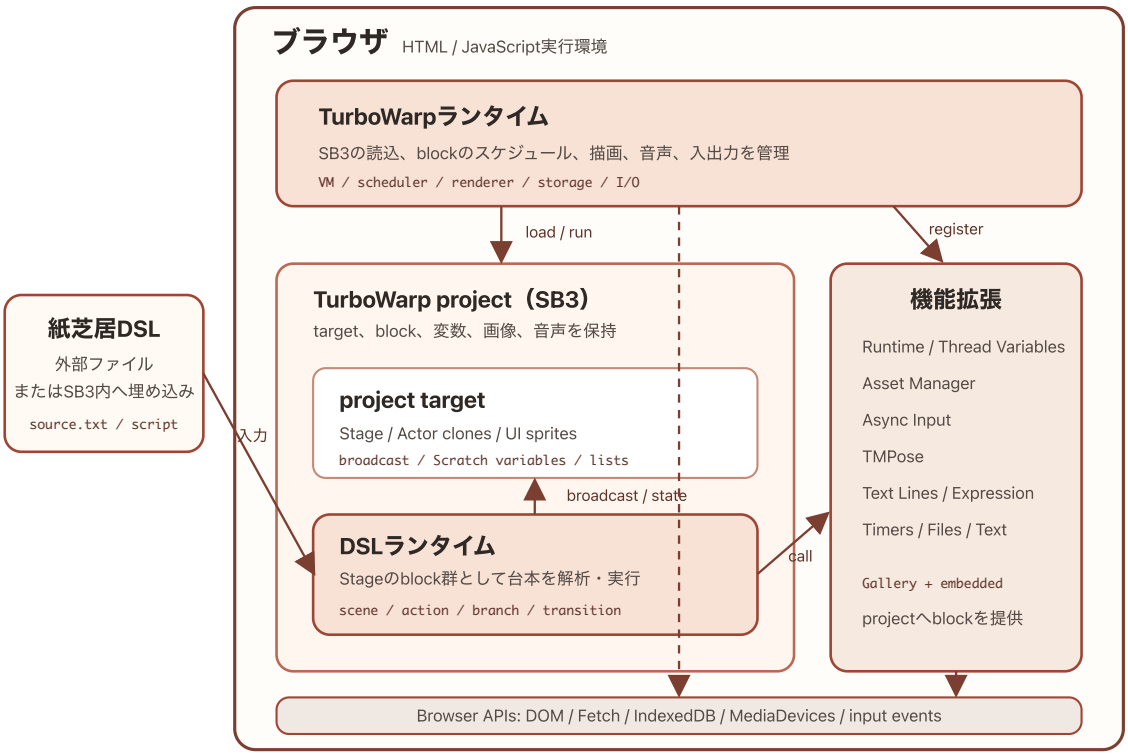
1.1 内部構造のレイヤー

次の図は、汎用アプリSB3の実行時責務を上位から下位へ並べた概念図です。 上位レイヤーは下位レイヤーへ処理を委譲し、eventや共有変数を介して結果と状態を受け取ります。 各レイヤーの具体的なtarget、変数、message、custom blockは後続の章で説明します。

紙芝居アプリ内部構造のレイヤー

1.2 実行アーキテクチャ

次の図は、DSLがブラウザ上で実行されるまでの包含関係と主要な接続を示します。 TurboWarp ランタイムはSB3 projectと機能拡張を読み込み、project内のStageに実装された DSLランタイムが、外部ファイルまたはSB3へ埋め込まれたDSLを解析・実行します。 DSLランタイムはbroadcastや共有変数でproject内のtargetを統括し、画像・音声、入力、 姿勢認識などの処理を機能拡張へ委譲します。



紙芝居アプリの実行アーキテクチャ

1.3 実装スナップショット

項目	件数
target（Stageを含む）	20
block	1751
event hat	98
カスタムブロック定義	42
Scratch変数	6
Scratch list	11
broadcast message	21
静的なruntime variable名	19
静的なthread variable名	36
TurboWarp機能拡張	14

本書に掲載するblock IDの意味と安定性は「event、カスタムブロック、呼出し関係」（第4章）で説明します。

1.4 使用する機能拡張

ID	役割	取得形態
`sipcconsole`	デバッグ用console	Gallery
`lmsTempVars2`	runtime variable／thread variable	Gallery
`strings`	文字列処理	Gallery
`kubohiroyaassetmanager`	画像・音声の登録とLoading進捗	埋め込み
`tmpose`	カメラ姿勢認識	埋め込み
`localstorage`	台本と選択UI言語のローカル保存	Gallery
`kubohiroyatextlines`	行単位の台本処理	埋め込み
`kubohiroyaruntimeexpression`	分岐条件式の評価	埋め込み
`kubohiroyaasyncinput`	key／touch入力とscene移動	埋め込み
`lmsTimers`	`wait`と時間ベースactor actionのタイマー	Gallery
`files`	外部台本ファイルの選択	Gallery
`text`	テキスト描画・アニメーション	Gallery
`translate`	Scratch／TurboWarpの表示言語を取得	標準
`kubohiroyaweblink`	HTTPSの公式Webサイトを新しいタブで開く	埋め込み

GitHub由来の管理対象となる埋め込み拡張では、由来、固定commit、SHA-256を `app/embedded-extensions.json` の `source` に記録します。更新方法は `sb3-toolchain` のワークフロー に従います。`kubohiroyaweblink` はこのproject内で管理する小規模な拡張なので、 `source` を持ちません。

2. SB3の構成

Scratch／TurboWarpのprojectは、1つの**Stage（ステージ）** targetと、0個以上のsprite targetから構成されます。Stage targetはproject全体の舞台を表し、背景（backdrop）を表示します。 Stage自身にblock、variable、listを持てますが、spriteのように座標を変えて動かしたり、 cloneを作ったりする対象ではありません。

このアプリでは、Stage targetを舞台の表示だけでなく、紙芝居全体の制御役として使います。 Stageに置かれたblock群が、台本の読込・解析、assetとactorの生成、sceneとactionの実行、 カメラと入力、画面遷移、共有runtime状態を統括します。したがって、本書やシーケンス図で `Stage` と書いた場合は、画面に見える舞台だけでなく、このStage targetとそこに置かれた 制御用block群を指します。

