



目次

- 1. 改訂情報
- 2. はじめに
 - 2.1. 本書の目的
 - 2.2. 対象読者
 - 2.3. 本書の構成
- 3. Accel Studio テスト機能 とは
 - 3.1. 全体像
 - 3.2. 連携フロー
- 4. セットアップ
 - 4.1. エージェントの実行環境について
 - 4.1.1. 必要なネットワーク環境
 - 4.1.2. 利用されるブラウザ
 - 4.1.3. エージェントの実行環境OS
 - 4.2. テスト機能API実行者へのロールの付与
 - 4.3. APIキーの取得
 - 4.4. 設定ファイルの変更
 - 4.5. 非同期タスクキューの設定（2025 Spring(Kamille) からアップデートの場合）
 - 4.6. エージェントの実行
 - 4.6.1. エージェントのダウンロード
 - 4.6.2. オフライン環境の場合の事前準備
 - 4.6.3. エージェントの起動
- 5. 設定ファイル
 - 5.1. Accel Studio テスト機能 の設定
 - 5.1.1. 概要
 - 5.1.2. リファレンス
 - 5.2. Accel Studio テスト機能 Copilot の設定
 - 5.2.1. 概要
 - 5.2.2. リファレンス
- 6. 機能仕様
 - 6.1. テスト定義
 - 6.1.1. 国際化
 - 6.1.2. テスト定義カテゴリ
 - 6.1.3. カテゴリの階層化
 - 6.1.4. カテゴリの削除
 - 6.1.5. 前処理、後処理
 - 6.1.6. テストケース
 - 6.1.7. テストリソース
 - 6.1.8. テストの定期実行について
 - 6.2. テスト実行
 - 6.2.1. 実行ユーザ
 - 6.2.2. テストコード
 - 6.2.3. IM-Workflow API の利用
 - 6.3. テスト結果
 - 6.3.1. テスト結果ステータス
 - 6.3.2. テストケース結果
 - 6.3.3. 実行時ログ
 - 6.3.4. 実行時エラーログ
 - 6.3.5. 実行時タイムライン
 - 6.4. テスト生成
 - 6.4.1. テスト生成について
 - 6.4.2. 全体像

- 6.4.3. 連携フロー
- 6.4.4. テスト生成の流れ
- 6.4.5. テスト定義一括生成
- 6.4.6. テスト生成一覧
- 6.4.7. テスト生成詳細
- 6.4.8. ワークフローテスト生成
- 6.5. インポート/エクスポート
 - 6.5.1. インポート/エクスポートで扱う情報
 - 6.5.2. インポート/エクスポート時の動作
- 7. チュートリアル
 - 7.1. チュートリアルの概要
 - 7.2. 準備・環境設定
 - 7.3. テスト定義の作成
 - 7.4. 前処理・後処理のフロー定義作成
 - 7.5. 前処理・後処理の指定
 - 7.6. 操作記録によるテストコードの作成
 - 7.7. テスト実行
 - 7.8. テスト結果の確認
 - 7.9. テスト定義一括生成
- 8. 著作権および特記事項

改訂情報

変更年月日	変更内容
2025-04-01	初版
2025-10-01	<p>第2版 下記を追加・変更しました。</p> <ul style="list-style-type: none"> ▪ 「Accel Studio テスト機能 とは」 - 「連携フロー」のシーケンス図に、「エージェント」から「Playwright」の処理を追加 ▪ 「機能仕様」 - 「テスト結果」 - 「実行時タイムライン」にテストステップが出力される旨追加 ▪ 「チュートリアル」 - 「テスト結果の確認」のテスト結果のデザインを変更 ▪ 「設定ファイル」 - 「Accel Studio テスト機能 Copilot の設定」にテスト生成機能の設定を追加 ▪ 「機能仕様」 - 「テスト生成」にテスト生成機能の機能仕様を追加 ▪ 「チュートリアル」にテスト定義一括生成の手順を追加
2026-04-01	<p>第3版 下記を追加・変更しました。</p> <ul style="list-style-type: none"> ▪ 「機能仕様」 - 「テスト結果」 - 「テスト結果ステータス」に「テスト実行エージェント接続エラー」ステータスを追加 ▪ 「設定ファイル」 - 「Accel Studio テスト機能 の設定」に「エージェント接続を行う際のタイムアウト値」に関する補足情報を追加 ▪ 「セットアップ」 - オフライン環境に対応したエージェントのセットアップ手順に変更（「オフライン環境の場合の事前準備」として使用ライブラリとブラウザデータの事前準備方法を追加、バージョン解決の優先順位に関する補足情報を追加、プロパティ一覧を更新） ▪ 「機能仕様」 - 「テスト生成」に「ワークフローテスト生成」を追加 ▪ 「機能仕様」 - 「テスト実行」で「<code>getSystemMatterIdFromActive</code>」と「<code>getSystemMatterIdFromCompleted</code>」にリトライオプション（<code>retryOption</code>）を追加

はじめに

項目

- 本書の目的
- 対象読者
- 本書の構成

本書の目的

本書は、Accel Studio テスト機能 を利用するユーザのみなさまの支援を目的としたドキュメントです。

対象読者

本書では以下のユーザを対象としています。

- Accel Studio で作成したアプリケーションのE2Eテストを実行したい

本書の構成

本書は以下のように構成されています。

- [Accel Studio テスト機能 とは](#)
本書、および、Accel Studio テスト機能 の概要について説明します。
- [セットアップ](#)
Accel Studio テスト機能 のセットアップ手順の仕様を説明します。
- [設定ファイル](#)
Accel Studio テスト機能 の設定ファイルについて説明します。
- [チュートリアル](#)
Accel Studio テスト機能 のチュートリアルです。
- [著作権および特記事項](#)
著作権および特記事項について記載します。

Accel Studio テスト機能 とは

Accel Studio テスト機能 は、E2Eテストの自動化ツールです。

E2Eテストを自動化することで、以下の場合に、テストに費やす時間を短縮できます。

- intra-mart Accel Platform のアップデート時
- Accel Studio アプリケーションの変更時
- ブラウザのアップデートや実行環境変更時

Accel Studio テスト機能 は、Playwrightを利用して、テストを実行します。

Accel Studio で作成したアプリケーションに対して、以下の機能を提供します。

- Accel Studio のリソースとしてのテストシナリオ管理
- Accel Studio アプリケーションに対するテスト実行管理
- Accel Studio アプリケーションに対するテスト結果管理
- Accel Studio アプリケーションをテストする上で最適なテスト支援機能

Accel Studio テスト機能 Copilot モジュールを利用することで、以下の機能を提供します。

- テスト定義の一括生成
- テスト定義生成の実行ログ参照

Playwrightは、Microsoft によって開発された、オープンソースのブラウザテストの自動化ライブラリです。

Accel Studio テスト機能 の動作、サポートは Playwright のサポートポリシーに準拠します。

コラム

Accel Studio テスト機能 は、テスト機能のため、開発環境や検証環境での利用を想定しています。

そのため、本番環境で Accel Studio テスト機能 を利用しない場合、「[Accel Studio テスト機能 の設定](#)」で動作不可にできます。

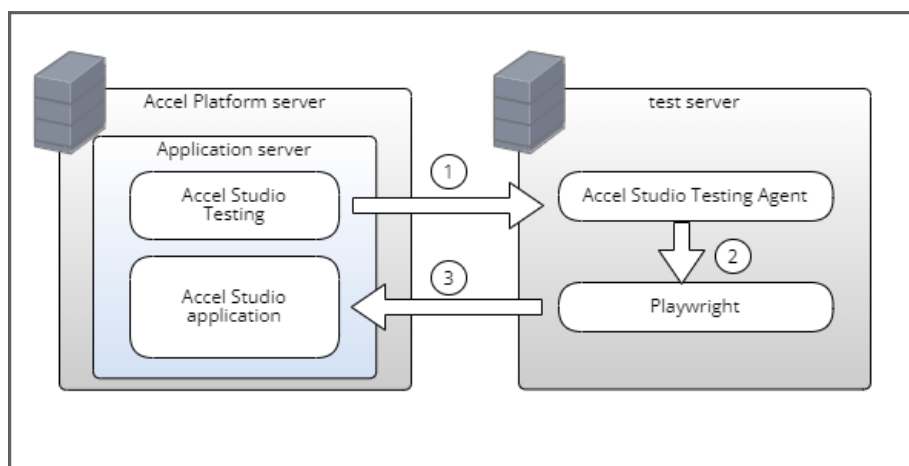
全体像

Accel Studio テスト機能 は、Accel Studio テスト機能 テスト実行エージェント を通してテストを実行します。

Accel Studio テスト機能 テスト実行エージェント は、ユーザ利用環境と同様のテスト実行環境で、実行することを想定しています。

intra-mart Accel Platform の実行環境が、Linuxなどユーザ利用環境と異なる場合に、テスト実行環境を分けることができます。

intra-mart Accel Platform の実行環境が、ユーザ利用環境と同様の場合、同一マシンに Accel Studio テスト機能 テスト実行エージェント をインストールすることも可能です。



(1) Accel Studio テスト機能 からテストを実行すると、Accel Studio テスト機能 テスト実行エージェント に実行命令を送ります。

intra-mart Accel Platform から Accel Studio テスト機能 テスト実行エージェント に通信ができる必要があります。

(2) Accel Studio テスト機能 テスト実行エージェント は、Playwrightを利用して、テストを実行します。

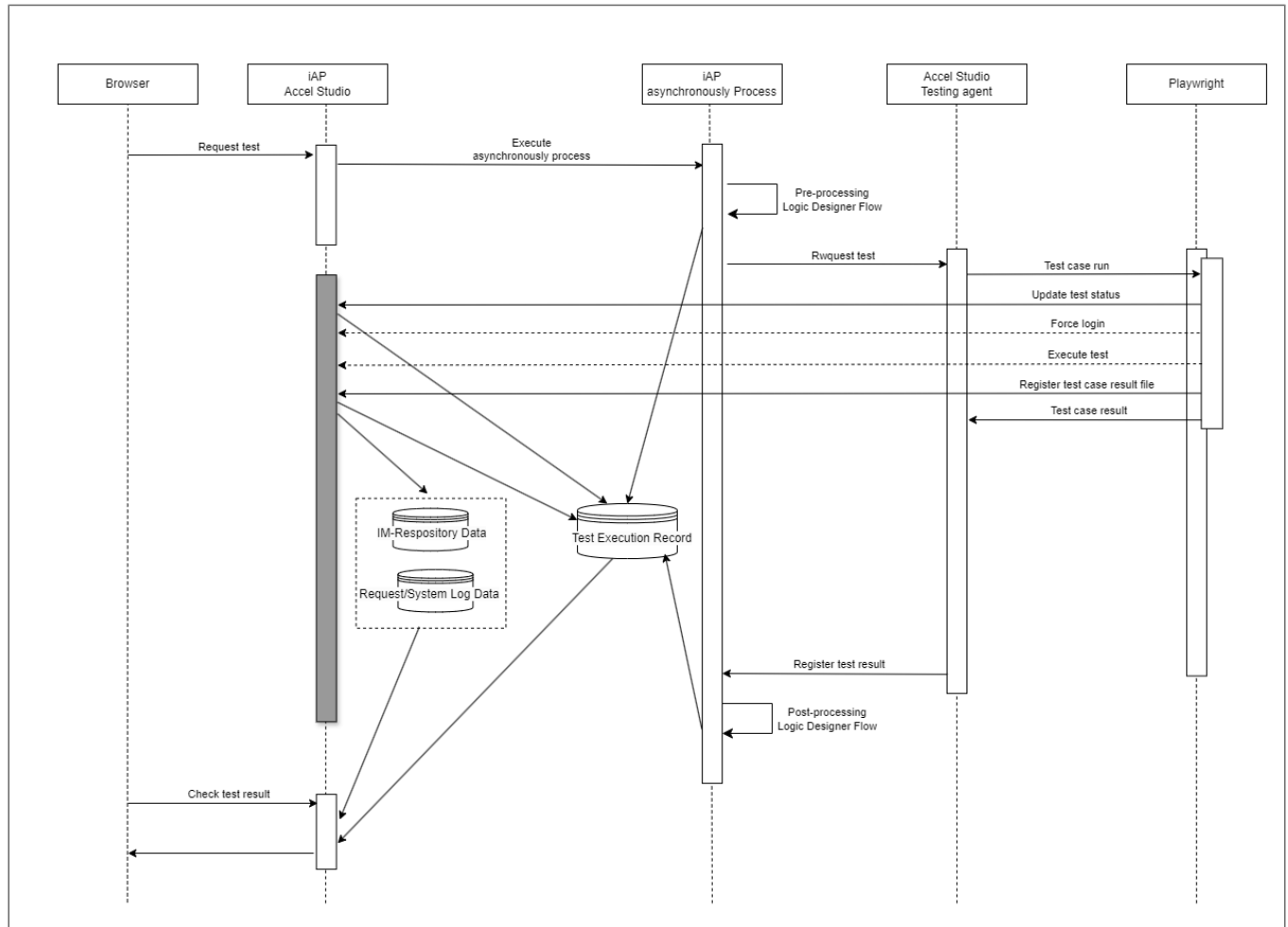
(3) Playwrightはヘッドレスブラウザを操作して、テストを実行します。

