Testablish 使い方マニュアル

目次

I. はじめに

Testablish とは

Testablish は、Webアプリケーションのテスト自動化を支援するツールです。

- **Webアプリケーションのテストシナリオ作成を支援します。**
  テストシナリオ作成のもととなる情報を収集し、実施した操作結果からテストシナリオひな型を生成することができます。
  収集した情報を参照しながら、アサーションをGUIで設定することが可能です。
- **Webアプリケーションでの実際の操作内容を収集し蓄積します。**
  収集した情報をもとに、テスト用に設定や編集を行い、テストコード、テスト仕様書を出力することができます。
  Testablish タスクトレイアプリで収集し蓄積した操作を繰返し実施することができます。
- **テストコードを自動生成します。**
  Seleniumを使用し、作成したテストシナリオからテストコードを自動生成します。
- **テスト仕様書を生成します。**
  作成したテストシナリオからテスト仕様書を自動生成します。

Testablish 概要図を以下に示します。

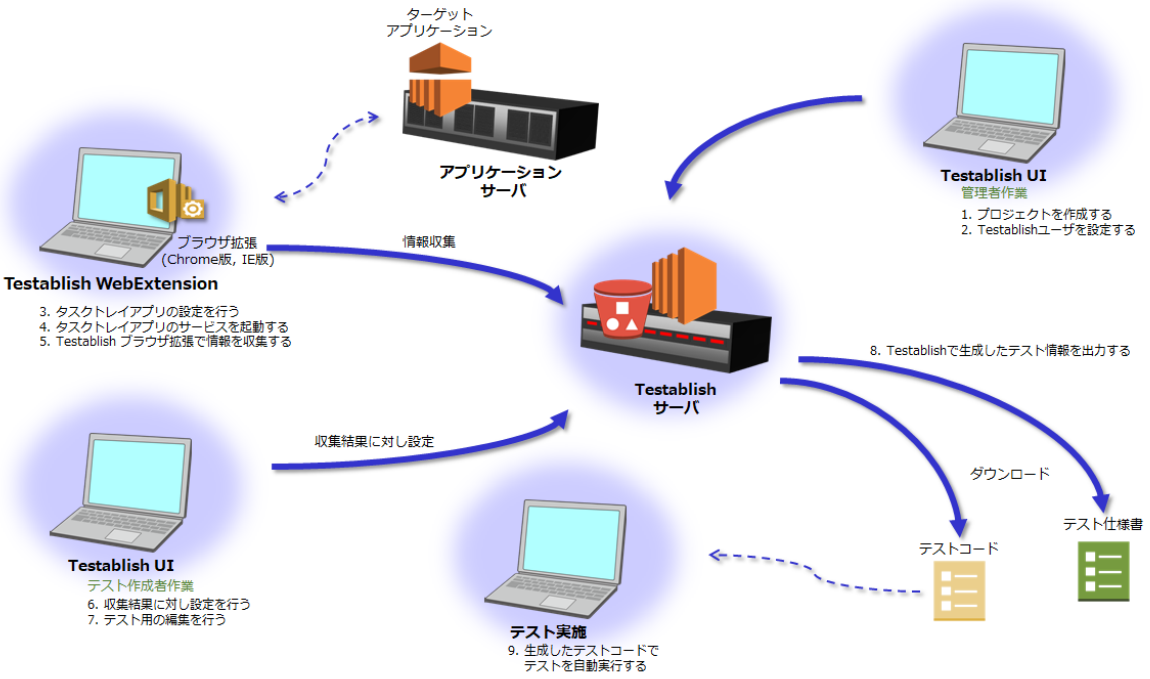


*Fig. Testablish*概要

- Testablish UI 管理者作業
  ユーザが Testablish を使用するための設定を行います。

- WebExtension
  操作した内容の情報収集を行います。

- Testablish UI テスト作成者作業
  テスト仕様書、テストコードを作成できるよう、GUIでテストシナリオを設定編集します。
- 自動生成
  作成したテストシナリオから、テスト仕様書、テストコードを自動生成します。
- テスト実施
  自動生成したテスト仕様書、テストコードを使用し、テストを実施します。

使い方マニュアル 目次 に戻る

II. 各画面の説明

1. 各画面共通
2. ホーム メニュー
3. フロー メニュー
4. ページ メニュー
5. テスト メニュー
6. 変数 メニュー
7. 統計 メニュー
8. メディア メニュー
9. コマンド メニュー
10. 設定 メニュー

1. 各画面共通

Testablishの画面には、常に表示される上部のヘッダエリアと、左側に表示されるメインメニューがあります。
※ メインメニューは ホーム画面でプロジェクトを選択した後の画面に表示されます。

Fig. 1 基本の画面構成

1.1 ヘッダエリア

Testablish 画面上部のヘッダエリアは常に表示されるエリアです。

No.	名称	説明
1	Testablish アイコン	プロジェクト一覧画面に戻ります。
2	プロジェクト名	選択しているプロジェクトを「プロジェクト名 / プロジェクトID」 の形式で表示します。
3	ユーザメニュー	ログインしているユーザ名を表示します。 クリックすると現在のユーザの権限に応じた管理メニューを表示します。 • 設定 • パスワード変更 • システム設定(管理者のみ) • ユーザー設定(管理者のみ) • ログアウト

Fig. 1.1 権限に応じたユーザメニュー

1.1.1 設定

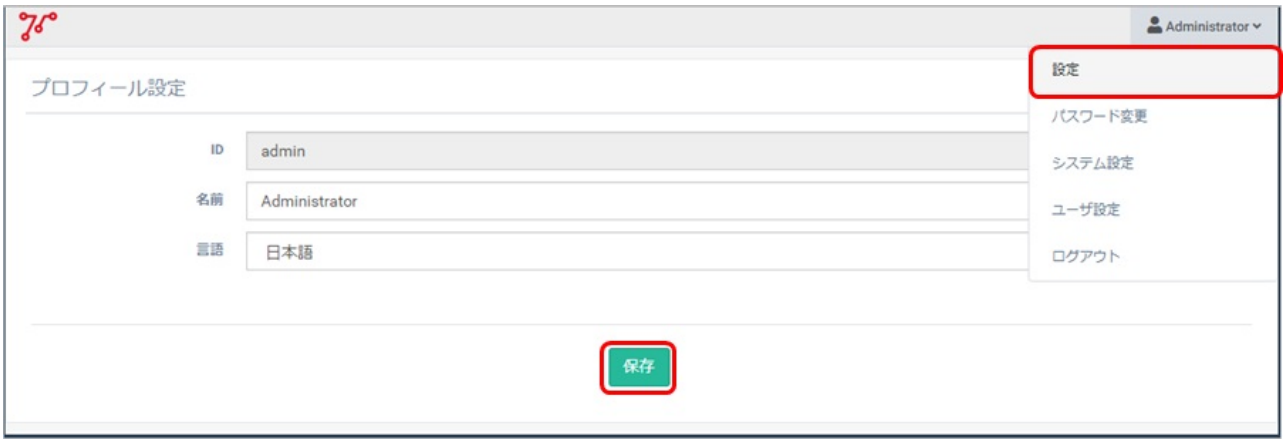


Fig. 1.1.1 プロフィール設定画面

このメニューを選択すると プロフィール設定画面が表示されます。
現在ログインしているユーザのの**名前**およびTestablishでの**使用言語**を設定することができます。
設定した名前は再ログイン後に有効になります。
言語は **保存**ボタンをクリック後すぐに有効になります。

1.1.2 パスワード変更

Fig. 1.1.2 パスワード変更画面

このメニュー項目を選択すると、 パスワード変更画面が表示されます。
現在のユーザのパスワードを変更できます。
現パスワード、新パスワードを入力し、**保存**ボタンをクリックして保存してください。

1.1.3 システム設定

このメニューは**管理者権限**をもつユーザにのみ表示されます。
このメニュー項目を選択すると、 システム設定画面のサービスタブが表示されます。

1.1.3.1 サービス タブ

システム設定画面で「Services」タグをクリックしてください。
Testablish server side service dashboard （以下Dashboard）画面が表示されます。
Dashboard画面では、システム設定画面で設定した各サービスの監視と起動ができます。

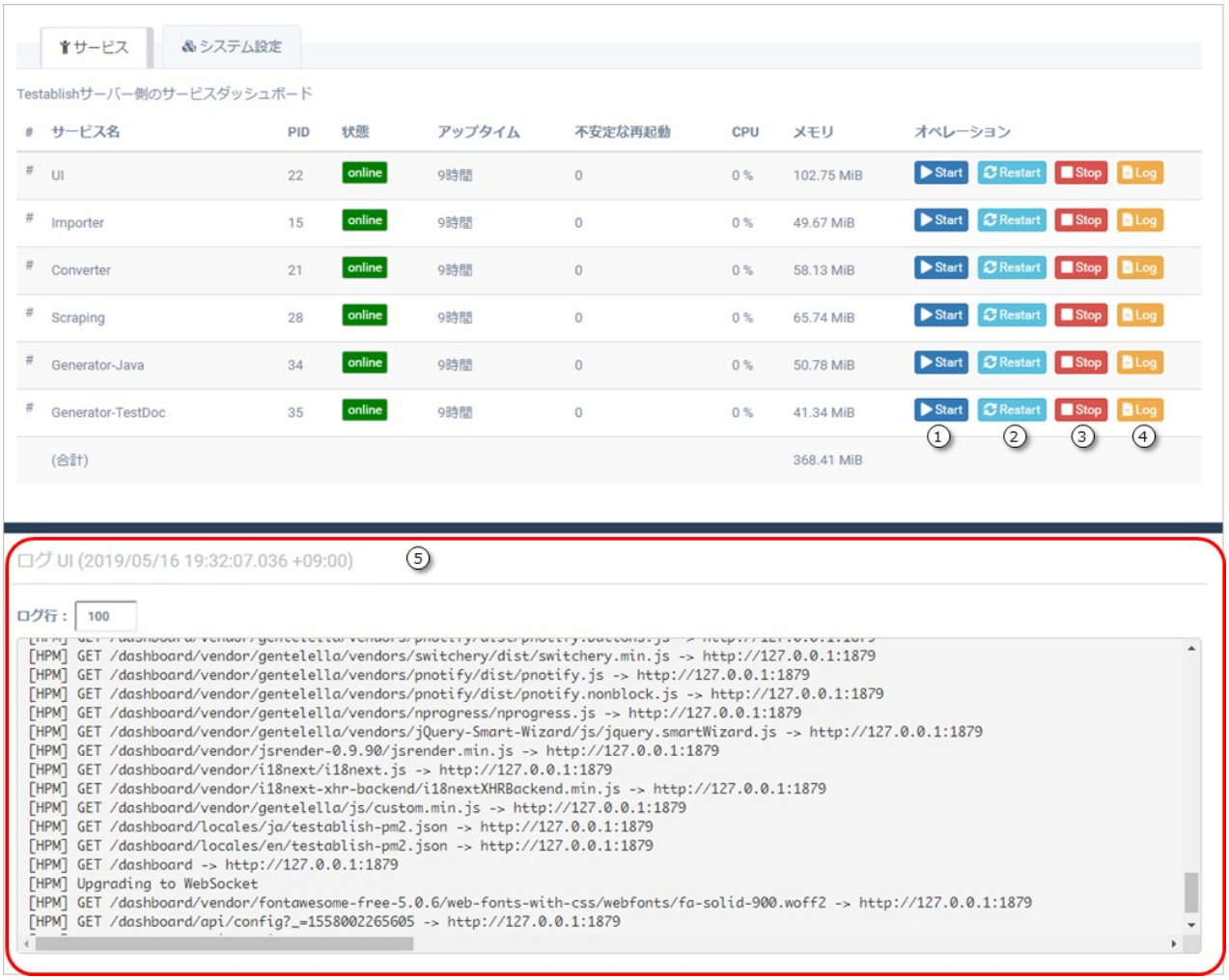


Fig. 1.1.3.1 Dashbord

No.	名称	説明
1	Start ボタン	サービスを起動します。
2	Restart ボタン	サービスを再起動します。
3	Stop ボタン	サービスを停止します。
4	Log ボタン	サービスのログを画面下のLog表示欄に表示します。
5	Log 表示欄	Logボタンをクリックした際にサーバーサイドのログが表示されます。

1.1.3.2 システム設定 タブ

システム設定画面で「システム設定」タグをクリックしてください。
システム設定画面の前半部分は、Testablishの各システムのホストアドレスとポート番号の設定項目です。

Testablishサーバでは各サービスを同一ホスト上に作成するため、ホストの項目は初期設定ではすべてlocalhost（127.0.0.1）に設定されています。

またTestablishでは各システムを同一ホスト上の別ポートで区分けしているため、ポート番号は各システムが重複しない番号が設定されています。
重複しない番号であれば、ポート番号を変更することも可能です。

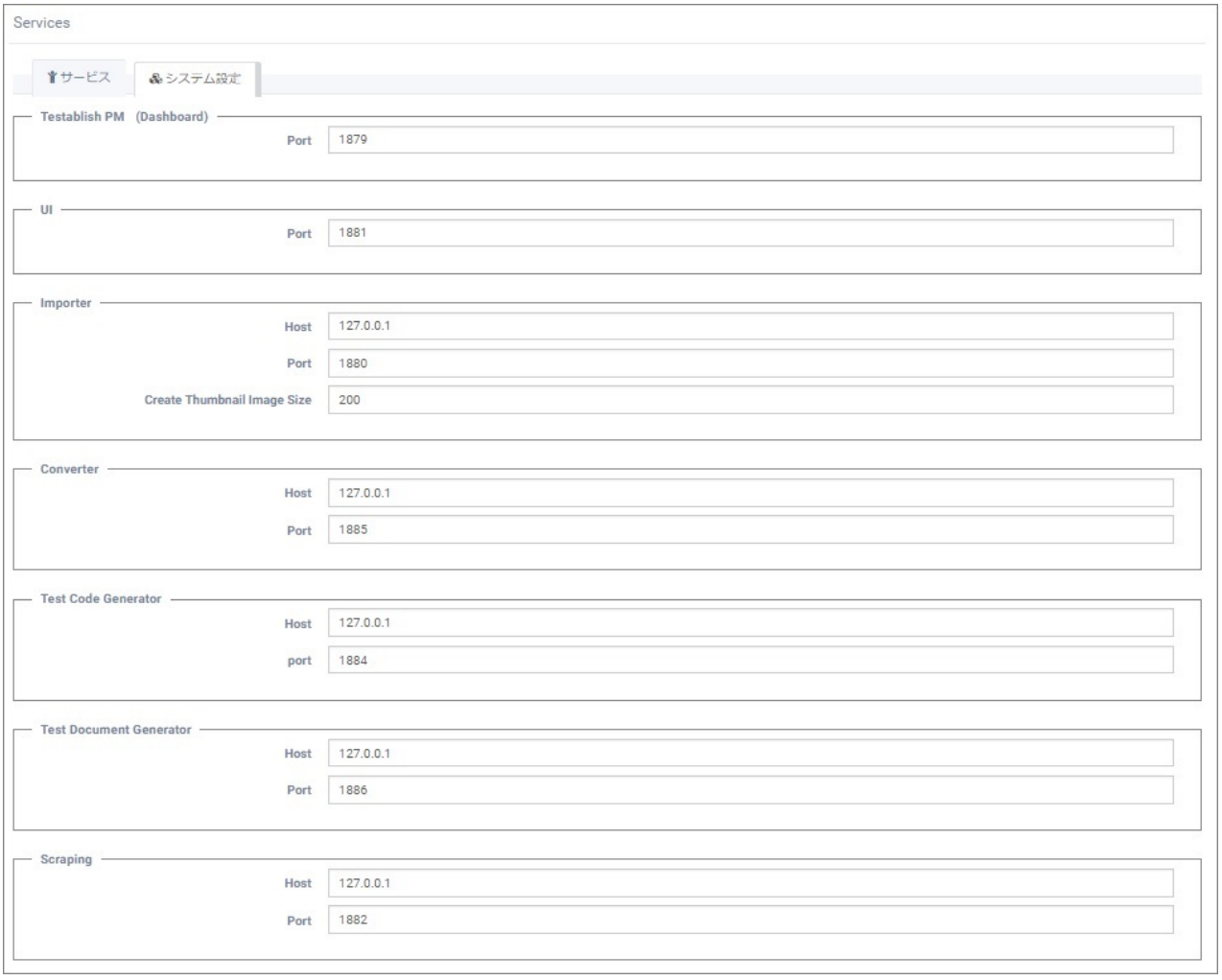


Fig. 1.1.3.2-1 システムのホスト・ポート設定

後半部分は、Testablishが使用するデータベースの設定項目です。

Testablishでは複数のデータベースを別ホストに作成します。
初期設定では、導入時に設定したホストのホスト名、ポート番号、ユーザ情報があらかじめ設定されています。

新たなデータベースを使用するためにホストを新たに作成した場合など、 各データベースの接続先を変更する必要がある場合は、この項目から設定変更してください。

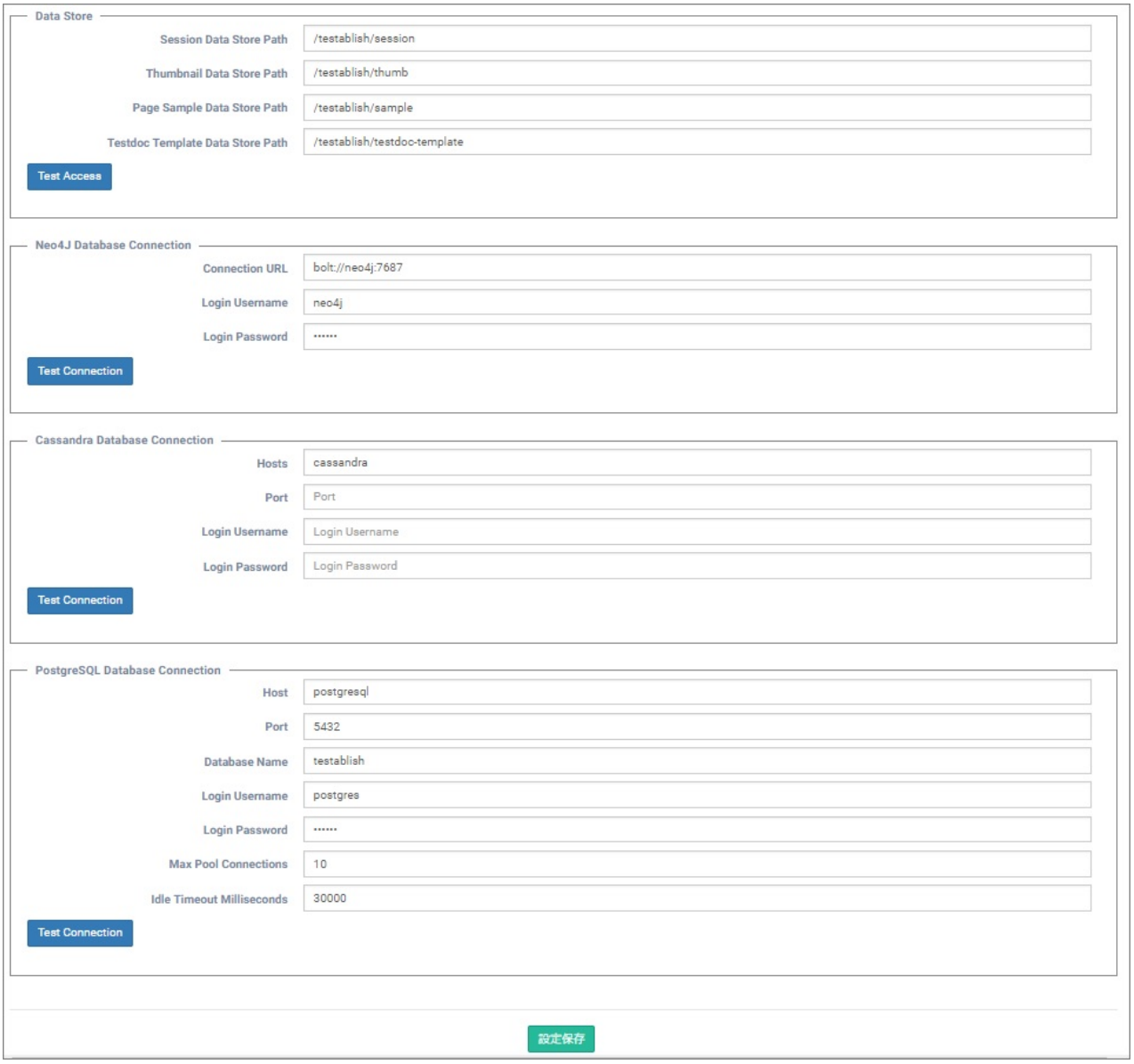


Fig. 1.1.3.2-2 データベース設定

1.1.4 ユーザ設定

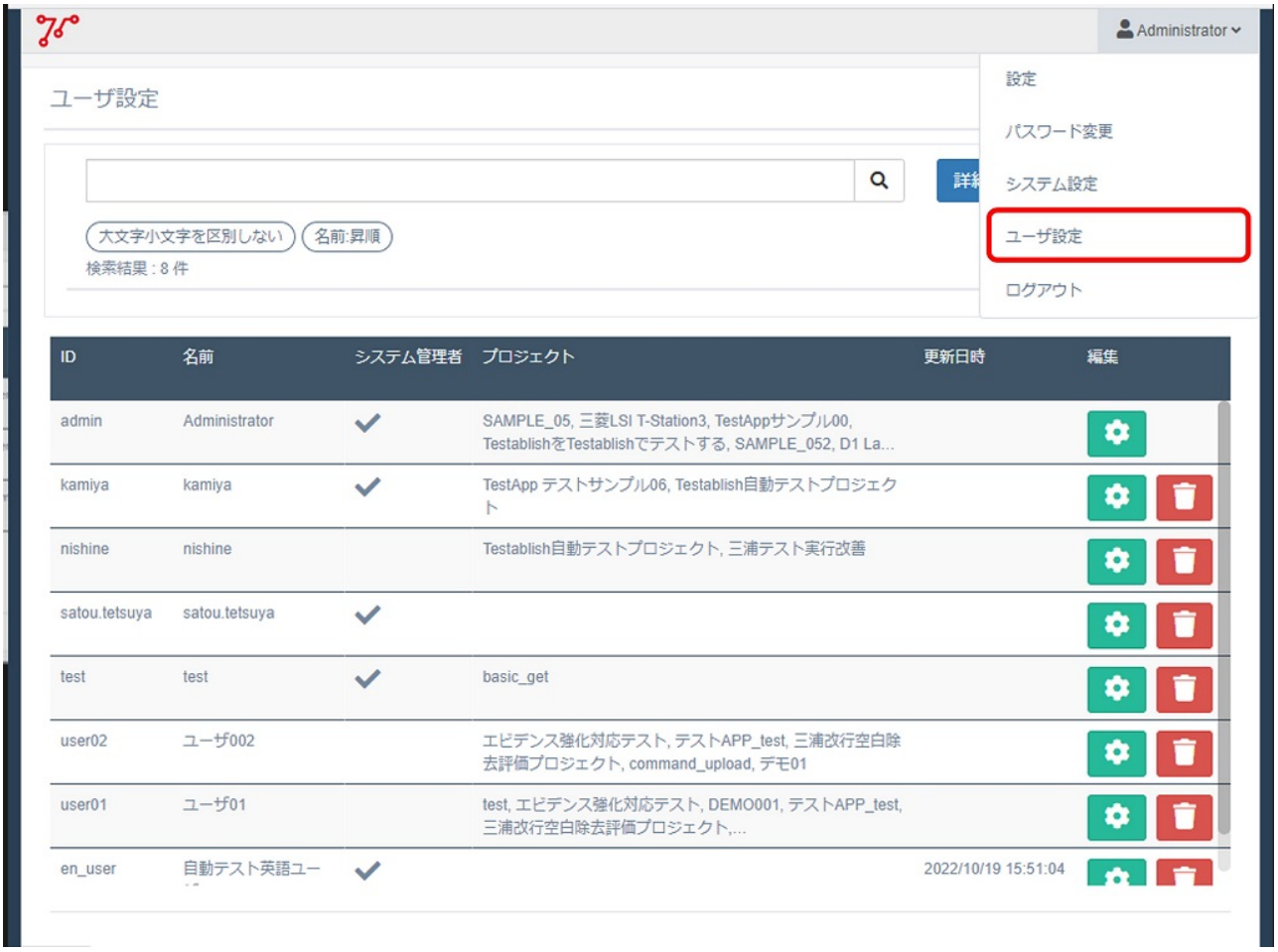


Fig. 1.1.4-1 ユーザ設定画面

このメニューは**管理者権限**をもつユーザにのみ表示されます。
このメニュー項目を選択すると、 ユーザ設定画面が表示されます。
Testablish を利用する ユーザ の新規追加・編集・削除が行えます。

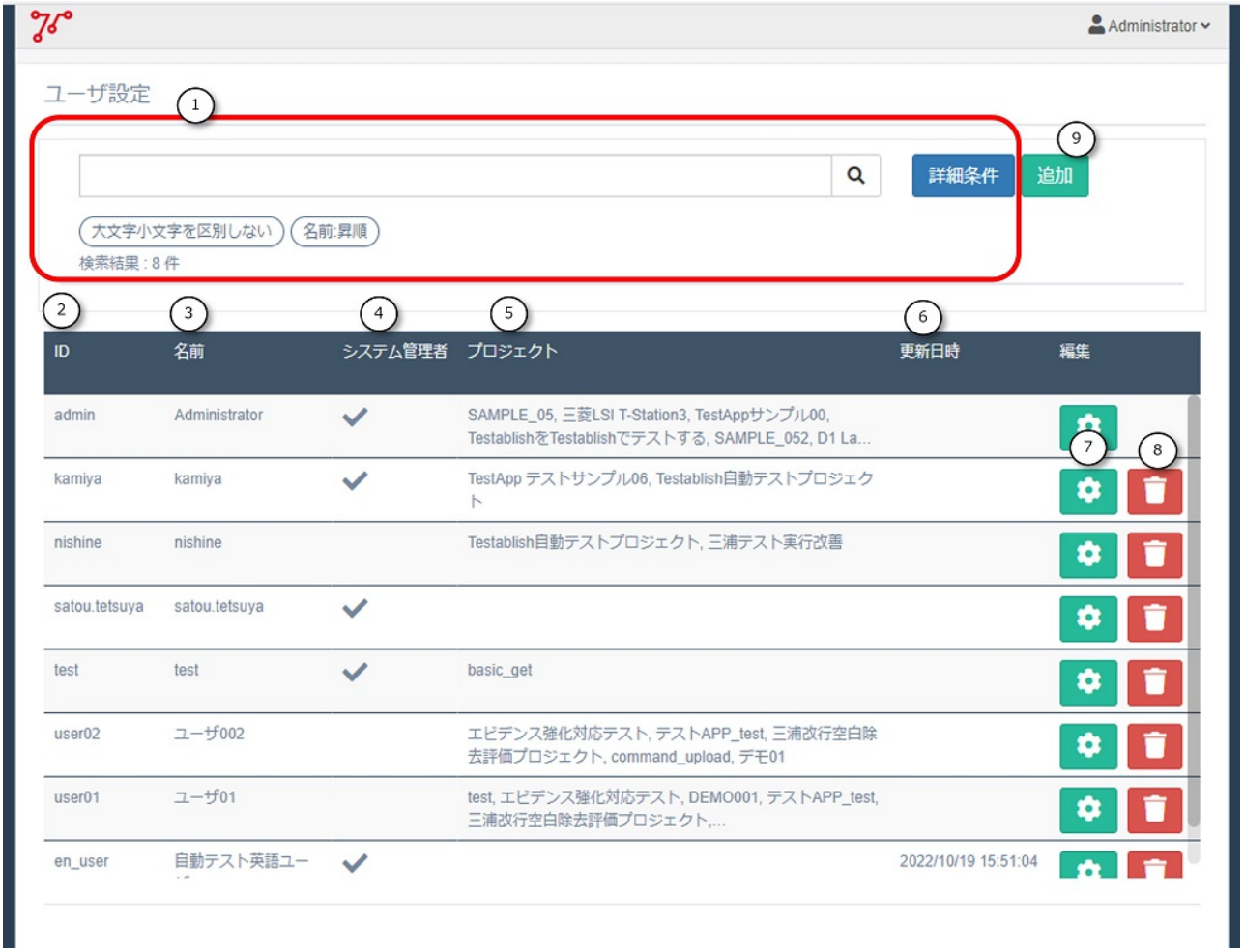


Fig. 1.1.4-2 ユーザ設定 編集画面

No.	名称	説明
ユーザー覧		
1	一覧検索・ソート	ユーザー一覧の表示内容の絞り込みと検索を行えます。 詳細条件ボタンを押下すると大文字小文字の区別と並べ替えを変更することができま
2	ID	IDを表示します。
3	名前	名前を表示します。
4	システム管理者	システム管理者権限を持つ場合は チェック（?）が表示されます。
5	プロジェクト	ユーザーが所属するプロジェクトが表示されます。
6	更新日時	ユーザー情報が更新された日時が表示されます。 ※v1.4.5以前から更新されていないユーザーの更新日時は表示されません。
7	編集ボタン	一覧の下部に編集エリアを表示し、該当のユーザ情報を表示します。
8	削除ボタン	該当のユーザを削除します。 ユーザが削除された場合、紐づいたユーザ情報は削除されます。 削除されたユーザ名は フロー画面 などでは **n/a** で表示されます。
9	追加ボタン	一覧の下部に新規ユーザの編集エリアを表示します。