2.1 target一覧

target	種別	役割	初期costume／sound
`Stage`	Stage	初期化、台本解析、scene/action実行、カメラ、入力、遷移を統括	`Title`, `TitleRuntime`, `Stars`, `LoadingBackdrop`
`Actor`	sprite 雛形	物語上の登場人物ごとにcloneされ、移動・見た目・音・時間actionを実行	`button1`／音声なし
`prompt`	UI sprite	操作案内、pose案内、台本エラーをAsset Managerのcostumeで表示	`ui-placeholder`
`openButton`	UI sprite	外部台本ファイルを選択して`startStory`へ渡す	`ui-placeholder`
`reloadButton`	UI sprite	保存済みの直前の台本を再読込する	`ui-placeholder`
`showTitleButton`	UI sprite	menuからtitleへ戻す	`ui-placeholder`
`languageButton`	UI sprite	menu末尾から言語選択画面を開く	`ui-placeholder`
`japaneseLanguageButton`	UI sprite	UI言語を日本語に変更して保存	`ui-placeholder`
`englishLanguageButton`	UI sprite	UI言語を英語に変更して保存	`ui-placeholder`
`titleHeading`	UI sprite	`about.title`を実行時SVGテキストとして表示	`ui-placeholder`
`titleVersion`	UI sprite	versionとbuild dateを実行時SVGテキストとして表示	`ui-placeholder`
`titleLicenseApp`	UI sprite	`about.license.app`を実行時SVGテキストとして表示	`ui-placeholder`
`titleLicenseStory`	UI sprite	`about.license.story`を実行時SVGテキストとして表示	`ui-placeholder`
`titleAuthorOrganization`	UI sprite	`about.author.organization`を実行時SVGテキストとして表示	`ui-placeholder`

target	種別	役割	初期costume／sound
`titleAuthorName`	UI sprite	`about.author.name` とemailを実行時SVGテキストとして表示	`ui-placeholder`
`officialWebsiteLabel`	UI sprite	公式Webサイト名を実行時SVGテキストとして表示し、押下時に開く	`ui-placeholder`
`officialWebsiteButton`	UI sprite	titleの３行目から公式Webサイトを開く	初期表示用／実行時表示用の言語非依存costume
`closeTitleButton`	UI sprite	title右上の閉じるボタンからStage clickと同じ遷移を実行する	`title-close-button`
`Loading`	UI sprite	Asset Managerの読込開始・進捗・完了に合わせてcostumeを表示	`loading`／音声なし
`LoadingBubbleAnchor`	UI sprite	Loading進捗メッセージ用のspeech bubble位置を固定	`loading-bubble-anchor`

`Actor` の本体は非表示で、cloneだけを登場人物として表示します。`prompt`、menu／言語選択sprite、 title用テキストsprite、`Loading`、`LoadingBubbleAnchor` の実画像は、Asset Managerへ登録します。台本が定義する 予約済みUIテキストは `ui.prompt` だけです。`ui.open`、`ui.reload`、`ui.about`、`ui.language`、`ui.invalidScript` はアプリの言語定義から設定します。

アプリUIの定義元は `scripts/sb3/app-shell-locales.mjs` です。ロケール別の `about.title`、`about.officialWebsite.name`、`about.license.app`、`about.license.story`、`about.author.organization`、`about.author.name` と、共通の `about.officialWebsite.url`、`about.author.email` を、green flag後にAsset Managerの実行時SVGテキストとして表示します。 versionとbuild dateはbuild時に `about.version` へ設定します。言語変更時は表示中の同じspriteの テキストskinを更新するため、ロケール別backdropは使用しません。

title用テキストの配置と文字サイズはScratch標準のStage解像度である480×360を基準にします。 長いライセンス文はロケール定義で明示的に２行へ分け、縮小表示に依存せず、タイトル、version、 公式Webサイト名、ライセンス、開発者情報を480×360の画面上で読める大きさに保ちます。

SB3読込直後からAsset Manager初期化完了まで、または初期化に失敗した場合にも画面を空に しないため、`Title` と `official-website-button` には英語の固定フォールバックをbuild時に 埋め込みます。初期化が完了して `showTitle` を送ると、Stageは文字なしの `TitleRuntime` へ、 公式Webボタンは文字なしの実行時costumeへ切り替え、title用テキストspriteを重ねます。 公式Webボタンの両costumeには `site/favicon.png` を埋め込みます。

green flag 時は localStorage の `uiLanguage` が `ja` または `en` ならその値を優先します。未保存なら 標準 `translate` 機能拡張の （言語） reporter が `日本語` 、 `ja` 、 `ja-JP` のいずれかを返す場合は 日本語、それ以外は英語を選びます。menu の `languageButton` から選び直した値は localStorage へ 保存し、 `languageChanged` でアプリ UI のテキストを即時更新します。言語選択画面では現在値に `✓` を付けます。台本の UTF-8 本文には翻訳や言語切替を適用しません。

2.2 アクターへ命令を届けるしくみ

Scratch／TurboWarp で 1 つの sprite から複数の登場人物を作る場合、clone ごとに スプライトローカル変数の識別子を持たせれば、同じ block を共有しながら個体を区別できます。 このアプリはその考え方を、clone へ命令を届けるところまで拡張しています。

本書では、clone の雛形として使う `Actor` target を**アクタースプライト**、そこから作られ、 スプライトローカル変数の `actorName` を割り当てられた各 clone を**アクター**と呼びます。 アクタースプライトの本体は表示せず、物語の登場人物として表示するのはアクターだけです。 すべてのアクターは同じ block 定義を共有し、 `actorName` によって互いを区別します。

Stage からアクターへ命令を届ける流れは次のとおりです。

1. Stage が DSL の Actor action を、宛先となるアクター名、command、引数に分解する
2. `actionTarget` 、 `actionCommand` 、 `actionParam` 、 `actionParam2` という runtime variable へ それぞれを格納する。このまとまりを本書では **action envelope** と呼ぶ
3. Stage が `execActorAction` を broadcast する
4. すべての `Actor` clone が message を受け取り、自分の `actorName` が `actionTarget` の 対象に含まれるかを調べる
5. 対象になったアクターだけが、 `actionCommand` を入れ子の条件分岐で振り分け、 移動、見た目、音、時間に関する処理を実行する

