



UiPath コーディング規約（EUC 向け）

UiPath 株式会社

Ver 1.0

Last Updated: 2020/11/20

変更履歴

No	ver	修正日	名前	修正内容
1	0.1	2020/07/15	UiPath	ドラフトバージョン
2	1.0	2020/11/20	UiPath	最初のバージョン
3				
4				

製品の対応バージョン

本ドキュメントは、UiPath Studio 2020.4 以降を対象に記述されています。

免責事項

- 本資料は UiPath 製品のご利用又はご利用のご検討をされている方を対象に、情報提供を目的として提供するものです。目的外のご利用はご遠慮下さい。
- 本資料に記載されている情報については、弊社では細心の注意を払っておりますが、その正確性、完全性又は妥当性を保証するものではありません。
- 本資料に関する一切の権利は弊社に帰属します。本資料に関する著作権は、弊社又はその他の権利者に帰属しており、著作権法その他の法令で保護されています。お客様は、本資料を内部利用目的の範囲内でのみ使用、変更又は改変することができます。本資料で使用する会社名、製品の商品名及びロゴマークは、弊社、そのグループ会社、又はそれぞれの権利者に帰属している商標又は登録商標です。権利者の許諾を得ることなくこれらを使用することは法令により禁止されておりますので、事前に弊社にご連絡の上許諾を得ていただくようお願いいたします。
- 本資料に記載されている情報は、現状有姿で提供されるものであり、想定とは異なる結果をもたらす様々なリスクや不確実性を含むことがあります。また、本資料の情報は最新でない場合があります。
- 本資料の使用に関し弊社はいかなる責任も負いません。弊社は、本資料の情報を最新の情報にアップデートし、又は改訂するいかなる義務も負いません。

目次

(基礎編).....	1
1. 概要	1
1.1. 目的	1
1.2. 適用範囲	1
1.3. 対象読者と前提条件	1
1.4. 用語定義	1
2. 禁止事項.....	3
3. プロジェクトについて	3
3.1. プロジェクトフォルダー	3
3.2. 必要なファイルは、プロジェクトフォルダーに入れる	4
3.3. プロジェクトテンプレートの活用	4
3.4. 設定ファイルの活用	4
4. ワークフローの設計	4
4.1. プロジェクトの大きさについて	4
4.2. シーケンスとフローチャートの使い分けについて	4
4.3. シーケンスの別ファイルへの抽出について	5
5. 命名規則.....	5
5.1. プロジェクトの命名規則 (例).....	6
5.2. XAML ファイルの命名規則 (例).....	6
5.3. アクティビティの表示名の命名規則 (例).....	6
6. 変数と定数について	7
6.1. わかりやすい名前にする	7
6.2. 既定値を設定する.....	7
6.3. スコープを狭くする	7
6.4. 【繰り返し (コレクションの各要素)】の item 変数について	8
7. ワークフローを読みやすくする	9
7.1. 表示名の変更	9
7.2. 注釈の設定	9
8. 不要な内容は削除する.....	9
8.1. 参考スクリーンショットの保持	9
8.2. 使用していない変数の除去	10
8.3. 不要になったアクティビティやコメントの削除	10
9. 安定したセレクトターの構築.....	10
9.1. セレクトターが期待通り動作しないときは	11
9.2. 意味がある値を含む属性を使用し、変化しそうな値を含む属性は使わない	11
9.3. セレクトターエディターで、セレクトターを修復する	12
9.4. 画像認識のアクティビティはなるべく使わない.....	12
(応用編).....	12
10. 例外処理	12

10.1. 例外処理とは.....	12
10.2. アプリケーションエラーとビジネスエラーの区別.....	13
10.3. 例外の作成とスロー.....	14
10.4. 例外のキャッチ.....	14
10.5. キャッチハンドラーにおける一般的なエラー処理.....	16
ログを出力する.....	16
スクリーンショットを採取する.....	16
10.6. 【リトライスコープ】アクティビティの活用.....	16
含まれるアクティビティのいずれかが、任意の例外をスローしたとき.....	17
【リトライスコープ】の条件に指定したアクティビティが False を返したとき.....	17
【リトライスコープ】の使用例.....	17
11. プロセス実装における作法.....	18
11.1. 【待機】アクティビティの無分別な利用を避け、使用する場合には必ず注釈を記載する.....	18
11.2. 【クリック】アクティビティの入力方法を適切に選択する.....	19
11.3. 【文字を入力】アクティビティの入力方法を適切に選択する.....	19
11.4. 【ホットキーを押下】アクティビティには、必ずセレクトアプロパティを設定する.....	19
11.5. メール送信時の宛先と本文に注意する.....	19
11.6. メール受信時の添付ファイルに注意する.....	19
12. ログ出力.....	20
12.1. ログメッセージを出力する場所.....	20
12.2. ログメッセージの書式.....	20
12.3. 【メッセージをログ】アクティビティに指定すべきログレベル.....	21

(基礎編)

1. 概要

1.1. 目的

本書は、情報漏洩などのインシデントのリスクを回避しながら安定して動作し、かつ修正しやすいワークフローを作成するために守るべき事項を記載します。本書は基礎編（1章～8章）と応用編（9章～11章）のふたつから構成されています。まずは基礎編をお読み頂き、応用編は必要に応じてご参照下さい。

1.2. 適用範囲

本書が規定する事項は、下記の全てに適用されます。

- UiPath Studio を使用して開発するワークフロープロジェクト
- Orchestrator で定義する項目（アセットやキューなど）

1.3. 対象読者と前提条件

本書は、Attended で比較的シンプルな業務を自動化したい、プロのソフトウェア開発者ではない方を対象としています。また、本書を読む前に、下記の UiPath アカデミーコースまでを受講していることを前提とします。

UiPath Studio 開発 初級コース

https://academy.uipath.com/DirectLaunch?cid=XZadDhhvte0_&io=tgpwv6P5Xm0_&md=IS2TIgXikDA_

UiPath Studio 開発 中級コース

https://academy.uipath.com/DirectLaunch?cid=KlacBOygAYk_&io=tgpwv6P5Xm0_&md=IS2TIgXikDA_

ご参考: UiPath アカデミー トレーニングカタログ

<https://www.uipath.com/ja/academy/training?hsCtaTracking=f956363e-6b73-4f14-ac4d-523dde8f6e87%7Cc96f9f14-8ec3-45b4-89c3-33c089ff513a>