連携フロー

Accel Studio テスト機能 は、テスト実行時に、テスト実行履歴を管理します。

テスト実行履歴とテストシナリオの実行ユーザから、テスト実行によって出力された下記のデータをテスト結果として確認できます。

- システムログ管理機能
- リクエストログ管理機能
- エンティティ操作ログ
- テストシナリオ内で出力した画面キャプチャ



i コラム

テスト実行タスクは、intra-mart Accel Platform の非同期機能の直列タスクとして管理します。直列タスクのため、テスト実行をリクエストした順に、順次実行します。intra-mart Accel Platform の非同期機能については「[非同期仕様書](#)」を参照してください。

セットアップ

Accel Studio テスト機能 は、iAP とは別に、Accel Studio テスト機能 テスト実行エージェント を用意する必要があります。

項目

- エージェントの実行環境について
 - 必要なネットワーク環境
 - 利用されるブラウザ
 - エージェントの実行環境OS
- テスト機能API実行者へのロールの付与
- APIキーの取得
- 設定ファイルの変更
- 非同期タスクキューの設定 (2025 Spring(Kamille) からアップデートの場合)
- エージェントの実行
 - エージェントのダウンロード
 - オフライン環境の場合の事前準備
 - 使用ライブラリの事前構築
 - ブラウザデータの事前準備
 - エージェントの起動

エージェントの実行環境について

必要なネットワーク環境

Accel Studio テスト機能 テスト実行エージェント は、Playwrightを経由してアプリケーションサーバと通信します。また、アプリケーションサーバは Accel Studio テスト機能 テスト実行エージェント に対して、テスト実行などのリクエストを送ります。そのため、アプリケーションサーバとインストールされた端末で、双方向に通信が可能であることを確認してください。Accel Studio テスト機能 テスト実行エージェント は、アプリケーションサーバとの通信時にプロキシを経由せずに接続できる必要があります。

利用されるブラウザ

Accel Studio テスト機能 は、Playwright を利用してテストを実行します。テストの実行は Chromium で行われます。動作するブラウザバージョンは、Playwright のサポートポリシーに準拠します。

エージェントの実行環境OS

エージェントは、intra-mart Accel Platform の「[クライアント要件](#)」の種別PCでの利用をサポートしています。利用するクライアント環境に合わせたOS上でエージェントを起動してください。また、動作するOSは、Playwright のサポートポリシーに準拠します。使用する Playwright のバージョンがサポートするOSを確認してください。

Playwright 公式ドキュメント: <https://playwright.dev/docs/intro#system-requirements>



コラム

エージェントは、サーバ要件の環境でも動作を確認しています。テスト実行環境は、クライアント要件に合わせることを推奨しますが、iAP サーバと別環境が用意できない場合、サーバ環境への構築も可能です。サーバ要件は、「[システム要件](#)」を確認してください。

テスト機能API実行者へのロールの付与

Accel Studio テスト機能 テスト実行エージェント とアプリケーションの通信を行うユーザとして、テスト機能API実行者を設定します。設定されたユーザでAPIキーの発行を行い、テストの結果登録、ログインの解決などアプリケーションサーバとの通信を担当します。ロールには、テストを実行する上で必要となるテスト機能のREST APIに対する認可が設定してあります。

- 「Accel Studio テスト機能 API実行者」ロール

APIキーの取得

「Accel Studio テスト機能 API実行者」ロールを持つユーザで、OAuth APIキーを取得してください。
下記のスコープでOAuth APIキーを取得してください。
Playwright でのテスト実行時に、このOAuth APIキーを利用して通信します。
OAuth APIキーは、後続手順のエージェントの実行時に、引数として利用します。

- im-accel-studio-testing-execution
- im-workflow-rest

取得手順は、「[API キーを発行する](#)」を参照してください。

設定ファイルの変更

[Accel Studio テスト機能 の設定](#) の下記を変更してください。

Accel Studio テスト機能 は、開発環境や検証環境での利用を想定しています。

そのため、本番環境で Accel Studio テスト機能 を利用しない場合、testing-enabledをfalseとすることで、テスト実行を無効にできます。

バーチャルテナントによる複数テナント を利用する場合、agentはテナントの数だけ定義してください。

- testing-enabled
- agent base-url
- agent tenant-id

[Accel Studio テスト機能 の設定](#) の下記でテスト定義画面からリソースをアップロードする際のサイズ制限を設定できます。

環境に応じて設定を変更してください。

テスト実行中のキャプチャ取得、定義のインポート時等、上記操作以外の処理は制限の対象外です。

- resource-size-limit
- resource-total-size-limit

Accel Studio テスト機能 Copilot を利用する場合、IM-Copilot 生成AI連携ドライバ設定で使用するモデルなどの個別変更が可能です。

IM-Copilot 生成AI連携ドライバ設定については、[IM-Copilot 生成AI連携ドライバ設定](#) を参照してください。

非同期タスクキューの設定（2025 Spring(Kamille) からアップデートの場合）

バーチャルテナント環境でデフォルトテナント以外のテナントでもテスト機能を利用できるようにします。

<https://issue.intra-mart.jp/issues/38762>

上記の対応において、2025 Spring(Kamille) で作成した非同期タスクキュー（タスクキューID：`imbqasttestingtaskmanager`）が不要となりました。

そのため、「[2025 Spring \(Kamille\) からアップデート](#)」を参照して非同期タスクキューを削除してください。

エージェントの実行

エージェントのダウンロード

Accel Studio テスト機能 テスト実行エージェント は、「[プロダクトファイルダウンロード](#)」よりダウンロードしてください。
ダウンロードには、ライセンスキーが必要です。

オフライン環境の場合の事前準備

Accel Studio テスト機能 テスト実行エージェント の実行環境で外部へのネットワークアクセスが許可されない場合、エージェント起動前に使用ライブラリとブラウザデータの事前準備が必要です。

オンライン環境の場合はエージェント起動時に自動でセットアップされるため、この手順は不要です。

使用する Playwright のバージョンがサポートするバージョンの Node.js を選択してください。

最新バージョンの Node.js で過去バージョンの Playwright を使用した場合は動作しない可能性があります。

使用ライブラリの事前構築

1. 別途、オンライン（実際に Accel Studio テスト機能 テスト実行エージェント を稼働させる環境と同じ OS）かつ、Node.js が利用できる環境で、以下のようなコマンド等により `node_modules` ディレクトリを構築してください。
 - Accel Studio テスト機能 Copilot を利用しない場合
 - `npm install @playwright/test`
 - Accel Studio テスト機能 Copilot を利用する場合
 - `npm install @playwright/mcp @playwright/test`
2. オンライン環境で構築した `node_modules` ディレクトリをオフライン環境にコピーしてください。
例: `c:\accel_studio_testing_agent\node_modules`
3. [エージェントの起動](#) で説明しているオプションに対して、値を設定してください。
`imbq.ast.agent.prebuilt-node-modules-path` にコピーした `node_modules` のルートディレクトリへのパスを指定してください。
例: `-Dimbq.ast.agent.prebuilt-node-modules-path=c:\accel_studio_testing_agent\node_modules`

ブラウザデータの事前準備

以下について、原則として実際に Accel Studio テスト機能 テスト実行エージェント を稼働させる環境と同じ OS で実施する必要があります。

1. Node.js が利用できる環境で、以下のようなコマンド等により、`PLAYWRIGHT_BROWSERS_PATH` を指定してブラウザデータを任意のディレクトリにダウンロードしてください。
`--with-deps` による依存関係のインストールは原則 Accel Studio テスト機能 テスト実行エージェント を稼働させるマシン上で実施する必要があります。
ネットワークの制限がある場合は、必要に応じてライブラリを手動でコピーしてください。
Playwright のバージョンを指定する場合はここでもバージョンを指定してください。
 - `npx cross-env PLAYWRIGHT_BROWSERS_PATH="xxx" npx playwright install --with-deps`
例: `npx cross-env PLAYWRIGHT_BROWSERS_PATH="C:\Playwright" npx playwright install --with-deps`
2. [エージェントの起動](#) で説明しているオプションに対して、値を設定してください。
`imbq.ast.agent.playwright-browsers-path` にブラウザデータのルートディレクトリへのパスを指定してください。
例: `-Dimbq.ast.agent.playwright-browsers-path=c:\accel_studio_testing_agent\playwright-browsers`

エージェントの起動

Accel Studio テスト機能 テスト実行エージェント は、実行可能jar形式で同梱しています。
jar ファイルを配置した各端末にて、以下の手順を実施し、エージェントを起動してください。

1. Java JDK をインストールします。（JDKバージョン：intra-mart Accel Platform のシステム要件に準拠）
2. 「`accel_studio_testing_agent-8.x.x.jar`」を任意のフォルダに配置します。
3. jar ファイルを配置したフォルダにて、以下のコマンドを実行し、エージェントを起動します。
オンライン環境向け:

```
java -Dfile.encoding=UTF-8 -Dimbq.ast.agent.oauth-api-key="OAuth APIキー" -Dimbq.ast.agent.testing-target-base-url="iAP URL" -Dimbq.ast.agent.playwright-mcp-version="Playwright MCPバージョン" -jar accel_studio_testing_agent-8.x.x.jar
```

オフライン環境向け:

```
java -Dfile.encoding=UTF-8 -Dimbq.ast.agent.oauth-api-key="OAuth APIキー" -Dimbq.ast.agent.testing-target-base-url="iAP URL" -Dimbq.ast.agent.prebuilt-node-modules-path="node_modules/パス" -Dimbq.ast.agent.playwright-browsers-path="ブラウザデータ配置パス" -jar accel_studio_testing_agent-8.x.x.jar
```

4. エージェントが起動すると、リクエスト受付の待機状態が始まります。
アプリケーションサーバからのリクエストを受信すると、Accel Studio テスト機能 が自動的に起動され、シナリオが実行されます。

i コラム

エージェントの起動時に、以下の表のシステムプロパティをJavaの引数として `-Dxxx=xxx` 形式で指定可能です。
`-Dfile.encoding=UTF-8` は必ず指定してください。
 次のコマンドのように、引数は `-jar` よりも前に指定してください。

下記はオンライン環境向けのコマンド例です。

```
java -Dfile.encoding=UTF-8 -Dimbq.ast.agent.oauth-api-key=xxx -Dimbq.ast.agent.testing-target-base-url=http://127.0.0.1:8080/imart/ -Dimbq.ast.agent.playwright-mcp-version=0.0.68 -jar accel_studio_testing_agent-8.x.x.jar
```

下記はオフライン環境向けのコマンド例です。

```
java -Dfile.encoding=UTF-8 -Dimbq.ast.agent.oauth-api-key=xxx -Dimbq.ast.agent.testing-target-base-url=http://127.0.0.1:8080/imart/ -Dimbq.ast.agent.prebuilt-node-modules-path=C:/accel_studio_testing_agent/node_modules -Dimbq.ast.agent.playwright-browsers-path=C:/playwright -jar accel_studio_testing_agent-8.x.x.jar
```

プロパティ名	デフォルト値	必須	意味
imbq.ast.agent.oauth-api-key	なし	○	OAuth APIキー
imbq.ast.agent.testing-target-base-url	http://127.0.0.1:8080/imart/	○	iAP のベースURL
imbq.ast.agent.prebuilt-node-modules-path	なし	—	事前構築済み <code>node_modules</code> ディレクトリパス（未指定時は外部への通信が可能と判定し <code>npm install</code> を自動実行）
imbq.ast.agent.playwright-browsers-path	なし	—	Playwright で使用するブラウザデータ配置先ディレクトリパス（未指定時はブラウザデータを自動ダウンロード）
imbq.ast.agent.playwright-version	なし	—	使用する Playwright のバージョン（未指定時は最新バージョンを取得、 <code>imbq.ast.agent.prebuilt-node-modules-path</code> が指定されている場合は無視）
imbq.ast.agent.playwright-mcp-version	なし	—	使用する Playwright MCP のバージョン（未指定時は Playwright MCP のインストールをスキップ、 <code>imbq.ast.agent.prebuilt-node-modules-path</code> が指定されている場合は無視）
imbq.ast.agent.script-playwright-config-path	なし	—	Playwright 用設定ファイル
imbq.ast.agent.script-playwright-mcp-config-path	なし	—	Playwright MCP 用設定ファイル
imbq.ast.agent.ignore-playwright-report	なし	—	Playwright のカスタムレポートを出力しない 2025 Spring(Kamille) 用互換オプション
server.port	8188	—	エージェントのポート番号
logging.file.name	accel_studio_testing_agent.log	—	出力ログファイル名
logging.level.root	INFO	—	出力ログレベル
logging.level.jp.co.intra_mart	logging.level.rootの値	—	製品実装部分の出力ログレベル