上記追加ボタン(⑨）を押下すると以下の画面が表示され、ユーザーを追加することができます。

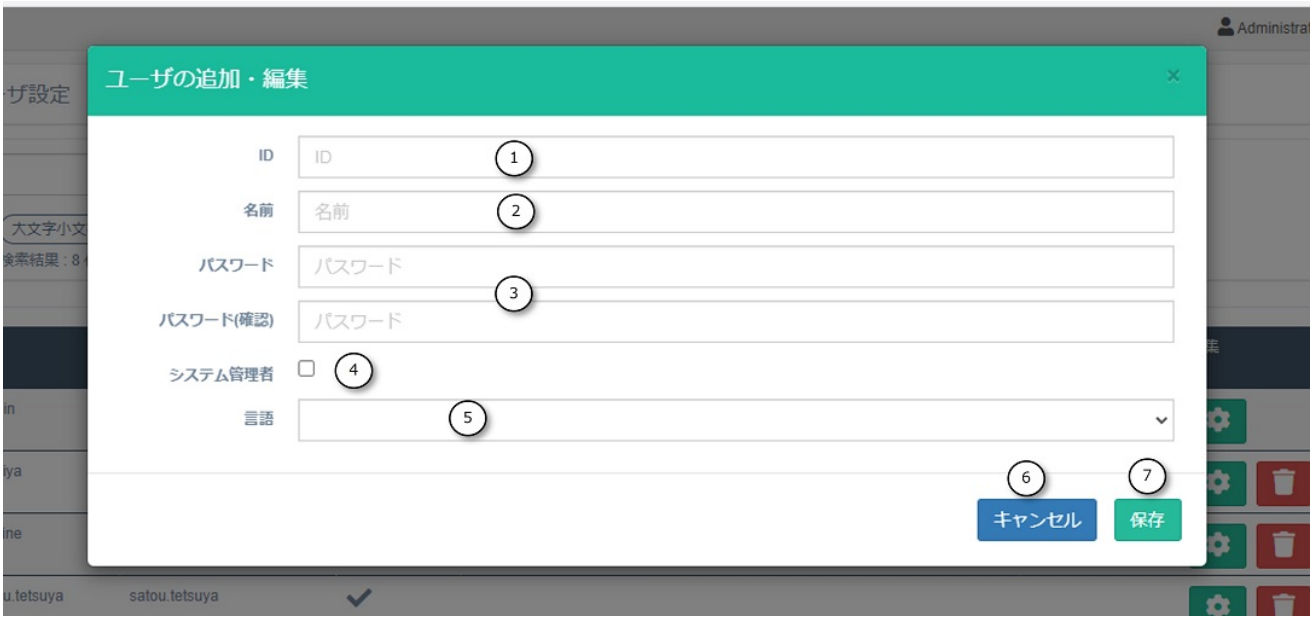


Fig. 1.1.4-3 ユーザ追加画面

No.	名称	説明
ユーザの追加・編集		
1	ID	ユーザIDを入力します。一度設定したユーザIDは編集できません。
2	名前	ユーザ名を入力します。
3	パスワード	半角英数記号で設定してください。日本語は使用できません。
4	システム管理者	システム管理者権限を与える場合は ON 、 一般ユーザ権限の場合はOFFにします。
5	言語	日本語 または English を選択します。
6	キャンセルボタン	編集エリアの入力をキャンセルしてエリアを閉じます。
7	保存ボタン	編集エリアの入力を保存してエリアを閉じます。 保存した情報は上部の一覧に反映されます。

1.1.5 ログアウト

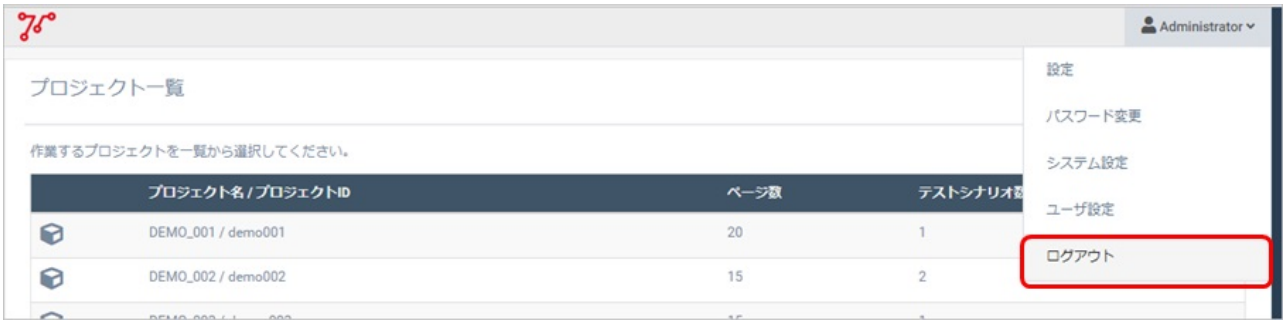


Fig. 1.1.5 ログアウト

このメニュー項目を選択すると、 Testablishからログアウトします。
ログアウト後はログイン画面に戻ります。

1. 各画面共通 目次 に戻る

1.2. メインメニュー

Testablish 画面左のメインメニューは、プロジェクト一覧からプロジェクトを選択した後に表示されます。

No.	アイコン	説明
1		プロジェクト一覧画面に戻ります。
2		選択しているプロジェクトの フロー/ビューセット編集 画面を表示します。
3		選択しているプロジェクトの ページ一覧 画面を表示します。
4		選択しているプロジェクトの テスト一覧 画面を表示します。
5		選択しているプロジェクトの 変数一覧 画面を表示します。
6		選択しているプロジェクトの テスト統計情報 画面を表示します。
7		選択しているプロジェクトの メディア 画面を表示します。
8		選択しているプロジェクトの コマンド一覧 画面を表示します。
9		選択しているプロジェクトの プロジェクト設定 画面を表示します。

1. 各画面共通 目次 に戻る

1.3 一覧表示での検索・ソート機能

Testablish の 一覧形式で表示されている画面では、一覧の表示内容の絞り込みとソート(並べ替え)を行うことができます。
一覧画面には共通の検索ボックスと検索ボタン、詳細条件を設定するためのボタンが配置されています。
また、一覧表示の件数や条件のラベルが表示されます。

Fig. 1.3-1 一覧表示での検索・ソート機能

No.	名称	説明
検索		
1	検索ワード入力欄	名称で絞り込むことができます。
2	検索ボタン	1 に入力した語句で検索を実行します。空白の場合すべて表示します。
ソート		
3	詳細条件ボタン	詳細条件ダイアログを表示します。
4	条件ラベル	3 で設定したソート条件を表示します。
5	検索結果件数	2 で実行した検索結果の件数を表示します。

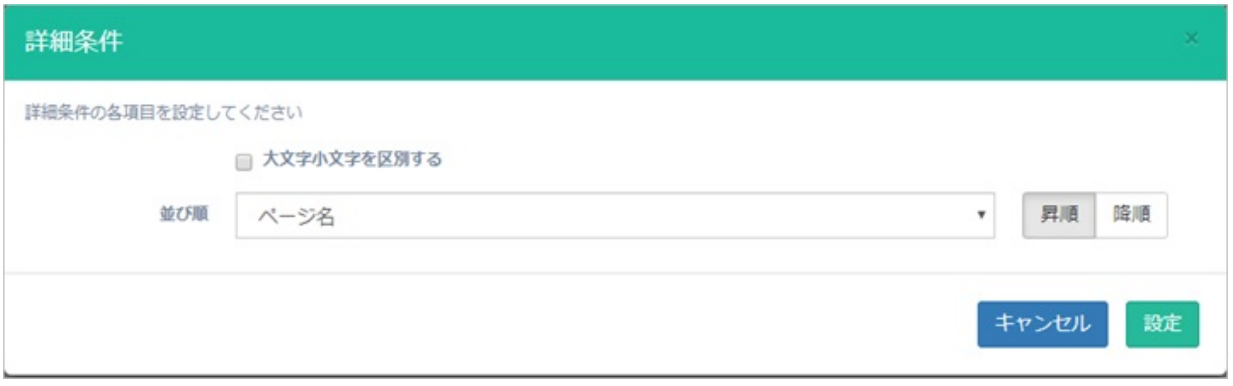


Fig. 1.3-2 詳細条件ダイアログ（ページ一覧）

一覧画面の種類によって、設定できる詳細条件やプルダウンの項目はは多少異なります。
また、設定した詳細条件によって一覧に表示される条件ラベルの表示数も異なることがあります。

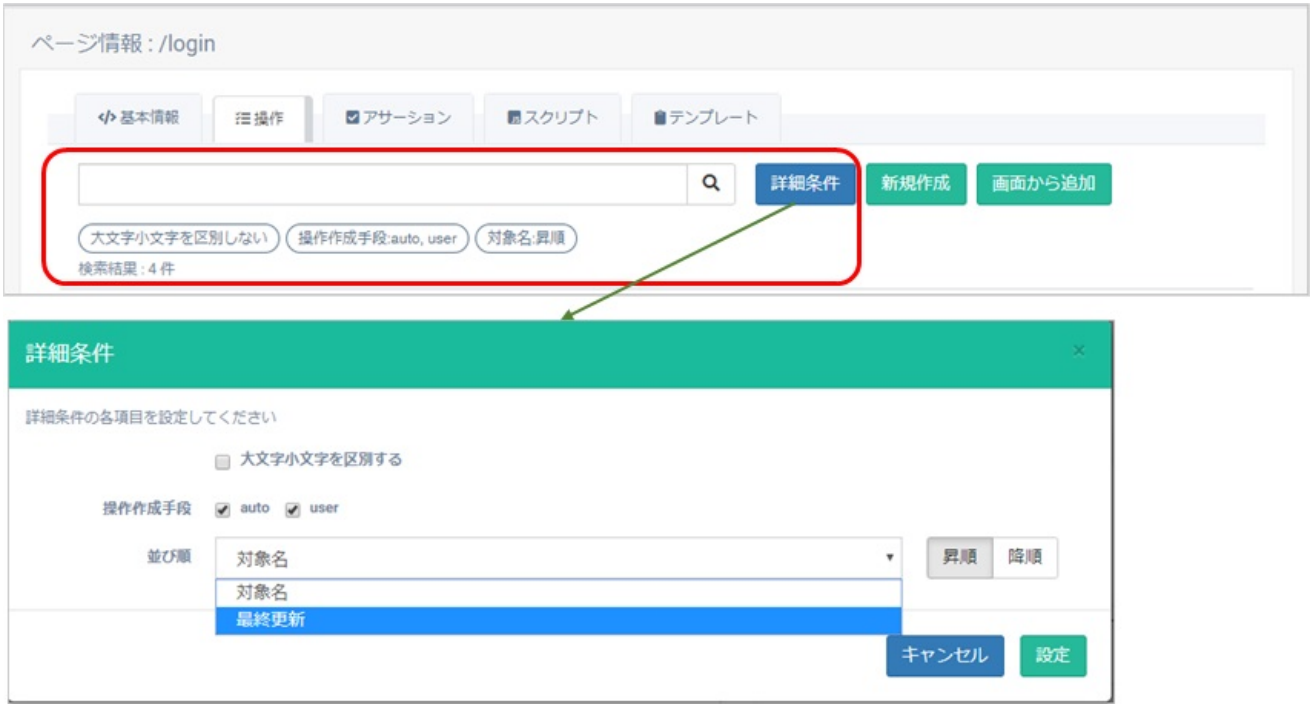


Fig. 1.3-3 詳細条件ダイアログ（ページ情報：操作一覧）

1. 各画面共通 目次 に戻る

1.4 依存一覧リンク

ページ、ページ操作、ページアサーション、ページスクリプト等の一覧の項目の右に**依存一覧リンク**
テスト依存一覧 テストテンプレート依存一覧 が表示されることがあります。
依存一覧にはテストの中で使用されている『テスト依存一覧』とテストテンプレートの中で使用されている『テストテンプレート一覧』があります。

このアイコンをクリックすると、その項目を使用しているテストの一覧画面を別ウィンドウで開きます。
テスト一覧で依存一覧リンクをクリックした場合にのみ、そのテストを使用しているテストスイート一覧が別ウィンドウで開きます。

別ウィンドウで開いたテスト一覧画面のタイトル横およびウィンドウタイトルに遷移元の画面名が ( ) 付きで表示されます。

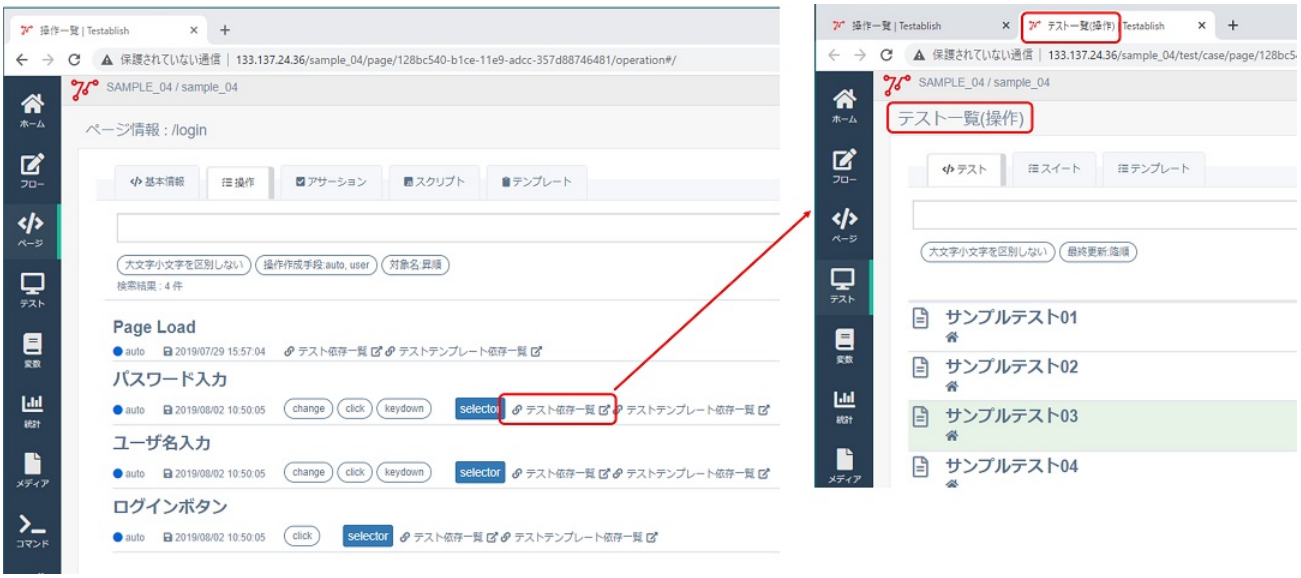


Fig. 1.4 依存一覧リンクで開くテスト一覧画面

操作やテストを編集したり削除したりする際に、依存一覧を参照することで、その影響範囲を確認することができます。

1. 各画面共通 目次 に戻る

1.5 一覧表示でのフォルダ操作

テスト一覧画面 および ページ一覧ではフォルダによる階層構造の表示・編集、一覧アイテムの複製・削除を行うことができます。

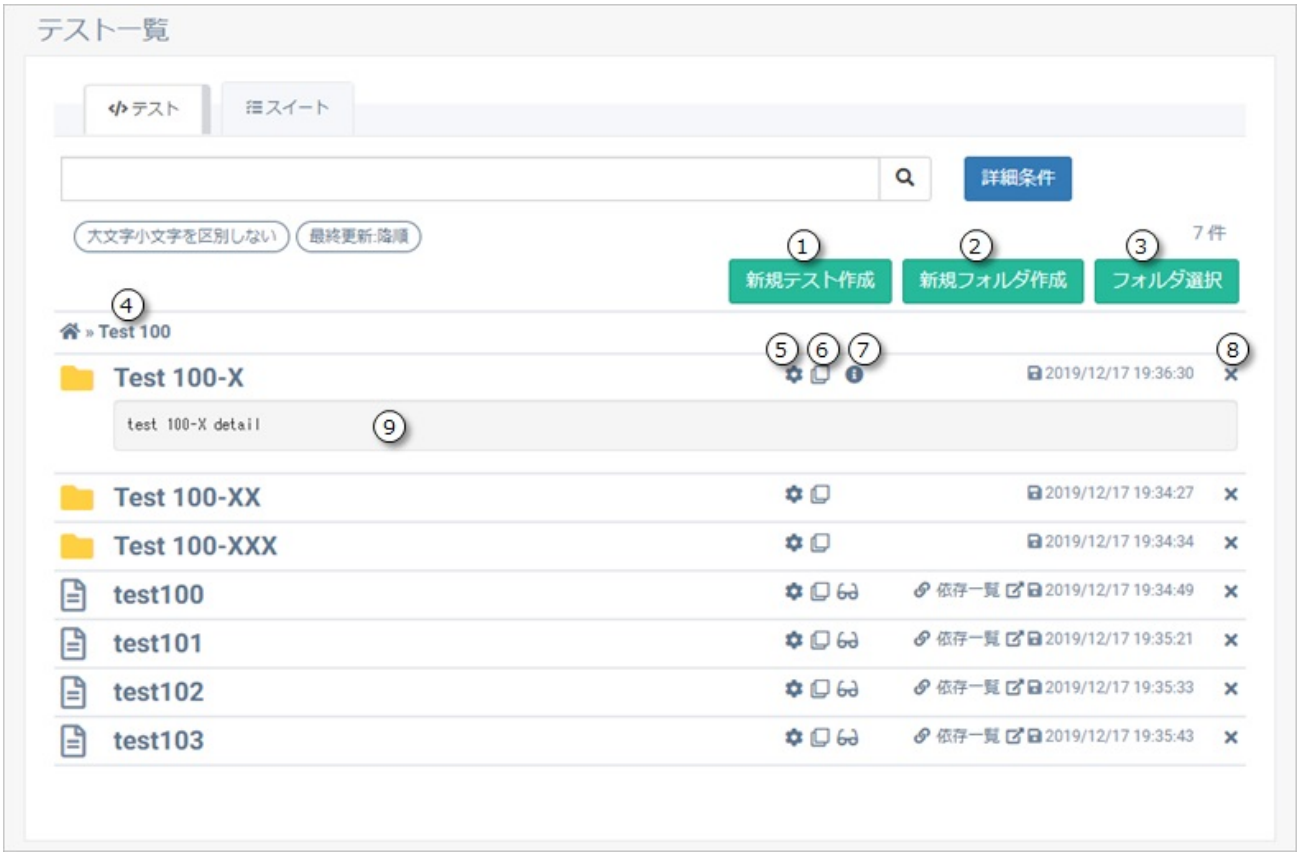


Fig. 1.5 一覧表示でのフォルダ操作

No	名称	説明
1	新規作成ボタン	ダイアログを表示し、一覧のアイテムを新規作成します。
2	新規フォルダ作成ボタン	ダイアログを表示し、新規フォルダを作成します。
3	フォルダ選択ボタン	ダイアログを表示し、選択したフォルダの階層を開きます。
4	現在位置表示	現在開いているフォルダの階層を表示しています。 フォルダ名をクリックすることでその階層を開くことができます。
5	設定アイコン	一覧のアイテムまたはフォルダの情報を設定することができます。
6	移動・複製アイコン	一覧のアイテムまたはフォルダの移動および複製を行うことができます。
7	情報アイコン	5 でフォルダの説明が設定されている場合、 このアイコンが表示されます。 クリックすることで 9 の情報表示欄を表示・非表示を切り替えます。
8	削除アイコン	一覧のアイテムまたはフォルダを削除することができます。
9	情報表示欄	5 で設定した情報のうち、説明欄の内容を表示します。