1.4. 用語定義

本書が使用する用語を以下に整理します。

表 1 本書で使用する用語の定義

用語	意味
Attended Robot (AR)	有人の環境で動作するロボット。人がロボット UI からプロセスの実行を指示すると、この Windows 端末上で自動化処理が実行される。
Orchestrator	AR/UR を管理するための UiPath 社のサーバー製品。ブラウザから操作できる。
RPA	Robotic Process Automation。ソフトウェアロボットによる業務自動化。
Studio	ワークフローを開発する UiPath 社のソフトウェア製品。
UI 要素	自動化プロセスが操作対象とする、Windows OS の画面に表示されるユーザーインターフェイスの要素。例えば、ウィンドウ、ボタン、エディットボックス、ラジオボタン、チェックボックス、メニュー項目など。
Unattended Robot (UR)	無人の環境で動作するロボット。Orchestrator からプロセスの実行を指示すると、当該の Windows 端末にロボットがログインして自動化が実行され、処理終了後にログオフする。
XAML	XML をベースとしたマークアップ言語。UiPath 製品においてはワークフロー実装の成果物であり、拡張子が.xml で BOM 付き UTF-8 のテキストファイル。
アクティビティ	ワークフローを構成するための部品。この開発には C# を使用する必要がある、ある程度のノウハウが必要。一般にユーザーが作成する必要はない。
アクティビティパッケージ	複数のアクティビティを含むパッケージファイル。Studio に同梱されているものと、外部サイトからダウンロードしてくるものがある。
シーケンス	ワークフローのルートにも配置できる特殊なアクティビティで、複数のアクティビティを含むコンテナとして機能する。配置されたアクティビティはその順で逐次実行される。
ジョブ	Orchestrator から実行開始を指示されたプロセス。
スケジュール	Orchestrator で、プロセス実行開始（ジョブの作成）を予約するもの。
スコープ	各変数/定数を利用可能な範囲。
パッケージ	Microsoft .NET テクノロジーにおいて、コンパイル済みのコード(.dll)やそのコードに関連する他のファイル、パッケージのバージョン番号などが記述されたマニフェストファイルなどを含むアーカイブファイル。.nupkg の拡張子をもつ。実行の際には外部のパッケージを必要とする場合があり、それらのパッケージ名とバージョンのリストが内部に管理されている。
パブリッシュ	プロジェクトにおいて、パッケージファイルを構築する操作。
フローチャート	ワークフローのルートにも配置できる特殊なアクティビティで、複数のアクティビティを含むコンテナとして機能する。配置されたアクティビティの実行順は有向線分により自由に構成できる。
プロジェクト	実行可能な自動化処理を構築する単位。プロジェクトを定義する project.json ファイルや、複数のワークフローファイルなどがひとつのフォルダーにまとめられている。プロジェクトには、プロセスパッケージを構築するプロセスプロジェクトと、ライブラリパッケージを構築するライブラリプロジェクトの 2 種類がある。
プロセス	ロボットが実行できる自動化の単位。オートメーションプロセス、自動化プロセスのこと。Windows のタスクマネージャに表示されるプロセスとは別のものなので注意が必要。
プロセスパッケージ	単一のプロセスを含むパッケージファイル。なお、プロセスパッケージを Orchestrator にアップロードすると、これは Orchestrator 上では「パッケージ」として表示される。これをロボットグループに関連づけることで、Orchestrator 上に「プロセス」として表示されることに留意のこと。

ライブラリ	ワークフローにより作成できる、再利用可能な部品。アクティビティと全く同様に利用できる。ユーザーが作成したワークフローを共有するために使用する。
ライブラリパッケージ	複数のライブラリを含むパッケージファイル。
ロボット	ワークフローを実行する UiPath 社のソフトウェア製品。
ロボット端末	ロボットを動作させる端末。
ワークフロー	自動化の処理の流れを定義したもの。XAML ファイルで実装される。

なお、本書においてアクティビティ名は【】で括弧で表記します。

2. 禁止事項

次のような業務の自動化は禁止されています。

- 業務システム、社外システムにログインする為のパスワード等のセキュア情報はワークフロー、設定ファイル、ログファイル等への記録を禁止する（適切なパスワード管理方式は、次節にて解説）
- システムに対して多大な負荷を与える自動化を禁止する
- 社外メールアドレスに対して業務担当者の確認なしにメールを送信する事を禁止する。ロボットはドラフト作成までを行い、業務担当者が内容を確認して送付するなど、人間による最終チェックが必要。
- 重要なデータに対する更新の自動化を禁止する。更新処理は必ず人間の確認の手順を設けること。更新後のデータ変更が不可・または困難な場合、データ更新前に人間が更新予定のデータ内容をチェックするステップを設けること。
- 承認業務を自動化することを禁止する。
- 重要なファイルのバックアップを取らずに直接更新することを禁止する。
- ロボットの停止時にビジネスへ甚大な影響を及ぼし得る業務の自動化を禁止する。
- ネットバンクを使った振込等、リカバリーが不可能かつ誤作動時の業務影響が大きい業務の自動化は禁止する。
- 個人情報を収集・社外公開する業務の自動化は禁止する。
- その他、人間が禁止されている行為について、ロボットを使う場合であっても禁止する。
 - 端末内のディスクへの重要データの保存
 - 重要データの社外送信
 - など

3. プロジェクトについて

3.1. プロジェクトフォルダー

Studio で新規に作成したプロジェクトの内容は、すべてプロジェクトフォルダーに配置されます。このフォルダー内には、好きな名前で作フォルダを作成したり、設定ファイルなどを配置したりして構いません。設定ファイルについては、ファイル名や形式（.xlsx や.csv など）を定めて、全てのプロジェクトで統一しましょう。

3.2. 必要なファイルは、プロジェクトフォルダーに入れる

プロジェクトフォルダー内に配置した任意のフォルダーやファイルは、パブリッシュにより生成されるプロセスパッケージに含まれます。これらのファイルは、【ワークフローファイルを呼び出し】アクティビティや【Excel アプリケーションスコープ】アクティビティなどについているブラウズボタンから参照できます。

一方で、不要なファイルや機密性の高いファイルは、プロセスパッケージファイルに含まれないようにするため、プロジェクトフォルダー内に配置しないで下さい。

3.3. プロジェクトテンプレートの活用

組織に応じて、適切なフレームワークをテンプレートとして使用してください。フレームワークとは、あるソフトウェア開発を行う上で必要な共通要素をテンプレート化／共通化したものです。本ドキュメントでは、「EUC 開発フレームワーク」の使用を推奨します。このフレームワークは、設定ファイルの読み込みとエラー処理の機能を提供するものです。この本体と説明書は、EUC 開発フレームワークに同梱されています。また Studio20.10 からは、Backstage ビューにテンプレートタブが追加されました。ここから、Orchestrator やインターネット上にあるフィードに登録されたテンプレートパッケージを指定して、新規にプロセスを作成できます。

3.4. 設定ファイルの活用

設定ファイルを用意しておき、このファイルにプロセス固有の設定値をまとめておくことは非常に良い習慣です。設定ファイルの扱い方をチーム全体で統一しておく、その後の保守も容易となるため、なるべく早い段階で設定ファイルの管理や操作方法を統一しておくとい良いでしょう。

4. ワークフローの設計

4.1. プロジェクトの大きさについて

自動化したい一連の操作について、ひとつのプロジェクトを作成してください。これは、人手で実行した場合には 1 分弱から数十分程度で完了する程度の大きさが適切です。ただし、プロセスを実行中にはその端末を人が操作できなくなるため、あまり長くなりすぎないように注意してください。

4.2. シーケンスとフローチャートの使い分けについて

基本的には、分岐する処理はフローチャートで作成してください。連続する処理はシーケンスで作成します。フローチャートの中に連続する処理を作成したいときは、フローチャートの中にシーケンスを配置してください。繰り返して実行する処理は、繰り返しのアクティビティをフローチャートもしくはシーケンスに配置して作成してください。

4.3. シーケンスの別ファイルへの抽出について

意味のある処理のまとまりがある程度の大きさになったら、それを別のワークフローに分割すると見通しが良くなります。これには、分割したいシーケンスを右クリックして、メニューから「ワークフローとして抽出」を選択してください。自動で【ワークフローファイルを呼び出し】が配置され、選択した部分のアクティビティが別のワークフローファイルに切り出されます。