※ `imbq.ast.agent.playwright-browsers-path` の詳細は [ブラウザデータの事前準備](#) を参照してください。
 ※ `imbq.ast.agent.prebuilt-node-modules-path` の詳細は [使用ライブラリの事前構築](#) を参照してください。

i コラム

バージョン解決の優先順位

Playwright 関連パッケージのバージョンは、以下の優先順位で解決されます。

1. `imbq.ast.agent.prebuilt-node-modules-path` が指定されている場合、指定した `node_modules` 内のパッケージが使用されます。
この場合、`imbq.ast.agent.playwright-version` および `imbq.ast.agent.playwright-mcp-version` の指定は無視されます。
2. `imbq.ast.agent.prebuilt-node-modules-path` が未指定の場合、`imbq.ast.agent.playwright-version` と `imbq.ast.agent.playwright-mcp-version` の指定に基づいてバージョンが決定されます。
`imbq.ast.agent.playwright-version` を指定せずに `imbq.ast.agent.playwright-mcp-version` を指定すると、`@playwright/mcp` の依存先である `playwright-core` が自動インストールされ、`npm playwright --version` でバージョンが検出されます。
`imbq.ast.agent.playwright-mcp-version` を指定しない場合は Playwright MCP がインストールされません。
3. `imbq.ast.agent.prebuilt-node-modules-path` ・ `imbq.ast.agent.playwright-mcp-version` ・ `imbq.ast.agent.playwright-version` のいずれも未指定の場合、Playwright の最新バージョンが取得され、Playwright MCP はインストールされません。

i コラム

Playwright 用設定ファイルについて

設定ファイルを配置することで、Playwright の動作を調整できます。

設定値は、Playwright の公式サイト（<https://playwright.dev/docs/test-configuration#expect-options> や <https://playwright.dev/docs/test-use-options>）を参考にしてください。

設定ファイルは、各種テストステップの動作調整を目的として利用してください。それ以外の目的による設定は推奨しません。

また、以下の設定項目はエージェントが内部で使用するため、変更できません。

- `outputDir`
- `snapshotPathTemplate`
- `globalSetup`
- `globalTeardown`
- `reporter`
- `use.baseURL`
- `use.storageState`

下記は、タイムアウト時間と Visual Regression Test の画像比較にて、異なる可能性のあるピクセルの許容量をデフォルト設定から変更、またテスト実行時の画面収録とスクリーンショットの取得を設定した例です。

```
import { PlaywrightTestConfig } from '@playwright/test';

const config: PlaywrightTestConfig = {
  name: 'custom-config',
  timeout: 50000,
  globalTimeout: 100000,
  use: {
    video: 'on', // テスト実行時の画面録画をテスト結果として取得するオプションです。
    screenshot: 'on', // テスト終了時やエラー発生時の画面キャプチャをテスト結果として取得するオプションです。
  },
  expect: {
    toMatchSnapshot: { maxDiffPixelRatio: 0.02, maxDiffPixels: 200 },
    toHaveScreenshot: { maxDiffPixelRatio: 0.02, maxDiffPixels: 200 },
  },
};

export default config;
```



コラム

Playwright MCP 用設定ファイルについて

設定ファイルを配置することで、Playwright MCP の動作を調整できます。

設定値は、Playwright MCP の公式サイト（<https://github.com/microsoft/playwright-mcp/blob/main/README.md#configuration-file>）を参考にしてください。

設定項目によっては正常に動作しなくなる可能性があるため、十分な検証の上ご利用ください。

設定ファイル

Accel Studio テスト機能 の設定

項目

- 概要
- リファレンス
 - テスト実行可否設定
 - エージェント設定
 - エージェント接続を行う際のタイムアウト値
 - リトライ最大回数設定
 - リトライ待ち秒数設定
 - 定義設定
 - 1ファイルあたりのリソースサイズ制限
 - 1定義に対するテストリソース全ファイルの最大サイズ制限

概要

Accel Studio テスト機能 に関する設定です。

モジュール	Accel Studio テスト機能
フォーマットファイル(xsd)	WEB-INF/schema/accel-studio-testing-config.xsd
設定場所	WEB-INF/conf/accel-studio-testing-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<accel-studio-testing-config xmlns="https://www.intra-mart.jp/accel-studio-testing-config"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="https://www.intra-mart.jp/accel-studio-testing-config accel-studio-testing-config.xsd"
  >

  <!--
    テスト機能を利用する場合は true に変更してください。
    If you want to use the testing function, change it to true.
    使用<>機能<>改<> true 。
  -->
  <testing-enabled>>false</testing-enabled>

  <agent base-url="http://localhost:8188/accel-studio-testing-agent" tenant-id="default">
    <timeout-seconds>1800</timeout-seconds>
    <retry>
      <max-count>3</max-count>
      <wait-seconds>10</wait-seconds>
    </retry>
  </agent>

  <definition>
    <resource-size-limit>1024</resource-size-limit>
    <resource-total-size-limit>102400</resource-total-size-limit>
  </definition>

</accel-studio-testing-config>
```

リファレンス

テスト実行可否設定

タグ名 testing-enabled

テスト実行を利用するかどうかを設定します。

false を設定した場合、テスト機能でテストを実行できません。

【設定項目】

```
<accel-studio-testing-config>
  <testing-enabled>>false</testing-enabled>
  :
</accel-studio-testing-config>
```

必須項目	○
複数設定	×
設定値・設定する内容	テスト実行可否
単位・型	なし
省略時のデフォルト値	false
親タグ	accel-studio-testing-config

エージェント設定

タグ名 agent

Accel Studio テスト機能 テスト実行エージェント に関する設定を定義します。

【設定項目】

```
<accel-studio-testing-config>
  :
  <agent base-url="http://localhost:8188/accel-studio-testing-agent" tenant-id="default">
  :
  </agent>
</accel-studio-testing-config>
```

必須項目	○
複数設定	×
設定値・設定する内容	Accel Studio テスト機能 テスト実行エージェント の接続設定
単位・型	なし
省略時のデフォルト値	なし
親タグ	accel-studio-testing-config

【属性】

属性名	説明	必須	デフォルト値
base-url	Accel Studio テスト機能 テスト実行エージェント のベース URL	○	なし
tenant-id	エージェント接続を行う対象のテナントID		デフォルトテナントID

エージェント接続を行う際のタイムアウト値

タグ名 timeout-seconds

エージェント接続を行う際のタイムアウト値を設定します。

0 を設定した場合、タイムアウトしません。

i コラム

2025 Spring および 2025 Autumn では、下記不具合によりタイムアウトが反映されません。

<https://issue.intra-mart.jp/issues/39958>

上記バージョンでエージェント接続における無限待機状態の発生が確認された場合は Accel Studio テスト機能 テスト実行 エージェント を停止してください。

【設定項目】

```
<accel-studio-testing-config>
:
<agent>
  <timeout-seconds>1800</timeout-seconds>
:
</agent>
</accel-studio-testing-config>
```

必須項目	○
複数設定	×
設定値・設定する内容	エージェント接続を行う際のタイムアウト値（単位：秒）
単位・型	整数値（0-）
省略時のデフォルト値	なし
親タグ	agent

リトライ最大回数設定

タグ名 max-count

リトライの最大回数を設定します。

0 を設定した場合、リトライしません。

【設定項目】

```
<accel-studio-testing-config>
:
<agent>
:
  <retry>
    <max-count>3</max-count>
  :
</retry>
</agent>
</accel-studio-testing-config>
```

必須項目	○
複数設定	×
設定値・設定する内容	リトライの最大回数
単位・型	整数値（0-）
省略時のデフォルト値	なし
親タグ	retry

リトライ待ち秒数設定

タグ名 wait-seconds

次にリトライするまでの待ち時間の秒数を設定します。

【設定項目】

```
<accel-studio-testing-config>
:
<agent>
:
  <retry>
  :
    <wait-seconds>10</wait-seconds>
  </retry>
</agent>
</accel-studio-testing-config>
```

必須項目	○
複数設定	×
設定値・設定する内容	リトライするまでの待ち時間（単位：秒）
単位・型	整数値（0-）
省略時のデフォルト値	なし
親タグ	retry

定義設定

タグ名 definition

テスト定義に関する設定を定義します。

【設定項目】

```
<accel-studio-testing-config>
:
<definition>
:
</definition>
</accel-studio-testing-config>
```

必須項目	○
複数設定	×
設定値・設定する内容	なし
単位・型	なし
省略時のデフォルト値	なし
親タグ	accel-studio-testing-config

1ファイルあたりのリソースサイズ制限

タグ名 resource-size-limit

1ファイルあたりのリソースサイズ制限を設定します。

【設定項目】

```
<accel-studio-testing-config>
:
<definition>
  <resource-size-limit>1024</resource-size-limit>
:
</definition>
</accel-studio-testing-config>
```

必須項目	○
複数設定	×
設定値・設定する内容	1ファイルあたりのリソースサイズ制限（単位：KB）
単位・型	整数値（0-）
省略時のデフォルト値	1024
親タグ	definition

1定義に対するテストリソース全ファイルの最大サイズ制限

タグ名 resource-total-size-limit

1定義に対するテストリソース全ファイルの最大サイズ制限を設定します。

【設定項目】

```
<accel-studio-testing-config>
:
<definition>
  <resource-total-size-limit>102400</resource-total-size-limit>
:
</definition>
</accel-studio-testing-config>
```

必須項目	○
複数設定	×
設定値・設定する内容	1定義に対するテストリソース全ファイルの最大サイズ制限（単位：KB）
単位・型	整数値（0-）
省略時のデフォルト値	102400
親タグ	definition

Accel Studio テスト機能 Copilot の設定

項目

- 概要
- リファレンス
 - 最大会話履歴数
 - テスト生成時の1回のリクエストあたりの最大生成数
 - 出力ランダム性

概要

Accel Studio テスト機能 Copilot に関する設定です。

Accel Studio テスト機能 Copilot を利用する際に、IM-Copilot 生成AI連携ドライバ設定で使用するモデルなどの個別変更が可能です。

IM-Copilot 生成AI連携ドライバ設定については、[IM-Copilot 生成AI連携ドライバ設定](#) を参照してください。

モジュール	Accel Studio テスト機能 Copilot
フォーマットファイル(xsd)	WEB-INF/schema/accel-studio-testing-copilot-config.xsd
設定場所	WEB-INF/conf/accel-studio-testing-copilot-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<accel-studio-testing-copilot-config xmlns="https://www.intra-mart.jp/accel-studio-testing-copilot-config"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="https://www.intra-mart.jp/accel-studio-testing-copilot-config accel-studio-testing-copilot-config.xsd"
>

<assistant>
  <max-conversation-history-count>100</max-conversation-history-count>
  <max-generation-count-per-request>450</max-generation-count-per-request>
  <temperature>0</temperature>
</assistant>

</accel-studio-testing-copilot-config>
```

リファレンス

最大会話履歴数

タグ名 max-conversation-history-count

生成AIとの会話の上限値を設定します。

【設定項目】

```
<assistant>
<max-conversation-history-count>100</max-conversation-history-count>
:
</assistant>
```

必須項目	×
複数設定	×
設定値・設定する内容	IM-Copilotとの会話履歴の最大保存数
単位・型	なし
省略時のデフォルト値	100
親タグ	assistant

テスト生成時の1回のリクエストあたりの最大生成数

タグ名 max-generation-count-per-request

テスト生成時の1回のリクエストあたりの最大生成数です。

【設定項目】

```
<assistant>
<max-generation-count-per-request>450</max-generation-count-per-request>
:
</assistant>
```

必須項目	×
複数設定	×
設定値・設定する内容	テスト生成時の1回のリクエストあたりの最大生成数
単位・型	なし
省略時のデフォルト値	450
親タグ	assistant

出力ランダム性

タグ名 temperature

出力ランダム性を制御するパラメータです。
値が高いほど創造的な出力を生成し、低いほど一貫性のある出力を生成します。

【設定項目】

```
<assistant>
<temperature>0</temperature>
:
</assistant>
```

必須項目	×
複数設定	×
設定値・設定する内容	生成AIの温度パラメータ（0以上の小数値）
単位・型	小数
省略時のデフォルト値	0
親タグ	assistant

機能仕様

テスト定義

テスト定義は、以下の情報を扱います。

- テスト定義ID
- テスト定義名
- 備考
- 前処理
- 後処理
- テストケース
- テストリソース

国際化

テスト定義の名称は、各ロケール用（製品標準では、日本語、英語、中国語）の表示名と標準表示名を持ちます。

原則として Accel Studio テスト機能 では、ユーザのロケールに合わせた表示名を利用します。

対象のユーザのロケールに合致する表示名が定義されていない場合、Accel Studio テスト機能 はその代替として標準表示名を利用します。

テスト定義カテゴリ

テスト定義カテゴリは、テスト定義の利用用途や適用範囲を分類するための情報です。

Accel Studio テスト機能 ではテスト定義の管理・分類するための情報としてテスト定義カテゴリを利用します。

カテゴリの階層化

テスト定義カテゴリは階層構造を設定できます。

テスト定義カテゴリに親カテゴリを設定することで、設定したカテゴリの子カテゴリとして設定され、階層構造を持ちます。

カテゴリの削除

テスト定義カテゴリの削除には、物理削除が利用されます。

またテスト定義カテゴリは、下記の条件の場合のみ削除可能です。

- 削除対象のカテゴリの属するテスト定義が存在しない場合
- 削除対象のカテゴリに子カテゴリが存在しない場合

前処理、後処理

テスト定義では、前処理、後処理として、ロジックフローを設定できます。

前処理

テスト実行前に処理されます。

入力

- `cases` (`object[]`) - テストケース情報の配列
 - `executionUserCd` (`string`) - 実行ユーザコード
 - `name` (`string`) - テストケース名
 - `sortNumber` (`string`) - ソート番号
 - `testingCaselId` (`string`) - テストケースID
- `categoryId` (`object[]`) - テスト定義が所属するカテゴリのID
- `name` (`string`) - テスト定義名
- `setupLogicFlowId` (`string`) - 前処理ロジックフローID
- `setupLogicFlowVersion` (`string`) - 前処理ロジックフローバージョン