1.5.1 フォルダの選択

一覧のアイテムがフォルダの場合、名称をクリックすることでそのフォルダを開くことができます。

開きたいフォルダの階層が深い場合などは、フォルダ選択ボタンを利用することによって、一気にその階層を開くことができます。

一覧画面右上の フォルダ選択ボタンをクリックすると、フォルダの選択ダイアログが表示されます。

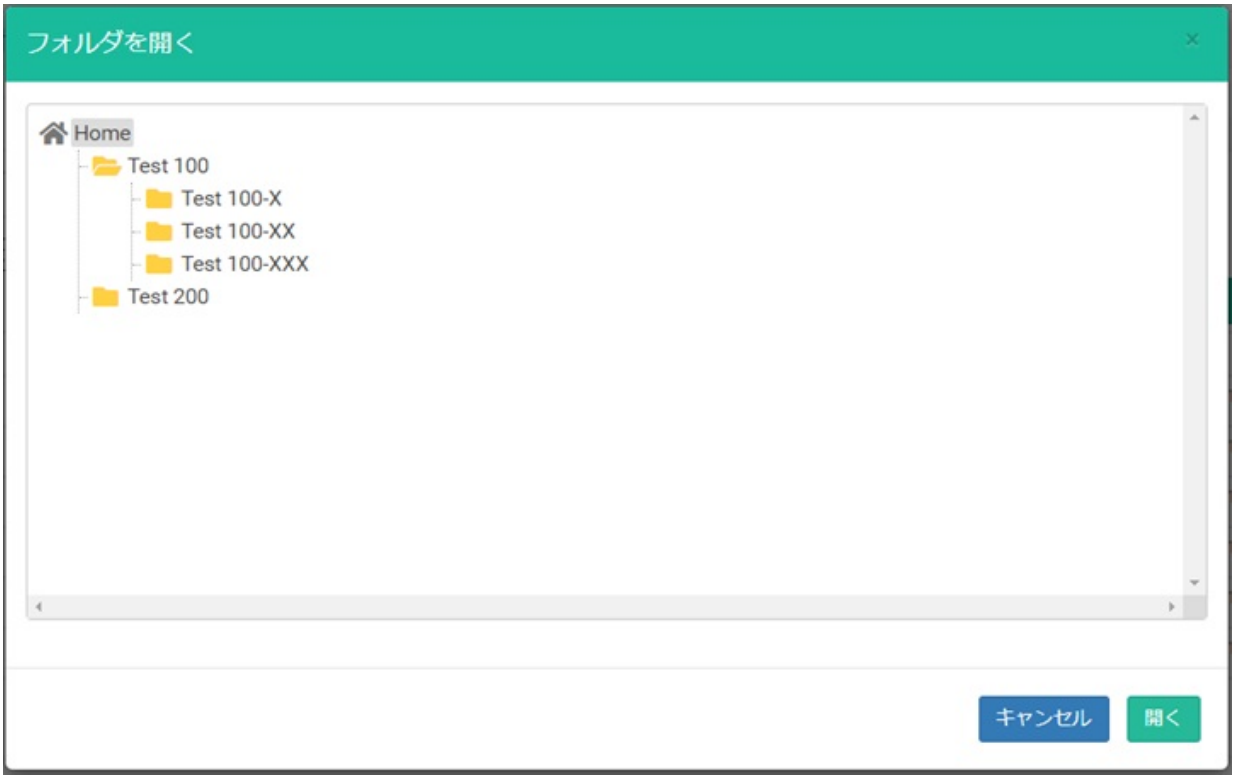


Fig. 1.5.1 フォルダ選択ダイアログ

開きたいフォルダを選択し、ダイアログ下部の 開く ボタンをクリックすると、指定のフォルダ階層を開くことができます。

1.5.2 一覧アイテムの移動・複製

一覧で移動・複製したいアイテムに表示されている 移動・複製アイコン をクリックすると、アイテムの移動・複製ダイアログが表示されます。

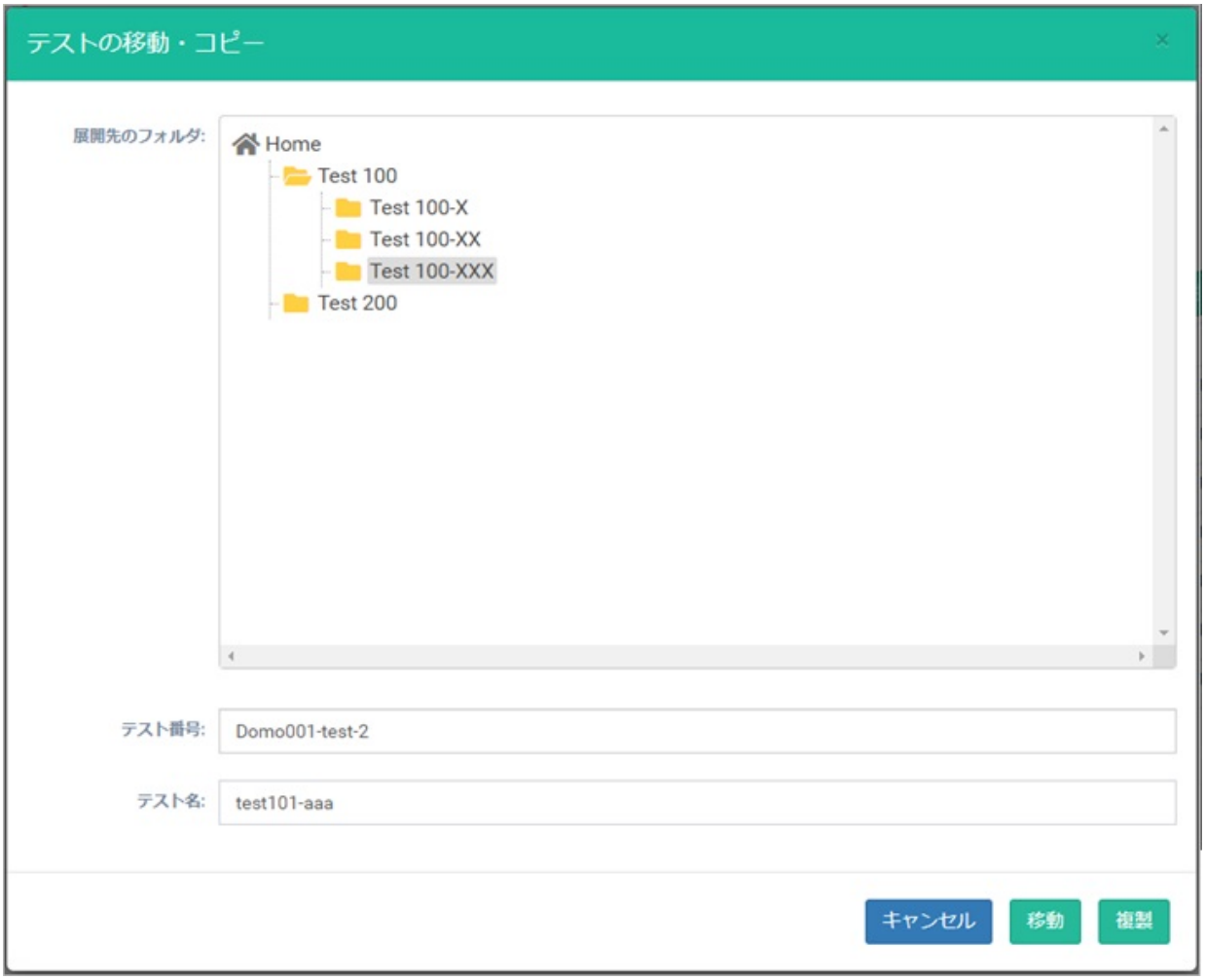


Fig. 1.5.2 アイテムの移動・複製ダイアログ

ダイアログ上部 展開先のフォルダエリアで、移動・複製先のフォルダを選択します。

ダイアログ下部で名称などの項目を入力します。
移動・複製先のフォルダに同名のアイテムがある場合はエラーになります。

移動ボタン または 複製ボタン をクリックすると、指定したフォルダ階層に指定した名称でアイテムが移動・複製されます。

1. 各画面共通 目次 | II. 各画面の説明 目次 | 使い方マニュアル 目次 に戻る

2. ホーム メニュー

2.1 プロジェクト一覧 画面

プロジェクト一覧画面は、登録されたプロジェクトを一覧表示する画面です。
プロジェクトを選んで、プロジェクトごとの画面に移動することができます。

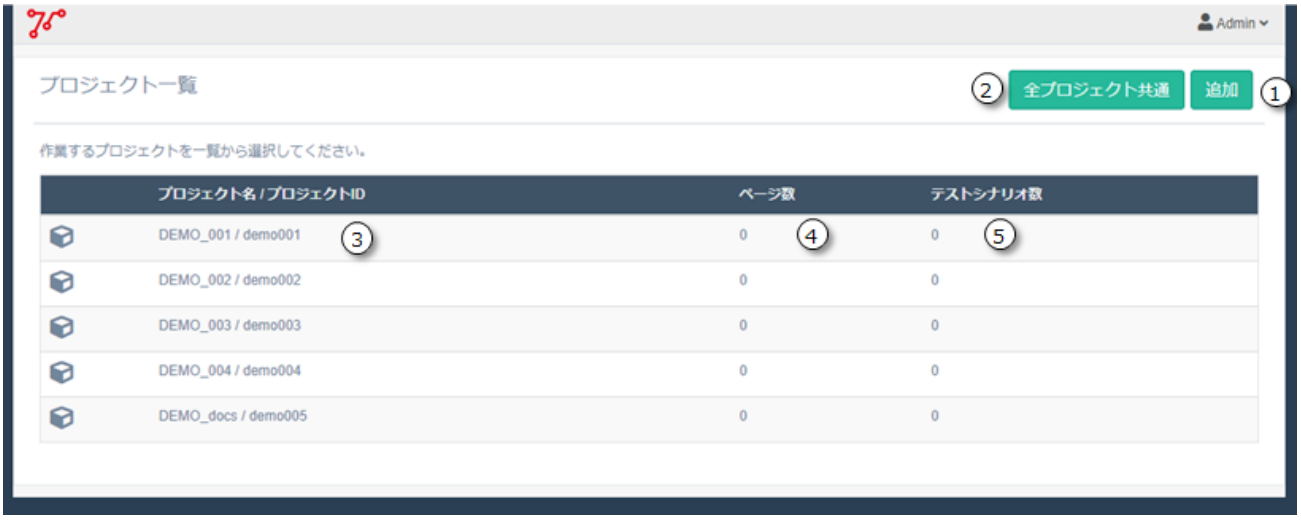


Fig. 2.1 プロジェクト一覧画面

No.	名称	説明
1	追加ボタン	プロジェクトを追加できます。
2	全プロジェクト共通ボタン	全てのプロジェクトで使用できる共通操作
3	プロジェクト名/ プロジェクトID	クリックすると、選択したプロジェクトの フロー/ ビューセット編集画面に移動することができます。
4	ページ数	プロジェクトに設定されているページの数
5	テストシナリオ数	プロジェクトに設定されているテストシナリオの数

2.1.1 追加 ボタン

追加ボタンをクリックすると、「プロジェクト設定」ダイアログが表示され、プロジェクトを**新規追加**することができます。
プロジェクトの追加は、**管理者権限**を持つユーザのみが行うことができます。（一般ユーザには追加ボタンは表示されません。）

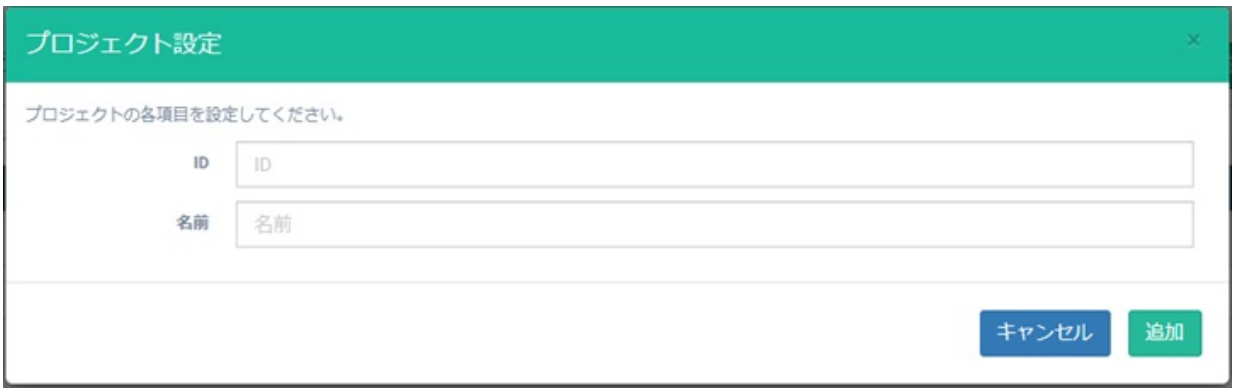


Fig. 2.1.1 プロジェクト一覧画面

「プロジェクト設定」ダイアログで「追加」ボタンをクリックすると新規プロジェクトが作成され、プロジェクト一覧に追加されます。
※ プロジェクト作成には 数十秒～数分程度、時間がかかる場合があります。
※ プロジェクトが一覧に表示されるまでお待ちください。

新しく追加されたプロジェクトを開き、メインメニューから**設定**をクリックして作成したプロジェクトの詳細設定を行ってください。⇒ 10.プロジェクト設定画面

2.1.2 全プロジェクト共通 ボタン

全プロジェクト共通ボタンをクリックすると、全プロジェクト共通機能(Common Project)と呼ばれる特殊なプロジェクトを設定できる画面に移動します。
全プロジェクト共通では、普通のプロジェクトと同様に、ページ情報の操作・アサーション・スクリプト・ページテンプレート、コマンドを設定することができ、それらはテスト作成の際に利用することができます。

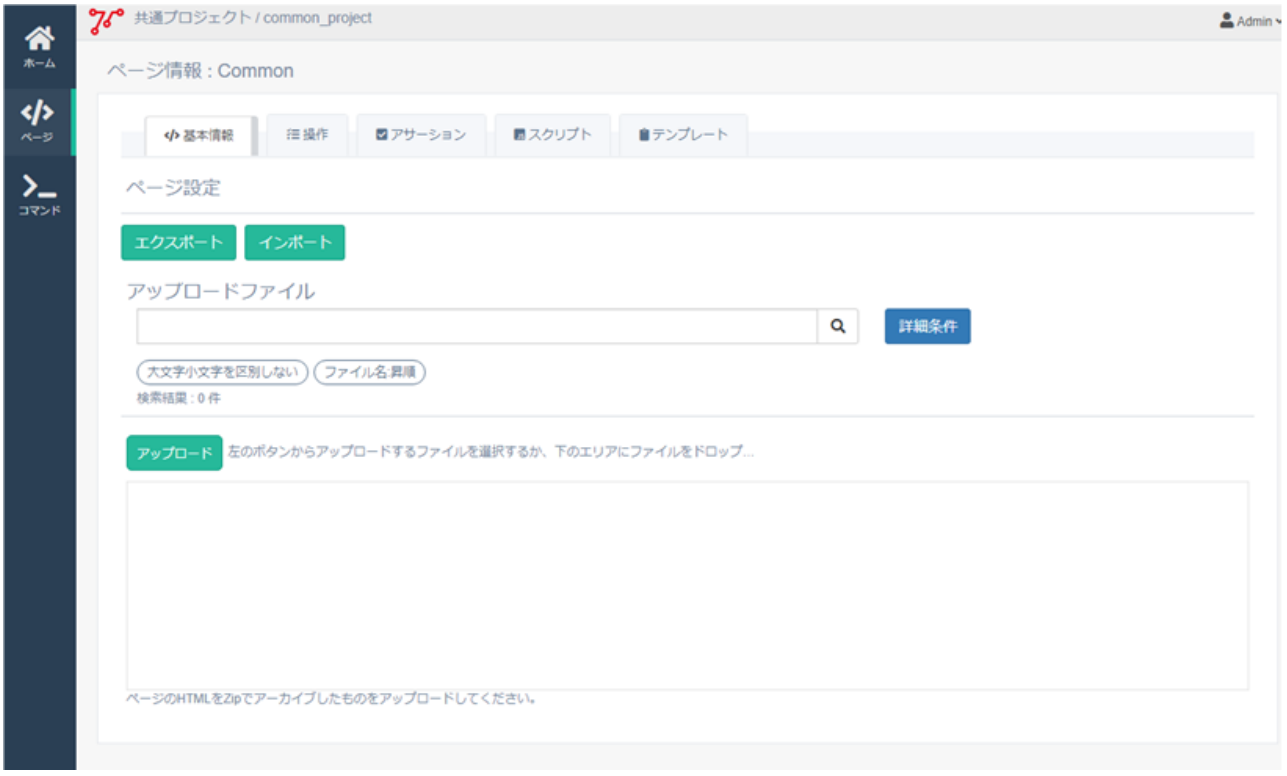


Fig. 2.1.2 全プロジェクト共通画面

全てのプロジェクトに共通して利用されているページ情報の操作、コマンドをここに定義しておくこと

で、テスト作成時にどの画面からも操作・アサーション・スクリプト・ページテストテンプレート、コマンドを設定することができます。

本ページで設定された操作等は、テスト編集の追加ダイアログでは「Common」と表示されます。

Fig. 2.1.3 全プロジェクト共通で設定された操作

2. ホーム メニュー 目次 | II. 各画面の説明 目次 | 使い方マニュアル 目次 に戻る

3. フロー メニュー

3.1 フロー/ビューセット編集 画面

フロー/ビューセット編集画面は、収集したWebアプリの操作フローを一覧表示する画面です。
収集した操作はフローグラフ図として表示されます。
1回の操作の収集を1セッションとし、フローはセッション単位で管理されています。
どんな操作のセッションかを見分けるために、セッションに名前を付けることができます。
また、収集した操作からテストを作成することができます。

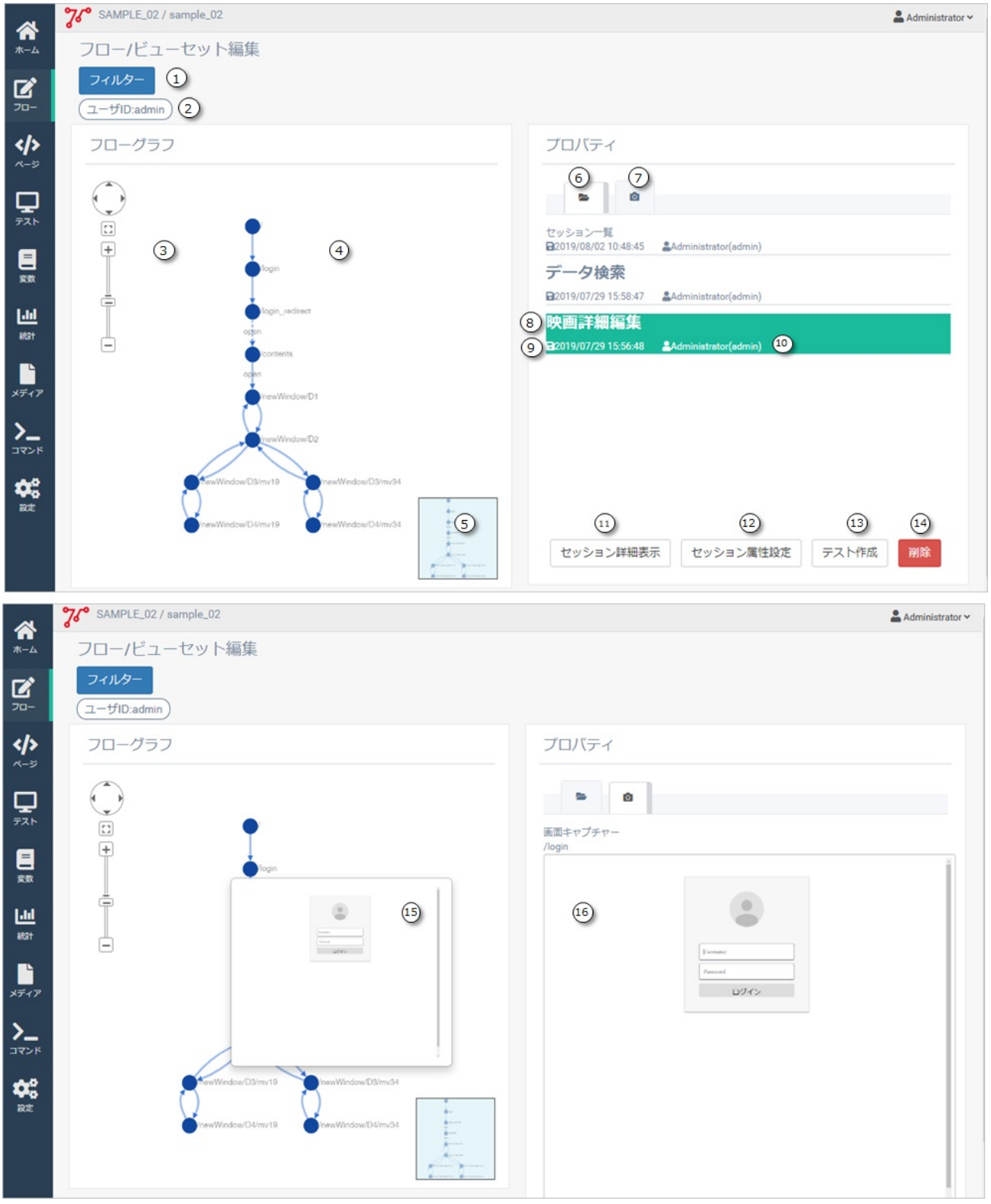


Fig. 3.1 フロー/ビューセット編集 画面

No.	名称	説明
1	フィルター ボタン	フィルター設定ダイアログを表示します。
2	フィルター表示	フィルター設定ダイアログで設定した条件を表示します。
3	フローグラフ操作スライダー	フローグラフの表示を操作します。 十字ボタンで図の上下左右の移動をします。 デフォルトポジションボタンで初期位置に戻せます。 スライダーの上下、＋/ーボタンで拡大縮小ができます。
4	フローグラフ	収集した操作のフローグラフを表示します。 初期表示はすべての操作を繋げた全体フローグラフです。 プロパティビューからセッションを選択すると、 各セッション単位の操作フローを表示します。
5	フローグラフ俯瞰図	現在のフローグラフの表示が、 全体のフローグラフのどの部分かを表示します。
6	セッション一覧タブ	二つのタブでプロパティビューの セッション一覧と画面キャプチャーの表示を切り替えます。
7	画面キャプチャータブ	
8	セッション名	セッション属性設定で設定したセッション名を表示します。
9	セッション実行日時	セッションが作成された日時を表示します。
10	セッション実行ユーザ	セッションを作成したユーザを表示します。
11	セッション詳細表示 ボタン	セッション詳細表示ダイアログを表示します。
12	セッション属性設定 ボタン	セッション属性設定ダイアログを表示します。
13	テスト作成 ボタン	テスト作成ダイアログを表示し、 選択したセッションからテストを作成します。
14	削除 ボタン	選択したセッションを削除します。 一度削除したセッションは復元できません。
15	フローグラフ画面サムネイル画像	フローグラフの各フロー(青い丸)をクリックすると、 フローの横および画面キャプチャータブに その画面のキャプチャーが表示されます。
16	画面キャプチャー	

3.1.1 セッション詳細表示 ボタン

セッション詳細表示ダイアログを表示します。

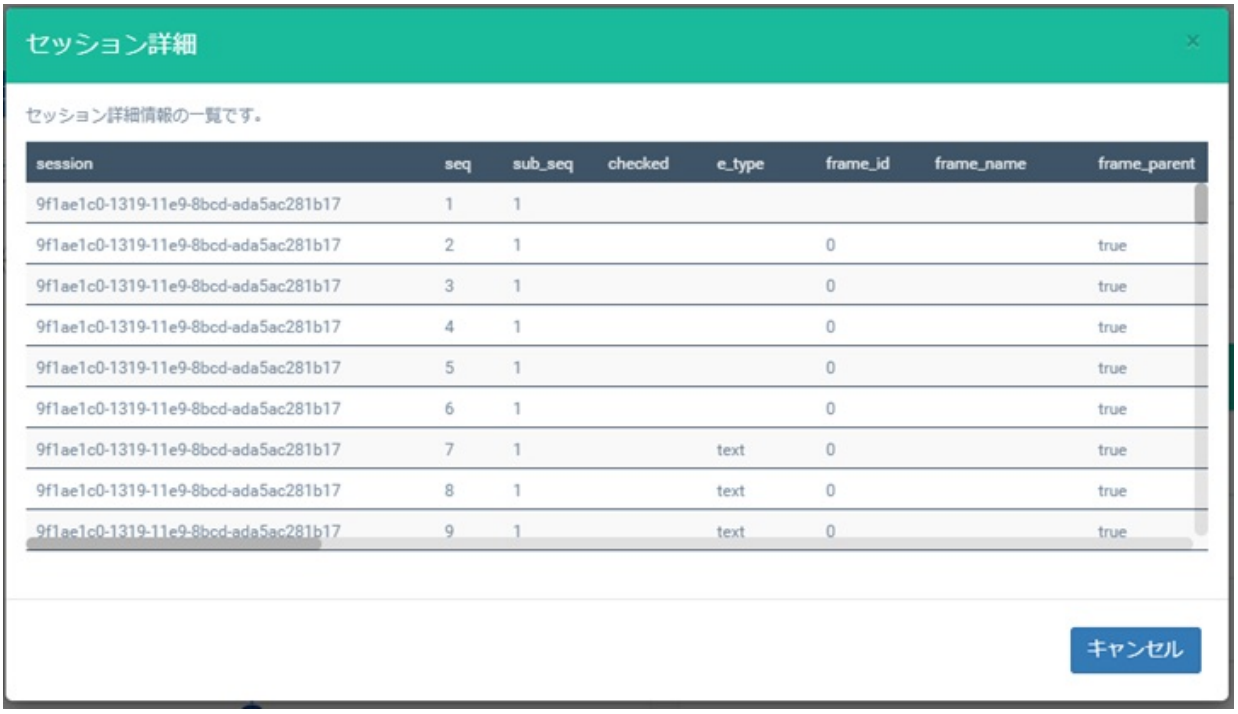


Fig. 3.1.1 セッション詳細表示ダイアログ

3.1.2 セッション属性設定 ボタン

セッション属性設定ダイアログを表示します。
選択したセッションに名称を設定することができます。

Fig. 3.1.2 セッション属性設定ダイアログ

3.1.3 テスト作成 ボタン

テスト作成ダイアログを表示して、選択したセッションからテストを作成します。
作成するテストを配置するフォルダ階層を選択し、テスト名を設定して**作成**ボタンをクリックすると、テスト編集画面に移動します。
なお、テスト名には以下の文字を含むことができませんのでご注意ください。

```
\ / : * ? " < > | () [] {} & $ ` ^ ~
```

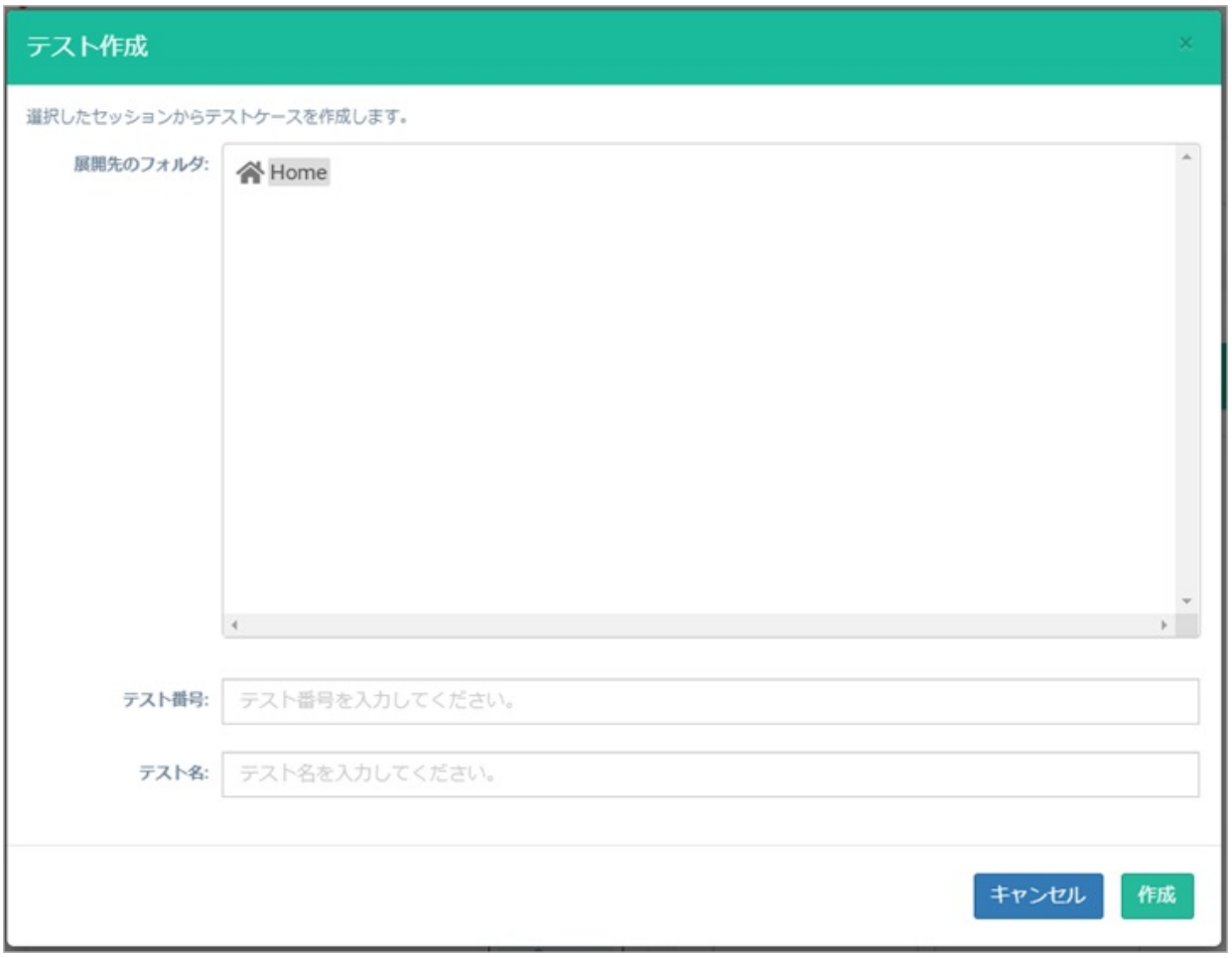


Fig. 3.1.3 テスト作成ダイアログ

3.1.4 フィルター ボタン

フィルターボタンをクリックすると、フィルター設定ダイアログを開きます。
ダイアログで日時、操作ユーザ、セッション名を選択して、表示するセッションを絞り込むことができます。
設定したフィルター条件は、フロー/ビューセット編集 画面のフィルターボタンの横に表示されます。

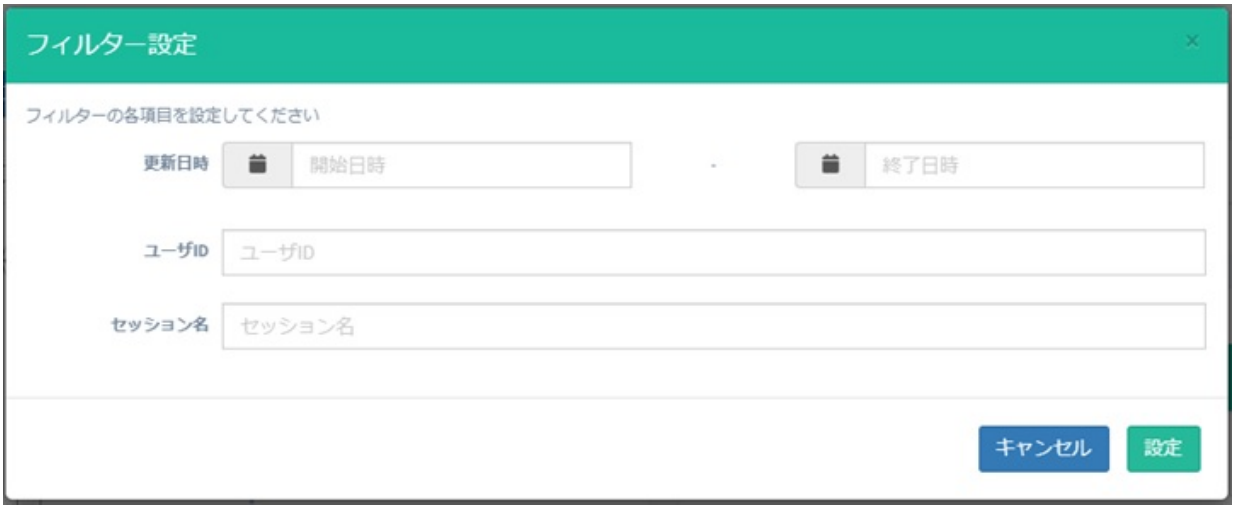


Fig. 3.1.4 フィルター設定ダイアログ

3. フロー メニュー 目次 | II. 各画面の説明 目次 | 使い方マニュアル 目次 に戻る

4. ページ メニュー

4.1 ページ一覧 画面

ページ一覧画面は、対象 Webアプリのページを一覧表示する画面です。
各ページ名はリンクになっており、ページごとの詳細を設定する画面に移動できます。
この画面から、手動で新たにページを追加することもできます。