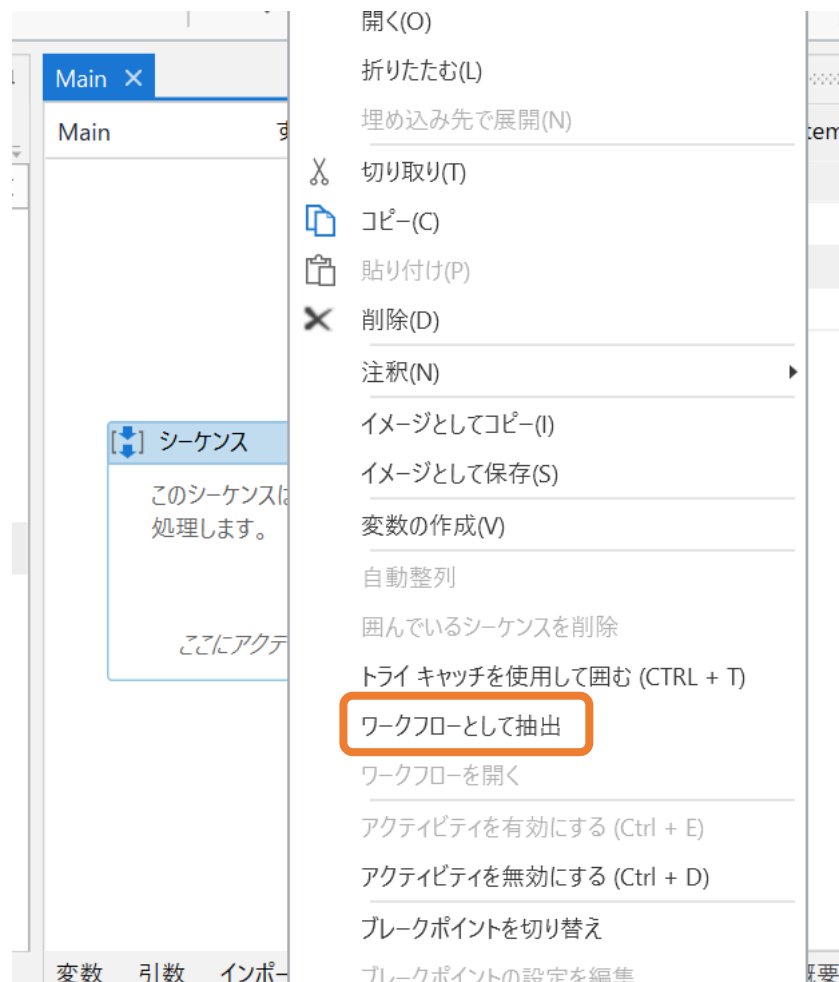


図 1 シーケンスを、ポップアップメニューから別のワークフローファイルに抽出

切り出したワークフローは、別の場所からも【ワークフローファイルを呼び出し】で呼び出して使うことができます。

5. 命名規則

プロジェクトやワークフローファイル、変数などの名前は、命名規則に則って決めましょう。命名規則に従っていないと、あとでプロジェクトの保守作業が大変になってしまいます。次節以降の例を参考に、プロジェクトで適用する命名規則を定めてください。

命名規則を定めるべき要素には次のものがあります。これらには、すべて日本語（かな漢字）を使って構いません。

- プロジェクト（プロジェクトフォルダー）

- プロジェクトに含まれるサブフォルダ
- 各ワークフローファイル (.xaml)
- 設定ファイル (.xlsx や .json など)
- 配置したアクティビティの表示名
- 変数と引数

5.1. プロジェクトの命名規則 (例)

プロジェクトの命名規則の例を表 2 に示します。

※プロジェクト名の命名ルールのみ順守必須とします。そのほかは各自で決めていただいて問題ありません。

表 2 プロジェクトの命名規則 (例)

No	項目名	命名ルールの例	命名の例
1	プロジェクト フォルダー	<p><u><組織コード>_<業務名称> (順守必須)</u></p> <p><u><組織コード>_<業務番号>_<業務名称></u></p> <p>上記のように業務 ID を付与する場合には、必要に応じて業務 ID の長さや形式も定めて下さい。部署名を表す識別子などを含めてユニークな名称とすることも考慮して下さい。</p> <p>この名称がそのままプロジェクトフォルダー名称となり、また実行時のプロセス名 (.nupkg ファイル名) となります。</p>	<ul style="list-style-type: none"> ● 50001345_採用リスト更新 ● 50001345_P000221_メンバーリスト更新

※命名に使用できる文字コードは UiPath のソフトウェア要件を満たす OS に準拠したものになります。

5.2. XAML ファイルの命名規則 (例)

XAML ファイルの命名規則の例を表 3 に示します。

表 3 XAML ファイルの命名規則 (例)

No	項目名	命名ルールの例	命名の例
1	XAML ファイル	<p><業務番号>_<処理名>.xaml</p> <p><業務番号>_<連番>_<処理名>.xaml</p> <p><共通部品番号>_<処理名>.xaml</p> <p>なお、Main ファイルは名称変更せず「Main.xaml」のまま使用します。</p> <p>処理名は、意味が分かりやすい簡潔なものとします。</p>	<ul style="list-style-type: none"> ● D000011_初期設定.xaml ● P000123_010_計算マクロ実行.xaml ● COM0001_画面キャプチャ取得.xaml

5.3. アクティビティの表示名の命名規則 (例)

アクティビティの表示名の命名規則の例を表 4 に示します。

表 4 アクティビティの表示名の命名規則 (例)

No	項目名	命名ルールの例	命名の例
----	-----	---------	------

1	アクティビティ	<p><u><処理名称></u></p> <p><u><処理番号>_<処理名称></u></p> <p>【シーケンス】や【フローチャート】の表示名はデフォルトのままとせず、必ず処理内容を理解できる任意の名称を日本語で設定します。必要に応じて、一意の処理番号を付与すると、トラブルシューティング時に問題個所の特定が容易になります。</p> <p>それ以外のアクティビティの表示名は修正を必須とはしませんが、なるべくわかりやすい名称を付与することを推奨します。例えば【クリック】アクティビティの表示名は「ログインボタンをクリック」のような名前を付けます。</p>	<ul style="list-style-type: none"> ● 初期設定 ● 本日のデータを取得 ● 020_ログインボタンをクリック
---	---------	---	---

6. 変数と定数について

管理しやすい変数を作成するポイントを列挙します。名前と既定値については、引数を使うときにも同じように考慮してください。

6.1. わかりやすい名前にする

読んで意味のわかる変数名にしましょう。下記に例を示します。

型	悪い変数名の例	良い変数名の例	メモ
Int32	データ	代金	「データ」では、意味が分かりません。
String	file	ファイル名	変数名には、なるべく日本語を使いましょう。
Boolean	チェックフラグ	存在するか	「チェックフラグ」では、値が True/False のときに、それぞれどのような意味となるか分かりません。

6.2. 既定値を設定する

既定値とは、変数に自動でセットされる最初の値です。既定値を設定できるときには、必ず設定してください。これは変数パネルで変数ごとに設定できます。

6.3. スコープを狭くする

スコープとは、その変数を使える範囲のことです。スコープは、変数パネルで変数ごとに設定できます。このスコープを広くすると、管理しなければならない変数の数が増えてしまうために不具合が発生しやすくなります。それぞれの変数に、なるべく狭いスコープを設定してください。

また、item 変数の名前を変更することもできます。これには、アクティビティの左上に要素として表示された項目を修正します。ほかの変数と同じように、この変数にもわかりやすい名前をつけてください。

7. ワークフローを読みやすくする

7.1. 表示名の変更

アクティビティの表示名プロパティに、わかりやすい名前をつけましょう。とくに、【シーケンス】や【条件分岐】などの表示名は、わかりやすいものに変更することを強くお勧めします。これによりワークフローがとても読みやすくなります。