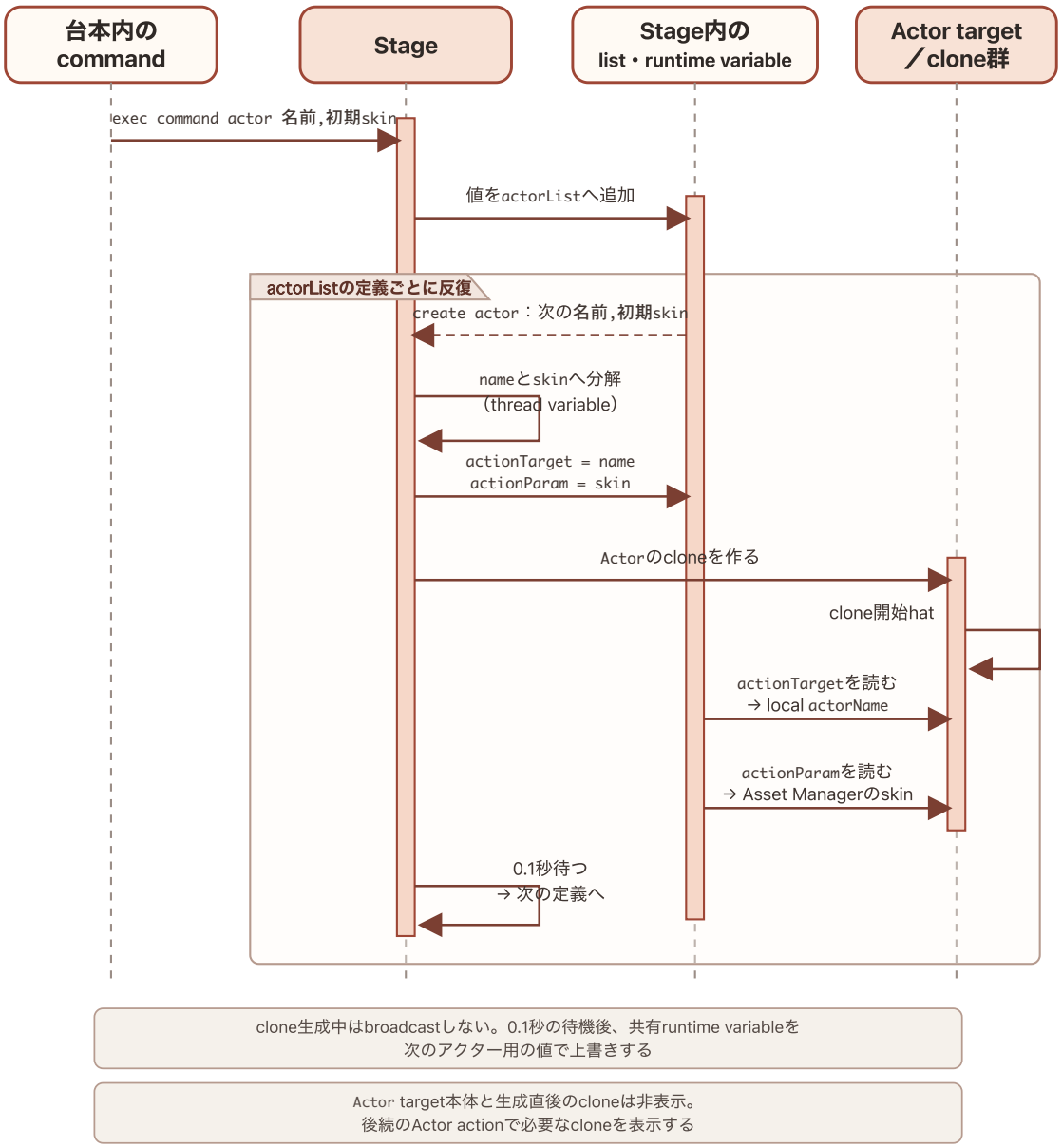
つまり、action envelope が「誰に・何を・どの引数で実行させるか」を表し、broadcast が その命令を全アクターへ配送します。ここでいう command は、アクターに対する メソッドに相当しますが、TurboWarp 上ではカスタムブロックと条件分岐で実装されています。

通常の Scratch project では costume や sound は sprite に属します。このアプリでは TurboWarp Asset Manager を使い、 SB3 内の costume、backdrop、sound、text や、外部 URL の画像・音声を、名前を持つ asset として 登録します。アクターの見た目や音の action はこの asset 名を参照するため、振る舞いを実装する アクタースプライトと、実際に使う画像・音声を分けて組み合わせられます。

台本 DSL は、この asset の定義、アクターの定義、scene ごとにアクターへ送る action の定義を テキストで記述するための言語です。TurboWarp で作られたこのアプリは DSL を解析し、asset を 読み込み、アクターを生成し、action envelope と broadcast を使って命令を実行します。 したがって、このアプリ全体は、紙芝居 DSL を解析・実行する処理系であり、その実行基盤 （runtime）でもあります。

台本からアクターclone生成までのシーケンス

アクターは、台本準備時に`actor=` commandから生成します。これは生成済みのアクターへ Actor action を送る処理とは実行時期もデータの渡し方も異なるため、別の図に示します。



台本の actor 定義からアクターcloneを生成するシーケンス

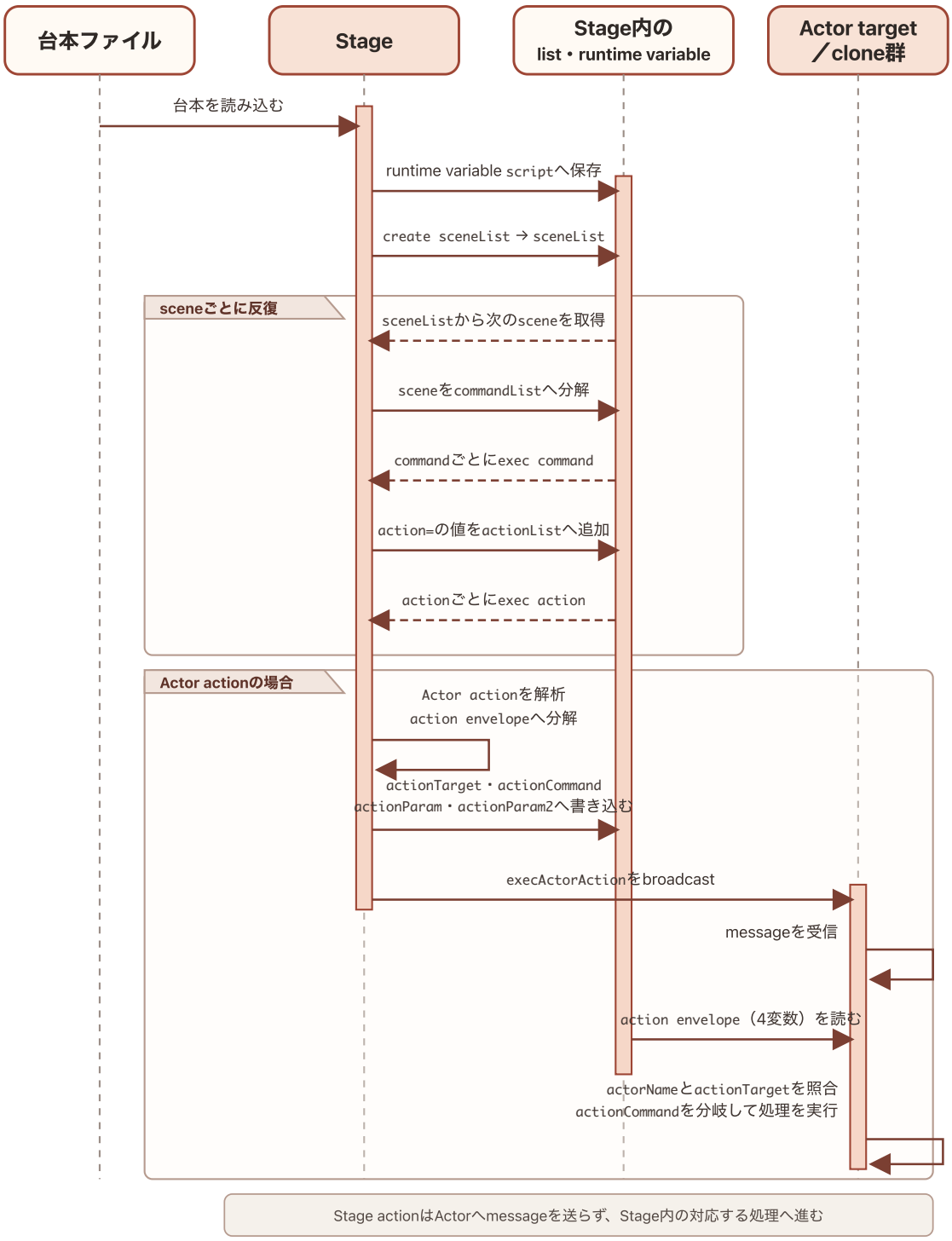
`exec command %s %s` は `actor=` の値を `actorList` へ追加します。各値は `アクター名,初期skin名` の形式です。その後、Stageの `create actor` が `actorList` を順に読み、 各値を thread variable の `name` と `skin` へ分けます。Stageは共有runtime variableの `actionTarget` へ `name` 、 `actionParam` へ `skin` を設定してから、 `Actor` target のcloneを 作ります。