- `teardownLogicFlowId` (string) - 後処理ロジックフローID
- `teardownLogicFlowVersion` (integer) - 後処理ロジックフローID
- `testingId` (string) - テスト定義ID

```
{
  "cases": [
    {
      "executionUserCd": "",
      "name": "",
      "sortNumber": 0,
      "testingCaselId": ""
    }
  ],
  "categoryId": "",
  "name": "",
  "setupLogicFlowId": "",
  "setupLogicFlowVersion": 0,
  "teardownLogicFlowId": "",
  "teardownLogicFlowVersion": 0,
  "testingId": ""
}
```

出力

なし

後処理

テスト実行後に処理されます。

入力

- `cases` (object[]) - テストケース情報の配列
 - `executionUserCd` (string) - 実行ユーザコード
 - `name` (string) - テストケース名
 - `sortNumber` (integer) - ソート番号
 - `testingCaselId` (string) - テストケースID
- `categoryId` (object[]) - テスト定義が所属するカテゴリのID
- `name` (string) - テスト定義名
- `setupLogicFlowId` (string) - 前処理ロジックフローID
- `setupLogicFlowVersion` (integer) - 前処理ロジックフローバージョン
- `teardownLogicFlowId` (string) - 後処理ロジックフローID
- `teardownLogicFlowVersion` (integer) - 後処理ロジックフローID
- `testingId` (string) - テスト定義ID
- `result` (object) - テスト結果情報
 - `cases` (object[]) - テストケース結果情報の配列
 - `executionStartDatetime` (date) - 実行開始日時
 - `executionTime` (integer) - 実行に要した時間 (ミリ秒)
 - `status` (string) - 結果ステータス
 - `testingCaseResultId` (string) - テストケース結果ID
 - `executionStartDatetime` (date) - 実行開始日時
 - `executionTime` (integer) - 実行に要した時間 (ミリ秒)
 - `status` (string) - 結果ステータス
 - `testingResultId` (string) - テスト結果ID

```

{
  "cases": [
    {
      "executionUserCd": "",
      "name": "",
      "sortNumber": 0,
      "testingCaselId": ""
    }
  ],
  "categoryId": "",
  "name": "",
  "setupLogicFlowId": "",
  "setupLogicFlowVersion": 0,
  "teardownLogicFlowId": "",
  "teardownLogicFlowVersion": 0,
  "testingId": "",
  "result": {
    "cases": [
      {
        "executionStartDatetime": null,
        "executionTime": 0,
        "status": "",
        "testingCaseResultId": ""
      }
    ],
    "executionStartDatetime": null,
    "executionTime": 0,
    "status": "",
    "testingResultId": ""
  }
}

```

出力

なし

テストケース

テストケースは、テスト定義に含まれるテストの単位です。テストケース毎に実行ユーザを指定できます。

コラム

テストケースの実行ユーザは、テスト実行時に指定されたユーザによってテストが実行されます。テストケース内で独自のログイン操作を行うこともできますが、テストケースの実行ユーザは必須で登録しなければなりません。

テストリソース

テストリソースは、テストケースのテストコード上で利用できるファイルやデータのことです。画面キャプチャやテスト用ファイルをテストリソースとして登録し、各テストケースで利用できます。

コラム

テストケース内に `toMatchSnapshot` や `toHaveScreenshot` によるアサーションがあり、「[テスト実行](#)」で「スナップショット更新実行」を実施した場合は正解ファイルがテストリソースに登録されます。

テストの定期実行について

Accel Studio テスト機能 では、テスト定義に対して定期実行を設定できます。

詳しくはジョブネットの設定を参照してください。

ジョブ機能の詳細は「[ジョブ・ジョブネットリファレンス](#)」-「[テスト実行ジョブ](#)」を参照してください。

テスト実行

テスト実行には2種類あります。

- **通常実行**
テストの実行を行います。
- **スナップショット更新実行**
スナップショットの更新では、実行時に取得されたスクリーンショットをテストリソースとして登録、更新します。
内部で下記の処理が実行されています。
[Visual comparisons - Updating screenshots](#)

実行ユーザ

テストコードで指定された操作を実行するユーザです。クライアントから実際に操作する場合と同様に、指定されたユーザで操作を行います。

テスト実行時に指定されたユーザでログインを行い、取得したCookieを利用して画面にアクセスします。

この関係上、テストコード内で指定する画面遷移等におけるURLのベースURL部分は、エージェント起動時に指定したベースURLと同一である必要があります。

なお、エージェント起動時に指定したベースURLは、Playwright のオプション ([TestOptions - baseURL](#)) として使用されるため、通常は画面遷移時に使用する `goto` メソッドなどでベースURL部分を指定する必要がありません。

指定されたユーザが存在しない場合、テスト実行は失敗します。

テストコード

テストコードは Playwright の仕様に従っています。使用可能なモジュールは下記の通りです。

- **@playwright/test**
- **playwright**
- **playwright-core**
- **im-workflow-util**

その他、Node.js の標準組み込みモジュールが使用できますが、実行環境のデータを不正に操作できないよう、以下のモジュールは制限されています。

- **async_hooks**
- **child_process**
- **cluster**
- **crypto**
- **dgram**
- **diagnostics_channel**
- **dns**
- **domain**
- **events**
- **fs**
- **http**
- **https**
- **inspector**
- **module**
- **net**
- **os**
- **perf_hooks**
- **process**
- **readline**
- **repl**
- **stream**
- **timers**
- **timers/promises**
- **tls**
- **util**
- **v8**

- **vm**
- **worker_threads**
- **zlib**

テストリソースの利用

テストリソースとしてテスト定義にアップロードされた資材は、テストコード上で利用できます。
下記のようにファイル名を指定して利用します。

```
import { test, expect } from '@playwright/test';

test('test', async ({ page }) => {
  // テストリソース sample1.png を利用
  await expect(page).toHaveScreenshot('sample1.png');
});
```

IM-Workflow API の利用

テスト機能のテストコード上から利用できる IM-Workflow API を提供しています。
下記のコードの記述を行うことでテストコード上でAPIが利用できます。
内部で IM-Workflow REST API を呼び出しています。

```
import * as workflowUtil from 'im-workflow-util';
// 未完了案件の参照情報リストを取得例
workflowUtil.getActiveMattersReference(condition);
```

なお、APIを利用するにはテスト機能API実行ユーザに発行したOAuth APIキーでアクセス範囲に IM-Workflow REST API へのアクセス(スコープ `im-workflow-rest`)が許可されている必要があります。

getActiveMattersReference

未完了案件の参照情報リストを取得します。
内部仕様の詳細は、「[未完了案件参照情報リスト取得](#)」を参照してください。

引数

- **condition** - 検索条件

```

{
  "advancedSearchCondition": [
    {
      "columnId": "string",
      "matterProperty": true,
      "value1": "string",
      "value2": "string"
    }
  ],
  "count": 0,
  "index": 0,
  "orders": [
    {
      "asc": true,
      "columnId": "string",
      "matterProperty": true
    }
  ],
  "authCondition": {
    "scope": "string"
  },
  "detailSearchCondition": {
    "applyAuthUserCd": "string",
    "applyBaseDateFrom": "string",
    "applyBaseDateTo": "string",
    "applyDateFrom": "string",
    "applyDateTo": "string",
    "matterName": "string",
    "matterNumber": "string",
    "priority": "string"
  },
  "flowGroupId": "string",
  "flowId": "string",
  "ignoreDisplayPattern": true
}

```

戻り値

- **RestAPIResult<ActiveMattersReferenceResponse>** - 案件情報

```

{
  "data": {
    "records": [
      {
        "applyAuthUserName": "string",
        "applyBaseDate": "string",
        "applyDate": "string",
        "defaultFormatApplyBaseDate": "string",
        "defaultFormatApplyDate": "string",
        "displayCopyNewLink": true,
        "displayDeleteLink": true,
        "displayDetailLink": true,
        "displayFlowLink": true,
        "displayHandleLink": true,
        "displayHistoryLink": true,
        "flowName": "string",
        "handleLevel": "string",
        "matterName": "string",
        "matterNumber": "string",
        "matterProperty": {},
        "priority": "string",
        "systemMatterId": "string",
        "userDatId": "string"
      }
    ]
  }
}

```

getSystemMatterIdFromActive

未完了案件の参照情報リストから、申請者ユーザコードの最新のシステム案件IDを取得します。未完了案件が存在しない場合はnullを返します。

retryOptionを指定した場合、システム案件IDが取得できないときにリトライします。

引数

- **flowId** (*string*) - フローID
- **applyAuthUserCd** (*string*) - 申請者ユーザコード
- **retryOption** (*RetryOption*) - リトライオプション（任意）
 - **retryCount** (*number*) - リトライ回数
 - **retryInterval** (*number*) - リトライ間隔（ミリ秒）

戻り値

- **string | null** - システム案件ID

getMattersInfo

案件情報を取得します。案件の状態（未完了、完了、過去）に関わらず取得できます。後続ノードが多い場合には、案件情報が更新されていない可能性があるため、遅延時間を設定してください。

内部仕様の詳細は、「[完了案件参照情報リスト取得](#)」を参照してください。

引数

- **systemMatterId** (*string*) - システム案件ID
- **delayTime** (*number*) - 遅延時間（ミリ秒）

戻り値

- **RestAPIResult<MattersInfoResponse>** - 案件情報

```
{
  "data": {
    "applyAuthUserCd": "string",
    "applyAuthUserName": "string",
    "applyBaseDate": "string",
    "applyDate": "string",
    "applyExecuteUserCd": "string",
    "applyExecuteUserName": "string",
    "archiveYearMonth": "string",
    "defaultFormatApplyBaseDate": "string",
    "defaultFormatApplyDate": "string",
    "defaultFormatMatterCplDate": "string",
    "defaultFormatMatterLastProcessDate": "string",
    "defaultFormatMatterStartDate": "string",
    "flowId": "string",
    "flowName": "string",
    "flowVersionId": "string",
    "matterCplDate": "string",
    "matterEndStatus": "string",
    "matterLastProcessDate": "string",
    "matterName": "string",
    "matterNumber": "string",
    "matterStartDate": "string",
    "matterStatus": "string",
    "priorityLevel": "string",
    "systemMatterId": "string"
  }
}
```

getStatusInfo

案件情報取得から指定したシステム案件IDのステータス情報を取得します。案件処理直後は案件情報が更新されていない可能性があるた

め、遅延時間を設定してください。

引数

- **systemMatterId** (*string*) - システム案件ID
- **delayTime** (*number*) - 遅延時間 (ミリ秒)

戻り値

- **string | null** - ステータス情報 (「0」: 未完了、「1」: 完了、「2」: 過去完了)

getConfirmAuthUsers

確認対象者を取得します。案件の状態 (未完了、完了) に関わらず取得できます。過去案件に対しては実行できません。内部仕様の詳細は、「[確認対象者取得](#)」を参照してください。

引数

- **systemMatterId** (*string*) - システム案件ID

戻り値

```
{
  "data": {
    "confirmAuthUsers": [
      {
        "authUserCd": "string",
        "authUserName": "string",
        "confirmed": true
      }
    ]
  }
}
```

isUserConfirmed

指定したユーザの確認状態を返します。

引数

- **systemMatterId** (*string*) - システム案件ID
- **confirmAuthUserCd** (*string*) - 確認対象者ユーザコード
- **delayTime** (*number*) - 遅延時間 (ミリ秒)

戻り値

- **boolean | null** - 確認状態 (確認済み: true、未確認: false、確認対象者ではない: null)

getCompletedMattersReference

完了案件の参照情報リストを取得します。内部仕様の詳細は、「[完了案件参照情報リスト取得](#)」を参照してください。

引数

- **condition** (*CompletedMattersReferenceSearchCondition*) - 検索条件

戻り値

- **RestAPIResult<MattersInfoResponse>** - 完了案件の参照情報リスト

```

{
  "advancedSearchCondition": [
    {
      "columnId": "string",
      "matterProperty": true,
      "value1": "string",
      "value2": "string"
    }
  ],
  "count": 0,
  "index": 0,
  "orders": [
    {
      "asc": true,
      "columnId": "string",
      "matterProperty": true
    }
  ],
  "authCondition": {
    "scope": "string"
  },
  "detailSearchCondition": {
    "applyAuthUserCd": "string",
    "applyBaseDateFrom": "string",
    "applyBaseDateTo": "string",
    "applyDateFrom": "string",
    "applyDateTo": "string",
    "matterName": "string",
    "matterNumber": "string",
    "priority": "string"
  },
  "flowGroupId": "string",
  "flowId": "string",
  "ignoreDisplayPattern": true
}