- 4.1.1 新規作成ボタン
- 4.1.2 全ページ共通について

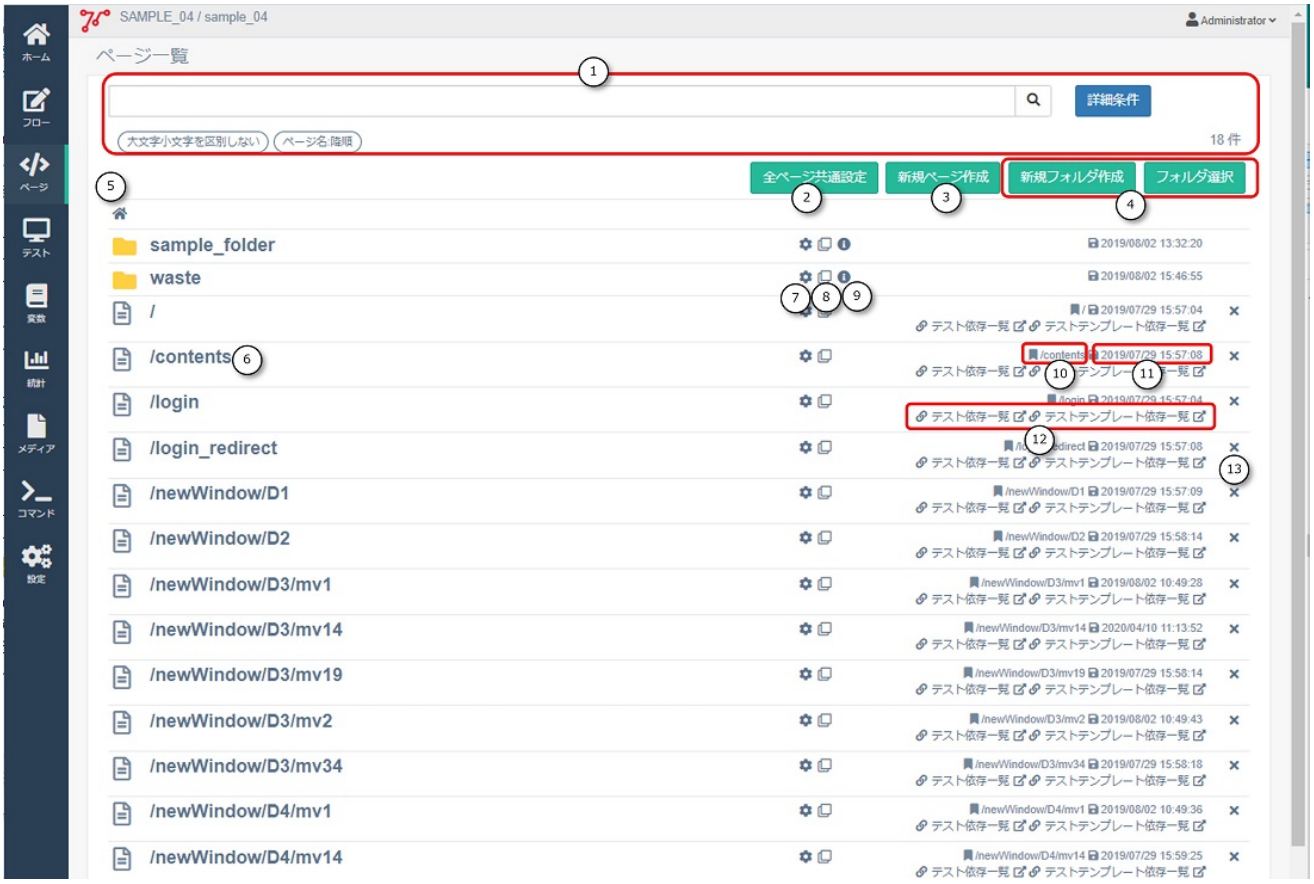


Fig. 4.1 ページ一覧画面

No.	名称	説明
1	一覧検索・ソート	一覧の表示内容の絞り込みと検索とその結果表示です。 詳しくは、1.各画面共通»1.3 一覧表示での検索・ソート機能 を参照してください。
2	全ページ共通設定ボタン	全ページに共通する操作を設定することができます。 クリックすると、全ページ共通のページ基本情報画面に遷移します。
3	新規ページ作成ボタン	新規ページを作成することができます。
4	新規フォルダ作成ボタン フォルダ選択ボタン	ページ一覧でのフォルダの追加やフォルダ移動等に関するボタンです。 詳しくは、1.各画面共通»1.5 一覧表示でのフォルダ操作 を参照してください。
5	現在位置表示	現在開いているフォルダの階層を表示しています。 フォルダ名をクリックすることでその階層を開くことができます。

6	ページ名	ページ名を表示します。 クリックすると、そのページのページ基本情報画面に遷移します。
7	設定アイコン	ページやフォルダの設定、移動・複製を行うことができます。 詳しい操作方法は、1.各画面共通»1.5 一覧表示でのフォルダ操作 を参照してください。
8	移動・ 複製アイコン	
9	情報アイコン	
10	ページのURL	ページのURLを表示します。 URLは、 収集したWebアプリで最初に表示する画面をホームとしたときのパスで表示します
11	最終更新日	
12	依存一覧リンク	このページを使用しているテストの一覧画面を別ウィンドウで開きます。 詳しくは、1.各画面共通»1.4 依存一覧リンク を参照してください。
13	ページ削除ボタン	ページを削除することができます。

4.1.1 新規ページ作成ボタン

新規ページ作成ボタンをクリックすると、新規ページ作成ダイアログが表示され、ページを新たに追加することができます。

Fig. 4.1.1 新規ページ作成ダイアログ

新規作成ダイアログでは **ページキー** と **ページ名** を入力して、**作成して進む** ボタンをクリックします。
ページキーは操作定義やアサーション定義の設定、新規テスト作成の際に必要となります。
新規ページが作成され、ページ基本情報画面 が表示されます。

フレームを使用する チェックボックスにチェックが入っている場合、フレームセットのページ基本情報の新規作成画面が表示されます。

4.1.2 全ページ共通について

ページ一覧画面から、**全ページ共通(common)**と呼ばれる特殊なページを定義することができます。
全ページ共通 では、普通のページ情報と同様に、操作・アサーション・スクリプト・ページテストテンプレートを設定することができ、 それらは、テスト作成の際にどのページにも設定することができます。

複数の画面に共通して利用されている操作（ex.常駐メニューなど）をここに定義しておくことで、テス

ト作成時にどの画面からも操作やアサーション、スクリプトを設定することができます。

4.1 ページ一覧 画面 目次 / 4. ページ メニュー 目次 に戻る

4.2 ページ編集画面

ページ編集画面では、ページの詳細な情報を設定することができます。
この画面には 基本情報・ 操作・ アサーション・ スクリプト・ テンプレート の5つのタブがあり、 それぞれの設定画面に切り替えることができます。

ページ情報の詳細の作成にはいくつかの方法があります。
対象のWebアプリケーションが稼働している場合は、WebExtensionを利用した**操作キャプチャによりページ情報を自動生成**することができます。

操作キャプチャが実施できない場合(対象のWebアプリケーションがまだ稼働していない、等)でも、ページを新規に**手動で追加して登録**することができます。

ページの基本情報を手動作成した後、操作やアサーション等の情報を一つずつ登録するしていくうえで、これらを**一括して登録**する方法も用意されています。

画面のモックアップがある場合、そのHTMLファイルをアップロードすることで、インスペクション画面から操作やアサーションを登録することができます。
また、エクセルファイルを介して詳細の情報を一括してエクスポート/インポートすることもできます。

4.2.1 基本情報 タブ

ページ基本情報画面は、選択したWebアプリのページの基本情報を設定する画面です。

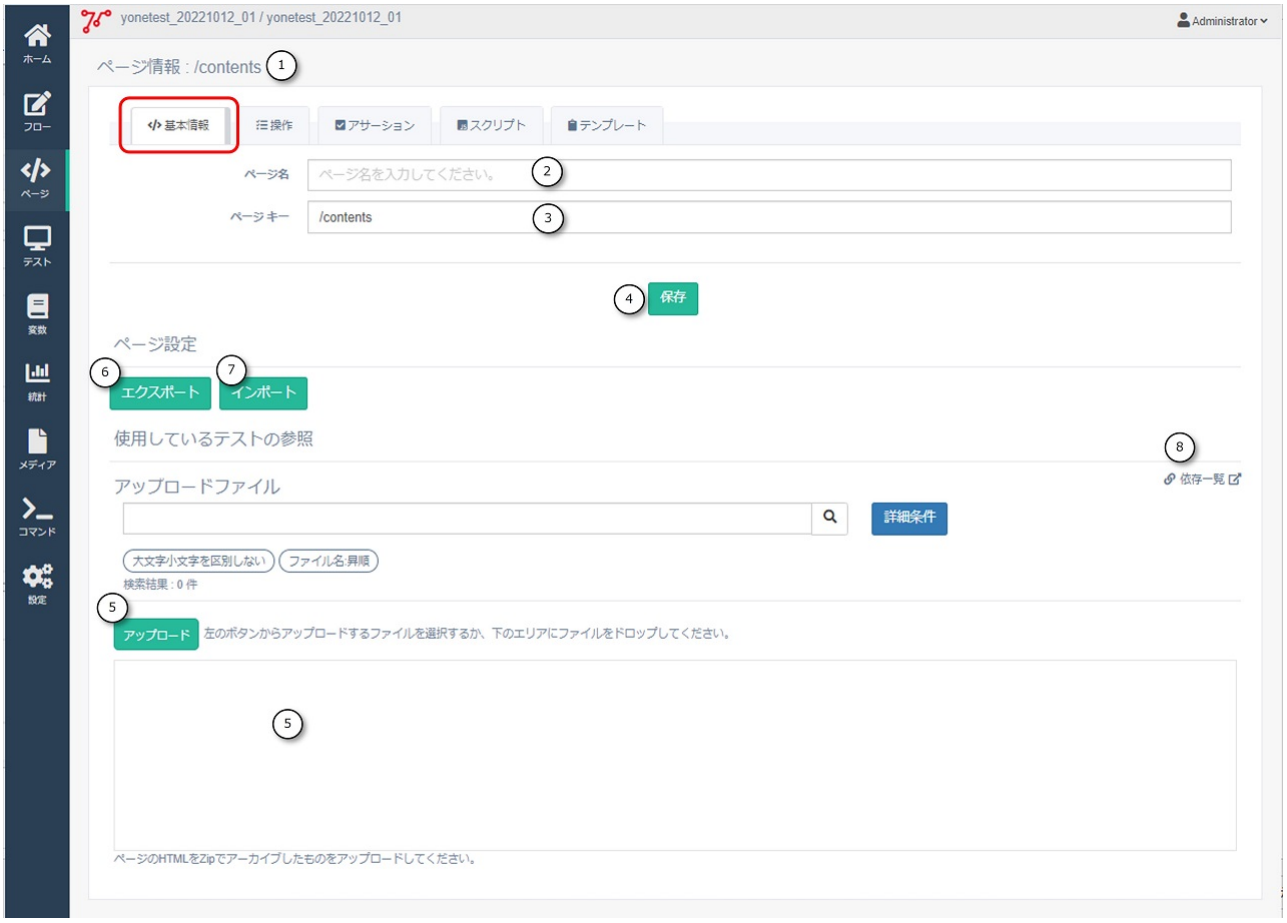


Fig. 4.2.1 ページ基本情報画面

No.	名称	説明
1	ページ名表示	現在選択しているページ名(未設定の場合は ページキー)を表示します。
2	ページ名	ページ名を設定することができます。
3	ページキー	ページのURLを表示します。 セッションログに存在するページキーは修正できません。
4	保存 ボタン	クリックすると、ページ基本情報の設定内容を保存します。
5	アップロードフォーム	HTMLをzipでアーカイブしたものをドラッグ& ドロップでアップロードできます。 アップロードボタンからファイルを選択してアップロードすることもできます アップロードしたものはインスペクション画面で使用します。
6	エクスポート ボタン	クリックするとページ設定をエクスポートすることができます。
7	インポート ボタン	クリックするとページ設定をインポートすることができます。
8	依存一覧リンク	このページを使用しているテストの一覧画面を別ウィンドウで開きます。 詳しくは、1.各画面共通»1.4 依存一覧リンク を参照してください。

4.2.1.1 ページ情報のファイルアップロード

ページ情報をアップロードすることによって、セッション情報の取得を行わなくても、ページ操作定義やアサーション定義の設定ができます。

(1) アップロードしたいHTMLファイルとCSSファイルは zipファイルに圧縮しておきます。

・HTMLファイルはzipのルート階層に設置してください。
・HTMLファイルとCSSフォルダをフォルダに入れたものzipファイルにするのではなく、
　HTMLファイルとCSSフォルダを直接zipファイルにしてください。
・フォルダ構造はHTMLファイルに記述されている構造で作成してください。
・複数のHTMLファイルがある場合は1ページずつHTMLファイルのみ読み込まれます。
・Javascriptの記述があっても適用できません。

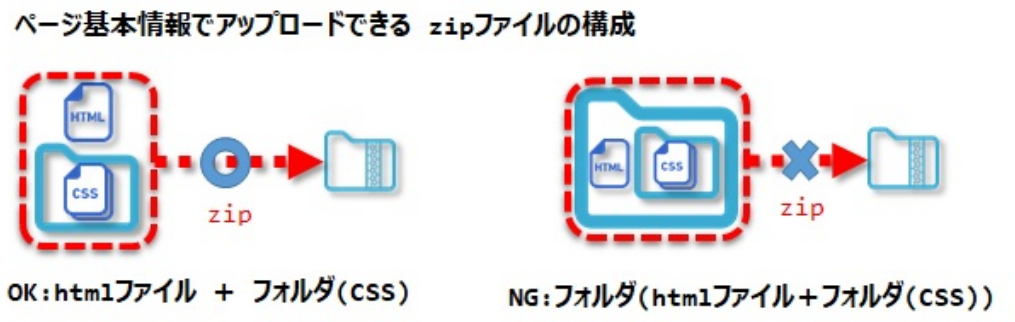


*Fig. 4.2.1.1-1 zip*ファイル構成

(2) アップロードしたいzipファイルを選択し、**ここにファイルをドロップしてください。** と表示されているエリアにドラッグ＆ドロップしてください。

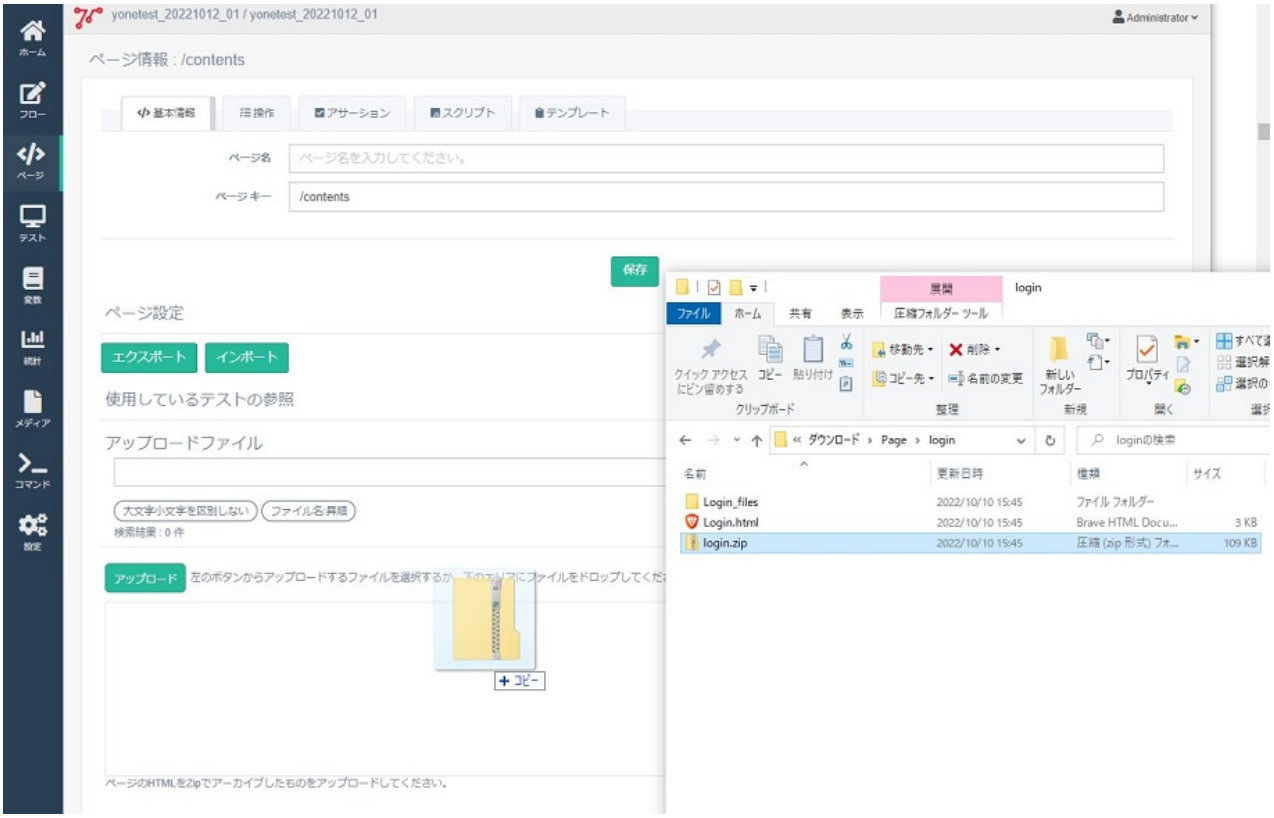


Fig. 4.2.1.1-2 ファイルドロップ

(3) ファイルがアップロードされていれば成功です。

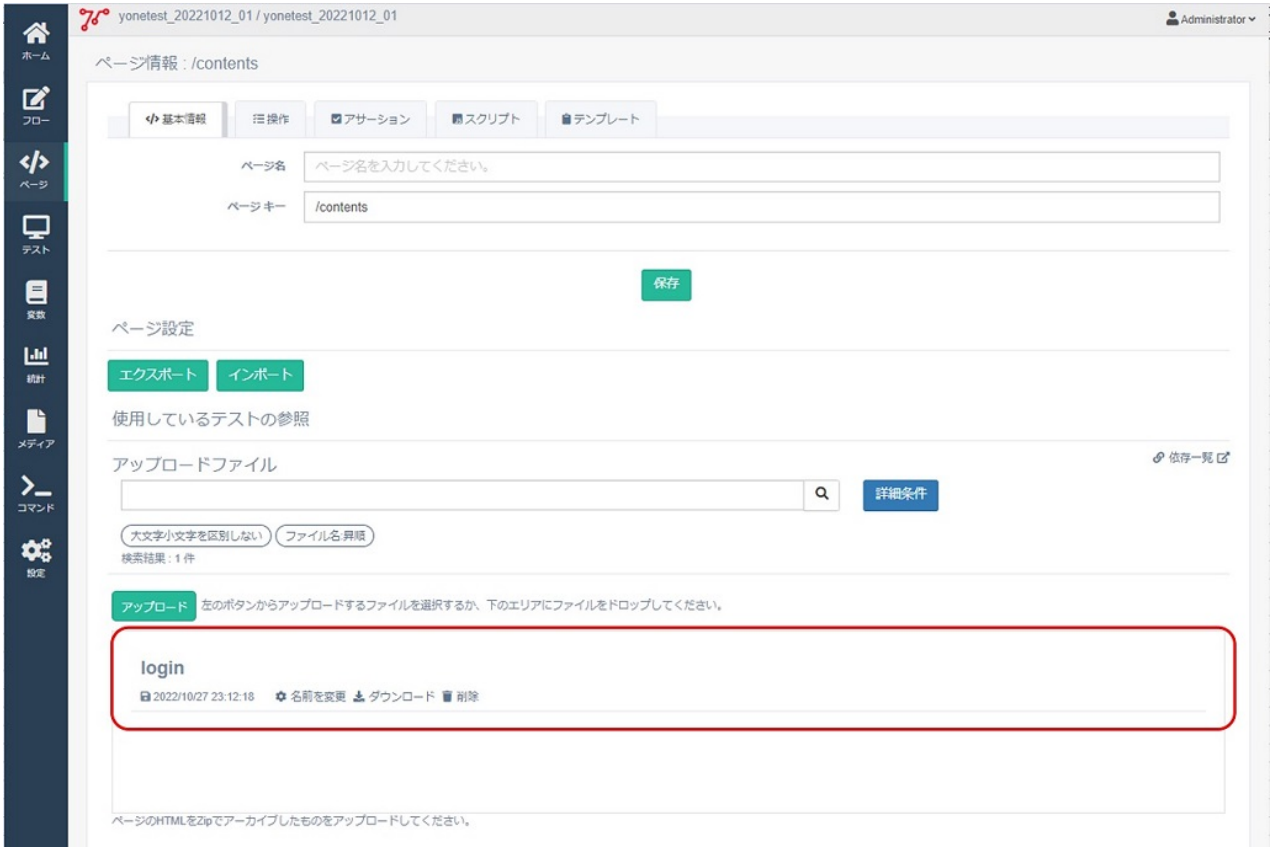


Fig. 4.2.1.1-3 アップロード成功

アップロードしたファイルは ページ操作やアサーション定義を作成する際の**画面から追加**ボタンで表示されるマルチインスペクション画面やインスペクション画面で使用します。
インスペクション画面の使い方については、4.2.6 インスペクションを参照してください。

4.2.1.2　ページ設定のエクスポート/インポート

ページ情報を **ページ設定ファイル** としてエクスポートし、このファイルを編集してインポートすることで、ページ情報を一括して編集・更新ができます。

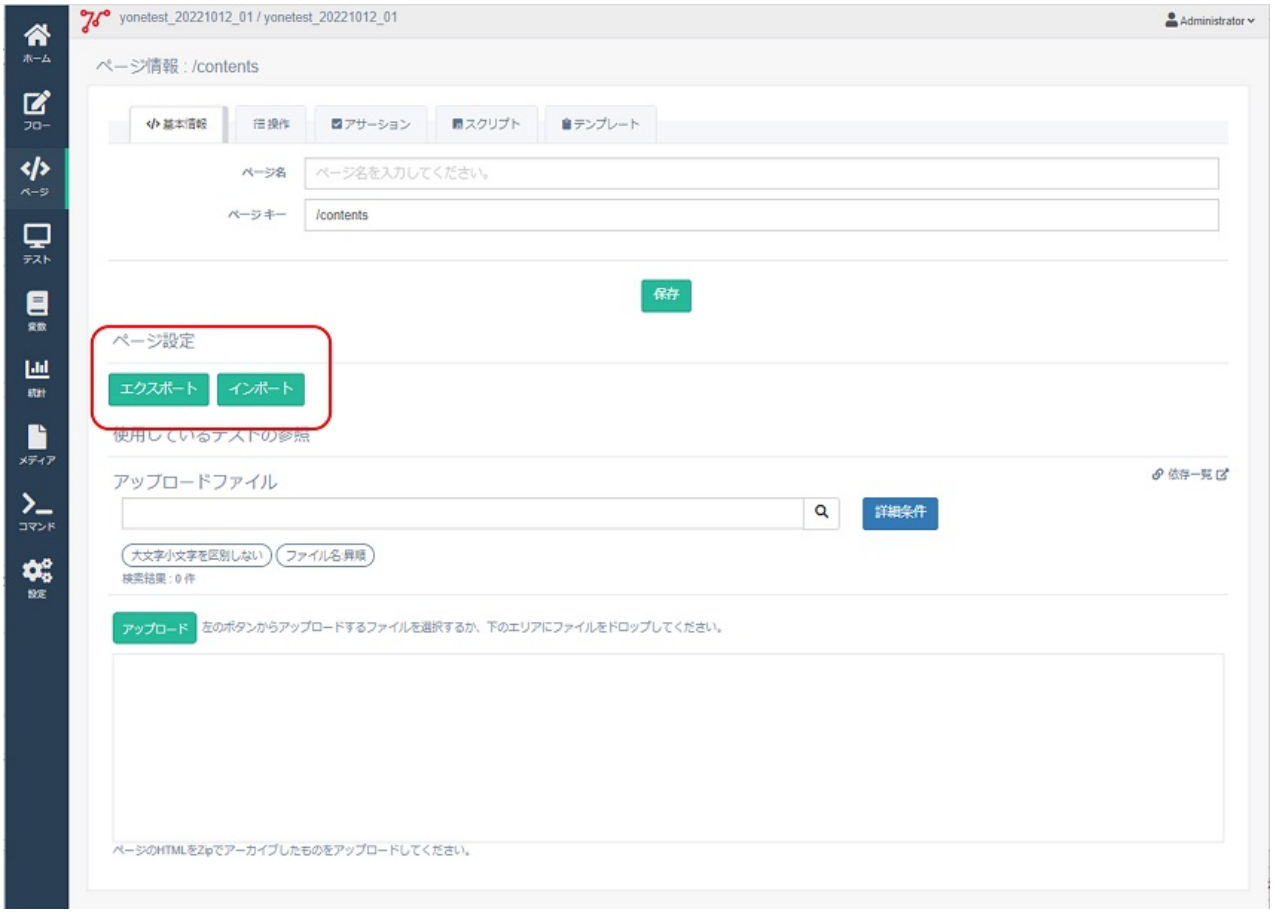


Fig. 4.2.1.2 ページ設定ファイルのインポート・エクスポート

ページ設定ファイルでは、「基本情報」の更新および「操作」「操作前の流れ」「操作後の流れ」「アサーション」の新規追加・更新することができます。
ページ設定ファイルのインポートでは、ページそのものを新規作成することはできません。

ページ設定ファイルの**ファイル形式・出力内容・編集要領**については、別マニュアル：**Testablish 入出力ファイル仕様** の **3. Testablish ページ設定ファイル** を参照してください。

4.2.1.3　フレームページの設定

新規ページ作成ダイアログで「フレームを使用する」のチェックボックスをONにして「作成して進む」と、**フレームセット**を設定する基本情報画面が新規作成されます。