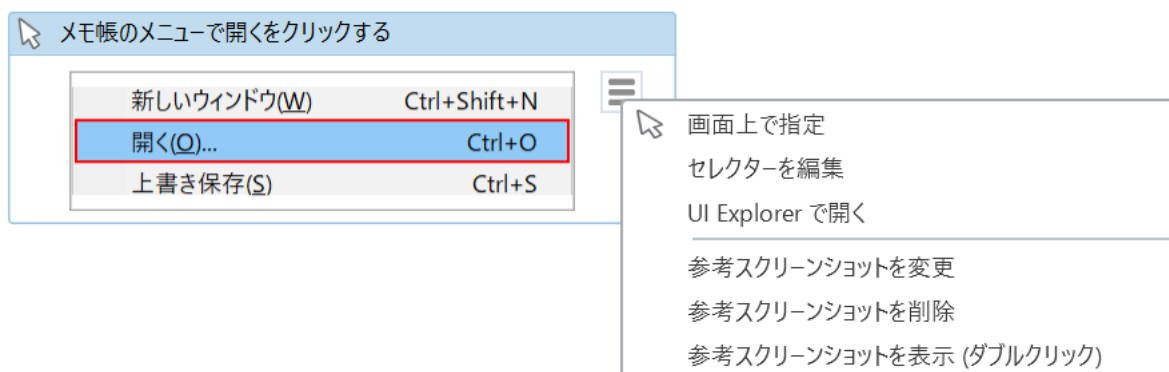
7.2. 注釈の設定

アクティビティの表示名には、長いテキストを設定すると見切れてしまいます。より長い説明文を記載しておきたいときには、注釈を使いましょう。アクティビティを右クリックして「注釈」→「注釈の追加」を選択して、注釈を追加できます。とくに、各ワークフローの最上位のアクティビティ（【シーケンス】や【フローチャート】など）には注釈を追加し、そのワークフローの処理概要を記載しておきましょう。

8. 不要な内容は削除する

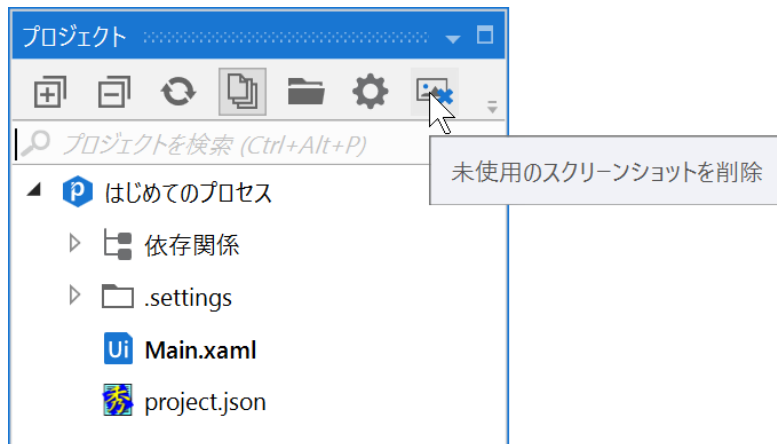
8.1. 参考スクリーンショットの保持

【クリック】などのアクティビティでは、セレクトア設定時に自動でスクリーンショットがプロジェクトの.screenshots フォルダに保存されます。これを参考スクリーンショットといいます。参考スクリーンショットは、当該のアクティビティ上に表示されます。



参考スクリーンショットは、このワークフローの処理内容を説明するドキュメントとして非常に重要なものです。このため、プロジェクトの.screenshots フォルダにある画像ファイルは削除しないでください。

ただし、参考スクリーンショットを撮り直しても、以前の参考スクリーンショットは自動で削除されません。使われていない参考スクリーンショットは、定期的に削除してください。これはプロジェクトパネルの上部にあるツールボタンで行えます。



8.2. 使用していない変数の除去

使っていない変数は不具合の元になるので、削除してください。Studio のデザインリボンにある「未使用の変数を削除」ボタンをクリックして削除できます。

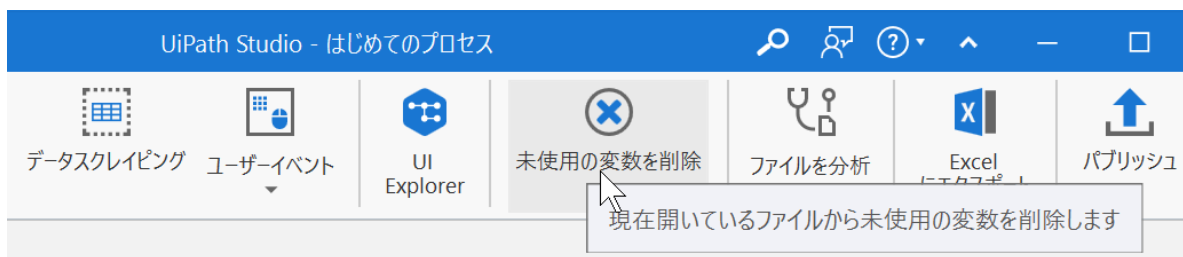


図 3 「未使用の変数を削除」ボタン

8.3. 不要になったアクティビティやコメントの削除

ワークフローの開発中には、フローチャート上に有向線分で接続されていないアクティビティや、シーケンス上にコメントアウトしたアクティビティを一時的に配置したままとすることがあります。しかし、開発が完了したら、そのようなアクティビティは削除してください。これは、後日このワークフローを保守する担当者が、実行されないアクティビティを見て混乱しないように、なるべく簡潔で見やすいワークフローを引き渡せるようにするためです。

9. 安定したセレクターの構築

セレクターとは、アクティビティが操作対象とする UI 要素を特定するための文字列データです。例えば、【クリック】や【文字を入力】アクティビティには、セレクタープロパティがあります。Studio で対象の UI 要素を指定する操作をすると自動でセレクターが生成されますが、状況に応じてこのセレクターを調整・修正する必要があります。プロセスの動作が安定しない原因の多くは、セレクターが適切に構成されていないことによります。本節は、セレクターの構成方法について説明します。

9.1. セレクターが期待通り動作しないときは

セレクターが期待通りに動作しないときは、UI フレームワークの切り替えを試して下さい。セレクターが動作するようになることがあります。UI フレームワークには、下記の 3 種類があります。切り替える場合には、既定->UI Automation->Active Accessibility の順でお試し下さい。

- 既定
- UI Automation (UIA)
- Active Accessibility (AA)

UI フレームワークを切り替えるには、セレクターを設定するときに [F4] キーを押します。

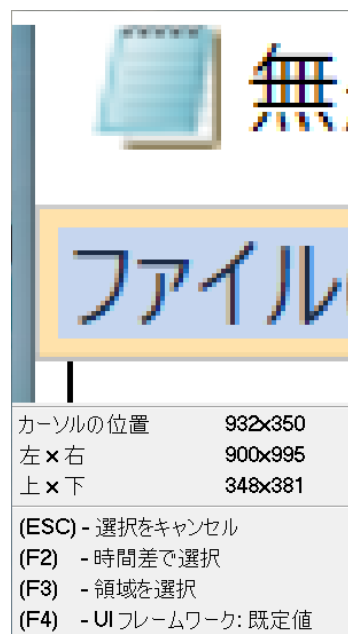


図 4 セレクター設定時のプレビューウィンドウ

9.2. 意味がある値を含む属性を使用し、変化しそうな値を含む属性は使わない

人間が読んで意味がある値（例：MenuBar など）を含む内容は、操作対象アプリケーションのバージョンアップなどによる環境の変化の影響を受けにくいといえます。一方で、何らかの意味を含まない値（例：意味のない文字や数字の羅列など）は、環境の変化を受けやすく、開発中には動作したとしても運用時に突然動作しなくなる危険があります。セレクターエディターでセレクターの内容を確認し、安定した形になるように編集・調整します。

セレクターに含まれる属性の例	使って良いか	メモ
MenuBar	良い	このように人が読んで意味のわかる名前の属性は、変化しにくいと考えられる。
idx	避けるべき	ウィンドウ上の UI 要素に連番(index)を付与する。特にこの値