```

getSystemMatterIdFromCompleted

完了案件の参照情報リストから、申請者ユーザコードの最新のシステム案件IDを取得します。
 retryOptionを指定した場合、システム案件IDが取得できないときにリトライします。

引数

- **flowId** (*string*) - フローID
- **applyAuthUserCd** (*string*) - 申請者ユーザコード
- **retryOption** (*RetryOption*) - リトライオプション（任意）
 - **retryCount** (*number*) - リトライ回数
 - **retryInterval** (*number*) - リトライ間隔（ミリ秒）

戻り値

- **string | null** - システム案件ID

テスト結果

テスト結果は、以下の情報を扱います。

- テスト結果ステータス
 - テストの実行結果を示します。
- 開始日時
 - テストの実行開始日時です。
- 終了日時
 - テストの実行終了日時です。

- 処理時間
 - テストの実行時間です。
 - テストが実行開始された日時から、終了された日時までの時間を示します。
- テストケース結果
 - テストケース毎の実行結果です。
 - 実行時に出力されたログ情報を確認できます。

テスト結果ステータス

- 未実行
 - テストが実行待ち、または実行されていない状態です。
- 実行中
 - テストが実行中の状態です。
- 実行成功
 - テストが成功した状態です。
- アサーションエラー
 - テストがテストコードでのアサーションで失敗した場合の状態です。
- 前処理エラー
 - テストが前処理で失敗した場合の状態です。
- 後処理エラー
 - テストが後処理で失敗した場合の状態です。
- テスト実行エージェント接続エラー
 - テスト実行エージェントとの接続に失敗した場合の状態です。

テストケース結果

テストケース結果は、テストケース毎の詳細なテスト実行結果です。テストケース実行時に発生したログやエラー情報を確認できます。

実行時ログ

Playwrightでテストコード実行中に出力されたログが表示されます。テストコード内部でアサーションに失敗した場合のエラー箇所の情報も含まれます。

実行時エラーログ

テスト実行中に発生したシステムエラー情報が表示されます。

実行時タイムライン

下記ログが出力されます。

- リクエストログ
- システムログ
- エンティティ操作ログ
- 画像/動画/ファイル
- テストステップ

各ログの詳細についてはそれぞれのドキュメントを確認ください。



コラム

ログの出力について

上記のログは、テスト実行中に指定されたユーザによって出力されたログが表示されています。

そのため、同一ユーザでテスト実行中に別の操作を行った場合、テスト結果のログとしてテストに関係ないログが出力される可能性があります。

テスト結果の定期削除について

Accel Studio テスト機能 では、テスト結果の定期削除を設定できます。

ジョブ機能の詳細は「[ジョブ・ジョブネットリファレンス](#)」-「[テスト結果削除ジョブ](#)」を参照してください。

テスト生成

テスト生成機能は、テスト定義の一括作成を支援する機能です。

テスト生成について

テスト生成機能は、生成AIを活用してテスト定義の一括作成を支援する機能です。

例えば、商品マスタ登録画面のURLを指定すると、商品マスタ画面のフォーム入力や登録ボタン押下などの操作を含むテスト定義を生成できます。

テスト定義を作成する手間を削減し、効率的にテストを作成できます。

Playwright MCPを利用して実際の画面構造を取得することで、より正確なテスト定義の生成が可能です。

Accel Studio テスト機能 Copilot は、以下の機能を提供します。

- テスト定義一括生成
- テスト生成一覧の参照
- テスト生成詳細の参照



注意

本機能は、生成AIによってテスト定義の作成を支援するものです。

生成される内容については正確性・完全性・目的適合性を保証するものではなく、意図しない応答や目的外の情報が含まれる場合があります。

作成されるテスト定義は、基本的にそのままテスト実行できるものではなく、状況に合わせて修正する必要があります。

テスト対象の画面の規模やデータが大きい場合、使用する生成AIモデルの制限により、正常にテスト定義を作成できないことがあります。

生成物の著作権はユーザに帰属します。

利用者は、生成された応答や成果物を参考情報として取り扱い、社内利用・外部公開のいずれにおいても、内容の妥当性や安全性を確認したうえで、自己の責任で利用してください。

外部公開や配布を行う場合は、社内で適切なレビューを行い、第三者著作物やライセンス条件を遵守してください。

当社は、生成物の利用や外部公開による結果・損害について一切責任を負いません。



注意

テスト生成時には、テストコードの品質を高めるために対象の画面操作やワークフローを複数回実行します。このため、生成終了後に検証データが残る可能性があることに注意してください。



コラム

Accel Studio テスト機能 Copilot は 2025 Autumn(Lilac) から利用可能です。

Accel Studio テスト機能 Copilot を利用するためには IM-Copilot および Accel Studio テスト機能 テスト実行エージェントが必要です。

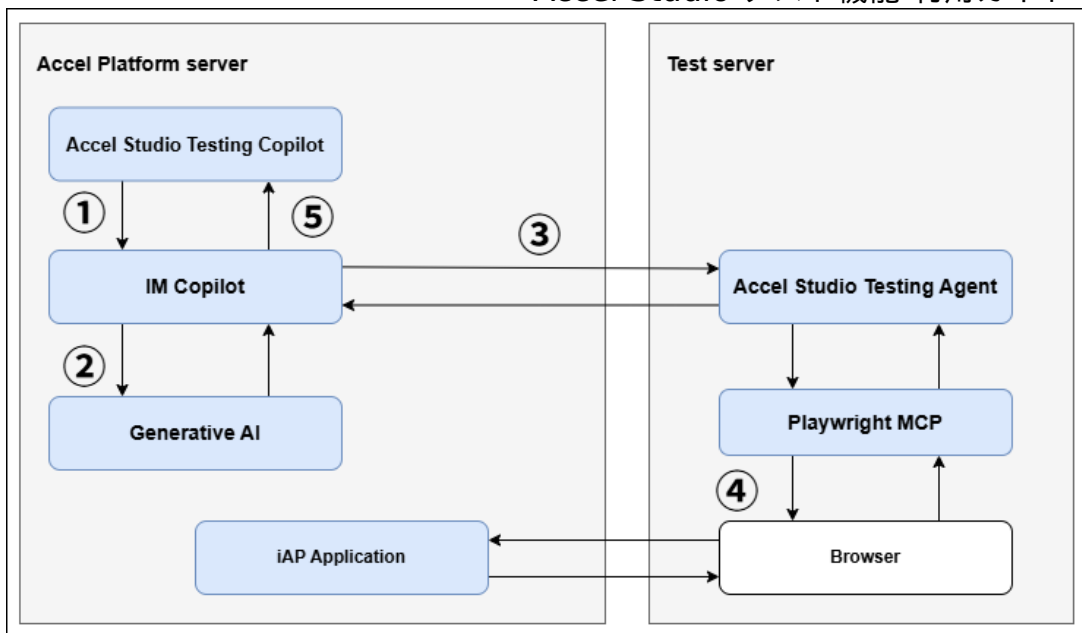
IM-Copilot については「[IM-Copilot 利用ガイド](#)」を参照してください。

全体像

Accel Studio テスト機能 Copilot は IM-Copilot と連携してテスト生成を行います。

IM-Copilot は Accel Studio テスト機能 テスト実行エージェント を通じて、Playwright MCP を利用して画面構造を取得します。

取得した画面構造を解析してテスト定義を生成します。



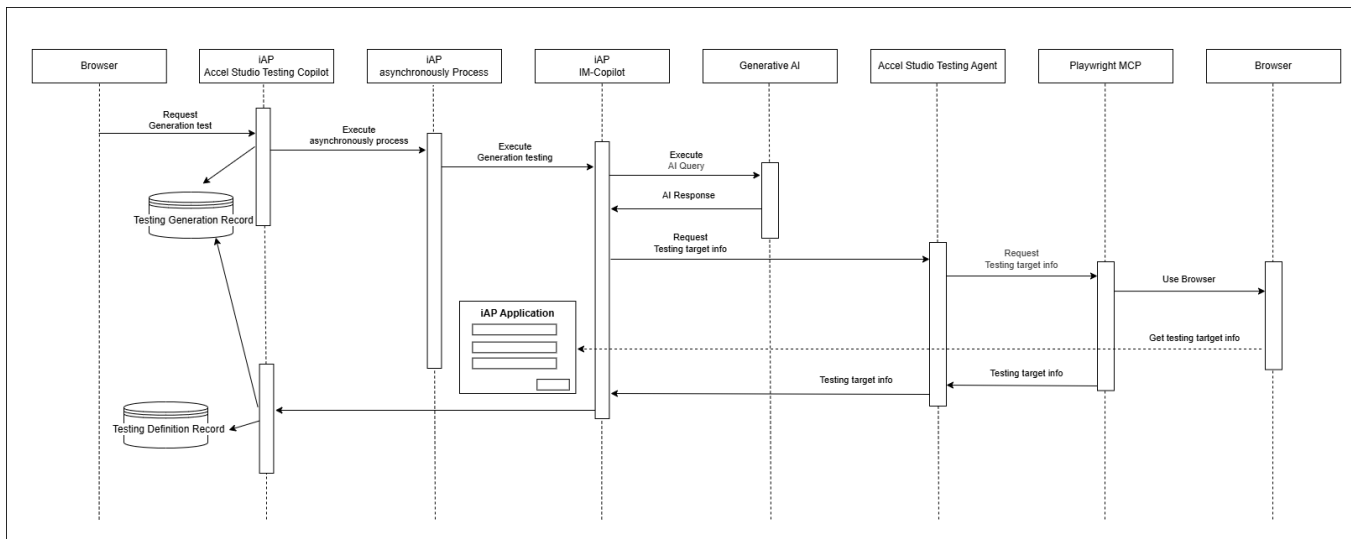
- (1) Accel Studio テスト機能 Copilot でテスト定義一括生成を実行すると、IM-Copilot へ生成依頼をします。
- (2) IM-Copilot は、生成AIへテスト生成の指示を行います。
- (3) IM-Copilot は、Accel Studio テスト機能 テスト実行エージェント を通して、Playwright MCPを利用して画面構造を取得します。
- (4) Playwright MCP は、ブラウザを操作して、対象のアプリケーション画面へ遷移し、画面構造を取得します。
- (5) 取得した画面構造を IM-Copilot が解析して、テスト定義を生成します。

i コラム

Playwright MCPは、Microsoft社が開発したPlaywrightをMCP (Model Context Protocol) 対応して拡張したツールです。ブラウザの自動操作、画面構造の取得、および要素の操作を可能にします。テスト生成においては、Webアプリケーションの動作を解析するために使用されます。Microsoft playwright-mcp GitHub (<https://github.com/microsoft/playwright-mcp>)

連携フロー

下記の図はブラウザからテスト生成を実行し、テストの分析をして、テスト生成情報を登録するまでの一連の流れを示します。テストの生成は下記の一連の流れを繰り返し実行します。



テスト生成の流れ

テスト生成は段階を踏んで実行されます。各フェーズの実行時のログはテスト生成詳細の実行ログから参照できます。下記の5つの手順で実行されます。

- (1) 分析

テスト対象の情報を取得し、横断的に分析を行います。

対象のURLにアクセスして画面構造を取得し、主要な機能や操作フローを特定します。

(2) 計画

分析結果をもとに、テストシナリオを計画します。

どのようなテストケースを生成するか、どの順序で実行するか、どのようなデータを使用するかなどを決定します。

(3) 設計

計画に基づいて、シナリオに対して具体的なテストケースを設計します。

各テストケースの目的、前提条件、入力データ、期待結果などを詳細に定義します。

(4) 実装

設計に基づいたテストコードを実装します。

(5) デプロイ

実装したテストコードを Accel Studio のテスト定義として登録します。

コラム

テスト設計までにテスト実行ユーザが削除された場合、Playwright MCP の画面情報取得時に失敗します。

テスト定義一括生成

テストの対象となるURLを指定して、テスト定義を一括生成します。

- 登録先アプリケーション
 - テスト定義を登録する Accel Studio アプリケーションを指定します。
- 一括生成情報
 - テスト生成名
 - テスト生成名を指定します。
 - 対象種別
 - テスト対象の種別を指定します。
 - URL
 - テスト対象のURLを直接指定します。
 - ルーティング定義
 - IM-BloomMaker のルーティング定義を指定します。
 - 指定できるルーティング定義は **GET**メソッド の定義のみです。
 - POSTメソッドや動的パラメータを含むルーティング定義は指定できません。
 - ワークフロー定義
 - IM-Workflow のワークフロー定義を指定します。
 - ワークフロー定義を選択すると、ルート選択・テストケース設定の画面に遷移します。
 - 詳細は「[ワークフローテスト生成](#)」を参照してください。
 - 登録先カテゴリ
 - テスト定義を登録するカテゴリを指定します。
 - 指定したカテゴリの配下にテスト定義が登録されます。
 - テスト定義の生成中に指定したカテゴリが削除された場合、デフォルトカテゴリ（カテゴリID：`imbq_testing_copilot_category`）にテスト定義を登録します。
 - テスト実行ユーザ
 - テストケースの実行ユーザを指定します。