clone開始hatは、 `actionTarget` をそのcloneのスプライトローカル変数 `actorName` へ保存し、 `actionParam` を TurboWarp Asset Managerへ渡して初期skinを設定します。Stageはcloneを 作るたびに0.1 秒待ち、この初期化が走る機会を設けてから、共有runtime variableを次の アクター用の値で上書きします。clone生成時には `execActorAction` をbroadcastしません。

`Actor` target本体はcloneの雛形として非表示です。cloneはこの表示状態を引き継ぐため、 生成直後も非表示であり、sceneの後続のActor actionによって必要な時点で表示されます。

台本からActor actionまでのシーケンス

台本ファイルからActor内の処理までを、データの変換と実行の順に並べると次のようになります。 外部ファイルだけでなく、再生用SB3へ組み込んだ台本も`script`へ入った後は同じ経路を通ります。



台本ファイルからアクターでの命令実行までのシーケンス

`create sceneList` は `script` を scene 単位に分けます。 `exec scene # %s with %s` は現在の scene を 行単位の `commandList` に分け、 `exec command %s %s` で command を順に処理します。このうち `action=` の値を `actionList` へ集め、 `exec actionList` が各 action を `exec action %s` へ渡します。 Stage action は Stage 内で実行されます。Actor action では、Stage が `actionTarget` 、 `actionCommand` 、 `actionParam` 、 `actionParam2` へ action envelope を書き込んでから、 `execActorAction` を broadcast します。message を受信した各 `Actor` clone は 4 変数を読み、 自分の `actorName` と宛先を照合します。実際に command を分岐して処理するのは、宛先に一致したアクターだけです。

2.3 target 間の責務

`Stage` は台本を `sceneList` 、 `commandList` 、 `actionList` へ段階的に変換します。 さらに scene と共有 runtime 状態を管理し、Stage action を実行します。 `Actor` clone の責務は、 生成時に自分の `actorName` と初期 skin を確定し、その後、 「アクターへ命令を届けるしくみ」（2.2 節）で配送された Actor action のうち 自分を対象とするものを実行することです。

UI sprite は表示状態を `showMenu` ／ `hideMenu` 、 `showPrompt` ／ `hidePrompt` 、 Asset Manager の Loading message で受け取ります。台本の実行状態を UI sprite 側へ 複製せず、Stage を状態の所有者とします。

3. 変数と list

変数は、SB3 へ永続化される Scratch 変数／list、thread ごとの一時値、project 全体で共有する runtime variable の 3 種類に分けます。

3.1 Scratch 変数

所有者	変数	初期値	役割
`Stage`	`ポーズ認識`	`0`	pose 認識中の表示・互換用状態
`Stage`	`チャージ`	`0`	pose 成立までの charge 表示・互換用状態
`Stage`	`actionIndex`	`1`	現在処理する `actionList` の位置
`Stage`	`poseIndex`	`1`	現在処理する `poseList` の位置
`Stage`	`__tmpose_embedded_script`	空文字	`player` profile の組み込み台本予約領域
`Actor`	`actorName`	`_template_`	clone が担当する台本上の actor 名

`__tmpose_embedded_script` は Stage に一つだけ存在し、monitor を持ちません。 `generic` と `editor` では空、 `player` では builder が変換済み台本を設定します。

3.2 Scratch list

すべてStage所有で、汎用SB3では空の初期状態です。

list	役割
`skinList`	actor action中のcostume候補
`poseList`	pose action中の認識label候補
`soundList`	actor action中のsound候補
`actionList`	現在sceneのaction列
`sceneList`	台本から抽出したscene block
`commandList`	sceneの前に評価する設定command
`actorList`	台本から生成するactor定義
`assetList`	登録するasset定義
`durationList`	pose候補ごとの継続時間
`sceneLabelList`	scene labelとindexの対応
`lines`	Text Linesで分割した台本行

3.3 runtime variable

静的な名前を持つruntime variableは次の19個です。

変数	生存期間／役割
`script`	読み込んだ変換済み台本。title、reload、`startStory` 間で共有
`version`	台本の `kamishibai` version
`startSceneIndex`	台本で指定した開始scene
`sceneIndex`	現在実行中のscene index
`actionTarget`, `actionCommand`, `actionParam`, `actionParam2`	Stageから Actor cloneへ渡すaction envelope
`nextSceneLabel`	key／touch入力が要求した遷移先scene label
`skipMode`	`Space`、`Right`、`Down` による未消費の進行要求
`skipContext`	`title`、`action`、`pose`、`scene` のどの境界が要求を消費できるか
`poseRecog`, `poseCharge`, `poseIdle`	pose認識のしきい値、charge時間、idle時間。既定値は `0.5`、`10`、`0`
`poseRecognitionSound`	`setPoseRecognitionSound` で指定した認識中の音声アセット名
`poseRecognitionSound2`	`setPoseRecognitionSound` で指定した認識成立時の音声アセット名
`loadingCostume`	Loading spriteへ適用するcostume名
`message`	Loading bubbleへ表示する現在の進捗文言
`uiLanguage`	アプリUIの表示言語。`ja` または `en`