		が大きいたまは、画面の変更により影響を受けやすいため、セレクトアでの使用は避ける。
--	--	---

9.3. セレクトアエディタで、セレクトアを修復する

操作したい対象のセレクトアは、当該のアプリケーシオンを実行するたびに変化してしまう場合があります。このようなたときには、当初に作成したセレクトアは期待通りに動作しなくなってしまう。このような場合には、セレクトアエディタの修復機能でセレクトアを修復してください。最初に指定した UI 要素と、修復時に指定した UI 要素の両方に合致するセレクトアが自動で生成されます。このように自動で修復されたセレクトアの中には、* (アスタリスク)が含まれることがあります。これは、任意のテキストと合致するワイルドカードとして機能します。なお、手作業でセレクトアを編集するたも、* を埋め込むことができます。

9.4. 画像認識のアクティビティはなるべく使わない

UiPath には、セレクトアを使わずに画像認識により UI 要素を特定できるアクティビティが多く用意されています。例えば【画像をクリック】や【画像が出現したとき】などです。

これらのアクティビティは、どうしてもセレクトアを期待通りに動作させることができない場合の代替として用意されたものです。画像一致によるアクティビティは、ロボット端末の解像度の変化に弱いため、一般にセレクトアで動作するアクティビティの方が安定して動作します。状況によっては、対象の画像を見つけられなかったり、別のものを誤って見つけたと判断してしまったりすることもあります。このため、どうしてもその必要がある場合を除いては、画像一致によるアクティビティは使用しないで下さい。

(応用編)

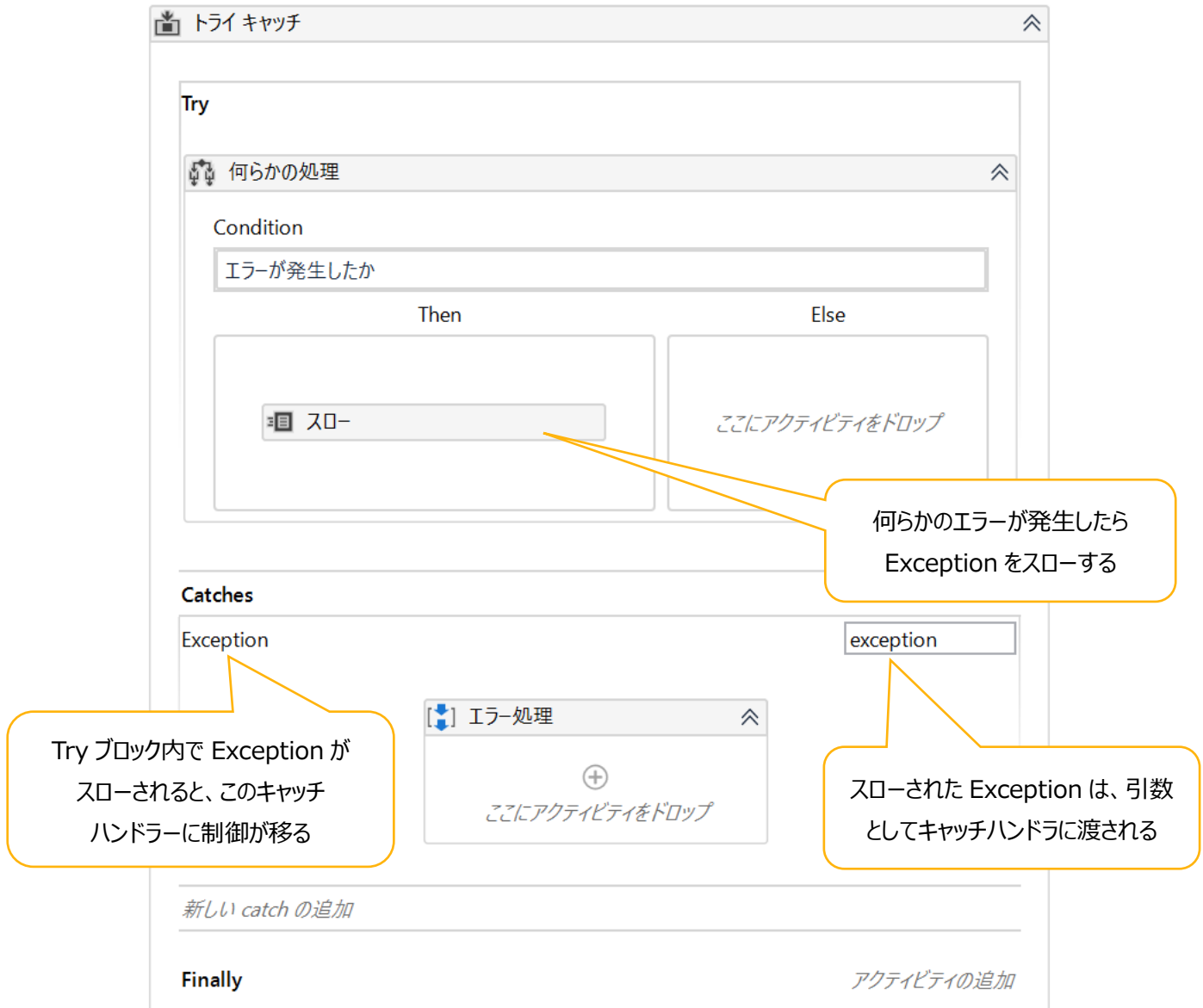
10 章以降は、応用編となります。Attended プロセスを作成するたにも安定性や保守性の観点から有効なテクニックではありますが、必須のものではありません。必要に応じてご参照下さい。

10. 例外処理

10.1. 例外処理とは

例外処理とは、何らかのエラーが発生した場合には、このエラー処理を行う箇所に制御を移すための仕組みです。例外処理において、エラー処理を行う箇所をキャッチハンドラーといいます。キャッチハンドラーには、発生したエラーの情報を引数として渡すことができます。このエラー情報を例外 (Exception) といいます。何らかのエラーが発生したら、この例外オブジェクトを作成してキャッチハンドラーにスローし (投げ) ます。例外をスローするには、【スロー】アクティビティを使います。これにより、後続のアクティビティ実行はすべてスキップされ、直近のキャッチハンドラーに制御が移ります。キャッチハンドラーでは、引数としてキャッチした例外オブジェクトから必要なエラー情報を取り出してエラー処理を行えます。スローした例外をキャッチするハ

ンドラーが見つからないときは、この例外はプロセスの外部にまで漏れていき、このプロセスは異常終了します。
例外処理の構造は【トライキャッチ】アクティビティで記述します。この基本的な構造を図 5 に示します。



必要なエラー処理の具体的な内容は、失敗した処理内容や発生したエラーの種類に依存するため、一般に説明することはできません。次節より、例外オブジェクトの扱い方と、処理順の制御方法について説明します。

10.2. アプリケーションエラーとビジネスエラーの区別

例外をスローするときは、その型を適切に選択する必要があります。これには、エラー発生の原因をアプリケーション由来のも

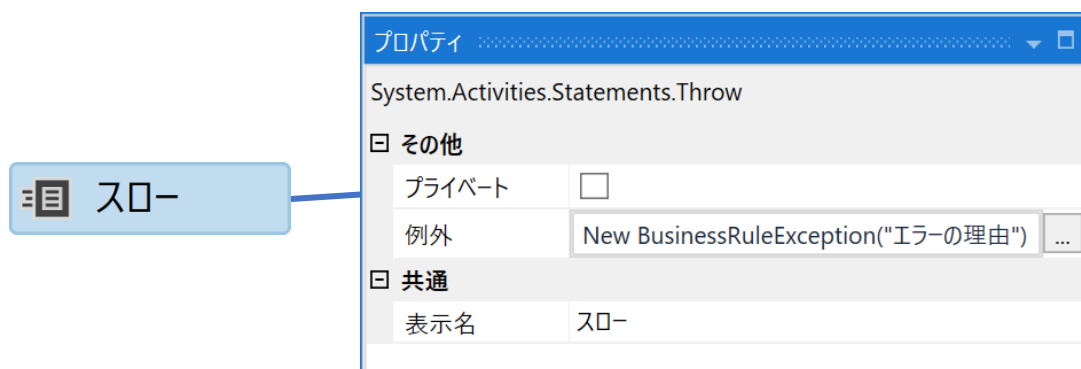
のとビジネス由来のものと区別することが役に立ちます。便宜上、それぞれをアプリケーションエラーとビジネスエラーといいます。アプリケーションエラーとは、ネットワークフォルダにアクセスできないとか、対象のアプリケーションがフリーズして操作できないなど、技術的な問題が原因で発生するエラーです。このときは、標準アクティビティが自動で例外をスローします。