テスト生成一覧

作成したテスト生成情報を一覧で参照できます。

テスト生成詳細

テスト生成時情報や実行ログを参照できます。

基本情報

テスト生成時の情報を参照できます。

登録先アプリケーション

- テスト生成時に指定した登録先アプリケーションです。

テスト生成名

- テスト生成名です。

対象種別

- テスト対象の種別です。
 - URL: テスト生成対象にURLを指定
 - IM-BloomMaker ルーティング定義: テスト生成対象に IM-BloomMaker のルーティング定義を指定
 - ワークフロー定義: テスト生成対象に IM-Workflow のワークフロー定義とルートを指定

テスト対象の識別ID/URL

- URL指定時 * テスト生成時に指定したURLが表示されます。
- ルーティングID指定時 * テスト生成時に指定した IM-BloomMaker のルーティング定義IDが表示されます。
- ワークフロー定義指定時 * テスト生成時に指定したワークフロー定義のフローIDが表示されます。

登録先カテゴリ

- テスト生成時に定義を登録したカテゴリです。

テスト実行ユーザ

- テスト生成時にテストケースの実行ユーザに設定したユーザです。

生成ステータス

- テスト生成のステータスを表示します。
 - 生成中: テスト生成が進行中であることを示します。
 - 生成完了: テスト生成が正常に完了したことを示します。
 - 生成失敗: テスト生成が失敗したことを示します。

生成ログ

テスト生成時の実行ログを参照できます。

Accel Studio テスト機能 テスト実行エージェント が実行したツールのログです。

生成ログを参照することで、テスト定義を生成する過程で実行した操作内容とその結果を確認できます。

カテゴリ

テスト生成時に実行したツールのカテゴリを表示します。

ツールを利用して、画面構造の取得や解析、テスト定義の生成を行います。

カテゴリは以下のとおりです。

- Playwright MCP
 - Playwright MCPの操作で、画面構造の取得、ブラウザ操作、ページ遷移などです。
- テスト機能ツール
 - テスト対象の分析やテスト定義の作成などの操作です。
- 会話
 - 生成AIとの会話履歴です。

操作内容

ツールの操作内容を表示します。

操作内容は、カテゴリごとに異なります。

操作内容は以下のとおりです。

- Playwright MCP
 - 主にブラウザ操作の内容が表示されます。（例: ページ遷移、クリック、フォーム入力など）
- テスト機能ツール
 - テスト対象の分析やテスト定義の生成に関する内容が表示されます。（例: 画面構造の解析、テスト定義の生成など）
- メッセージ
 - 生成AIのメッセージが表示されます。

ステータス

ツールの実行ステータスを表示します。

ステータスは以下のとおりです。

- 成功
 - ツールが正常に完了したことを示します。
- 失敗
 - ツールの実行に失敗したことを示します。
 - 例えば、期待する画面要素が見つからない、無効な操作を試みたなど、生成AIが誤った操作を行った場合に発生します。
- システムエラー
 - ツールの実行中に予期しない例外が発生したことを示します。

実行開始日時

ツールの実行開始日時を表示します。

実行時間

ツールの実行にかかった時間を表示します。

ワークフローテスト生成

ワークフローテスト生成は、IM-Workflow のワークフロー定義からテスト定義を自動生成する機能です。

ワークフローのルートに応じたテスト定義を生成し、テスト設計工数を削減できます。

生成されたテストケースは Playwright のテストコードとして Accel Studio テスト機能 のテスト定義に登録されます。

テンプレートに合わせてユーザコンテンツ画面の入力欄に対する入力用コードと、処理モジュール内の項目に対する設定/入力用のコードが生成されます。

生成中にサーバエラー等が発生した場合は、テンプレートの部分のみでテストコードが生成されます。

入力フォームへの入力が不要な承認画面では最低限動作するテストコードは生成されますが、入力が必要な画面では修正が必要な場合があります。



コラム

Accel Studio で作成されたアプリケーションを対象とするため、Accel Studio のリソースとして管理されているフロー定義がテスト対象です。

また、作成されるテストは処理モジュールを前提として調整されています。

ワークフローテスト生成の流れ

ワークフローテスト生成は、以下のステップで実行されます。

(1) ワークフロー定義の選択

IM-Workflow に登録されているワークフロー定義の一覧から、テスト対象のワークフローを選択します。

システム基準日で申請可能なフローが表示されます。

(2) ルート選択

選択したワークフロー定義のフロー図上でノードをクリックして、テスト対象のルート経路を構築します。

開始ノードから終了ノードまでの経路を選択します。分岐がある場合は、分岐先の候補が提示されます。

「テストケースに反映」ボタンで、設定したルートの各ノードに対応するテストケースが自動生成されます。

(3) テストケースの設定

テストケース名、テスト実行ユーザを設定します。

テストケースを移動や削除させることができますが、通常のケースではいずれも必要ありません。必要な場面は以下のとおりです。* 移動: 複数選択可の分岐や同期ノードでルートが分かれていても、実質順序性が求められる場合 * 削除: システムノードなど、任意のノードで人が介入することなく自動処理されることが前提となっている場合

(4) テスト定義の生成

ルート選択・テストケース設定の完了後、「テスト定義一括生成」ボタンをクリックして、テスト定義を生成します。

Playwright MCPを利用して申請画面や承認画面の画面構造を取得し、テストコードを生成します。

生成されるテストケースの内容

一括生成画面でワークフローテスト生成情報を追加後、生成を開始すると、各ノードの処理種別に応じた Playwright テストコードが、テストケース単位で自動生成されます。

Playwright テストコードは、ページ遷移やボタン操作などの基本操作を定義するテンプレートと、IM-Copilot が生成する画面固有のフォーム入力コードを組み合わせて構成されます。

生成されるテストコードのテンプレートは、処理種別ごとに以下の操作を含みます。

- 申請ノード、承認ノード、動的承認ノード、横配置ノード、縦配置ノード

- 申請・処理対象のコンテンツ画面への遷移→必要情報の入力→申請・処理モジュールの表示→必要情報の入力→申請・処理実行
- **確認ノード**
 - 確認画面へ遷移→確認実行→確認状態の検証



コラム

利用者処理ノードがテストケースの生成対象となり、開始・終了・システム・同期・分岐などの制御ノードは対象外です。テンプレート置換ノードはルート設定時点で展開され、展開後のノードに応じたテストコードが生成できます。縦配置、横配置ノードは、テストケース設定では1つのノードとして扱います。必要に応じて、生成後にテストケースを修正・複製してください。



コラム

申請・処理系ノードのテストコードには、生成した画面固有のフォーム入力コードが含まれます。フォーム入力コードの生成に失敗した場合は、テンプレートのみでテストコードを生成します。

インポート/エクスポート

Accel Studio テスト機能 のインポート/エクスポート機能について説明します。



コラム

Accel Studio テスト機能 のインポート/エクスポート機能 は 2025 Autumn(Lilac) から利用可能です。

インポート/エクスポートで扱う情報

インポート/エクスポートでは以下の情報を扱います。

- **テスト定義**
 - テスト定義ID
 - テスト定義名
 - 説明
 - カテゴリID
 - 前処理フローID
 - 前処理フローバージョン
 - 後処理フローID
 - 後処理フローバージョン
 - テスト実行ジョブ対象フラグ
- **テスト定義カテゴリ**
 - カテゴリID
 - 親カテゴリID
 - カテゴリ名
 - ソート番号
 - 説明
- **テストケース**
 - テストケースID
 - テスト定義ID
 - テストケース名
 - ソート番号
 - 説明
 - 実行ユーザ
 - テストコード
- **テストリソース**
 - テストリソースID
 - テスト定義ID
 - リソースデータ

- ファイル名
- ファイルサイズ

インポート/エクスポート時の動作

Accel Studio テスト機能 のインポート/エクスポート機能の動作仕様は以下のとおりです。

エクスポート

エクスポート機能は、「[インポート/エクスポートで扱う情報](#)」に記載した内容を出力します。
テスト定義カテゴリは、テスト定義が分類されているカテゴリ、およびその親カテゴリすべてが出力対象です。



コラム

前処理および後処理に設定されている IM-LogicDesigner のフロー定義はインポート/エクスポートの対象外です。
IM-LogicDesigner のエクスポート/インポート機能を利用して、フロー定義のエクスポート/インポートを実施してください。

インポート

インポート機能は、エクスポート機能により出力されたzipファイルを元にデータを取り込みます。
インポート途中で失敗した場合には、失敗したテスト定義のインポートはスキップし、成功するテスト定義をインポートします。

チュートリアル

チュートリアルの概要

本章では、Accel Studio テスト機能 を使用して、ローコードツールで作成したアプリケーションをテストする方法を説明します。このチュートリアルで、以下を実現できます。

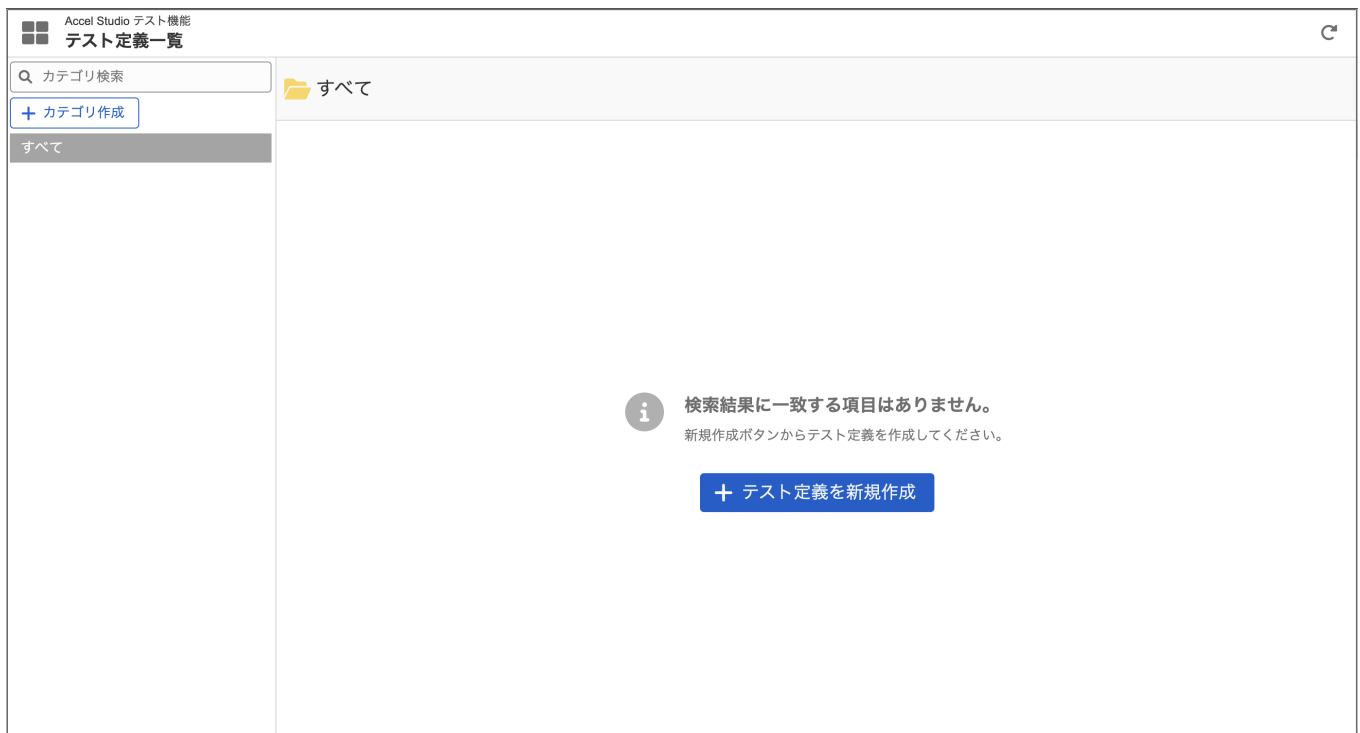
- Accel Studio で作成した画面のテストケースを作成
- 作成したテストケースを実行し、テスト結果を確認
- Accel Studio テスト機能 Copilot を利用して、テスト定義を一括生成

準備・環境設定

「[セットアップ](#)」を参照して、Accel Studio テスト機能 テスト実行エージェント を起動してください。

テスト定義の作成

サイトマップからテスト定義一覧に遷移し、テスト定義を作成します。



The screenshot shows the 'Test Definition Creation' (テスト定義作成) screen in Accel Studio. It is divided into two main sections: 'Basic Settings' (基本設定) and 'Test Definition Details - Sample Test 1' (テスト定義詳細 - サンプルテスト1).