このほか、`exec command %s %s` はDSLで指定されたruntime variable名を動的に設定します。 分岐条件は `branch:<branchName>` という名前で保存します。この２系列は入力から名前が決まるため、 静的な19個には数えません。

3.4 thread variable

カスタムブロック呼出しごとに分離され、呼出し終了後に共有状態として残さない値です。

用途	名前
共通loop・文字列処理	`index`, `length`, `line`, `lineIndex`, `name`, `value`, `key`, `keyValue`
台本解析	`sceneBlock`, `sceneLabel`, `sceneLabelList`, `condition`, `conditionList`
asset／actor生成	`asset`, `assetList`, `resourceId`, `actor`, `actorName`, `skin`
action実行	`action`, `actionResult`, `actionListResult`, `stageActionResult`, `actorActionResult`
command／branch／入力	`commandIndex`, `branchIndex`, `keyId`, `hasRead`
cover	`cover`, `coverBackground`, `coverBgm`
Actor clone	`x`, `y`, `scale`
その他	`durationList`, `sceneIndex`

runtime variableと同名の`sceneIndex` thread variableは、カスタムブロック内の局所的な 引数・計算値です。project全体の現在sceneはruntime variable側だけを正本とします。

4. event、カスタムブロック、呼出し関係

表の`target`はblockを所有するStageまたはsprite、`ID`はそのtargetの`blocks` objectにあるkeyです。SB3内では`project.json`の`targets[].blocks`、このリポジトリ では`app/project.source.json`の同じ位置に保存されます。IDはopcodeや表示名ではなく、 この実装スナップショット内のblockを特定するための内部識別子です。

`sb3-toolchain`のbuildとimportはblock IDを新規採番せず、入力に含まれるIDを保持します。 既存blockを再生成しないTurboWarp上の編集・保存でも通常は保持されます。一方、blockの 削除と再作成、複製やcopy & pasteによる新しいblockの作成、target／projectのimportなどで blockが再生成されるとIDは変わります。したがって、IDは外部仕様、永続ID、他の版をまたぐ 参照には使いません。アプリを編集してIDが変わった場合は、本章も現在の`app/project.source.json`に合わせて更新します。

4.1 event hat一覧

`procedures_definition`と、接続されていないreporter blockはevent hatに含めません。 「実行される内容」はhatを起点とする主要なカスタムブロック呼出し、broadcast、状態変更、 表示操作を要約したもので、標準blockを含む全処理の逐語的な列挙ではありません。

Stage

target	ID	trigger	実行される内容
`Stage`	`iM`	green flag	UI言語を保存値／標準（言語）から決定後、camera／pose／actorを初期化し`showTitle`送信
`Stage`	`i;`	key `space`	`showCover`送信
`Stage`	`i}`	key `down arrow`	現在の`sound`停止、sequence終端化、`finishTimedActorAction`送信、`skipMode=scene`
`Stage`	`jb`	key `right arrow`	現在の`sound`停止、sequence終端化、`finishTimedActorAction`送信、`skipMode=action`
`Stage`	`jX`	`startStory`受信	`start camera`, `create sceneList`, `exec scene # %s with %s`, `create asset`, `create actor`
`Stage`	`j/`	`stopStory`受信	`stop camera`, `stop pose recog`, `show cover`; `deleteAllActors`, `showMenu`送信
`Stage`	`j?`	`debugTestCamera`受信	TMPoseのcamera previewを直接確認
`Stage`	`kD`	`showCover`受信	`show cover`; `hidePrompt`, `deleteAllActors`送信
`Stage`	`l=`	Stage click	`closeTitle`送信
`Stage`	`titleCloseHat`	`closeTitle`受信	組み込み台本の有無に応じて`showCover`または`startStory`送信
`Stage`	`l[`	`showTitle`受信	`TitleRuntime`へ切替え、実行contextをclearし、`hidePrompt`, `deleteAllActors`送信
`Stage`	`m~`	`stopKeyInput`受信	Async Inputの全listenerを停止
`Stage`	`nx`	`stopTouchInput`受信	Async Inputの全listenerを停止
`Stage`	`uiLanguageChangedHat`	`languageChanged`受信	menuと`about.*`の実行時SVGテキストを選択言語で更新

Actor

target	ID	trigger	実行される内容
`Actor`	`nY`	`execActorAction` 受信	`isTimeBasedAction`, `wait for actor action %s seconds`
`Actor`	`oH`	`deleteAllActors` 受信	cloneを削除
`Actor`	`oJ`	clone開始	`actorName`、位置、scaleをruntime envelopeから初期化
`Actor`	`actorFinishHat`	`finishTimedActorAction`受信	対象actorと`skipMode`を照合して時間actionを完了

UI sprite

target	ID	trigger	実行される内容
`prompt`	`oS`	`showPrompt` 受信	案内costumeを表示
`prompt`	`oV`	`hidePrompt` 受信	非表示
`prompt`	`oX`	`invalidScript` 受信	エラーcostumeを表示
`openButton`	`o!`	green flag	非表示
`openButton`	`o%`	sprite click	file選択後に `hideMenu`, `startStory` 送信
`openButton`	`o*`	`hideMenu` 受信	非表示
`openButton`	`o,`	`showMenu` 受信	`ui.open` skinで表示
`reloadButton`	`o.`	green flag	非表示
`reloadButton`	`o:`	`hideMenu` 受信	非表示
`reloadButton`	`o=`	sprite click	保存済み台本で `hideMenu`, `startStory` 送信
`reloadButton`	`o[`	`showMenu` 受信	台本が保存済みなら表示
`showTitleButton`	`o\``	green flag	非表示
`showTitleButton`	`o|`	`hideMenu` 受信	非表示
`showTitleButton`	`o~`	sprite click	`hideMenu`, `showTitle` 送信
`showTitleButton`	`pb`	`showMenu` 受信	title以外なら表示
`languageButton`	`languageButtonFlag`	green flag	非表示
`languageButton`	`languageButtonHideMenu`	`hideMenu` 受信	非表示