フレームセット の基本情報画面では、ウインドウをいくつかのフレームに分割する設定と、それぞれの **フレーム** に読み込む **ページ** の指定を行います。

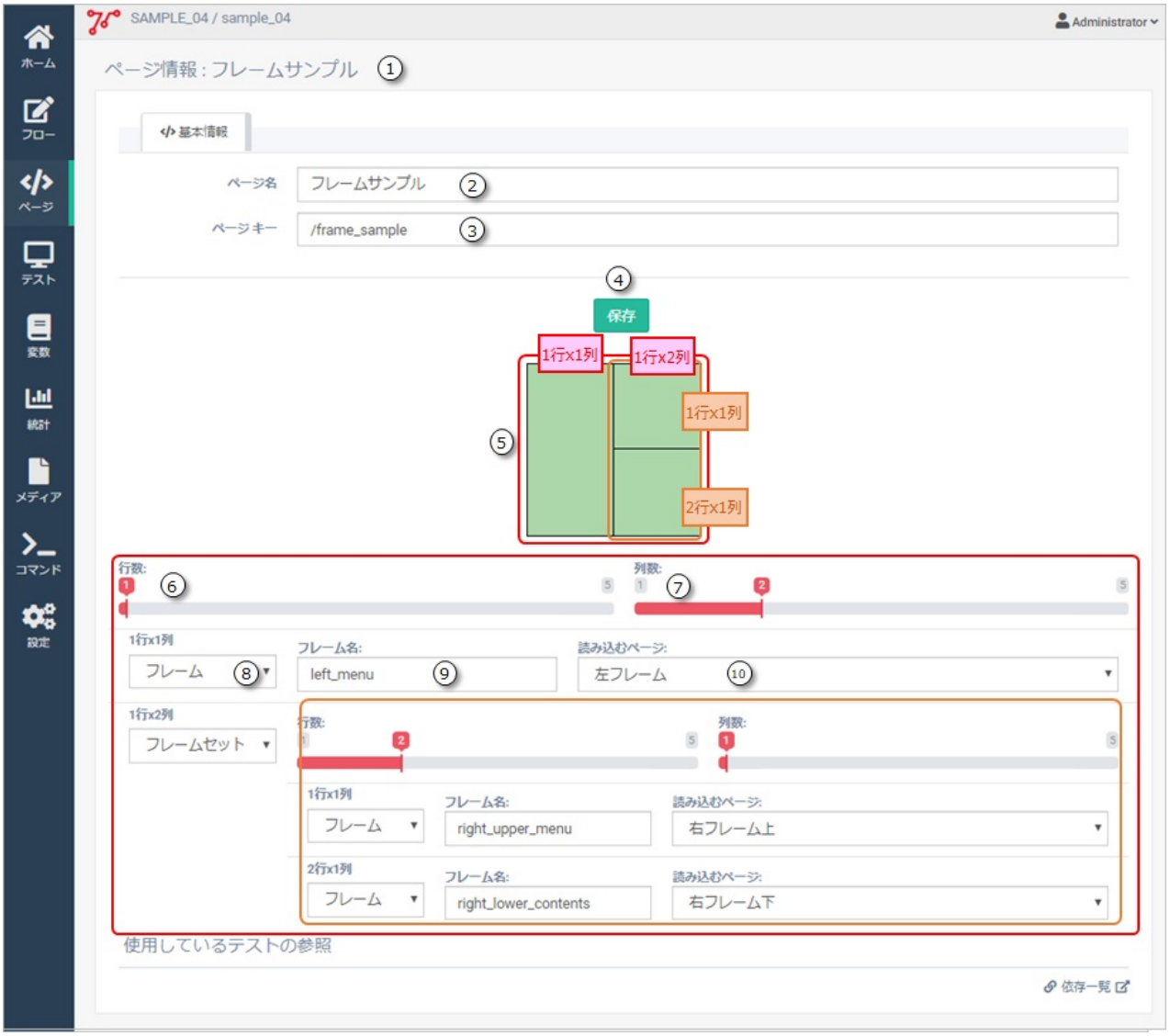


Fig. 4.2.1.3 フレームページの基本情報画面

No.	名称	説明
1	ページ名表示	現在選択しているページ名を表示します。
2	ページ名	フレームセットページ名を設定します。
3	ページキー	ページのURLを表示します。 セッションログに存在するページキーは修正できません。
4	保存 ボタン	クリックすると、ページ基本情報の設定内容を保存します。
5	フレーム分割図	設定されたフレームの分割状況を図示します。
6	行数 スライドバー	このフレームセットで分割する行数をスライドバーで指定します。
7	列数 スライドバー	このフレームセットで分割する列数をスライドバーで指定します。
8	フレームタイプ プルダウン	フレーム名を設定します。
9	フレーム名	表示されるプルダウンメニューでフレームセット/ フレームを選択します。
10	読み込むページ プルダウン	フレームに読み込むページ名をプルダウンから選択します。 読み込むページはあらかじめ作成しておく必要があります。

4.2 ページ編集 画面 目次 / 4. ページ メニュー 目次 に戻る

4.2.2 操作 タブ

操作タブをクリックすると、選択しているページの操作一覧を表示します。

4.2.2.1 操作一覧 画面

ページ操作一覧画面は、選択したページで行う操作を一覧表示する画面です。
この画面で、選択したページにどのような操作が行われるかを確認することができます。
この画面からは、ページで行う各操作の操作編集画面に移動することができます。

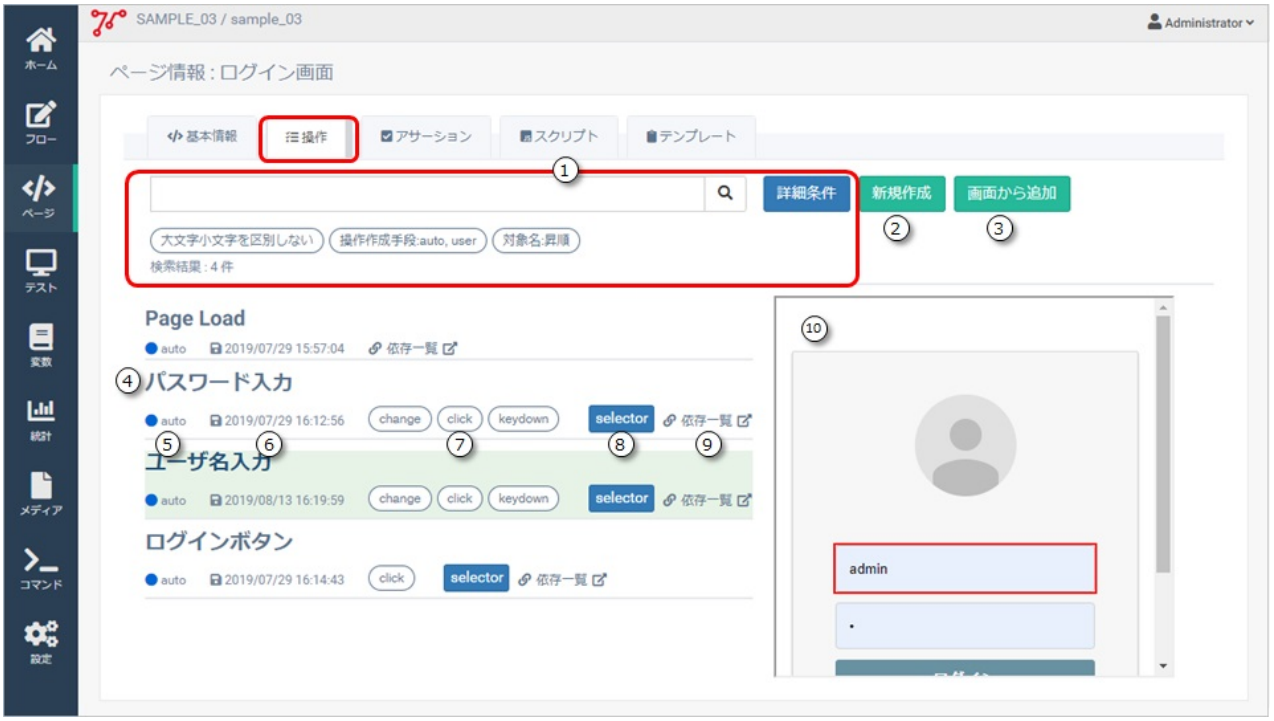


Fig. 4.2.2.1 操作一覧画面

No.	名称	説明
1	一覧検索・ソート	一覧の表示内容の絞り込みと検索とその結果表示です。 詳しくは、1.各画面共通»1.3 一覧表示での検索・ソート機能 を参照してください。
2	新規作成 ボタン	操作を新規追加します。
3	画面から追加 ボタン	操作をマルチインスペクション画面を使って新規追加します。
4	操作名	操作の対象名を表示します。 対象名が未設定の場合はセレクタを表示します。 クリックすると、操作編集画面へ移動します。
5	操作作成手段	操作がどの方法(auto/user)で作成されたかを表示しています。 操作の収集(キャプチャ)で自動作成された場合、autoが表示されます。 新規作成ボタンからユーザが作成した場合、userが表示されます。
6	最終更新日	
7	操作タイプ	設定されている操作のタイプを表示します。
8	セレクタボタン	クリックすると、10の 操作画面ビューで操作に対応する要素を枠で囲って表示します。

9	依存一覧リンク	この操作を使用しているテストの一覧画面を別ウィンドウで開きます。 詳しくは、1.各画面共通»1.4 依存一覧リンク を参照してください。
10	操作画面ビュー	操作の収集で登録されたページの場合、 直近でキャプチャした画面が表示されます ページの画面キャプチャが存在しない場合、このビューは表示されません。

4.2.2.2 操作編集 画面

ページ操作編集画面は、選択した操作を編集する画面です。
この画面からは、テストを自動実行する際の操作の対象と、操作の内容を設定できます。

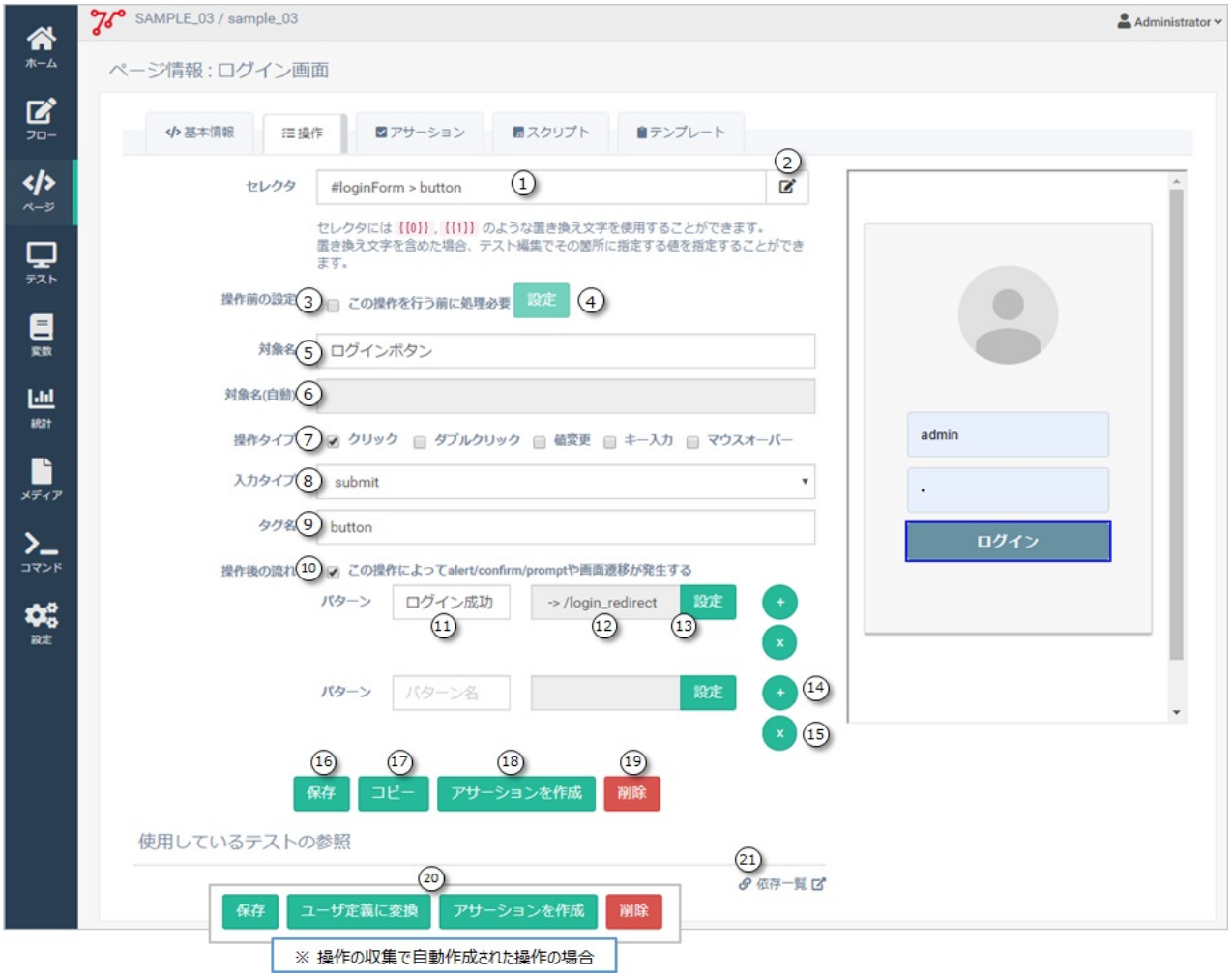


Fig. 4.2.2.2 操作編集画面

No.	名称	説明
1	セレクタ	操作を設定する画面要素を入力します。必須入力です。 セレクタボタンから設定することができます。
2	セレクタボタン	インスペクション画面のウィンドウを開きます。 このボタンから、操作の対象にする html要素を選択することができます。
3	操作前の設定 チェックボックス	この操作を行う前に必要な処理がある場合、チェックしてください。 チェックすると操作前の設定ボタンがクリックできるようになります。

		操作前の処理の詳細については、(1)操作前の処理の設定 を参照してください。
4	操作前の設定 ボタン	操作前の設定ダイアログを表示します。 この操作の前の処理を設定してください。
5	対象名	操作の名前を入力してください。
6	対象名(自動)	自動収集された操作の場合、自動で付けられた名前を表示します。 この名前は変更できません。
7	操作タイプ	この操作の操作タイプをチェックボックスから選択してください。 必須選択です。
8	入力タイプ	この操作の入力タイプを選択してください。
9	タグ名	操作対象のhtmlタグを入力してください。
10	操作後の流れ チェックボックス	この操作によってダイアログの表示、 または画面の移動が発生する場合、 「この操作によってalert/confirm/promptや画面遷移が発生する」 をチェックしてください。 操作後の流れの詳細については、(2)操作後の流れの設定 を参照してください。
11	パターン名	この操作のあとに発生する処理に名前を付けてください。 操作後の処理設定ダイアログのパターン名になります。
12	操作後の処理設定内容	パターン設定ボタンで表示されるダイアログで設定した 操作後の処理内容を表示します。
13	パターン設定ボタン	操作後の処理設定ダイアログを表示します。 発生する処理の種類や画面遷移を選択してください。
14	パターン追加ボタン	パターンを追加します。 この操作のあとに発生する処理が複数ある場合、 パターンを追加してください。
15	パターン削除ボタン	追加したパターンを削除します。
16	保存ボタン	操作の編集を保存します。
17	コピーボタン	ユーザ定義の操作の場合に表示されます。 編集中の操作のコピーを作成し、表示します。
18	アサーションを作成ボタン	同じ要素のセレクタで新しく手作業でアサーションを作成します。
19	削除ボタン	操作を削除します。
20	ユーザー定義に変換ボタン	自動収集された操作の場合に表示されます。 操作をユーザーが追加した操作に変換して、編集できるようにします。 ただし、変換すると自動収集した状態には戻りません。
21	依存一覧リンク	この操作を使用しているテストの一覧画面を別ウィンドウで開きます。 詳しくは、1.各画面共通»1.4 依存一覧リンク を参照してください。

(1) 操作前の処理の設定

Testablishでは**最後に使った要素の情報**を操作として設定します。
一連の複数の事前操作を経て項目を選択する場合、最後の項目選択に至るまでの操作を **操作前の処理** として設定しておかなければなりません。

例） 階層を辿って選択するメニューの場合、最後に使った要素（下記の例では"**京都府をクリック**"）を操作として設定します。 **京都府をクリック** するために辿った **近畿** については **操作前の処理** として設定します。

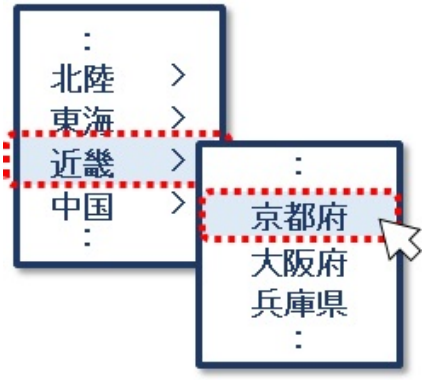


Fig. 4.2.2.2.1-1 操作前の処理概要

ページ操作編集画面で、対象の操作の 「**この操作を行う前に処理必要**」 にチェックをいれます。
チェックをいれると、右の **設定** ボタンが押せるようになります。
設定ボタンをクリックすると、**操作前の処理設定ダイアログ** が開きます。

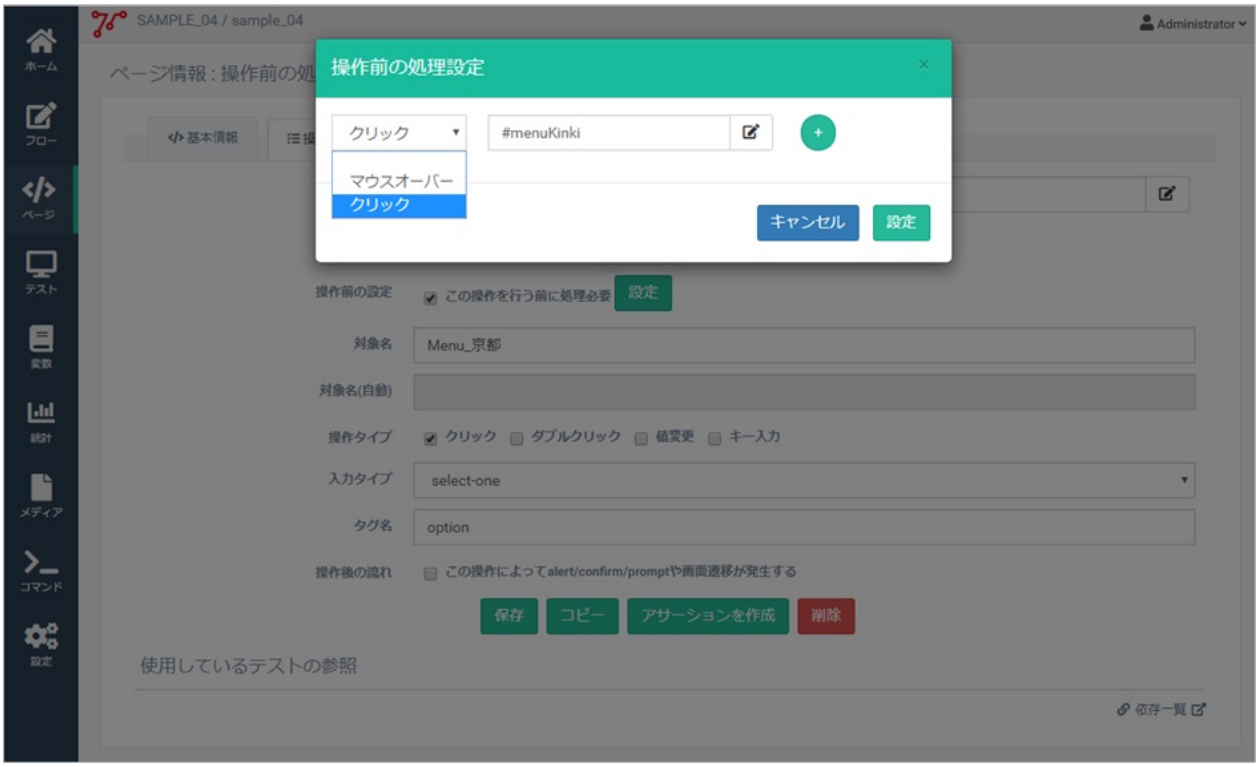


Fig. 4.2.2.2.1-2 操作前の処理設定ダイアログ

操作前の処理設定ダイアログが表示されますので、**操作タイプ、セレクタ** を設定してください。
- 操作タイプは **クリック** と **マウスオーバー** から選択できます。
- セレクタには操作対象のCSSセレクタを指定してください。
- 設定した処理は上から順に実行されます。

設定ボタンをクリックして設定を保存します。操作編集画面へ戻ります。

この設定により自動テストを実行したときに設定した処理が行われますので、テスト編集での追加の設定は不要です。

(2) 操作後の流れの設定

画面遷移が発生する操作は、ページ操作編集画面で **操作後の流れ** を設定します。
アラートダイアログや確認ダイアログなどを表示する場合もここから設定できます。

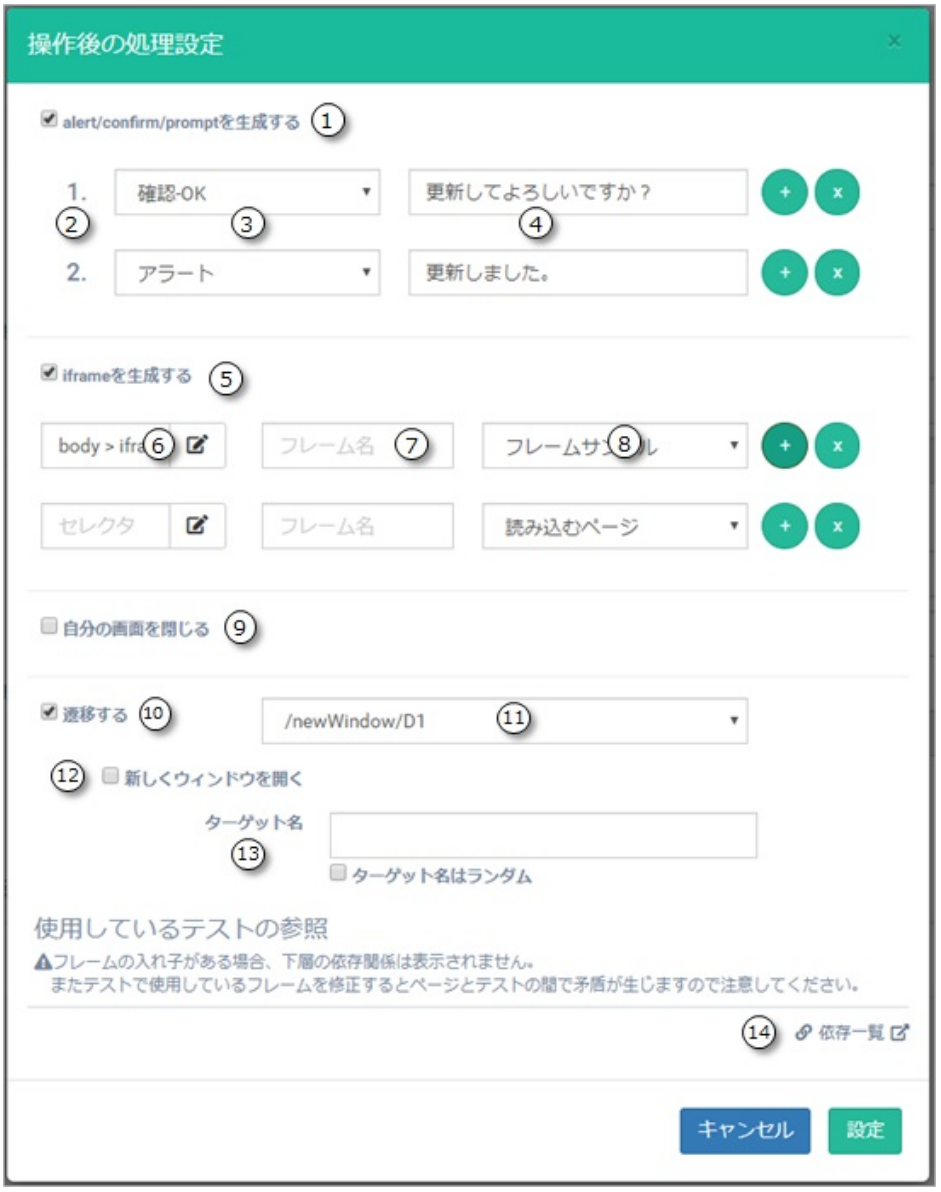


Fig. 4.2.2.2.2-1 操作後の処理設定ダイアログ

No.	名称	説明
1	alert/confirm/prompt を生成する チェックボックス	チェックすることで、ダイアログに関する設定ができるようになります。
2	ダイアログ番号	alert/confirm/prompt が表示される順序です。
3	タイプ プルダウン	ダイアログのタイプを選択できます。
4	メッセージ	ダイアログに表示するメッセージの文言を指定します。
5	iframeを生成する チェックボックス	チェックすることで、ページに読み込む iframe の設定ができるようになります。

6	セレクタ	読み込む iframe のセレクタ・フレーム名を指定します。 セレクタ または フレーム名のどちらかの入力が必須です。
7	フレーム名	
8	ページ名	iframe へ読み込むページをプルダウンから選択できます。
9	自分の画面を閉じる チェックボックス	操作後に自身のウィンドウを閉じる場合、ここをチェックします。
10	遷移する チェックボックス	チェックすることで、**遷移先画面**プルダウンが選択できるようになります。
11	遷移先画面 プルダウン	遷移先の画面をプルダウンから選択できます。
12	新しくウィンドウを開く チェックボックス	チェックするとターゲット名の設定ができるようになります。
13	ターゲット名	新しく開くウィンドウのターゲット名を指定します。 特に指定する必要がない場合は **ターゲット名はランダム** チェックボックスをチェックしてください。
14	依存一覧リンク	この操作後の処理を使用しているテストの一覧画面を別ウィンドウで開きま 詳しくは、1.各画面共通»1.4 依存一覧リンク を参照してください。

タイプ で選択できる種類は以下の通りです。それぞれ、メッセージの文言を設定できます。

・アラート
・確認-OK
・確認-キャンセル
・プロンプト-OK
・プロンプト-キャンセル

Fig. 4.2.2.2.2-2 操作後の処理：タイプ設定

必要な設定ができたら、 **設定** ボタンをクリックしてダイアログを閉じます。
操作編集画面に戻りますので、設定を保存してください。

操作後の処理は複数設定することができます。
遷移先を複数設定した場合には、テスト編集の際に遷移先を選択して指定することができます。

4.2 ページ編集 画面 目次 / 4. ページ メニュー 目次 に戻る

4.2.3 アサーション タブ

アサーション タブはテストに用いるアサーション(検証)を定義する画面です。
アサーションがある場合は一覧が表示されますが、最初はアサーションがひとつもない状態です。

4.2.3.1 ページアサーション一覧 画面

ページアサーション一覧画面は、選択したページで定義したアサーションを一覧表示する画面です。
この画面では、選択したページで利用できるアサーションを確認することができます。
この画面からは、ページで行う各アサーションの編集画面に移動することができます。

Fig. 4.2.3.1 アサーション一覧画面

No.	名称	説明
1	一覧検索・ソート	一覧の表示内容の絞り込みと検索とその結果表示です。 詳しくは、1.各画面共通»1.3 一覧表示での検索・ソート機能 を参照してください。
2	新規作成 ボタン	アサーションを新規追加します。 クリックするとアサーション編集画面に移動します。
3	画面から追加ボタン	操作・アサーションを マルチインスペクション画面 を使って新規追加します。
4	アサーション名	ページに登録されたアサーション名を表示します。 クリックすると、アサーション編集画面へ移動します。
5	最終更新日	
6	依存一覧リンク	このアサーションを使用しているテストの一覧画面を別ウィンドウで開きます。 詳しくは、1.各画面共通»1.4 依存一覧リンク を参照してください。

4.2.3.2 ページアサーション編集 画面

ページアサーション編集画面は、選択したアサーション定義を編集する画面です。
この画面で、テストの際に利用できるアサーション定義の内容の変更ができます。

Fig. 4.2.3.2 アサーション編集画面

No.	名称	説明
1	セレクタ	アサーションを設定する画面要素を入力します。必須入力です。
2	セレクタボタン	インスペクション画面のウィンドウを開きます。 このボタンから、アサーションの対象にする html要素を選択する
3	対象名	アサーション定義の名前を入力します。必須入力です。
4	検証タイプ	アサーション定義に設定するチェックのタイプを選択します。
5	説明	アサーション定義の説明を必要に応じて入力します。
6	保存ボタン	アサーション定義の編集内容を保存します。
7	コピーボタン	アサーション定義の編集内容をコピーします。
8	削除ボタン	編集中のアサーション定義を削除します。
9	テスト依存一覧リンク	このアサーションを使用しているテストの一覧画面を別ウィンド 詳しくは、1.各画面共通»1.4 依存一覧リンク を参照してくださ
10	テストテンプレート依存一覧リンク	このアサーションを使用しているテストテンプレートの一覧画面 詳しくは、1.各画面共通»1.4 依存一覧リンク を参照してくださ

(1) セレクタ

セレクタには 要素のセレクタがを記載します。

セレクタには '{{1}}' や '{{2}}' のような置き換え文字を使用することができます。
置き換え文字を含めた場合、テスト編集でその箇所に指定する値を指定することができます。