ビジネスエラーとは、ビジネスエラーとは、自動化プロセスで使用するデータが不完全または欠損していることが原因で発生するエラーです。このときは、【スロー】を使って `BusinessRuleException` をスローしてください。あるいは、同梱の EUC フレームワークに含まれる【ビジネスルール例外をスロー】を使って簡単にスローすることもできます。

上記のように何らかの例外をスローする可能性がある部分は、【トライキャッチ】の Try ブロックの中に入れます。この【トライキャッチ】には `Exception` と `BusinessRuleException` のハンドラーを作成し、それぞれのエラー処理を記述してください。

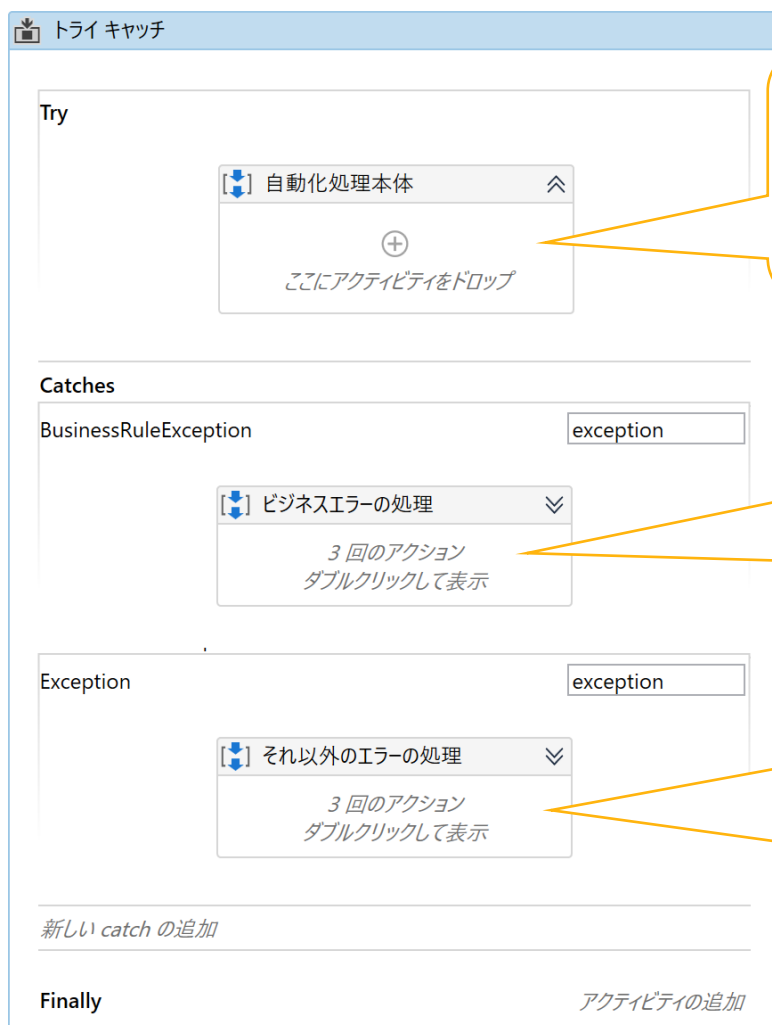
10.3. 例外の作成とスロー

スローすべき例外の型を見つけたら、この型のオブジェクトを new して、【スロー】アクティビティでスローします。上述の通り、ワークフローの開発者が明示的に例外をスローするときは、`BusinessRuleException` をスローすることになるでしょう。次のようになります。



10.4. 例外のキャッチ

例外をキャッチするには、同一の【トライキャッチ】に `BusinessRuleException` と `Exception` のハンドラーを作成してください。次のようになります。



ビジネスエラーが発生したら、この中で BusinessRuleException をスローする（それ以外の例外は、標準アクティビティが自動でスローする場合がある）

BusinessRuleException はここでキャッチされるので、この例外情報をログに出力したり、業務担当者にエラー発生を旨をメールしたりするなどの処理を行う

それ以外の例外は、すべてここでキャッチされるので、この例外情報をログに出力したり、画面写真を撮影したり、IT 担当者にエラー発生を旨をメールしたりするなどの処理を行う

なお、ある【トライキャッチ】に配置するキャッチハンドラーに応じて捕捉できる例外の種類が異なります。これを下表に整理します。状況に応じて使い分けてください。

同一の【トライキャッチ】に作成する Catch ハンドラー		この【トライキャッチ】で捕捉する 例外の種類
BusinessRuleException ハンドラー	Exception ハンドラー	
作成する	作成しない	BusinessRuleException ハンドラーで BusinessRuleException 例外を捕捉します。それ以外の例外は捕捉せず、この【トライキャッチ】の外に漏らします。
作成しない	作成する	この Exception ハンドラーで、BusinessRuleException 例外を含むすべての例外を捕捉します。
作成する	作成する	BusinessRuleException ハンドラーで、

		BusinessRuleException 例外を捕捉します。 Exception ハンドラーで、それ以外のすべての例外を捕捉します。
--	--	--

10.5. キャッチハンドラーにおける一般的なエラー処理

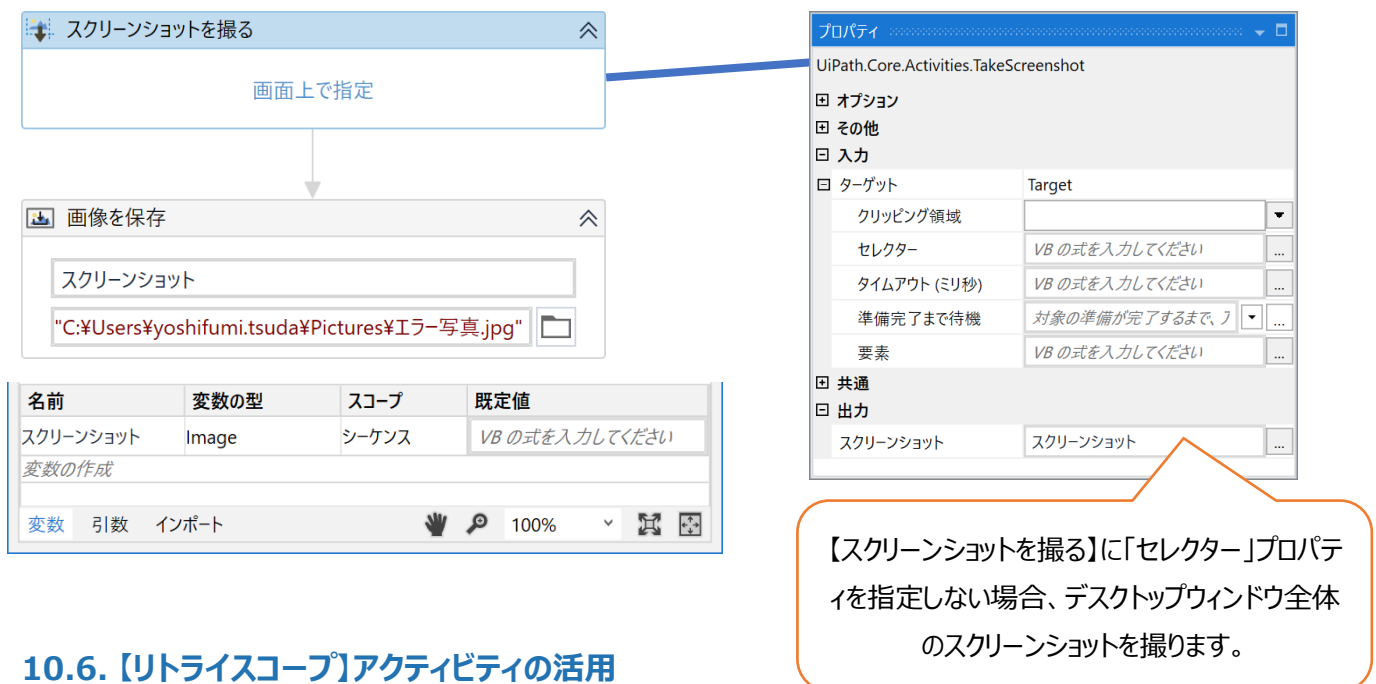
エラー処理には、状況に応じてさまざまな内容が考えられます。本節では、一般的なエラー処理の概要について補足します。