target	ID	trigger	実行される内容
`languageButton`	`languageButtonShowMenu`	`showMenu` 受信	`ui.language` skin で表示
`languageButton`	`languageButtonClick`	sprite click	`hideMenu` 後に `showLanguageMenu` 送信
`japaneseLanguageButton`	`japaneseLanguageFlag`	green flag	非表示
`japaneseLanguageButton`	`japaneseLanguageHideMenu`	`hideMenu` 受信	非表示
`japaneseLanguageButton`	`japaneseLanguageShowChoices`	`showLanguageMenu` 受信	現在値なら `✓ 日本語`、それ以外は `日本語` を表示
`japaneseLanguageButton`	`japaneseLanguageClick`	sprite click	`uiLanguage=ja` を保存・適用し menu へ戻る
`englishLanguageButton`	`englishLanguageFlag`	green flag	非表示
`englishLanguageButton`	`englishLanguageHideMenu`	`hideMenu` 受信	非表示
`englishLanguageButton`	`englishLanguageShowChoices`	`showLanguageMenu` 受信	現在値なら `✓ English`、それ以外は `English` を表示
`englishLanguageButton`	`englishLanguageClick`	sprite click	`uiLanguage=en` を保存・適用し menu へ戻る
`titleHeading`	`titleHeadingFlag`, `titleHeadingClick`, `titleHeadingShowTitle`, `titleHeadingHideMenu`, `titleHeadingStartStory`	green flag／click／title・menu・story	`about.title` を表示し、click では `closeTitle` 送信
`titleVersion`	`titleVersionFlag`, `titleVersionClick`, `titleVersionShowTitle`, `titleVersionHideMenu`, `titleVersionStartStory`	green flag／click／title・menu・story	`about.version` を表示し、click では `closeTitle` 送信

target	ID	trigger	実行される内容
`titleLicenseApp`	`titleLicenseAppFlag`, `titleLicenseAppClick`, `titleLicenseAppShowTitle`, `titleLicenseAppHideMenu`, `titleLicenseAppStartStory`	green flag／click／title・menu・story	アプリライセンスを表示し、clickでは`closeTitle`送信
`titleLicenseStory`	`titleLicenseStoryFlag`, `titleLicenseStoryClick`, `titleLicenseStoryShowTitle`, `titleLicenseStoryHideMenu`, `titleLicenseStoryStartStory`	green flag／click／title・menu・story	台本ライセンス案内を表示し、clickでは`closeTitle`送信
`titleAuthorOrganization`	`titleAuthorOrganizationFlag`, `titleAuthorOrganizationClick`, `titleAuthorOrganizationShowTitle`, `titleAuthorOrganizationHideMenu`, `titleAuthorOrganizationStartStory`	green flag／click／title・menu・story	開発者所属を表示し、clickでは`closeTitle`送信
`titleAuthorName`	`titleAuthorNameFlag`, `titleAuthorNameClick`, `titleAuthorNameShowTitle`, `titleAuthorNameHideMenu`, `titleAuthorNameStartStory`	green flag／click／title・menu・story	開発者氏名・emailを表示し、clickでは`closeTitle`送信
`officialWebsiteLabel`	`officialWebsiteLabelFlag`, `officialWebsiteLabelClick`, `officialWebsiteLabelShowTitle`, `officialWebsiteLabelHideMenu`, `officialWebsiteLabelStartStory`	green flag／click／title・menu・story	公式Webサイト名を表示し、clickではサイトを開く
`officialWebsiteButton`	`officialWebsiteFlag`	green flag	初期化前用の英語フォールバックを表示
`officialWebsiteButton`	`officialWebsiteClick`	sprite click	公式Webサイトを新しいタブで開く
`officialWebsiteButton`	`officialWebsiteShowTitle`	`showTitle`受信	文字なしの実行時costumeへ切り替えて表示
`officialWebsiteButton`	`officialWebsiteHideMenu`	`showMenu`受信	非表示

target	ID	trigger	実行される内容
`officialWebsiteButton`	`officialWebsiteStartStory`	`startStory` 受信	非表示
`closeTitleButton`	`closeTitleFlag`	green flag	右上に表示
`closeTitleButton`	`closeTitleClick`	sprite click	`closeTitle` 送信
`closeTitleButton`	`closeTitleShowTitle`	`showTitle` 受信	表示
`closeTitleButton`	`closeTitleHideMenu`	`showMenu` 受信	非表示
`closeTitleButton`	`closeTitleStartStory`	`startStory` 受信	非表示
`Loading`	`pf`	green flag	非表示
`Loading`	`pm`	`assetLoadingStarted` 受信	Loading costume を表示
`Loading`	`pj`	`assetLoadingProgress` 受信	costume を循環
`Loading`	`ph`	`assetLoadingCompleted` 受信	非表示、完了 sound
`LoadingBubbleAnchor`	`loadingBubbleFlag`	green flag	非表示、bubble を clear
`LoadingBubbleAnchor`	`loadingBubbleStarted`	`assetLoadingStarted` 受信	anchor を表示
`LoadingBubbleAnchor`	`loadingBubbleProgress`	`assetLoadingProgress` 受信	runtime variable `message` を say
`LoadingBubbleAnchor`	`loadingBubbleCompleted`	`assetLoadingCompleted` 受信	bubble を clear して非表示

4.2 カスタムブロック定義一覧

引数名は prototype の `argumentnames`、warp は prototype の `mutation.warp` から取得します。「呼び出す処理／送信する message」には定義内で呼ぶ別のカスタムブロックや機能拡張の 処理を示し、broadcast message は送信する名前を明記します。

初期化・parse・共通処理

target	ID	定義	引数	warp	呼び出す処理／送信するmessage
`Stage`	`c:`	`init skinList with %s`	`commaSeparatedText`	yes	`selectValue # %s separated by %s from %s`
`Stage`	`c_`	`init poseList with %s`	`commaSeparatedText`	yes	`selectValue # %s separated by %s from %s`
`Stage`	`dc`	`init soundList with %s`	`commaSeparatedText`	yes	`selectValue # %s separated by %s from %s`
`Stage`	`gH`	`init durationList with %s`	`commaSeparatedText`	no	`selectValue # %s separated by %s from %s`
`Stage`	`eM`	`selectValue # %s separated by %s from %s`	`index`, `separator`, `text`	yes	—
`Stage`	`hl`	`substr of %s after %s`	`text`, `firstDelim`	no	—
`Stage`	`dL`	`min %s %s`	`valueA`, `valueB`	yes	—
`Stage`	`d)`	`exec command %s %s`	`key`, `value`	no	`substr of %s after %s`, `selectValue # %s separated by %s from %s`, `setTMPoseURL with %s`; `invalidScript`
`Stage`	`e+`	`create sceneList`	—	yes	`selectValue # %s separated by %s from %s`
`Stage`	`fe`	`create asset`	—	no	組み込み `LoadingBackdrop`; Asset Manager; `assetLoadingStarted/Progress/Completed`
`Stage`	`fv`	`create actor`	—	no	`selectValue # %s separated by %s from %s`