(例) セレクタが「テーブルの2行目の1列目」のとき

```
#Product > table > tbody > tr:child(2) > td:child(1)
```

(例) テーブルの行数が可変、あるいは選択可能な列が複数列あるような場合。

```
#Product > table > tbody > tr:child({{1}}) > td:child({{2}})
```

置き換え文字については、正規表現で規則を定義することもできます。

(2) 検証タイプ

検証タイプ はアサーションで検証するタイプの定義です。複数が選択可能です。
インスペクション画面からアサーションの設定を行うと **検証タイプ** の初期設定としてチェックボックスにチェックが入っています。 設定は変更が可能です。

検証タイプ	説明
テキスト一致 テキストを含む テキスト不一致 テキストを含まない	htmlで表示されたテキストに対し指定したテキストが 一致する・一致しない、含まれる・含まれない
要素数一致	htmlの要素数が指定した数と一致する
値一致 値を含む 値不一致 値を含まない	inputなどにおけるvalueの値に対し指定した値が 一致する・一致しない、含まれる・含まれない
チェックされている チェックされていない チェック不確定	チェックボックスなどに、 チェックが入っている・入っていない・不確定
属性値一致 属性値を含む 属性値不一致 属性の指定あり 属性値を含まない 属性の指定なし	htmlタグの属性に対する属性値に対し指定した値が 一致する・しない、含まれる・含まれない、 属性の指定のあり・なし、属性そのものが指定されていない
表示されている 表示されていない	指定したhtmlの要素が画面に表示されている・表示されていない
存在する 存在しない	指定したhtmlの要素が存在する・しない

4.2 ページ編集 画面 目次 / 4. ページ メニュー 目次 に戻る

4.2.4 スクリプト タブ

対象のページにjavascriptを定義・登録しておくことで、テスト実行時の画面に対して、直接Javasctiptを実行させることができます。

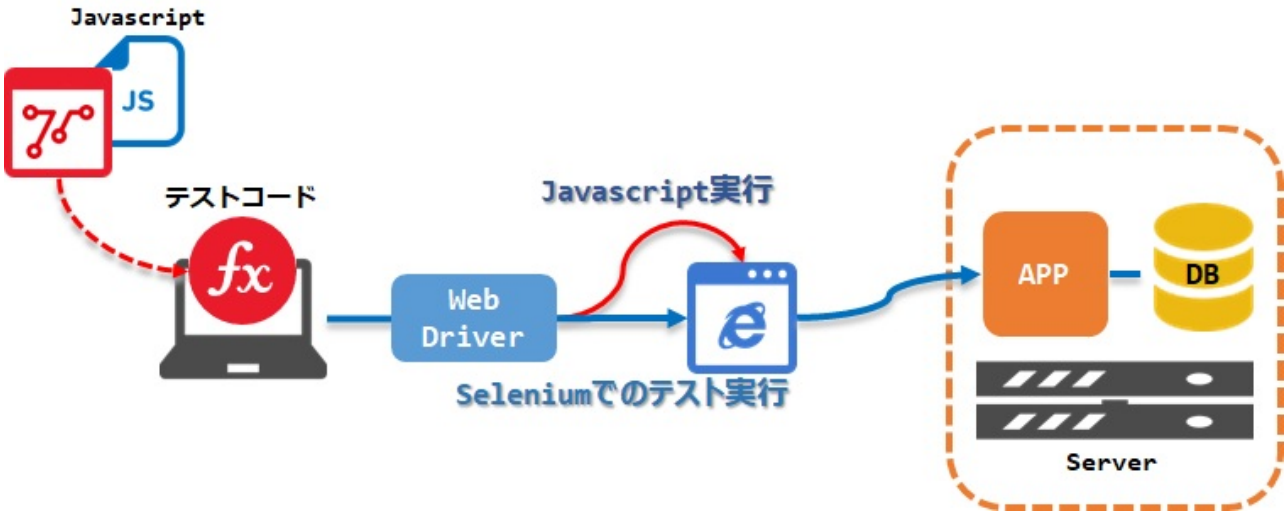


*Fig. 4.2.4 Javascript*実行機能概要

4.2.4.1 Javascript実行機能一覧 画面

Javascript実行機能一覧画面は、選択したページで定義したJavascriptコードのスクリプト名を一覧表示する画面です。
この画面では、選択したページで利用できるJavascriptコードを確認することができます。
この画面からは、ページで行う各Javascriptコードの編集画面に移動することができます。

*Fig. 4.2.4.1 Javascript*実行機能一覧画面

No.	名称	説明

1	一覧検索・ ソート	一覧の表示内容の絞り込みと検索とその結果表示です。 詳しくは、1.各画面共通»1.3 一覧表示での検索・ソート機能 を参照してください。
2	新規作成 ボタン	Javascriptコードを新規追加することができます。 クリックするとJavascriptコード編集画面に移動します。
3	スクリプト名	スクリプト名を表示します。 クリックするとJavascriptコード編集画面に移動します。
4	説明	スクリプトの説明を表示します。
5	最終更新日	
6	依存一覧リンク	このスクリプトを使用しているテストの一覧画面を別ウィンドウで開きます。 詳しくは、1.各画面共通»1.4 依存一覧リンク を参照してください。

4.2.4.2 Javascript実行機能編集 画面

Javascript実行機能編集画面は、選択したスクリプトを編集する画面です。
この画面からは、テストを自動実行する際にJavascriptを使って画面の操作を行うコード作成と、操作の内容を設定できます。

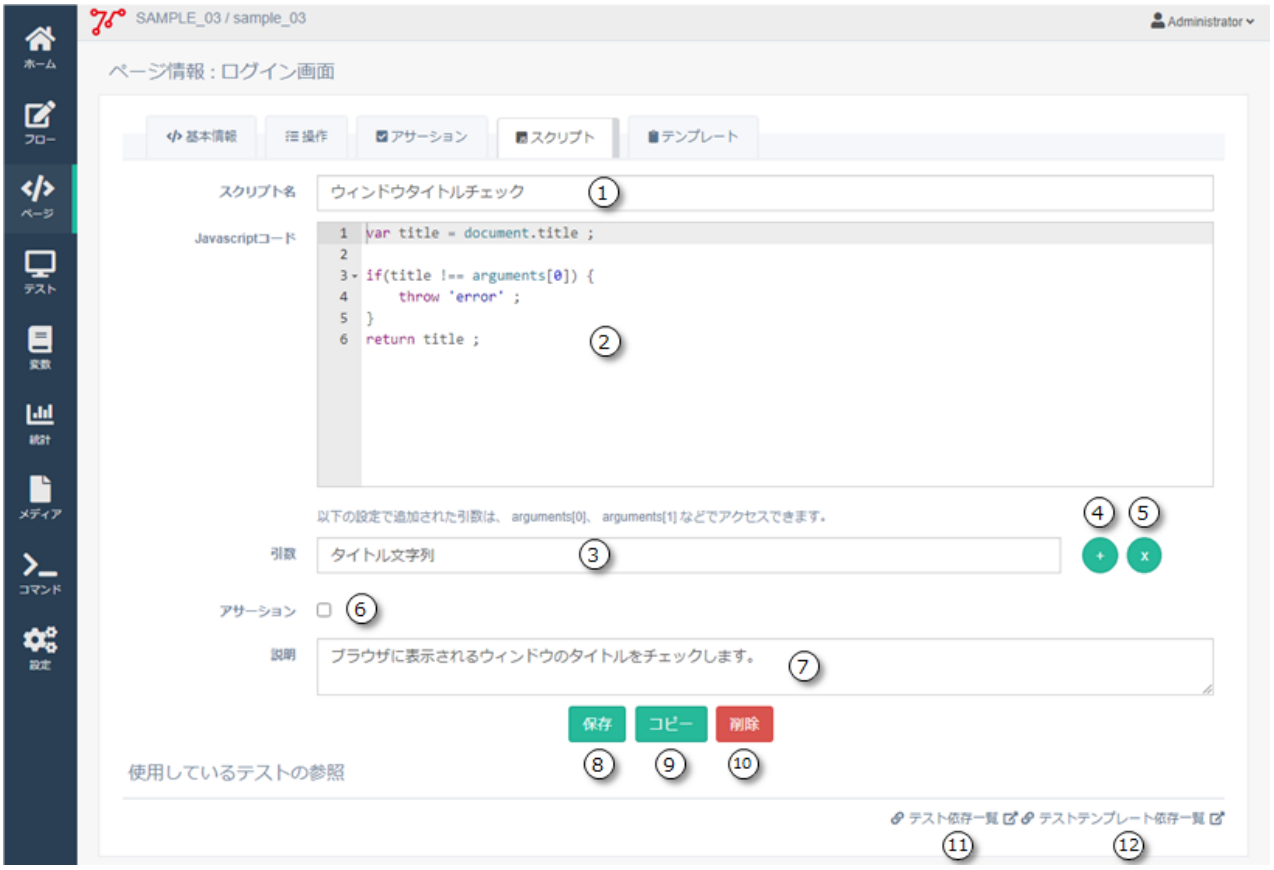


*Fig. 4.2.4.2 Javascript*実行機能編集画面

No.	名称	説明
1	スクリプト名	作成するJavascriptコードのスクリプト名を入力します。
2	Javascriptコード	実行するJavascriptのコードを入力します。

3	引数名	引数の論理名(値ではありません。)を入力します。 (実際の引数の値はテスト編集画面で入力します。) 設定した引数 arguments[1] などでアクセスできます。 初期設定では「引数は設定しない」と表示されています。
4	引数の追加 ボタン	引数の設定する場合、このボタンをクリックして追加してくださ
5	引数の削除 ボタン	追加した引数を削除します。
6	アサーション	作成したスクリプトをアサーションの色で表示します。
7	説明	作成したJavascriptコードの操作説明を入力してください。
8	保存ボタン	スクリプトの編集を保存します。
9	コピーボタン	スクリプトをコピーします。
10	削除ボタン	スクリプトを削除します。
11	テスト依存一覧リンク	このスクリプトを使用しているテストの一覧画面を別ウィンドウ 詳しくは、1.各画面共通»1.4 依存一覧リンク を参照してくださ
12	テストテンプレート依存一覧リンク	このスクリプトを使用しているテストテンプレートの一覧画面を 詳しくは、1.各画面共通»1.4 依存一覧リンク を参照してくださ

4.2.4.3 テストでのJavascriptの設定

作成したJavaScriptをテスト編集画面から設定します。
メインメニューの テスト からスクリプトを設定するテストを選択し、テスト編集画面を開きます。
ページ内のスクリプトを実行したいステップで、**下に操作を追加** ボタンをクリックして表示されるメニューから **スクリプト実行追加** を選択します。

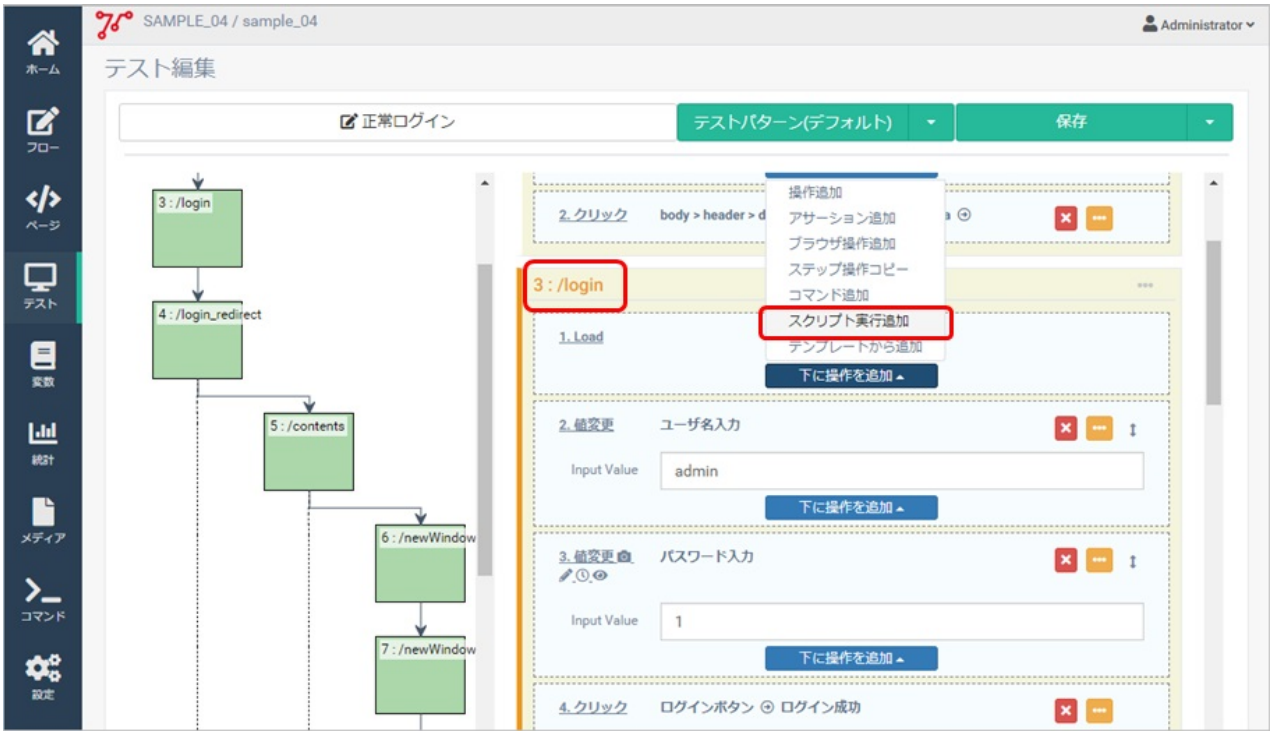


*Fig. 4.2.4.3-1 Javascript*テスト選択

表示されたスクリプト選択ダイアログから、追加するスクリプトを選択して**Add**ボタンをクリックし、

キャンセルボタンをクリックしてダイアログを閉じます。

*Fig. 4.2.4.3-2 Javascript*テストダイアログ

スクリプトが追加されているのを確認し、スクリプトに引数がある場合は、それらの値を設定していきます。

*Fig. 4.2.4.3-3 Javascript*テスト設定

4.2 ページ編集 画面 目次 / 4. ページ メニュー 目次 に戻る

4.2.5 テンプレート タブ

テストを作成する際、そのページで繰り返し設定する共通の操作やアサーション等があることがあります。
これらを事前にページテストテンプレートとして定義・作成しておくことができます。
定義したページテストテンプレートは、テスト編集画面で読み込んで利用することができます。

4.2.5.1 ページテストテンプレート一覧 画面

ページテストテンプレート一覧画面は、選択したページで定義したテストテンプレート名を一覧表示する画面です。
この画面からページテストテンプレートの定義の追加、変更を行います。
ページテストテンプレート名をクリックすると、ページに定義されている各テストテンプレートの編集画面に移動することができます。

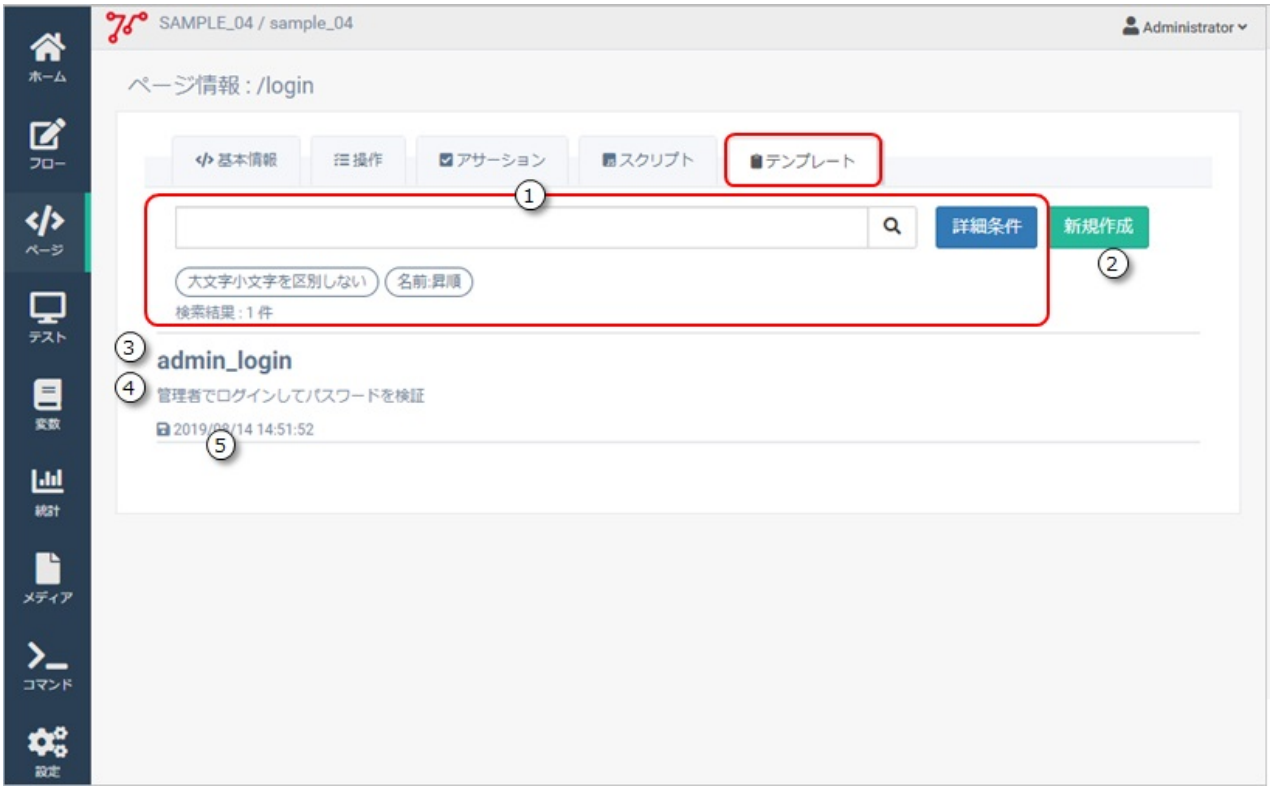


Fig. 4.2.5.1 ページテストテンプレート一覧 画面

No.	名称	説明
1	一覧検索・ソート	一覧の表示内容の絞り込みと検索とその結果表示です。 詳しくは、1.各画面共通»1.3 一覧表示での検索・ソート機能 を参照してください。
2	新規作成 ボタン	ページテストテンプレートを新規追加します。 クリックすると新規テンプレート作成ダイアログを表示します。
3	ページテストテンプレート名	ページテストテンプレート名を表示します。
4	説明	ページテストテンプレートの説明を表示します。 ページテストテンプレート編集画面のページテストテンプレート名をクリックして表示されるテンプレート情報編集ダイアログで設定できま
5	最終更新日	

(1) 新規作成ボタン

新規作成ボタンをクリックすると、新規テンプレート作成ダイアログが表示され、ページテストテンプレートを新たに追加することができます。

Fig. 4.2.5.1.1 新規テンプレート作成ダイアログ

新規作成ダイアログでは ページテストテンプレートの **名前** と **説明** を入力し、**作成して進む** ボタンをクリックします。
新規ページが作成され、ページテストテンプレート編集画面が表示されます。

4.2.5.2 ページテストテンプレート編集 画面

ページテストテンプレート編集画面は、選択したページテストテンプレートを編集する画面です。

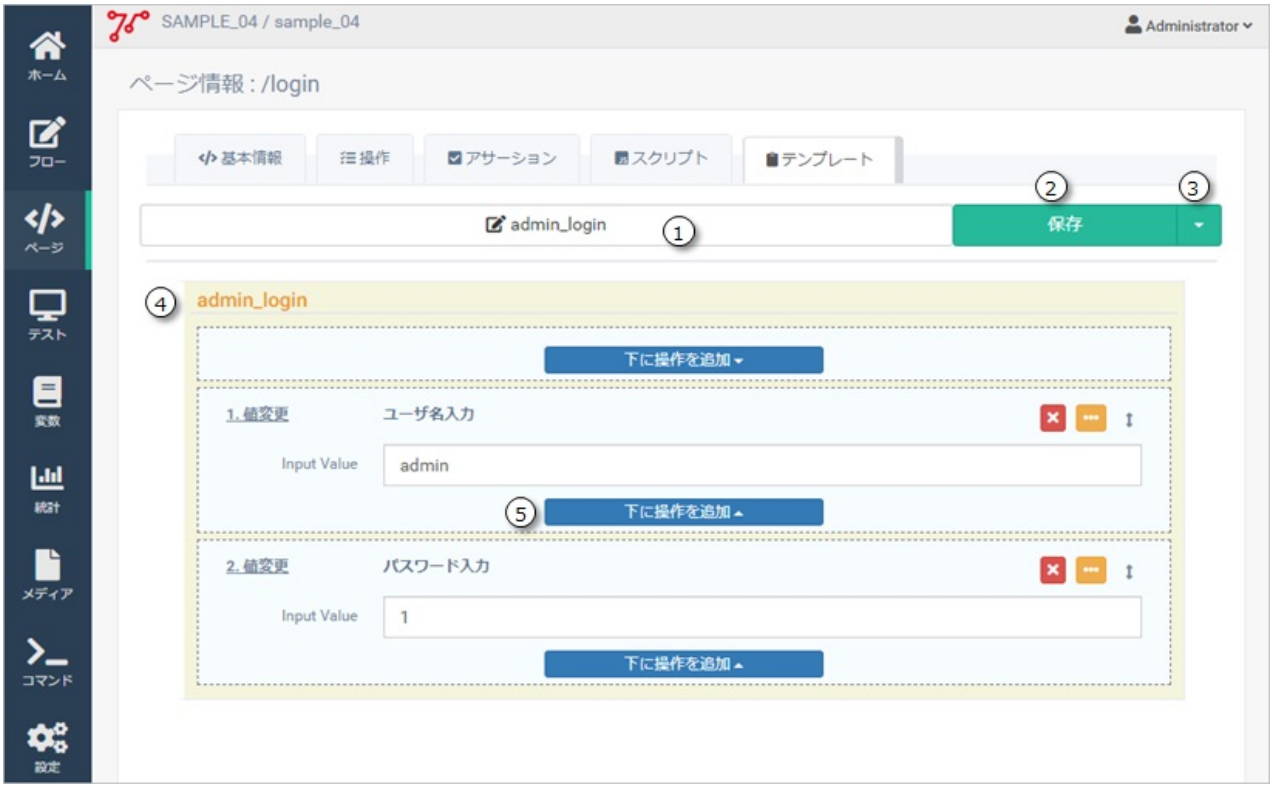


Fig. 4.2.5.2-1 ページテストテンプレート編集 画面

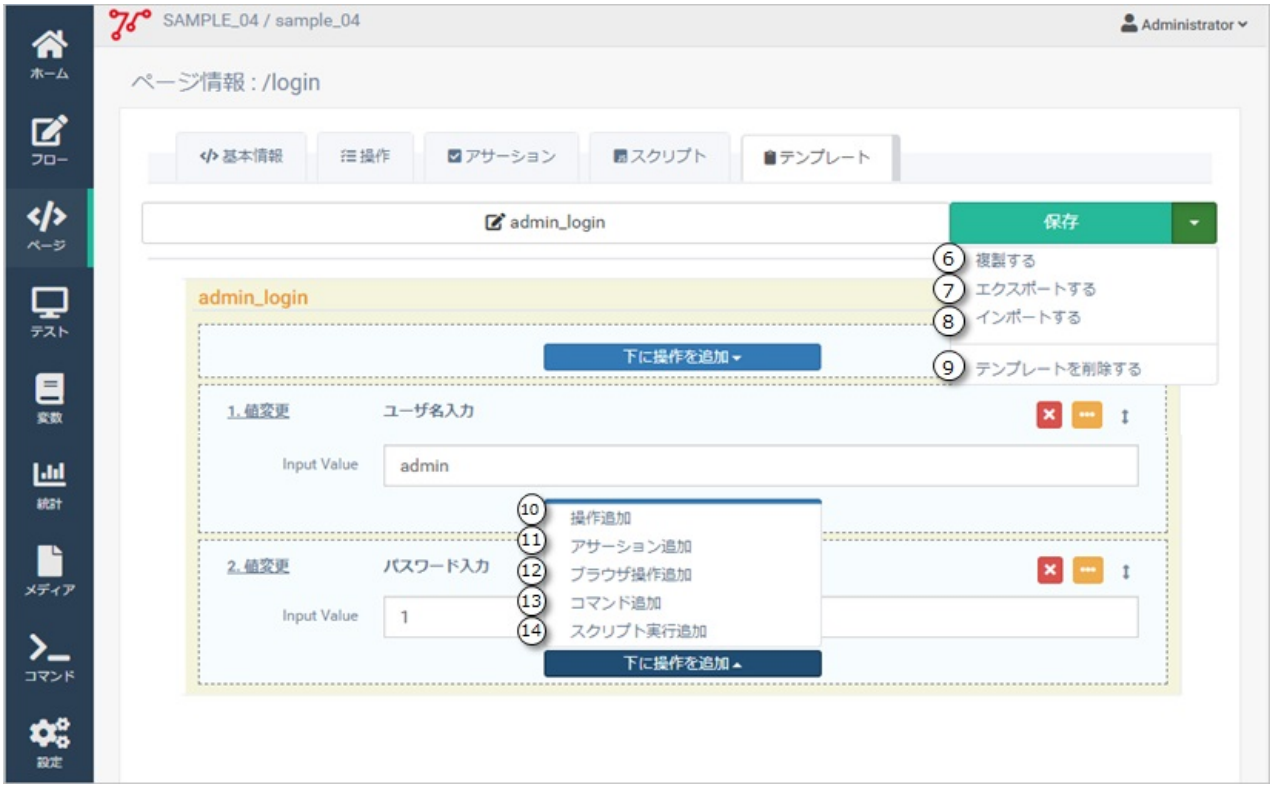


Fig. 4.2.5.2-2 ページテストテンプレート編集 画面*(プルダウン)*

No.	名称	説明
1	ページテスト テンプレート名	クリックすると、テンプレート情報編集ダイアログを表示します。 ページテストテンプレート情報の編集ができます。
2	保存ボタン	編集中のテンプレートを保存します
3	テンプレート編集 プルダウンボタン	テンプレート編集プルダウンメニューを表示します。 No.6～9のメニュー項目があります。
4	ページテスト テンプレート名（表示）	クリックすると、テンプレート情報編集ダイアログを表示します。
5	下に操作を追加 プルダウンボタン	下に操作を追加 プルダウンメニューを表示します。 No.10～14のメニュー項目があります。
テンプレート編集 プルダウンメニュー		
6	複製する	開いているページテストテンプレートを複製します。
7	エクスポートする	ページテストテンプレートを ファイル(xlsx)にエクスポートします。 ここでエクスポートしたページテストテンプレートファイルは、 テスト編集画面でも読み込むことができます。
8	インポートする	エクスプローラーを開き、ページテストテンプレートファイル(xlsx) を読み込んで画面に表示します。
9	テンプレートを削除する	編集中のページテストテンプレートを削除します。
下に操作を追加 プルダウンメニュー		
10	操作追加	操作の下に新しく操作を追加します。
11	アサーション追加	操作の下に新しくアサーションを追加します。
12	ブラウザ操作追加	操作の下にブラウザ操作を追加します。
13	コマンド追加	操作の下にコマンドを追加します。