基本設定 (Basic Settings):

- ID (必須):** sample_test1
- 名称:** サンプルテスト1 (標準 必須)
- 備考:** (標準)
- 親カテゴリ (必須):** サンプルカテゴリ
- テスト実行ジョブ対象:**

テスト定義詳細 - サンプルテスト1 (Test Definition Details - Sample Test 1):

- 実行 (Execute):**
- 登録テスト (Registered Test):**
- 実行パラメータ (Execution Parameters):**
 - 実行ユーザ (Execution User):** 青柳辰巳
- テストコード (Test Code):**

```

1 import { test, expect } from '@playwright/test';
2 import * as workflowUtil from 'im-workflow-util';
3
4 test('test', async ({ page }) => {
5
6     // Reference https://playwright.dev/docs/writing-tests
7     // await page.goto('https://www.intra-mart.jp/');
8     // await page.screenshot({ path: 'screenshot.png' });
9
10 });
                
```
- 前処理 (Pre-processor):**
- 後処理 (Post-processor):**

テストケース単位でテストコードを作成します。今回は登録処理のテストケースを作成します。

前処理・後処理のフロー定義作成

テスト実行時には、ブラウザでの操作と同様にアプリケーションが実際に操作されます。そのため、データ登録を伴うアプリケーションの場合、テスト実行前・実行後にデータをテストの想定値に合わせて登録・削除する必要があります。

このチュートリアルでは、データをクリアし、常にデータが空であることを保証する前処理・後処理を作成します。

アプリケーション詳細画面から、リソース追加、フロー定義の新規作成画面に遷移します。次に、SQL定義を追加し、利用するデータをクリアするSQL定義を作成してください。

SQL定義

データベース種別 *

TENANT ▾

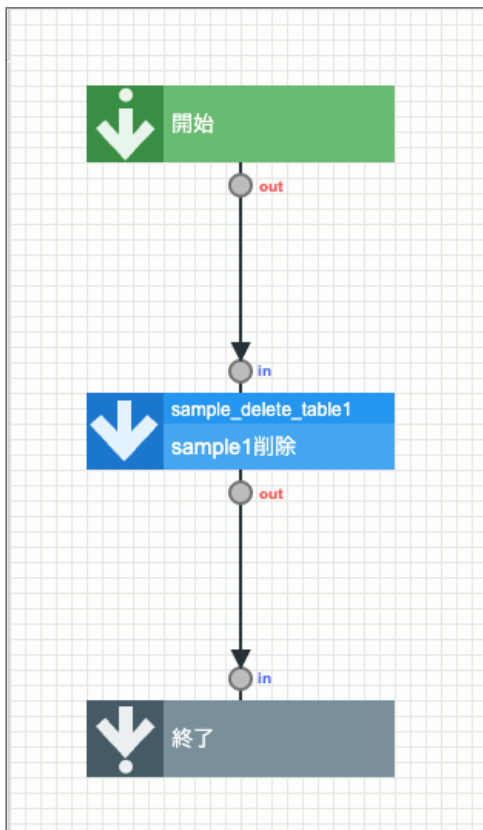
クエリ種別 *

DELETE ▾

クエリ *

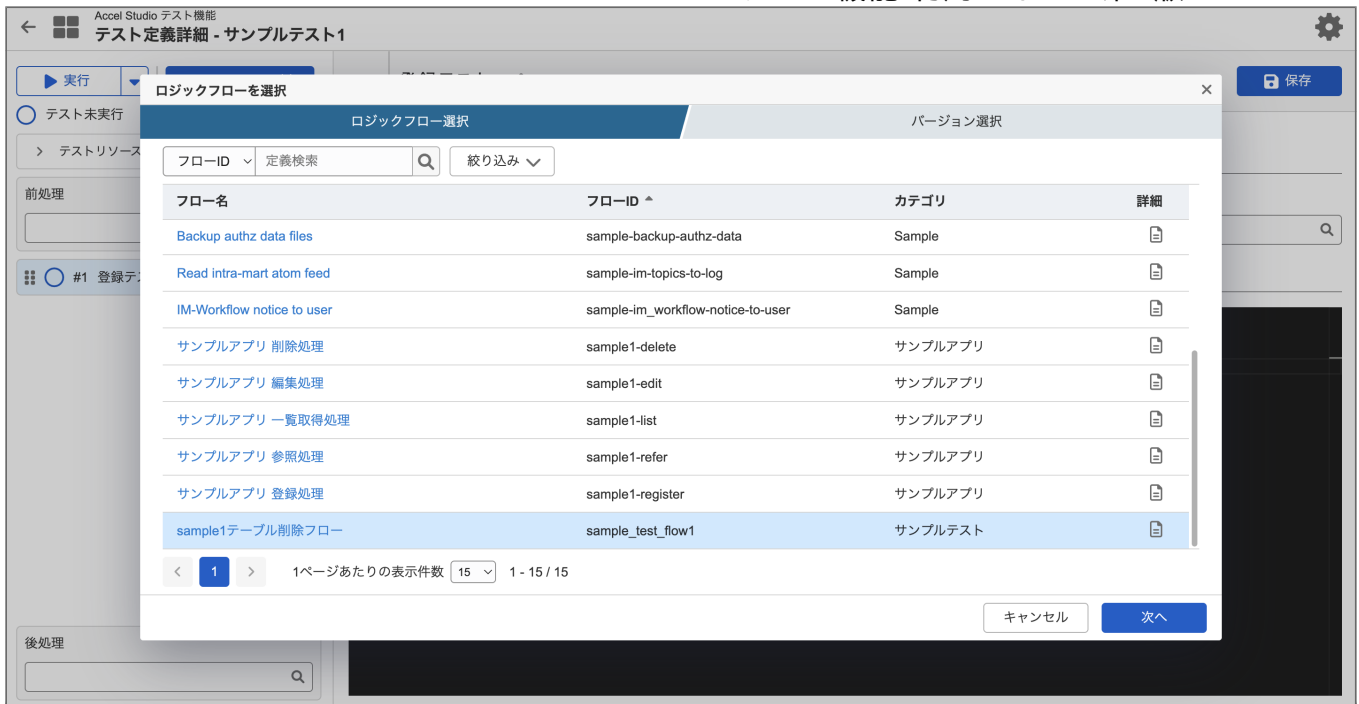
```
1 DELETE FROM sample1
2
```

このSQL定義を利用するシンプルなフロー定義を作成します。



前処理・後処理の指定

作成したフロー定義を前処理と後処理に指定します。



操作記録によるテストコードの作成

今回は、単純にデータを登録し、登録したデータが表示できるか確認するコードを作成します。VSCodeを利用して操作記録を行います。ツールの利用方法については、Playwright 公式ドキュメントを参照してください。

Test generator

コラム

実際に利用できるコードについては、Playwright の対応バージョンのドキュメントを参照してください。

API reference

記録、または記述したコードを利用してテストコードを作成します。下記がコードの例です。一覧画面に遷移し、新規登録を行い、詳細画面で登録したデータが表示されるか確認するテストコードです。

```
import { test, expect } from '@playwright/test';

test('test', async ({ page }) => {
  await page.goto('accel-studio-app/sample1/list');
  await page.getByRole('button', { name: '+ 新規作成' }).click();
  await page.locator('#im-element-48').fill('入力1');
  await page.locator('#im-element-63').fill('入力2');
  await page.getByRole('button', { name: '登録' }).click();
  await page.getByRole('button', { name: 'OK' }).click();
  await page.goto('accel-studio-app/sample1/refer?column1=%E5%85%A5%E5%8A%9B1');
  await expect(page.locator('#im-element-52')).toContainText('入力1');
  await expect(page.locator('#im-element-67')).toContainText('入力2');
});
```

テスト実行

テスト実行ボタンを押下するとテストを実行できます。

The screenshot shows the 'テスト定義詳細 - サンプルテスト1' (Test Definition Details - Sample Test 1) page. It includes a sidebar with navigation options like '実行' (Execute), 'テストケース追加' (Add Test Case), and '前処理' (Pre-processor). The main area is divided into '実行パラメータ' (Execution Parameters) with a user field set to '青柳辰巳' and 'テストコード' (Test Code) containing a Jest-style test script. The script includes actions like page navigation, form filling, and screenshot capture.

コラム
 テスト実行ボタンが押下されると、実行エージェントとの間で通信が発生します。なお、テストの実行は命令された順に直列で実行されるため、実行までに時間がかかる可能性があります。

テスト結果の確認

実行したテストは、テスト結果で確認できます。

test 関数ごとに結果ステータスと test タイトルがリストで表現されます。

The screenshot shows the 'テスト結果詳細 - サンプルテスト2 - 2025/09/18 14:44:07' (Test Results Details - Sample Test 2) page. On the left, a summary card shows '100% Passed' with 1 success and 0 failures. Below this is a table of test steps: '前処理: sample2テーブル削除フロー', '登録テスト' (6,603ms), and '後処理: sample2テーブル削除フロー'. The main area displays the '登録テスト' (Registration Test) details, including the user '青柳辰巳' and a success message. A '実行時刻ログ' (Execution Time Log) shows two test cases, 'test1' and 'test2', both passing.

test タイトルをクリックすると、テスト結果の詳細が表示されます。

The screenshot shows the 'Sample Test 2' results page. On the left, a summary card displays '100% Passed' with 1 success and 0 failures. The test 'test1' is marked as 'passed'. The main area shows a detailed execution timeline with various steps like 'Before Hooks', 'Navigate to "/>

コラム

アサーションエラー等が発生している場合、実行時タイムライン上部にエラーのサマリを表示するとともに、実行時タイムライン内でどのテストステップで発生しているのかを確認できます。

The screenshot shows the 'test1' results page where the test has failed. The error summary at the top states: 'Error: expect(locator).toContainText(expected) failed Locator: locator("#im-element-67") Expected string: "input3" Received string: "input2" Timeout: 5000ms Call log: ...'. The execution timeline below highlights step '#15: Expect "toContainText"' as failed. The detailed error message and call log are shown in a scrollable area, including the Playwright test code snippet:

```

Error: expect(locator).toContainText(expected) failed
Locator: locator("#im-element-67")
Expected string: "input3"
Received string: "input2"
Timeout: 5000ms
Call log:
- Expect "toContainText" with timeout 5000ms
- waiting for locator("#im-element-67")
  9 × locator resolved to <label id="im-element-67" class="im-hichee-inline imds-refer-field" data-uuid="ef77f9d2-87d6-4288-9054" >
    - unexpected value "input2"

13 |     await page.screenshot({ path: 'screenshot2.png' });
14 |     await expect(page.locator("#im-element-52")).toContainText('input1');
> 15 |     await expect(page.locator("#im-element-67")).toContainText('input3');
    |
16 |   });
17 |
18 |   test('test2', async ({ page }) => {
  
```

コラム

test 関数については、Playwright の対応バージョンのドキュメントを参照してください。

[API reference](#)

i コラム

2025 Spring(Kamille) で実行したテスト結果や、テスト実行時に何らかのエラーが発生した場合は test 関数やテストステップは表示されず、以下のデザインで表示されます。

The screenshot displays the 'Test Results Details' page for 'Sample Test 1' on 2025/03/18 at 18:37:18. The interface includes a summary section with a '100% Passed' indicator, showing 1 successful test and 0 failures. Below this, a list of test steps is shown, with '登録テスト' (Register Test) highlighted, indicating it took 2,951ms. The right-hand side of the interface shows the execution log, starting with '登録テスト' and '実行に成功しました。' (Execution successful). Below the log, there is a table of execution timelines with columns for request logs, system logs, entity operation logs, and screenshots/files. The table lists several HTTP requests (GET, POST, INSERT) and their corresponding system logs, all with a status of 200.