ログを出力する

Exception 型の変数には多くの情報が含まれています。これをログに出力する部品が EUC 開発フレームワークテーブルに含まれています。この使用方法を示します。

スクリーンショットを採取する

例外発生時のスクリーンショットがあると、問題解決に大変役に立ちます。スクリーンショットを採取する方法を下記サンプルに示します。



【スクリーンショットを撮る】に「セレクター」プロパティを指定しない場合、デスクトップウィンドウ全体のスクリーンショットを撮ります。

10.6. 【リトライスコープ】アクティビティの活用

【リトライスコープ】アクティビティは、この中に含まれるアクティビティを実行に失敗したとき、この中に含まれるアクティビティを最初から再実行(リトライ)します。リトライの前に、リトライの間隔プロパティに設定された時間を待機します。リトライする回数の上限は、リトライの回数プロパティで指定します。

実行に失敗したと判断する条件は、下記のふたつです。

含まれるアクティビティのいずれかが、任意の例外をスローしたとき

リトライスコープ内部のどこかで例外がスローされたら、リトライスコープ内部に含まれる後続の処理をスキップして、リトライスコープの最初から処理をリトライします。リトライの前に何らかの初期化処理やエラーからの回復処理（アプリケーションウィンドウやブラウザウィンドウをクローズするなど）をしたい場合には、リトライスコープの中に【トライキャッチ】を配置して、その Exception ハンドラーの中に当該の処理と【リスロー】を配置します。

もしもリトライの回数が上限に達していた場合にはリトライせず、この例外をそのままリトライスコープの外にスローする動作となります。

【リトライスコープ】の条件に指定したアクティビティが False を返したとき

自動化処理の開発において、ある操作をした結果、何らかのテキストメッセージやダイアログウィンドウが表示されたら成功、表示されなければ当該の操作を再試行したい、ということはよくあります。このような処理を簡単に実装できるように、【リトライスコープ】アクティビティの条件には次のようなアクティビティを配置できます。

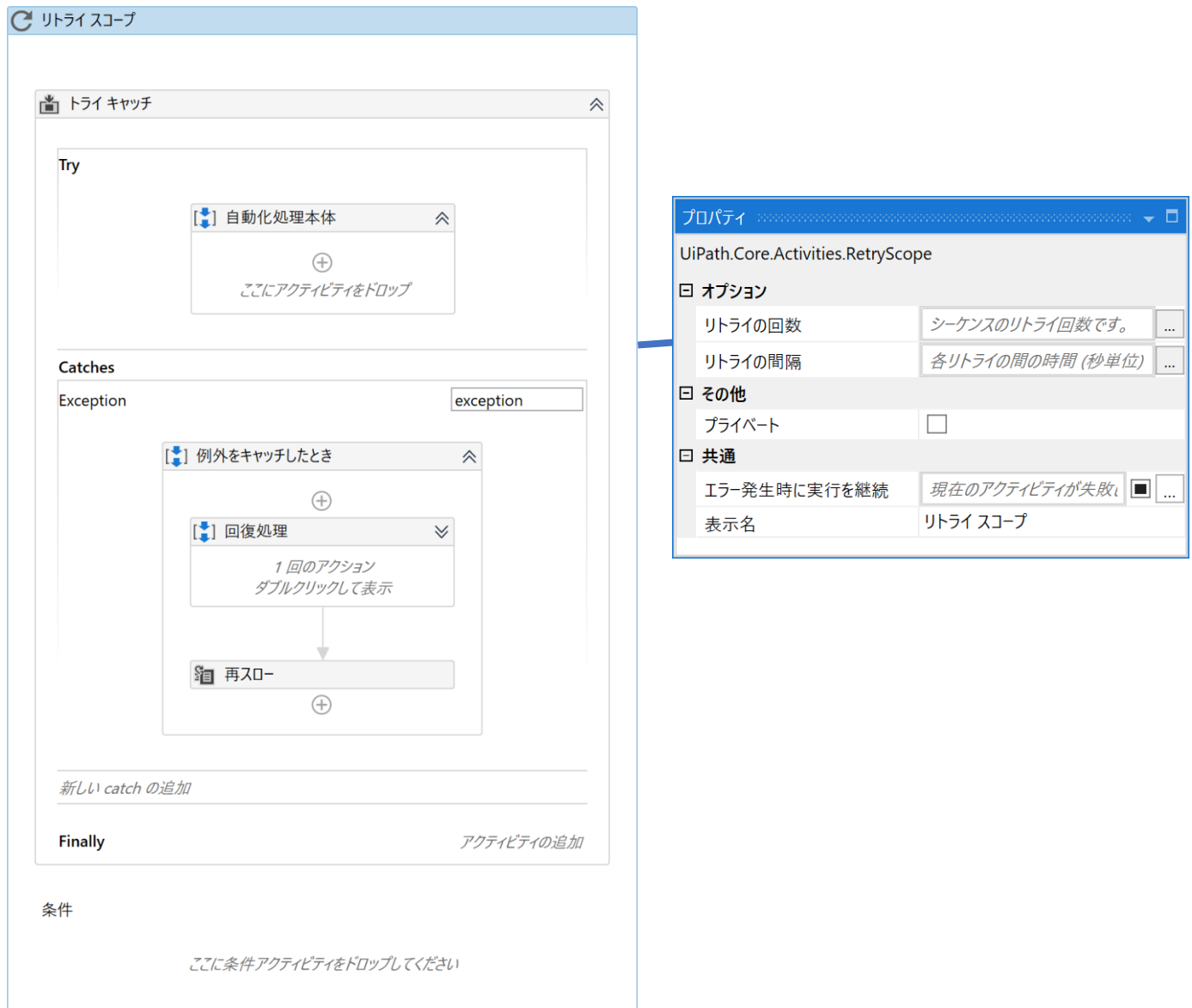
- 画像の有無を確認 (ImageFound)
- 要素の有無を検出 (UiElementExists)
- テキストの有無を確認 (TextExists)
- OCR でテキストの有無を確認 (OCRTextExists)
- コレクション内での有無 (ExistsInCollection<T>)