14	スクリプト実行追加	操作の下にJavascript実行機能を追加します。

(1) ページテストテンプレート名

クリックすると、テンプレート情報編集ダイアログを表示します。
ページテストテンプレート情報の編集ができます。

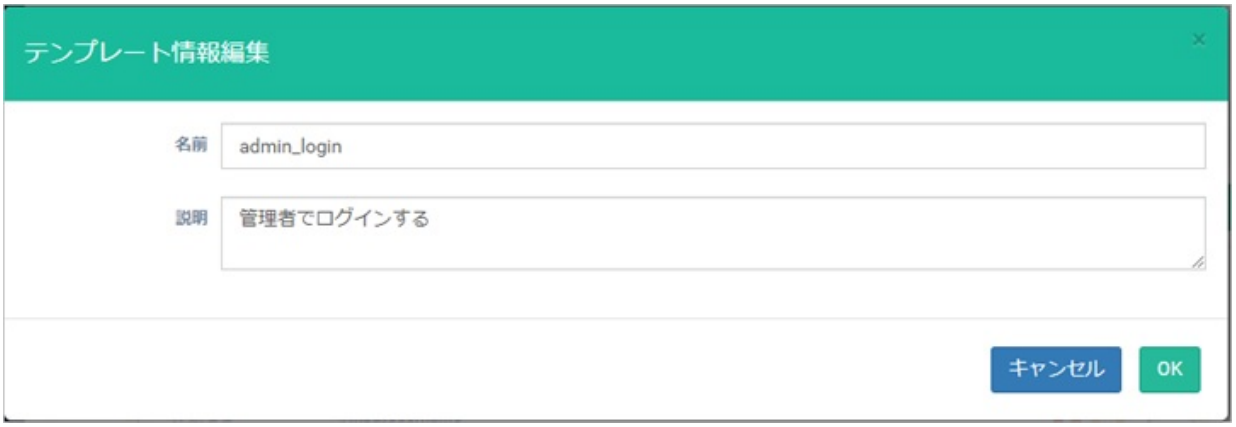


Fig. 4.2.5.2.1 テンプレート情報編集ダイアログ

(2) ページテストテンプレートのエクスポート・インポート

ページテストテンプレートを ファイル(xlsx)にエクスポートします。
ページテストテンプレートファイルは、**ページテストテンプレート編集画面** および **テスト編集画面** で読み込むことができます。
インポートするファイルは、**エクスポートしたファイルの形式と同じ**でなければなりません。

エクスポート/インポート されるファイルのフォーマットや詳細は、別マニュアルの **入出力ファイル仕様** の **4.Testablish ページテストテンプレートファイル** を参照してください。

(3) 操作追加

操作の下に新しく操作を追加します。
ページに定義されている操作が表示された 追加項目選択ダイアログを表示します。

Fig. 4.2.5.2.3 追加項目選択ダイアログ

テンプレートに追加する操作の**操作タイプ**を選択して **Add ボタン** をクリックします。
項目は連続して追加することができます。
ダイアログを閉じて、編集画面に戻るには、**キャンセル ボタン** をクリックします。

(4) アサーション追加

操作の下にアサーションを追加します。
ページに定義されているアサーションが表示された 追加アサーション選択ダイアログを表示します。

Fig. 4.2.5.2.4 追加アサーション選択ダイアログ

テンプレートに追加するアサーションの**検証タイプ**を選択して **Add ボタン** をクリックします。
アサーションは連続して追加することができます。
ダイアログを閉じて、編集画面に戻るには、**キャンセル ボタン** をクリックします。

(5) ブラウザ操作追加

操作の下にブラウザ操作を追加します。
共通項目選択ダイアログを表示します。

Fig. 4.2.5.2.5 共通項目選択ダイアログ

ブラウザを操作するための予めTestablishで用意している操作が選択できます。
（このダイアログに表示される共通項目は、対象のページの状態によって異なります。）

共通項目は連続して追加することができます。
ダイアログを閉じて、編集画面に戻るには、**キャンセル ボタン** をクリックします。

※ 共通項目については、5.2 テスト編集画面 » 5.2.4 操作追加プルダウンメニュー » 5.2.4.3 ブラウザ

操作追加 を参照してください。

(6) コマンド追加

操作の下にコマンドを追加します。
コマンド選択ダイアログを表示します。
このダイアログでは9. コマンド メニューで 定義されている外部コマンドが表示されます。

Fig. 4.2.5.2.6 コマンド選択ダイアログ

コマンドは連続して追加することができます。
ダイアログを閉じて、編集画面に戻るには、**キャンセル ボタン** をクリックします。

(7) スクリプト実行追加

操作の下にJavascript実行機能を追加します。
ページに定義されているスクリプトが表示された スクリプト選択ダイアログを表示します。

Fig. 4.2.5.2.7 スクリプト選択ダイアログ

スクリプト項目は連続して追加することができます。
ダイアログを閉じて、編集画面に戻るには、**キャンセル ボタン** をクリックします。

4.2 ページ編集 画面 目次 / 4. ページ メニュー 目次 に戻る

4.2.6 インスペクション

インスペクション画面では、セッションで収集されたHTMLや事前にページ基本情報に登録したHTMLから、そこに含まれている要素を選択し、その**CSSセレクタを取得**することができます。

Testablishでは、操作・アサーションを個別に設定するインスペクション画面と、複数の操作・アサーションを設定できるマルチインスペクション画面があります。

画面を見ながら追加要素のCSSセレクタが取得できるので、操作の収集で取得していない操作やアサーションを手動で追加する場合に便利です。

4.2.6.1 インスペクション 画面

インスペクション画面は、操作やアサーションの**セレクタボタン**をクリックすることで、別ウィンドウで表示されるサブ画面です。

Fig. 4.2.6.1-1 セレクタボタン

過去に収集したセッションを選択するか、事前に登録したHTMLファイルを選択し、画面を選ぶことができます。
インスペクション画面上で要素をクリックすると、呼び出し元のセレクタ欄にCSSセレクタが入ります。

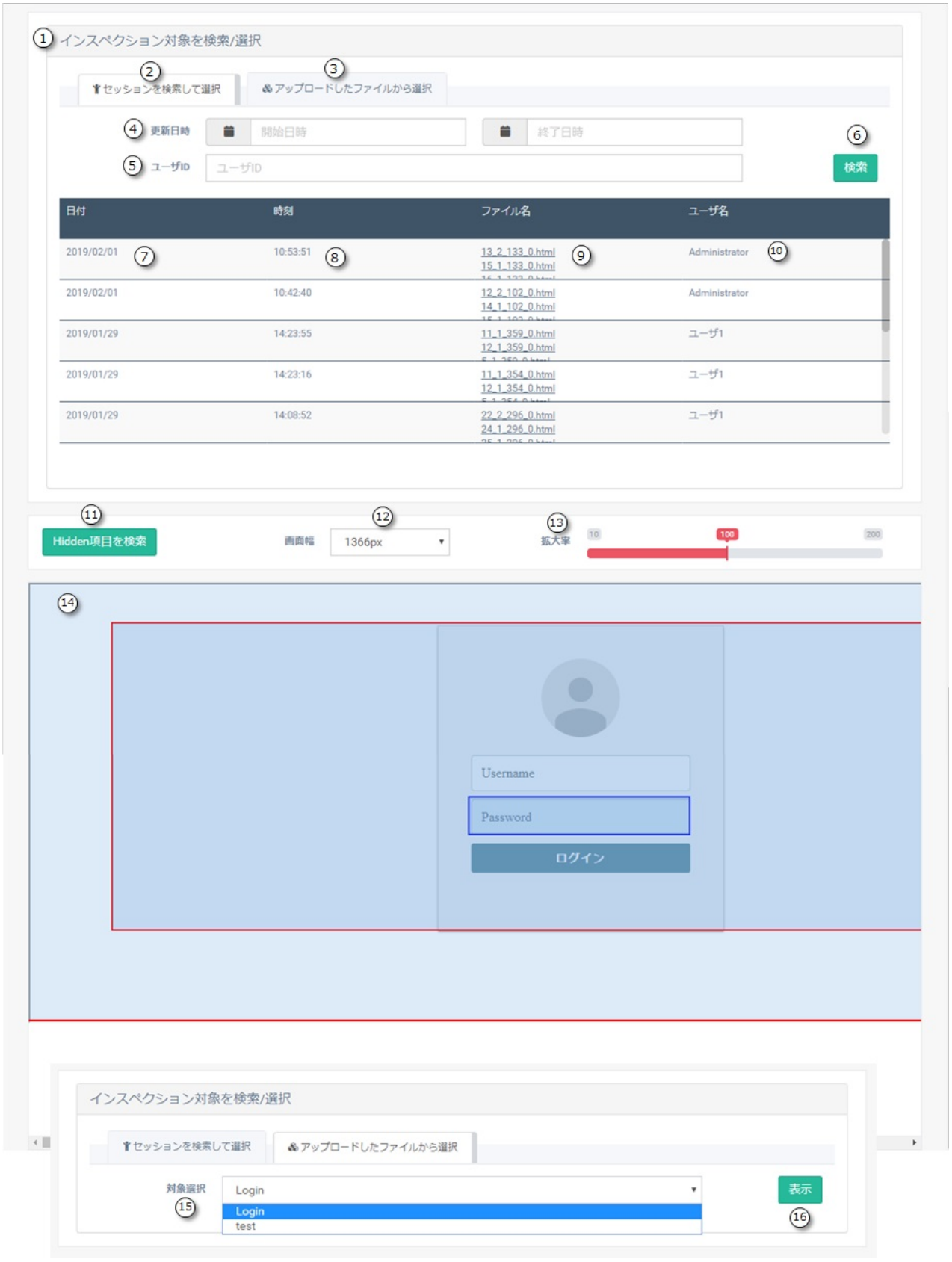


Fig. 4.2.6.1-2 インスペクション画面

No.	名称	説明
1	インスペクション ウィンドウタイトル	このエリアをクリックすることで セッションの検索/選択部分の表示・非表示を切り替えます。
2	セッションを検索して選択 タブ	過去に収集したセッションから選択する場合はこのタブで選択します

3	アップロードした ファイルから選択 タブ	アップロードフォームから取得した情報から選択する場合は このタブで選択します。
セッションを検索して選択 タブ		
4	更新日時	検索条件として、開始日時・終了日時を指定できます。 フィールド左のボタンからカレンダー/ 時計設定ポップアップを表示し、 日付と時刻を選択することができます。
5	ユーザID	検索条件として、セッションを実施したユーザIDを指定できます。
6	検索ボタン	入力した検索条件でセッションを検索し、 セッション一覧を表示します。 すべて空欄の場合はすべてのセッションを表示します。
7	セッションの実施日付	セッションの一覧です。 セッションで取得した画面htmlのリンクをクリックすると、 ウィンドウ下部に 操作画面ビューを表示します。
8	セッションの実施時刻	
9	セッションで取得した画面html	
10	セッションを実施したユーザ名	
要素選択エリア(共通)		
11	Hidden項目を検索 ボタン	画面に非表示な要素の検索をします。 inputタグのtype属性"hidden"の検索をします。
12	画面幅	画面の横幅サイズの変更ができます。
13	拡大率	操作画面ビューの拡大・縮小することができます。
14	操作画面ビュー	インスペクションで設定する要素を選択する画面を表示します。 このビュー内の要素をクリックすると、 クリックした要素がインスペクションウィンドウを呼び出し元の画面 セレクタ入力欄に入力されます。
アップロードしたファイルから選択 タブ		
15	対象選択	ページの基本情報タブのアップロードフォームから アップロードしたhtml画面を選択します。
16	表示ボタン	クリックすると、選択したファイルの内容を ウィンドウ下部の操作画面ビューに表示します。

使い方

1. セレクタボタンをクリックすると、別ウィンドウでインスペクション画面が開きます。
2. 選択タブや検索エリアが表示されていない場合は、インスペクションウィンドウタイトルをクリックします。
3. 「セッションを検索して選択 タブ」または「アップロードしたファイルから選択 タブ」をクリックします。
4. 「セッションを検索して選択 タブ」の場合
   ⇒ 検索ボタンをクリックして、セッション一覧を表示します。
5. 「アップロードしたファイルから選択 タブ」の場合
   ⇒ 対象選択プルダウンから事前に登録しているHTMLファイルを選択します。
6. 操作画面ビューに画面が表示されます。
7. 表示されている画面から、CSSセレクタを取得したい要素をクリックします。

この時、呼び出し元のセレクタ欄にはCSSセレクタが入力されています。

8. インスペクション画面を閉じて、呼び出した編集画面へ戻ります。

4.2.6.2 マルチインスペクション 画面

マルチインスペクション画面は、操作やアサーションの一覧にある**画面から追加**ボタンをクリックすることで表示される画面です。

マルチインスペクション画面では、ウィンドウ上部に **操作・アサーションリストエリア** があり、対象ページの操作とアサーションをまとめて**複数設定**することができます。
対象の操作・アサーションを検索・選択する方法は、4.2.6.1 インスペクション 画面と同じです。

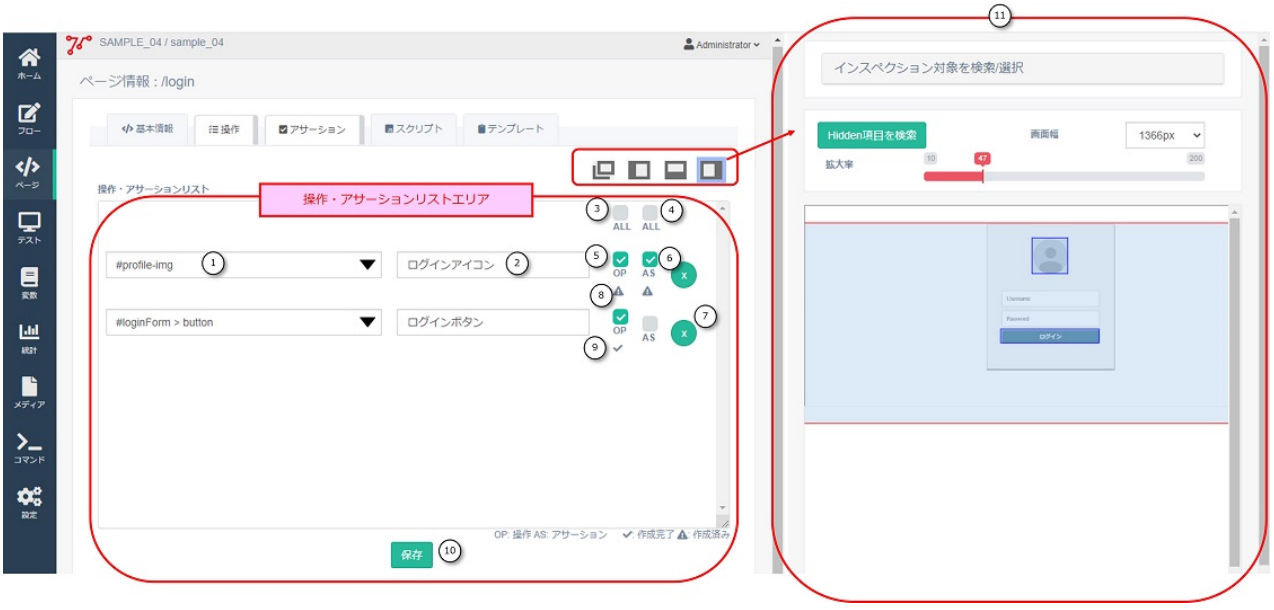


Fig. 4.2.6.2 マルチインスペクション画面

No.	名称	説明
1	操作・ アサーションCSSセレクタ	画面ビューでクリックした要素のCSSセレクタが入力されます。
2	対象名入力欄	操作・アサーションの名前を入力してください。
3	ALLチェックボックス(OP)	すべてのAS(アサーション)チェックボックス のチェックを ON/OFFします。
4	ALLチェックボックス(AS)	すべてのOP(オペレーション)チェックボックス のチェックを ON/OFFします。
5	OP(オペレーション) チェックボックス	操作定義の作成を行うときにチェックしてください。
6	AS(アサーション) チェックボックス	アサーション定義の作成を行うときにチェックしてください。
7	操作・アサーションリスト 削除ボタン	選択した要素を削除します。
8	作成済みマーク	保存ボタンをクリックした後に、 すでに定義が作成されているときに出現します。

9	作成完了マーク	保存ボタンをクリックした後に、 作成が完了できたときに出現します。
10	保存ボタン	設定した操作定義やアサーション定義を保存します。
11	インスペクション対象ビュー	インスペクション対象を検索/選択するエリアの使い方は 4.2.6.1 インスペクション 画面 を参照してください。 このウィンドウの配置場所は操作・ アサーションリストの右側にあるボタンで変更することができます。

使い方

1. ページ編集画面の操作一覧またはアサーション一覧画面で、**画面から追加ボタン**をクリックすると、マルチインスペクション画面に移動します。
2. 選択タブや検索エリアが表示されていない場合は、インスペクションウィンドウタイトルをクリックします。
3. 4.2.6.1 インスペクション 画面と同様に、インスペクション対象画面を選択し、操作画面ビューに画面を表示します。
4. 表示されている画面から、CSSセレクタを取得したい要素をクリックします。
   この時、ウィンドウ上部の操作・アサーションリストにクリックした要素が追加されていきます。
5. 必要に応じて対象名入力欄に名称を設定します。
6. CSSセレクタごとに、作成する操作・アサーションのチェックボックスを設定します。
   上部のALLチェックボックスでまとめてチェックをON/OFFすることもできます
7. 定義の設定ができたら、**保存ボタン**をクリックします。
8. チェックが入っているチェックボックスの下部に作成完了マーク(?)が表示されれば作成完了です。
9. 画面上部の操作タブまたはアサーションタブをクリックして、操作一覧またはアサーション一覧に戻って、作成を確認してください。

4. ページ メニュー 目次 ｜ II. 各画面の説明 目次 ｜ 使い方マニュアル 目次 に戻る

5. テスト メニュー

5.1 テスト一覧 画面

テスト一覧画面は、作成したテストを一覧表示する画面です。
他のユーザが編集中の場合は保存ができません。
各テスト名はリンクになっており、テストごとの設定画面に移動できます。
この画面から、手動で新たにテストを追加することもできます。

- 5.1.1 新規テスト作成ボタン

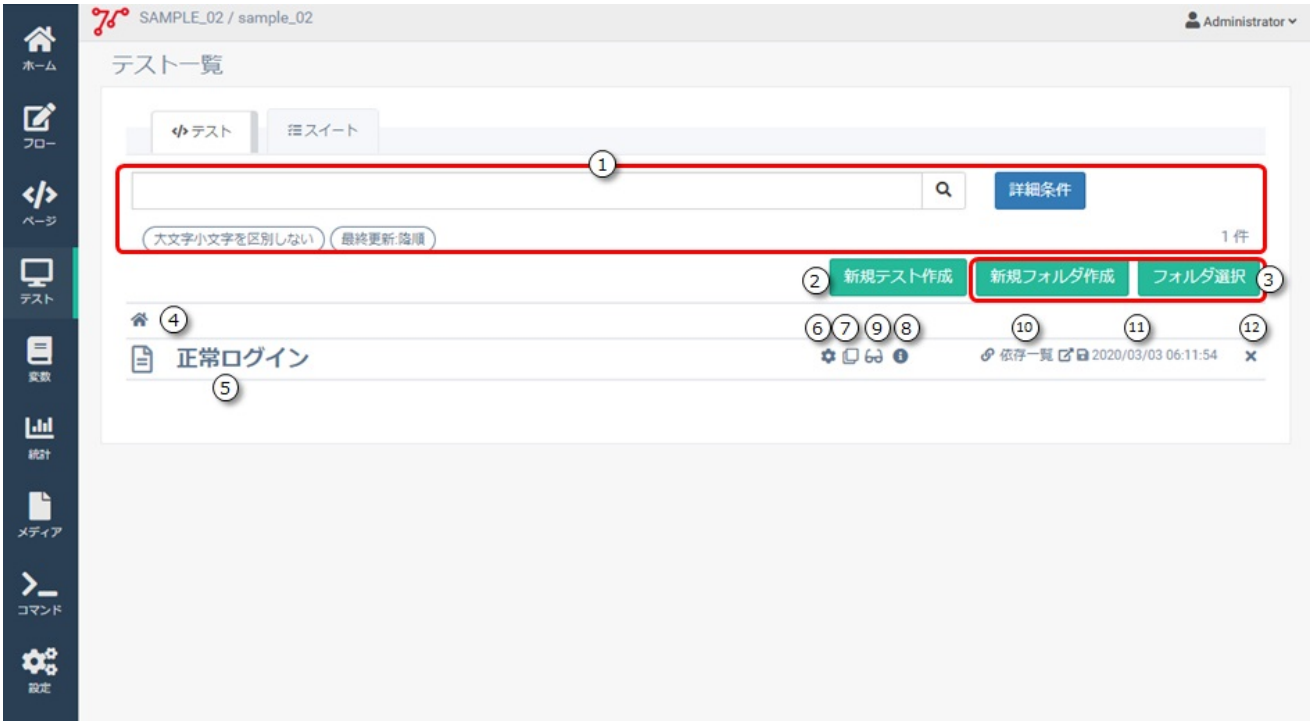


Fig. 5.1 テスト一覧画面

No.	名称	説明
1	一覧検索・ソート	一覧の表示内容の絞り込みと検索とその結果表示です。 詳しくは、1.各画面共通»1.3 一覧表示での検索・ソート機能 を参照してください。
2	新規テスト作成ボタン	新規テストを作成することができます。
3	新規フォルダ作成ボタン フォルダ選択ボタン	テスト一覧でのフォルダの追加やフォルダ移動等に関するボタンです。 詳しくは、1.各画面共通»1.5 一覧表示でのフォルダ操作 を参照してください。
4	現在位置表示	現在開いているフォルダの階層を表示しています。 フォルダ名をクリックすることでその階層を開くことができます。
5	テスト名	クリックすると、そのテストのテスト編集画面に移動します。
6	設定アイコン	テストの設定や移動・複製を行うことができます。 詳しい操作方法は、1.各画面共通»1.5 一覧表示でのフォルダ操作 を参照してください。
7	移動・複製アイコン	
8	情報アイコン	

9	参照アイコン	テスト編集画面を**参照モード**(ReadOnly)で開きます。
10	依存一覧リンク	このページを使用しているテストスイートの一覧画面を別ウィンドウで開きます。 詳しくは、1.各画面共通»1.4 依存一覧リンク を参照してください。
11	最終更新日時	
12	削除アイコン	テストの削除を行うことができます。

5.1.1 新規テスト作成ボタン

新規テスト作成ボタンをクリックすると、新規テスト作成ダイアログが表示され、テストシナリオを新たに追加することができます。

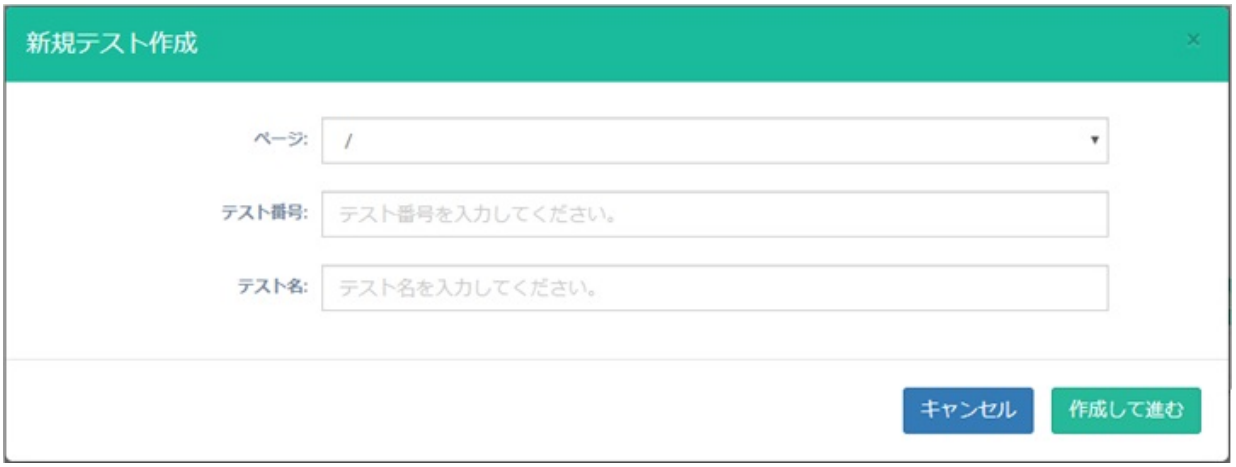


Fig. 5.1.1 新規テスト作成ダイアログ

新規作成ダイアログでは **開始ページ** をリストより選択し、**テスト番号** および **テスト名** を入力、**作成して進む** ボタンをクリックします。
新規テストが作成され、テスト編集画面が表示されます。
なお、テスト名には以下の文字を含むことができませんのでご注意ください。

```
\ / : * ? " < > | () [] {} & $ ` ^ ~
```

5.1 テスト一覧 画面 目次 / 5. テストメニュー 目次 に戻る

5.2 テスト編集画面

テスト編集画面は、選択したテストを編集する画面です。
この画面からは、作成するテストの操作の追加・削除・変更、入力値の設定、アサーションの設定などを行うことができます。
この画面から、テストコードの出力およびテストパターンと呼ばれるテストの編集ファイルのダウンロード・アップロード、テスト仕様書の出力を行うことができます。

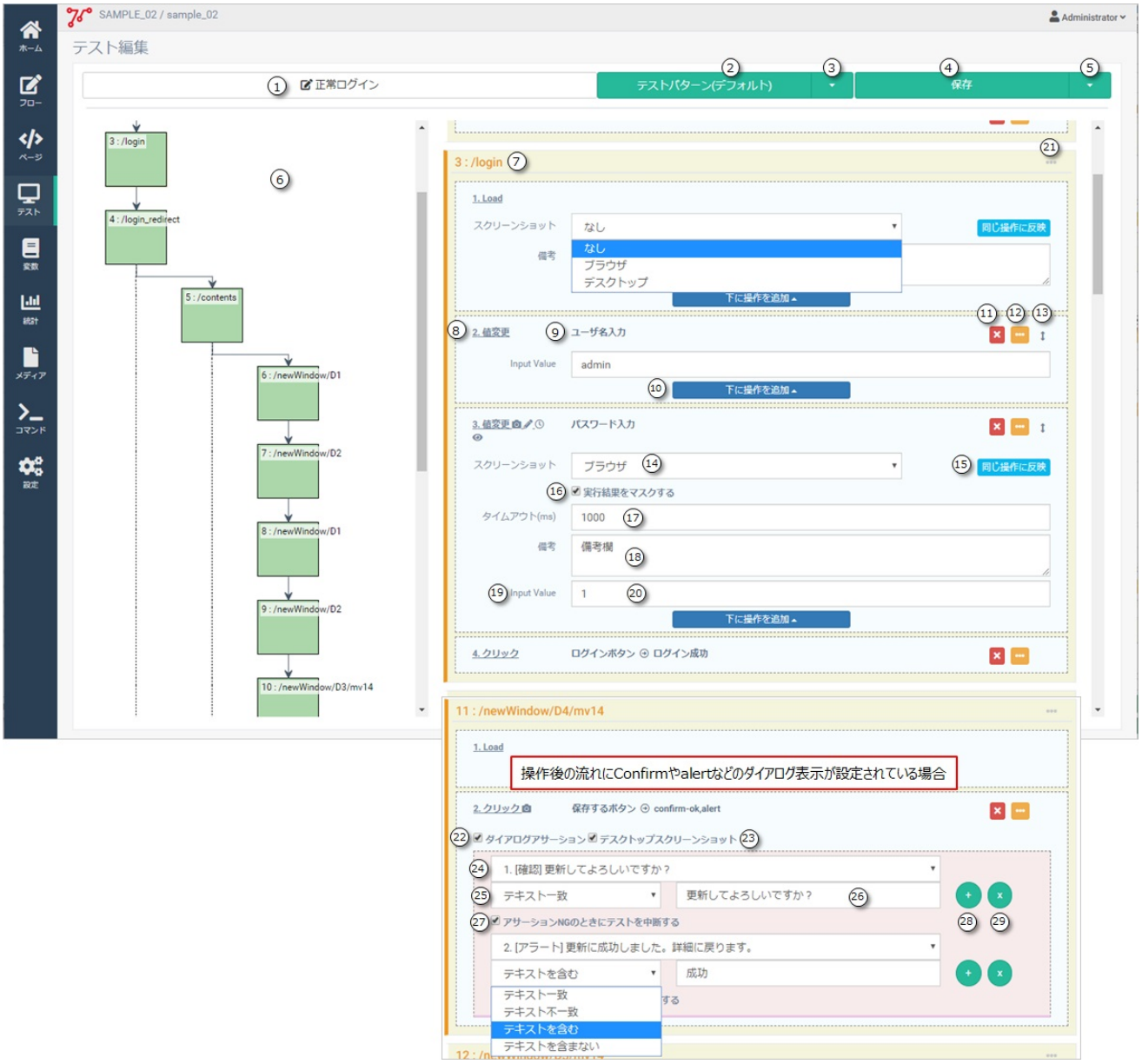


Fig. 5.2-1 テスト編集画面

No.	名称	説明
1	テスト名	クリックすると、テスト基本情報編集ダイアログを表示します。
2	テストパターン名	現在表示しているテストパターン名を表示します。
3	テストパターン選択 プルダウンボタン	テストパターン選択プルダウンメニューを表示します。
4	保存 ボタン	テストの編集を保存します。 参照モードで編集画面を開いている場合は保存ボタンは非活性になります
5	テストコード出力 プルダウンボタン	テストコード出力プルダウンメニューを表示します。
6	テストフロー	テストの画面遷移(ステップ)をフローチャートで表示します。
7	ステップ番号と画面名	テストで操作するステップ番号と画面名を表示します。 画面にフレームが含まれる場合、アクティブになっている フレームのフレーム名 または セレクタも併せて表示されます。
8	操作番号と操作種類	操作種類を表示します。 クリックすると、(14)～(18)の表示/非表示を切り替えます。