camera・pose

target	ID	定義	引数	warp	呼び出す処理／送信する message
`Stage`	`dQ`	`setTMPoseURL with %s`	`URL`	no	—
`Stage`	`dU`	`start camera`	—	no	—
`Stage`	`dX`	`stop camera`	—	no	—
`Stage`	`eD`	`start pose recog`	—	no	—
`Stage`	`dG`	`stop pose recog`	—	no	—
`Stage`	`dY`	`rate of pose recog`	—	no	—
`Stage`	`d!`	`label of pose recog`	—	no	—
`Stage`	`d%`	`start camera preview`	—	no	—
`Stage`	`dC`	`stop camera preview`	—	no	—
`Stage`	`dk`	`exec pose action %s`	`actorName`	no	`start camera preview`, `start pose recog`, `exec pose %s`, `stop pose recog`, `stop camera preview`; `showPrompt`, `hidePrompt`
`Stage`	`f[`	`exec pose %s`	`actorName`	no	`change skin...`, Asset Managerの認識音再生／停止、`min...`, `rate of pose recog`

scene・action・actor

target	ID	定義	引数	warp	呼び出す処理／送信するmessage
`Stage`	`fB`	`exec scene # %s with %s`	`sceneIndex`, `sceneData`	no	scene skip中もcommandを末尾まで解析し、`exec actionList`、`hide all actors`；`invalidScript`
`Stage`	`e=`	`exec actionList`	—	no	通常は`exec action %s`、scene skip中は`bgm`と`transition`だけを台本順に実行
`Stage`	`f(`	`exec action %s`	`action`	no	`exec stage action %s`, `exec actor action %s`
`Stage`	`gP`	`exec stage action %s`	`action`	no	`touchInputToChangeScene %s %s`, `exec keyInputToChangeScene %s %s`, `exec branch action %s`, `exec transition action %s`, `wait %s seconds`
`Stage`	`gp`	`exec actor action %s`	`action`	no	`selectValue # %s separated by %s from %s`, 3つのlist初期化、`exec pose action %s`；`execActorAction`
`Stage`	`ee`	`hide all actors`	—	no	`execActorAction`
`Stage`	`eh`	`change skin of %s to %s`	`actorName`, `skinName`	no	`execActorAction`
`Stage`	`eR`	`show cover`	—	no	`selectValue # %s separated by %s from %s`；`showMenu`
`Actor`	`it`	`isTimeBasedAction`	—	no	—
`Actor`	`actorWaitDef`	`wait for actor action %s seconds`	`seconds`	no	—

transition・branch・input

target	ID	定義	引数	warp	呼び出す処理／送信する message
`Stage`	`ga`	`exec transition action %s`	`transitionName`	no	`exec transition reset`, `exec transition fadeUp`, `exec transition fadeOut`, `exec transition fadeToWhite`, `exec transition fadeFromWhite`
`Stage`	`gf`	`exec transition fadeOut`	—	no	—
`Stage`	`gh`	`exec transition fadeUp`	—	no	—
`Stage`	`gj`	`exec transition reset`	—	no	—
`Stage`	`fadeToWhiteDef`	`exec transition fadeToWhite`	—	no	`exec transition fadeUp`
`Stage`	`fadeFromWhiteDef`	`exec transition fadeFromWhite`	—	no	`exec transition fadeOut`
`Stage`	`g]`	`exec branch action %s`	`branchName`	no	`selectValue...`
`Stage`	`hq`	`exec keyInputToChangeScene %s %s`	`keyIdList`, `sceneLabelList`	no	Async Input
`Stage`	`hu`	`touchInputToChangeScene %s %s`	`actorNameList`, `sceneLabelList`	no	Async Input
`Stage`	`hy`	`wait %s seconds`	`seconds`	no	More Timers

4.3 主要な呼出し経路

起点	経路
green flag	保存済みUI言語または標準（言語）を判定 → app shell文言初期化 → camera／pose停止 → actor非表示 → `showTitle`
`startStory`	台本検証 → `create sceneList` → asset／actor生成 → `exec scene # %s with %s`
scene実行	`exec command %s %s` → `exec actionList` → actionごとに`exec action %s`
Stage action	branch、transition、key／touch入力、`wait`などへdispatch
Actor action	runtime envelope設定 → `execActorAction` → 対象clone → 移動・見た目・音・時間action
pose action	camera preview／pose認識開始 → 第1音を再生 → `exec pose %s`反復（条件成立時は第2音を「ポーズ認識」更新前に再生）→音声／認識停止 → prompt非表示
asset loading	組み込みの黒背景 → `setLoadingBackdrop`指定背景 → Loading用画像 → 通常アセット
終了	`stopStory` → camera／pose停止 → actor削除 → cover → menu