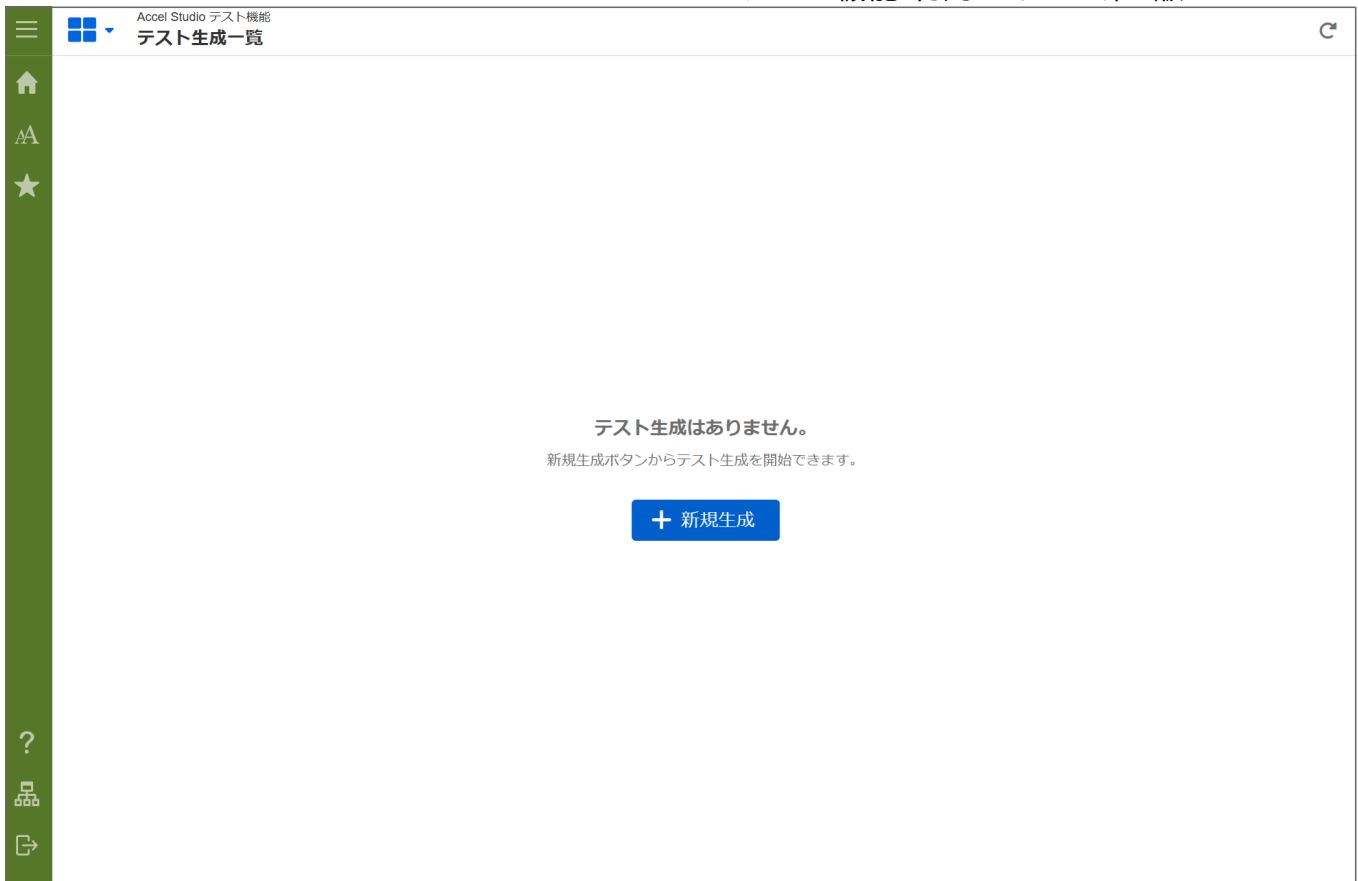
テスト定義一括生成

Accel Studio テスト機能 Copilot を利用して、テスト定義を一括生成します。

i コラム

Accel Studio テスト機能 Copilot は 2025 Autumn(Lilac) から利用可能です。Accel Studio テスト機能 Copilot を利用するためには IM-Copilot および Accel Studio テスト機能 テスト実行エージェントが必要です。IM-Copilot については「[IM-Copilot 利用ガイド](#)」を参照してください。

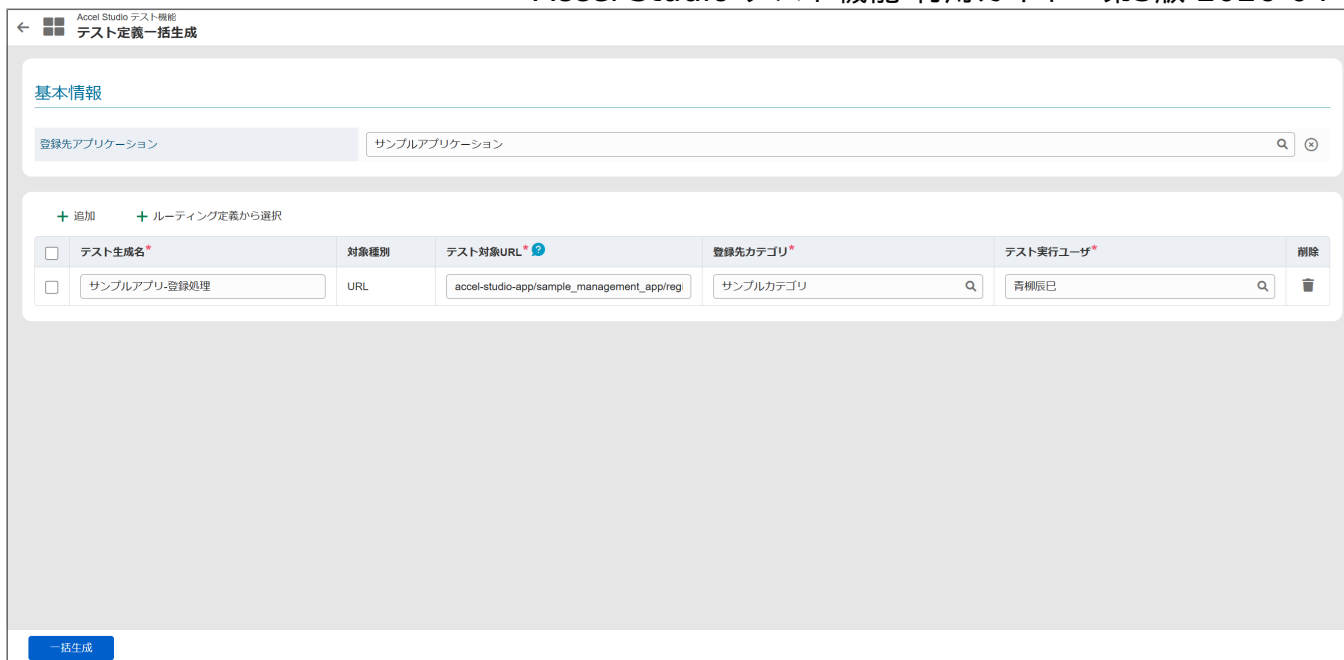
サイトマップからテスト定義一括生成画面に遷移します。



新規作成ボタンを押下し、テスト定義一括生成画面に遷移します。 テスト定義一括生成画面では、下記の項目を指定し、テスト定義を一括生成します。

今回はURLを指定してテスト定義を一括生成します。 IM-BloomMaker のルーティング定義を指定することも可能です。

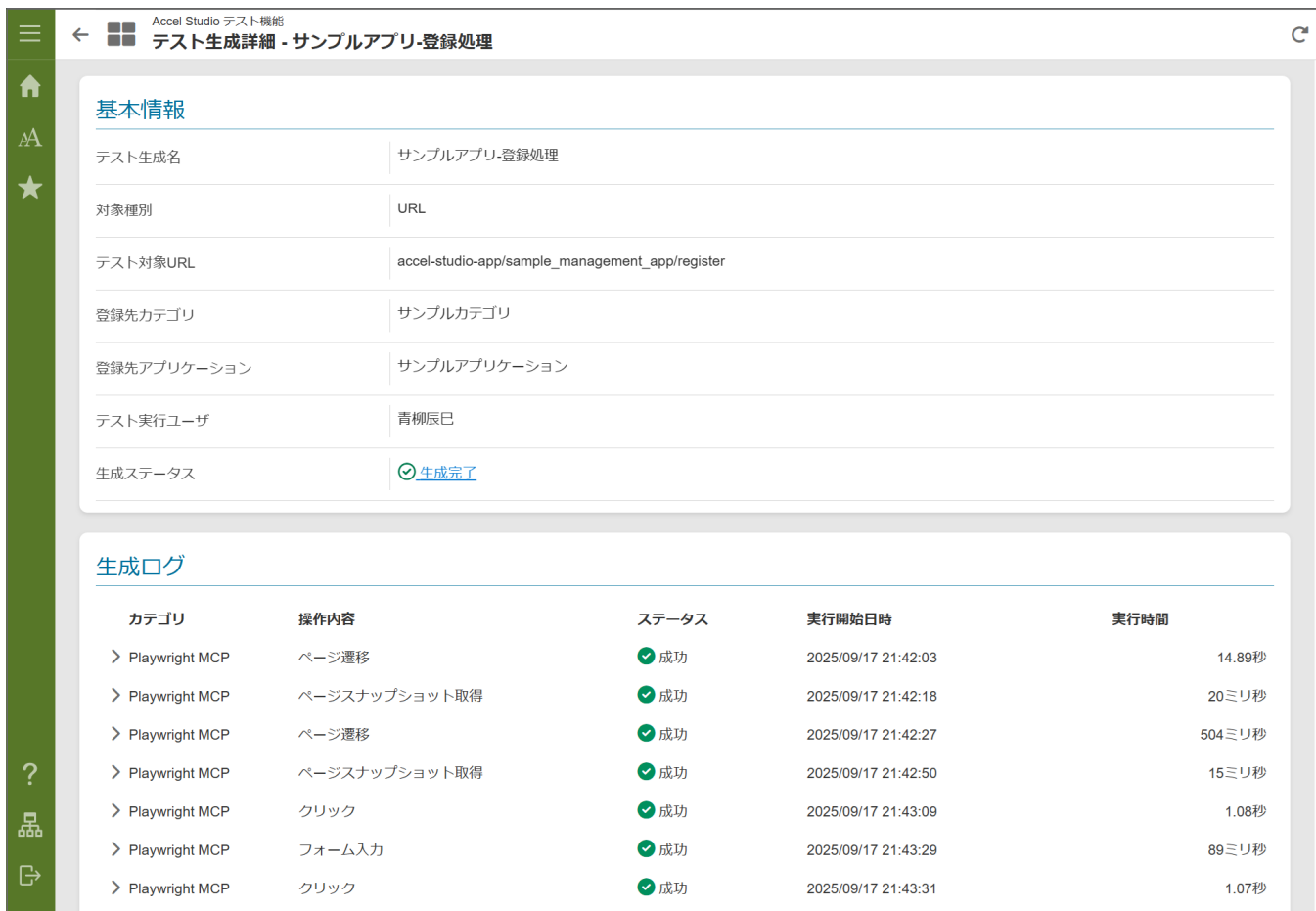
- 登録先アプリケーション
 - テスト定義を登録する Accel Studio アプリケーションを指定します。
- 一括生成情報
 - テスト名
 - 対象種別
 - URL: テスト対象のURLを直接指定
 - ルーティング定義: IM-BloomMaker のルーティング定義を指定
 - テスト対象URL
 - 登録先カテゴリ
 - テスト実行ユーザ



一括生成ボタンを押下すると、テスト定義が一括生成されます。テスト定義一覧画面で、「テスト生成名」のリンクをクリックすると、テスト生成詳細画面に遷移します。



テスト生成詳細画面で、テスト定義生成時のログや詳細情報を参照できます。



生成ステータスのリンクをクリックすると、生成されたテスト定義の詳細画面に遷移します。

Accel Studio テスト機能
テスト定義詳細 - 「accel-studio-app/sample_management_app/register」のテスト
⚙️

▶ 実行
+ テストケース追加

☐
必須項目を有効値で全て入力し、登録を実行すると成功し、一覧画面に新規...
更新

▶ Given 「登録画面 (/accel-studio-app/sample_management_app/register)」に遷移している...

実行パラメータ

実行ユーザ

テストコード

```

1 import { test, expect } from '@playwright/test';
2
3 test('必須項目を有効値で全て入力し、登録を実行すると成功し、一覧画面に新規レコードが表示されることを確認する', async ({ p
4   await test.step('登録画面 (/accel-studio-app/sample_management_app/register) に遷移する', async () => {
5     await page.goto('accel-studio-app/sample_management_app/register');
6     await page.waitForLoadState('domcontentloaded');
7     // 画面タイトルや見出しの表示確認
8     await expect(page.getByRole('heading', { name: '登録画面' })).toBeVisible();
9   });
10  await test.step('画面に「型番 *」入力欄と「登録」ボタンが表示されていることを確認する', async () => {
11    // 「登録」ボタン
12    const registerButton = page.getByRole('button', { name: '登録' });
13    await expect(registerButton).toBeVisible();
14    // 「型番」入力欄 (アクセシブルネームで取得を試み、なければ先頭のテキストボックスを使用)
15    let modelInput = page.getByRole('textbox', { name: /型番/ });
16    if ((await modelInput.count()) === 0) {
17      modelInput = page.getByRole('textbox').first();
18    }
19    await expect(modelInput).toBeVisible();
20    // 後続ステップで再利用できるようにテスト情報へ保存
21    await test.info().attach('field-locators', { body: JSON.stringify({ hasLabeledTextbox: (await page.getBy
22  });
23  await test.step('一意なテストデータ (『型番値』) を用意する', async () => {
24    const pad = (n) => String(n).padStart(2, '0');
25    const now = new Date();
26    const ms = String(now.getMilliseconds()).padStart(3, '0');
27    const 型番値 = `AT-${now.getFullYear()}${pad(now.getMonth()+1)}${pad(now.getDate())}${pad(now.getHours()
28    // 生成値の確認ログ
29    console.log('作成した型番値:', 型番値);
30  });
31  await test.step('「型番」入力欄に『型番値』を入力する', async () => {

```

前処理

#1 必須項目を有効値で全て入力し、...

後処理

著作権および特記事項

intra-mart は株式会社エヌ・ティ・ティ・データイントラマートの登録商標です。

Oracle と Java は、Oracle Corporation およびその子会社、関連会社の米国およびその他の国における登録商標です。

Playwright は Microsoft Corporation の登録商標です。

文中の社名、商品名等は各社の商標または登録商標である場合があります。

他の会社名、製品名およびサービス名などはそれぞれ各社の商標または登録商標です。

本製品を使用する場合は、本製品に含まれる各ソフトウェアのライセンスについても同意したものとします。

以上