【リトライスコープ】の内部に配置されたアクティビティを全て実行した後に、条件に配置されたアクティビティを実行し、この結果が False であればリトライします。

もしもリトライの回数が上限に達していた場合にはリトライせず、Exception 型の例外をリトライスコープの外にスローする動作となります。

【リトライスコープ】の使用例

自動化処理本体が例外をスローしたとき、回復処理を実行してから自動化処理本体を再実行したい場合には、次のように構成します。リトライスコープの内部で例外を再スローすることにより、このリトライスコープの先頭から処理を再実行できます。リトライ時に回復処理が不要な場合には、【リトライスコープ】の中に【トライキャッチ】を配置する必要はなく、自動化処理本体だけを直接配置すれば OK です。



11. プロセス実装における作法

本節では、プロセス実装において守るべき作法を紹介します。

11.1. 【待機】アクティビティの無分別な利用を避け、使用する場合には必ず注釈を記載する

【待機】アクティビティは、プロセスの安定稼働に寄与する場合があります。画面が落ち着いてから次の操作に進めるようになるからです。

しかし、【待機】アクティビティを無分別に使うと、プロセスの処理が遅くなってしまいます。また、なぜ、どのくらいの期間を待機しなければならないのかわからないと、配置された【待機】を削除できないため、保守も難しくなります。

このため、【待機】アクティビティは可能な限り使用すべきではありません。その代わりに、次の操作に進めることを示す何らか

の画面上のサインが現れるまで待ち、それから次の操作実行に進むことができます。これには、【画像の有無を確認】や【画像が出現したとき】などのアクティビティが役に立ちます。これにより、無駄な待ち時間を発生させることなく、効率的かつ安定した動作が得られます。

どうしても【待機】アクティビティを使用する必要がある場合には、必ずこの【待機】アクティビティに注釈を追加し、その必要性と待機時間の根拠を記載して下さい。記載できない場合には、この【待機】アクティビティは削除しましょう。

11.2. 【クリック】アクティビティの入力方法を適切に選択する

【クリック】アクティビティを使用するときは、入力方法を制御するプロパティを次の順で試し、動作するものを選択してください。

表 5 【クリック】アクティビティのプロパティによる動作比較

プロパティ	手作業との互換性	バックグラウンドで動作	処理速度
クリックをシミュレート	99% (web アプリ) 60% (desktop アプリ)	はい	100%
ウィンドウメッセージを送信	80%	はい	50%
チェックなし (既定)	100%	いいえ	50%

11.3. 【文字を入力】アクティビティの入力方法を適切に選択する

前節と同様のことが、【文字を入力】アクティビティにもいえますが、まずはこのアクティビティを試す前に【テキストを設定】アクティビティをお試しください。【テキストを設定】アクティビティが期待通り動作できなかった場合には、【文字を入力】アクティビティを使います。このプロパティ設定も、上記の表に基づいて行ってください（表中の「クリックをシミュレート」は、「入力」をシミュレート」に読み替えてください）。

11.4. 【ホットキーを押下】アクティビティには、必ずセクタープロパティを設定する

【ホットキーを押下】アクティビティにセクターを指定しない場合、このアクティビティを実行時にアクティブな（前面に表示されている）ウィンドウにホットキーを送信する動作となります。これは、このプロセスの不安定な動作を引き起こす可能性があります。安定した動作を期すために、【ホットキーを押下】アクティビティには必ずホットキーを送信したい先の UI 要素をセクタープロパティで指定してください。

11.5. メール送信時の宛先と本文に注意する

プロセスでは、誤って多くの宛先に大量にメールを送信してしまう事故が発生しやすいため、特に注意が必要です。開発中には実際の電子メールアドレスにメールを送ってしまうことがないように、メールアドレスは設定ファイルなどに記述して、運用時と開発時の設定ファイルを分けるなどの管理を行って下さい。メールの本文についても、正しいものとなるように留意して下さい。

11.6. メール受信時の添付ファイルに注意する

不特定の送信者から送信された外部メールの添付ファイルが、悪意を含む可能性に留意します。その前提に立った処理を自動化するのでない限りは、このような添付ファイルをプロセスで取得/展開することは避けて下さい。

同様に、不特定の送信者から送信された外部メールの受信をトリガーとして、何らかの処理を自動化することも控えましょう。フィッシングや各種スパムメール、攻撃型メールなど、意図しないメールに対して動作する可能性があるためです。

12. ログ出力

UiPath 製品はさまざまなログを出力しますが、この中で特に有益なのがロボットの実行ログ (yyyy-mm-dd_Execution.log) です。【メッセージをログ】アクティビティに指定したテキストが、この実行ログに出力されます。何らかの問題が発生したとき、適切なログメッセージが出力されていれば、問題解決にとっても役立ちます。

12.1. ログメッセージを出力する場所

基本的には、各処理の開始地点と終了地点に【メッセージをログ】アクティビティを配置して、当該の処理の開始/終了の旨をログに出力してください。ログメッセージを有益なものとするには、そのログがどこで出力されたのかがわかるようにする必要があります。

また、ログは、多く出力すればするほど良い訳ではありません。ログ行が多すぎると、必要な情報を探すことが大変になりますし、ディスク容量を圧迫したり、Orchestrator のパフォーマンス劣化を引き起こしたりする可能性もあります。特に、ループ処理の内部でログを出力した場合には、ログの量が多くなり過ぎることがあります。

12.2. ログメッセージの書式

ログメッセージの書式を統一しておくと、ログの分析が容易となります。特に、当該のログがどこで出力されたのかが、このログの分析に重要な情報となります。このため、ログの出力箇所が同じ書式で各ログメッセージに含まれるように、下記の構成を推奨します。

各ワークフローにて、XAML ファイル名を設定したログヘッダ定数を定義し、この定数を【メッセージをログ】アクティビティで参照するようにして、同じ書式で XAML ファイル名が出力されるようにします。XAML ファイル名は既定で fileName フィールドに出力されますが、このフィールドは Orchestrator で閲覧しにくいいため、ログメッセージ (message フィールド) にもファイル名を含めるための工夫です。

名前	変数の型	スコープ	既定値
ログヘッダ	String	シーケンス	"Main.xaml"
変数の作成			
変数 引数 インポート			

図 6 全ての XAML ファイルの先頭でログヘッダ定数を定義

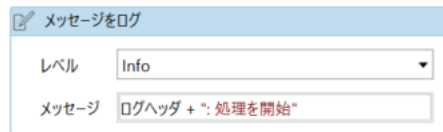


図 7 ログ出力時にログヘッダ定数を参照

12.3. 【メッセージをログ】アクティビティに指定すべきログレベル

各ログメッセージに適切なログレベルを設定することは非常に重要です。エラーではないのにエラーレベルが設定されたログメッセージが大量に出力されていると、本当にエラーのメッセージが埋もれて見えなくなってしまうからです。このため、各ログメッセージには適切なログレベルを設定してください。ログレベルは、【メッセージをログ】で指定できます。

基本的に、Error, Warn, Info のいずれかのみを使うことを推奨します。

表 6 各ログメッセージに対して設定すべきログレベル

ログレベル	このログレベルにて出力すべきメッセージの例
Fatal	回復できない致命的なエラー発生時に、このエラーの調査に有益な情報。ほかのプロセス実行にも影響があるような環境の不整合の検出時など。
Error	エラー発生時に、このエラーの調査に有益な情報。このエラーメッセージを受けて、運用担当者が何らかのアクションを必要とする時。
Warn	このプロセスの正常実行には失敗したが、自動で回復可能であり、運用担当者によるアクションは不要な時。
Info	何らの問題は発生していないが、何らかのまとまった処理の開始時や終了時など、ログに記録しておくべき情報。
Trace	この【メッセージをログ】アクティビティを通過したことを示して処理を追跡(トレース)できるようにし、デバッグ時やトラブルシュート時に有益となる情報。