5. broadcastと状態遷移

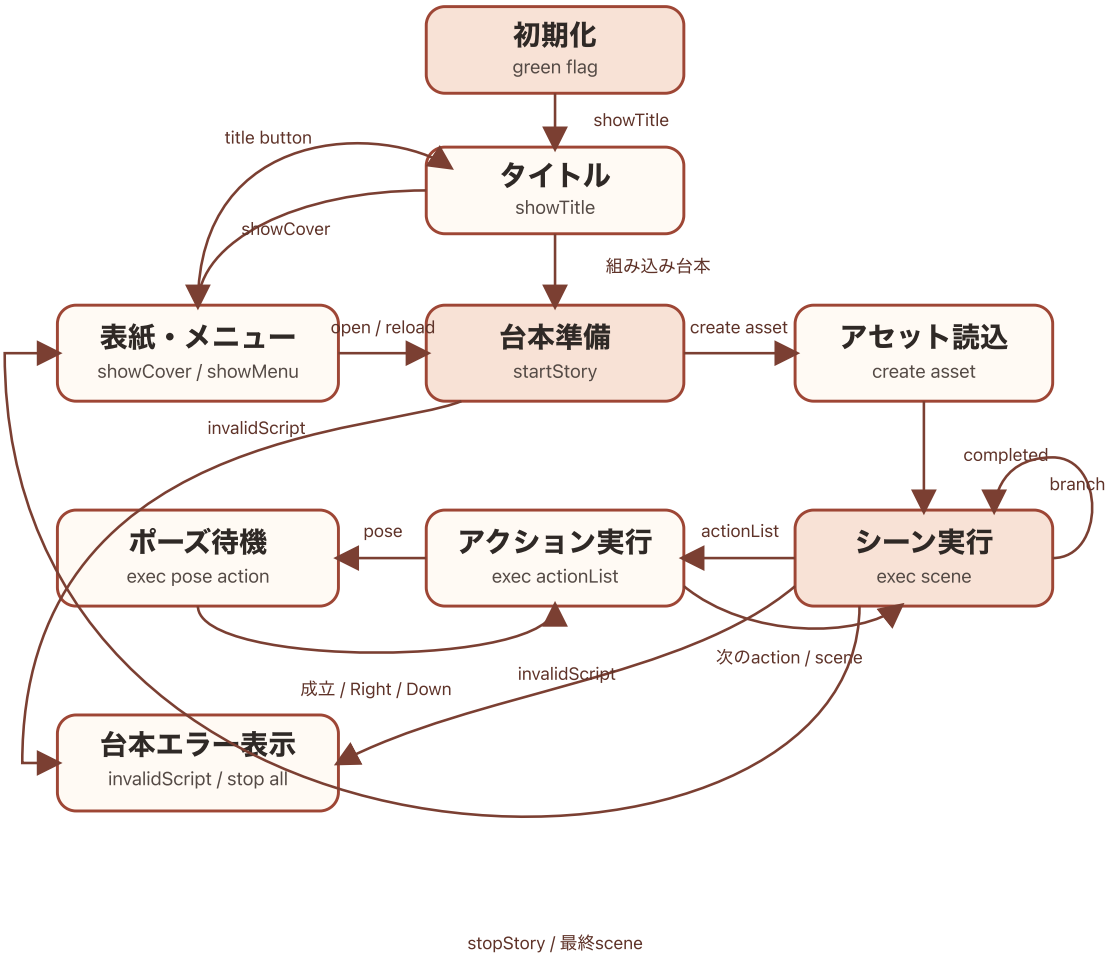
5.1 message一覧

message	主な送信者	受信者	役割
`showPrompt`	`Stage`	`prompt`	操作・pose案内を表示
`hidePrompt`	`Stage`	`prompt`	案内を非表示
`invalidScript`	`Stage`	`prompt`	台本エラーを表示
`hideMenu`	menu／言語選択button	menu／言語選択button	menuと選択肢を一括非表示
`showMenu`	`Stage`、言語選択button	4つのmenu button、Title用2ボタン	利用可能なmenuを表示しTitle用ボタンを隠す
`showLanguageMenu`	`languageButton`	2つの言語選択button	`日本語`と`English`の選択肢を表示
`languageChanged`	`Stage`、2つの言語選択button	`Stage`	app shellの予約済みテキストを選択言語へ更新
`startStory`	`Stage`, `openButton`, `reloadButton`	`Stage`、Title用2ボタン	台本の解析・実行を開始しTitle用ボタンを隠す
`stopStory`	`Stage`	`Stage`	実行を停止しcoverへ戻す
`showCover`	`Stage`	`Stage`	coverを構築してmenuを表示
`showTitle`	`Stage`, `showTitleButton`	`Stage`、Title用2ボタン	title状態へ戻しTitle用ボタンを表示する
`closeTitle`	`Stage`, `closeTitleButton`	`Stage`	Stage clickと閉じるボタンの遷移を共通化する
`execActorAction`	`Stage`	`Actor`	action envelopeをcloneへ通知
`deleteAllActors`	`Stage`	`Actor`	全cloneを削除
`assetLoadingStarted`	`Stage`／Asset Manager	`Loading`, `LoadingBubbleAnchor`	Loading表示を開始

message	主な送信者	受信者	役割
`assetLoadingProgress`	`Stage`／Asset Manager	`Loading`, `LoadingBubbleAnchor`	進捗costumeとmessageを更新
`assetLoadingCompleted`	`Stage`／Asset Manager	`Loading`, `LoadingBubbleAnchor`	Loading表示を終了
`stopKeyInput`	Async Input	`Stage`	key listenerを停止
`stopTouchInput`	Async Input	`Stage`	touch listenerを停止
`finishTimedActorAction`	`Stage`のRight／Down key hat	`Actor`	時間actionを確定状態へ進める
`debugTestCamera`	TurboWarp editorからの手動送信	`Stage`	camera previewの診断

`stopKeyInput`と`stopTouchInput`は標準broadcast blockではなく、Async Inputへ渡したcallback messageです。`debugTestCamera`は通常フローに送信元を持たない診断用messageです。

5.2 状態遷移



紙芝居アプリの主要状態遷移

主要状態はStageが所有します。UI表示そのものを状態の正本にせず、runtime variable、 broadcast、実行中のcustom blockから導出します。

状態	入口	主な出口
初期化	green flag	`showTitle`
title	`showTitle`	Stage click または右上の閉じるボタン。組み込み台本なら `startStory`、それ以外は cover
cover／menu	`showCover` または `stopStory`	open／reload で `startStory`、title button で `showTitle`
台本準備	`startStory`	正常なら asset loading と scene 実行、検証異常なら `invalidScript`
asset loading	`create asset`	`assetLoadingCompleted` 後に scene 実行
scene 実行	`exec scene # %s with %s`	次 scene／branch、最終 scene で `stopStory`、解析異常で `invalidScript`
action 実行	`exec actionList`	Right で action 境界、Down で scene 境界、完了で次 action
pose 待機	`exec pose action %s`	pose 成立、Right／Down で action 実行へ戻る
台本エラー表示・実行停止	台本・command・scene 解析エラー時の `invalidScript`	`prompt` が `ui.invalidScript` を表示し、送信元の `stop all` で実行を停止

`invalidScript` は pose 待機への遷移ではありません。Stage は台本検証、command 解析、 scene 解析のエラー時にこの message を送信し、各送信箇所の直後に `stop all` を実行します。 `prompt` は message を受信すると `ui.invalidScript` の skin を設定して表示します。

`skipMode` は要求、 `skipContext` は消費可能な境界です。要求は title、action、pose、scene の 該当境界だけが消費し、scene 開始、cover、stop で clear します。 `nextSceneLabel` は key／touch listener が設定し、scene loop が label を index へ解決したあと削除します。

`skipMode=scene` では、scene parser は残りの command を解析して action list を完成させます。 action loop は `bgm` と `transition` だけを選択して台本順に高速実行し、それ以外を読み飛ばします。 transition の反復待ちは `skipMode` の存在で終了しますが、最後の明るさ設定は必ず実行します。 `bgm` は scene skip 中も開始でき、down arrow handler は全音声を停止せず、現在の `sound` だけを 停止します。これにより、すでに再生中またはシーン残部で開始した BGM を次 scene へ引き継ぎます。

6. 関連ドキュメント

- `03-user-guide.md`: アプリの利用方法と成果物の使い分け

- `04-dsl-manual.md`：台本の構造と書き方
- `05-command-reference.md`：コマンドと action の外部仕様
- `06-developer-guide.md`：成果物とビルダーの利用、setup、変更、検証、公開
- `history.md`：DSL とアプリの変更履歴
- `README.md`：プロジェクト全体の入口