9	操作名	操作名(設定されていない場合はセレクタ)を表示します。
10	下に操作を追加 プルダウンボタン	操作追加プルダウンメニューを表示します。
11	 (削除ボタン)	操作を削除します。
12	 (項目操作ボタン)	項目操作プルダウンメニューを表示します。 メニューに表示される項目は、項目の状況によって異なります。
13	 (矢印アイコン)	この矢印アイコンをつかんで ドラッグ＆ドロップすることで、 操作ボックスを移動させることができます。
14	スクリーンショット プルダウン	テスト実行時にスクリーンショットを取得する場合に、 プルダウンから選択・指定します。 **なし**：スクリーンショットの取得なし **ブラウザ**：ブラウザの内容のスクリーンショットを取得 **デスクトップ**：フルスクリーンのスクリーンショットを取得 「ブラウザ」または「デスクトップ」を選択すると、 (8)の操作種類の横に カメラアイコン が表示されます。
15	同じ操作に反映 ボタン	スクリーンショットを取得する/しない の設定を同じ操作に反映します。
16	実行結果をマスクする チェックボックス	テスト実行後に出力されるテスト実行結果ファイルで、当該操作の入力値 設定値・エラー内容を「XXXXX」でマスクするかどうかを指定します。 チェックを入れると、(8)の操作種類の横に 目アイコン が表示されます。
17	タイムアウト(ms)	テスト実行時にこの要素を探索する時間をミリ秒で設定できます。 例えば、その要素が画面に表示されるまでに時間がかかるような場合、 このタイムアウトを長めに設定することで、 "テスト実行時に操作する要素が見つからない" といったエラーを 回避することができます。 タイムアウトを設定すると、(8)の操作種類の横に 時計アイコン が表示されます。
18	備考	操作の備考を入力できます。 備考を入力すると、(8)の操作種類の横に 鉛筆アイコン が表示されます
19	値ラベル	操作がvalue属性を用いる場合に表示します。 html要素名を項目ラベルとして表示します。
20	値	操作がvalue属性を用いる場合に表示します。value の値を入力します。
21	 (ステップ操作ボタン)	ステップ操作プルダウンメニューを表示します。 メニューに表示される項目は、ステップの状況によって異なります。
ダイアログアサーション関連		
22	ダイアログアサーション チェックボックス	操作の操作後の流れにダイアログ表示の設定がある場合に表示されます。 チェックすると、ダイアログに対するアサーションを設定できるように なります。
23	デスクトップ スクリーンショット チェックボックス	操作の操作後の流れにダイアログ表示の設定がある場合に表示されます。 チェックすると、ダイアログを表示した状態のフルスクリーンショットを 取得することができます。
24	ダイアログシーケンス選択 プルダウン	検証する対象のダイアログを選択します。

25	検証タイプ選択 プルダウン	検証タイプを選択します。 ダイアログアサーションで設定できる検証タイプは次の通りです。 「テキスト一致 / テキスト不一致 / テキストを含む / テキストを含まない」
26	検証値	検証値を設定します。
27	テスト中断 チェックボックス	チェックすると、アサーションNGの時にテストを中断します。
28	追加 ボタン	ダイアログアサーションを追加します。
29	削除 ボタン	ダイアログアサーションを削除します。

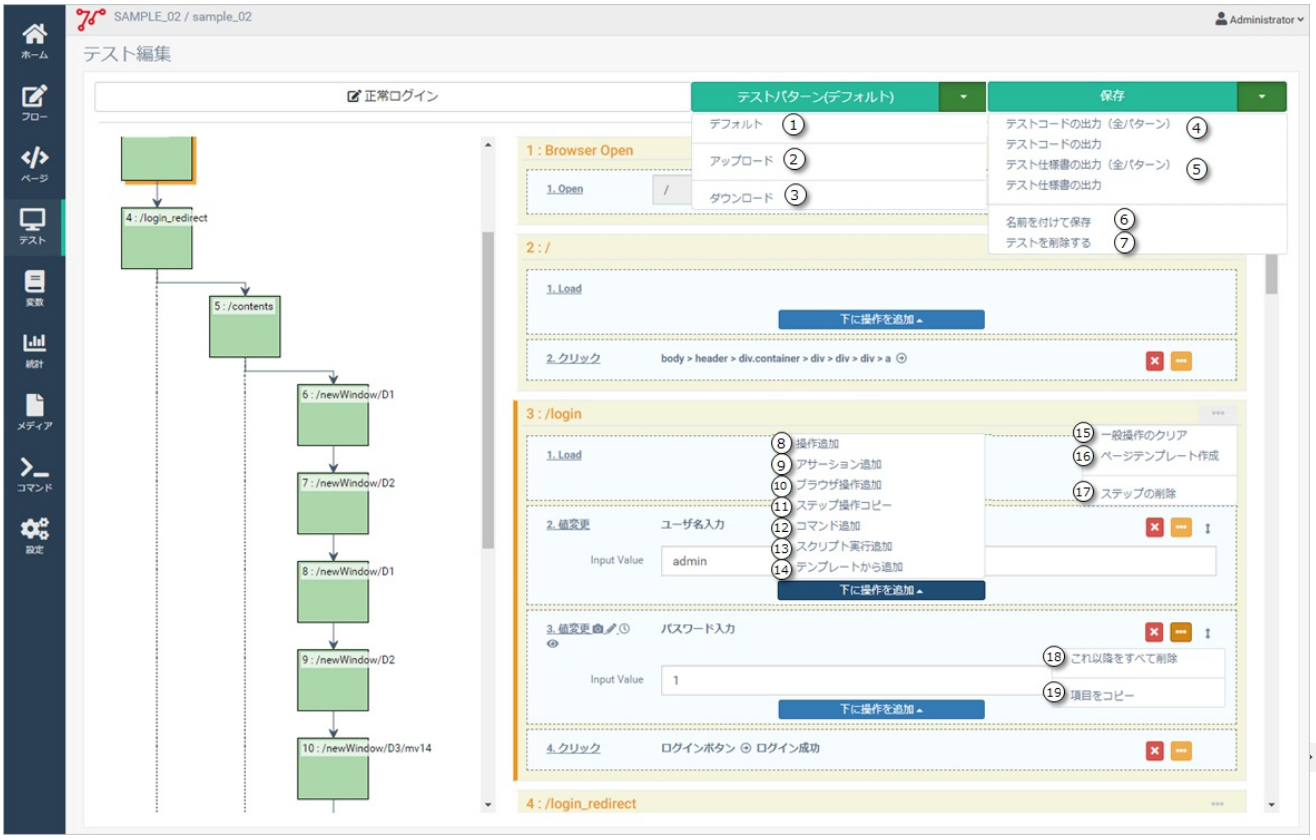


Fig. 5.2-2 テスト編集画面*(プルダウンメニュー)*

No.	名称	説明
テストパターン選択 プルダウンメニュー		
1	テストパターン名	テストパターンを選択します。 テストにテストパターンが複数ある場合は複数表示されます。 テストを作成した時点では「デフォルト」のパターン一つのみが存在します
2	アップロード	テストパターンファイル（xlsx）を読み込んで画面に表示します。
3	ダウンロード	テストパターンファイル（xlsx）をダウンロードします。
テストコード出力プルダウンメニュー		
4	テストコードの出力 (全パターン)	表示しているパターン または 全パターンの テストコードをzipファイルでダウンロードします。
5	テスト仕様書の出力 (全パターン)	表示しているパターン または 全パターンの テスト仕様書をzipファイルでダウンロードします。

		テスト仕様書はプロジェクトの設定に従い、Excel または Word (もしくは両方)のファイル形式で出力されます。
6	名前を付けて保存	現在編集中のテストを別名で保存します。
7	テストを削除する	テストを削除します。
操作追加プルダウンメニュー		
8	操作追加	追加項目選択ダイアログを表示し、操作の下に新しく操作を追加します。
9	アサーション追加	追加アサーション選択ダイアログを表示し、 操作の下に新しくアサーションを追加します。
10	ブラウザ操作追加	共通項目選択ダイアログを表示し、操作の下にブラウザ操作を追加します。
11	ステップ操作コピー	同じページがテスト内にある場合に、 操作やアサーション定義のコピーをすることができます。
12	コマンド追加	コマンド選択ダイアログを表示します。
13	スクリプト実行追加	スクリプト選択ダイアログを表示し、 操作の下にJavascript実行機能を追加します。
14	テンプレートから追加	テンプレートの読込ダイアログを表示します。 ページで定義したページテストテンプレートを読み込むことができます。 ページテストテンプレート編集画面でエクスポートしたExcelファイル(※) を読み込むこともできます。 ※ エクスポートしたファイルをインポートする際には、基本情報シートを削除し、 操作シートのみにしておく必要があります。
ステップ操作 プルダウンメニュー		
15	一般操作のクリア	Loadや画面遷移を伴う操作などの特殊な操作を除く 一般的な操作を削除することができます。
16	ページテンプレート作成	新規テンプレート作成ダイアログを表示し、 ステップをページテストテンプレートとして保存できます。 なお、ページ遷移を伴う操作はテンプレートに含めることはできません。
17	ステップの削除	ステップを削除します。 ステップが遷移でつながっている場合はこのメニュー項目は表示されません
項目操作プルダウンメニュー		
18	これ以降をすべて削除	項目操作ボタンをクリックした操作以降をすべて削除します。
19	項目をコピー	操作をコピーして追加します。画面遷移を伴う操作の場合、 このメニュー項目は表示されません。

5.2.1 テスト基本情報編集ダイアログ

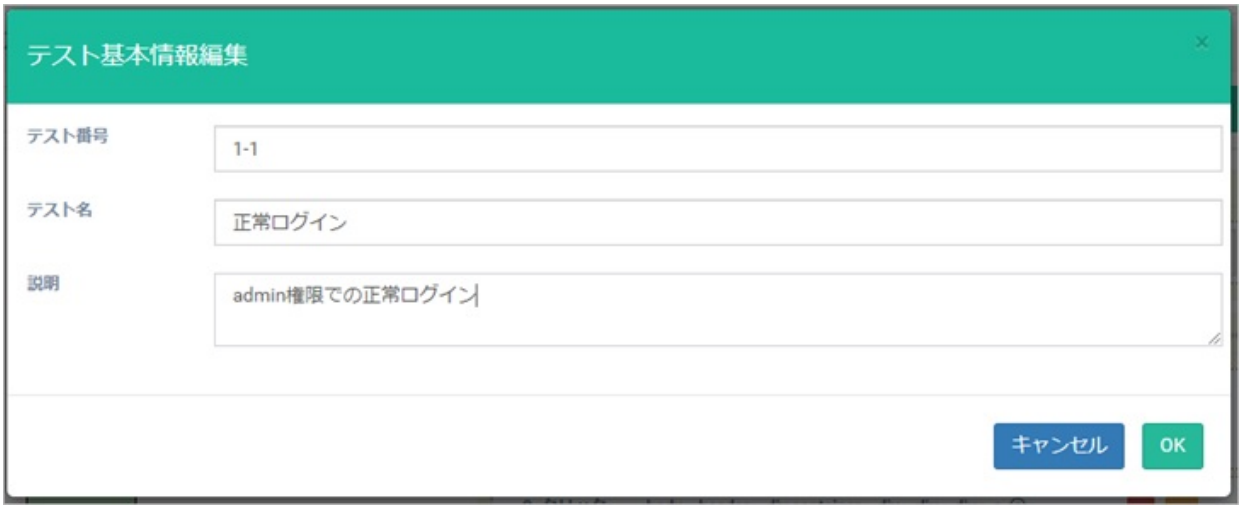


Fig. 5.2.1 テスト基本情報ダイアログ

テスト名をクリックすると、テスト基本情報編集ダイアログが表示され、ここでテストの基本情報を設定することができます。
テスト番号は任意の形式で設定することができます。**テスト仕様書**を出力する際には、このテスト番号の設定が**必須**になります。
説明欄を入力しておくと、テスト一覧画面の情報アイコンをクリックすることで、説明内容を確認することができます。

5.2 テスト編集 画面 目次 / 5. テストメニュー 目次 に戻る

5.2.2 テストパターン選択 プルダウンメニュー

テストシナリオを作成する中で、テストの流れは同じで、入力する値を変えて実施したい場合があります。
Testablishでは、入力値のパターンを複数作成して適用することが可能です。

編集中のテストシナリオにパターンが複数含まれる場合、プルダウンメニューには登録されているパターン名が表示されます。 (パターンがない場合は『テストパターン(デフォルト)』のみになります。)

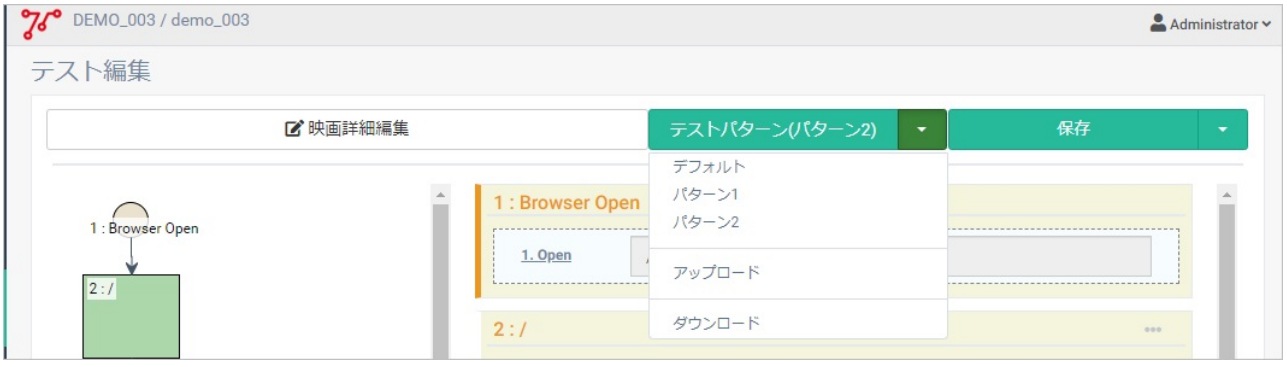


Fig. 5.2.2 テストパターン選択 プルダウンメニュー(複数パターン有)

5.2.2.1 テストパターン名

現在選択しているテストパターンの名称を表示します。
テストパターン選択 プルダウンメニューからパターン名を選択してパターンを切り替えることができます。

5.2.2.2 ダウンロード/アップロード

テストパターンを利用する流れは
1. テストパターン選択プルダウンメニュー » **ダウンロード**
2. ダウンロードした **テストパターンファイル** (excel形式) を **編集**
3. テストパターン選択プルダウンメニュー » **アップロード**
4. テストパターン選択プルダウンメニュー » (利用したい)**テストパターン名を選択・表示**

となります。

テストパターン選択 プルダウンメニューから **ダウンロード**を選択すると、Excelファイル形式（.xlsx）でテストパターンファイルがダウンロードされます。 ファイル名は **<テスト名>_output_<出力日時>.xlsx** です。

ダウンロードされたExcelファイルを編集することで、パターンを追加・削除することができます。

編集したテストパターンファイルを読み込む場合は **アップロード** を選択してください。
アップロードしたテストパターンが テストパターン選択プルダウンメニューに表示され、選択できるようになります。

テストパターンファイル編集の詳細については 別マニュアルの **入出力ファイル仕様 2. Testablish テストパターンファイル** を参照してください。

5.2 テスト編集 画面 目次 / 5. テストメニュー 目次 に戻る

5.2.3 テストコード出力プルダウンメニュー

このメニューから、テストコード・テスト仕様書の出力の他、テストシナリオの保存および削除を行うことができます。

5.2.3.1 テストコードの出力

保存したテストは、Selenium で実行できるテストコードとして出力することができます。 (保存されていないテスト編集画面の内容は出力されるコードには反映されません。)

テストコードはzipファイル形式で出力されます。
ファイル名は **<テスト名>_TestCode_<出力日時>.zip** です。

「テストコードの出力」を選択した場合、画面に表示されているパターンのテストコードが生成されます。

「テストコードの出力（全パターン）」を選択すると、一つのテストコードにパターンをすべて含めたコードが生成されます。
出力されたテストコードを実行すると、すべてのパターンが順に実行されます。

テストの自動実行の手順は、**III 操作の流れ/機能説明 » テストの自動実行** を参照してください。

5.2.3.2 テスト仕様書の出力

作成して保存したテストから、テスト仕様書が生成できます。 (保存されていないテスト編集画面の内容は出力される仕様書には反映されません。)

テスト仕様書はプロジェクトの設定に従いzipファイルで出力されます。
ファイル名は **<テスト名>_TestDocument_<出力日時>.zip** です。

設定メニュー の **テスト仕様書出力タイプ** で設定したしたファイル形式（エクセル(.xlsx) または ワード(.docx)もしくは両方）で作成されます。

「テスト仕様書の出力」を選択した場合、画面に表示されているパターンのテスト仕様書が生成されます。

「テスト仕様書の出力（全パターン）」を選択すると、複数のテストパターンがある場合、ファイルは一つで複数シートの仕様書が生成されます。

テスト仕様書の詳細については、 別マニュアルの **入出力ファイル仕様 1. Testablish テスト仕様書** を参照してください。

5.2 テスト編集 画面 目次 / 5. テストメニュー 目次 に戻る

5.2.4 操作追加プルダウンメニュー

ページのステップ内にある「下に操作を追加」ボタンをクリックすると **操作追加プルダウンメニュー**がボタン上方に表示されます。

5.2.4.1 操作追加

操作追加プルダウンメニューから、**操作追加** を選択してください。
追加項目選択ダイアログが表示されます。
このダイアログでは、操作追加を選択したページに定義されている操作が表示されます。

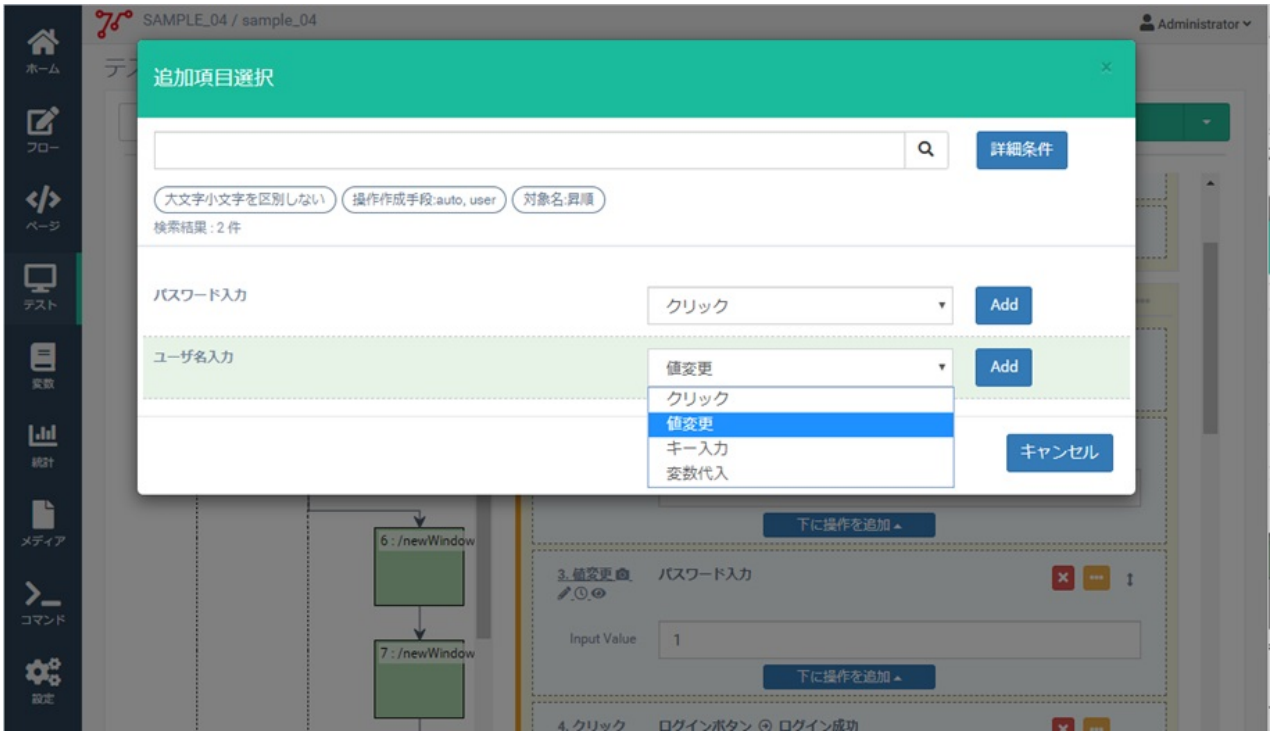


Fig. 5.2.4.1 追加項目選択イアログ

追加する操作を選んで、プルダウンから操作の種類を選択してください。

プルダウンには、その操作に設定されている操作タイプと「変数代入」が選択肢として表示されます。

Add ボタンで操作を追加します。
操作は続けて追加できます。ダイアログを閉じるには キャンセル ボタンをクリックしてください。
テスト編集画面に操作が追加されます。

5.2.4.2 アサーション追加

操作追加プルダウンメニューから、**アサーション追加** を選択してください。
追加アサーション選択イアログが表示されます。
このダイアログでは、アサーション追加を選択した**ページに定義されている**アサーションが表示されます。

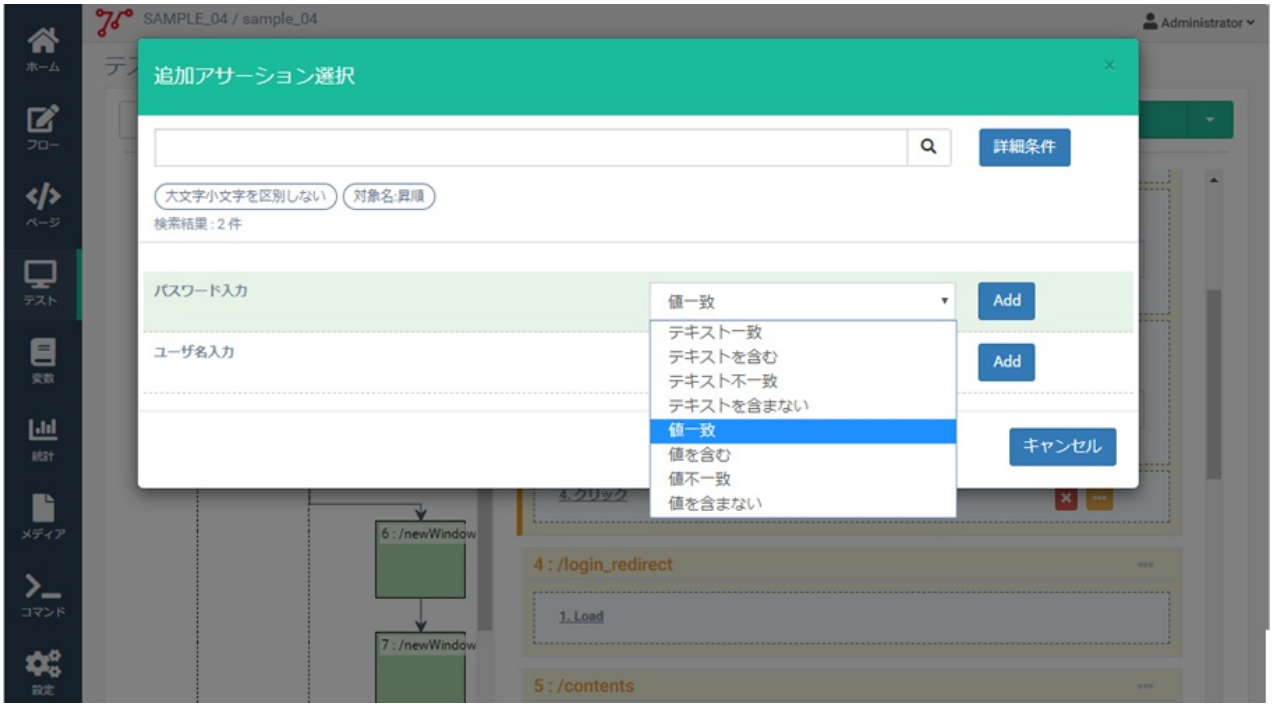


Fig. 5.2.4.2-1 追加アサーション選択イアログ

追加するアサーションを選び、プルダウンから検証の種類を選択してください。
Add ボタンで操作を追加します。
操作は続けて追加できます。ダイアログを閉じるには キャンセル ボタンをクリックしてください。
テスト編集画面にアサーションが追加されます。

追加されたアサーションの操作ボックスには、実行オプションがあります。

- 空白の取り扱い
  アサーション対象の値がテキストの場合に表示されます。
  「**何もしない**」：テスト実行時の入力値はそのまま検証されます。
  「**すべての空白を除去**」：テスト実行時の入力値から空白が除去されたうえで検証されます。

- 改行の取り扱い
  アサーション対象の値がテキストの場合に表示されます。
  「**すべての改行を除去**」チェックボックスにチェックを入れると、テスト実行時の入力値から改行が除去されたうえで検証されます。

- **アサーションNGの時にテストを中断する**
  テストを実行する際、検証が失敗した際にテストを中断する場合には、これをチェックしてください。
  チェックしない場合は、検証が失敗していても後続のステップが実行されます。

Fig. 5.2.4.2-2 アサーション 実行オプション

なお、変数に対するアサーションは ブラウザ操作として追加します。詳細は 5.2.4.3 ブラウザ操作追加 の ◆変数のアサーション を参照してください。

また、ある操作の操作後の流れにダイアログを表示するような設定がされている場合、 その操作のシーケンスに **ダイアログアサーション チェックボックス** が表示されます。
これをチェックすることで、アサーションを設定するエリアが開き、ダイアログに表示されている文字列を検証することができます。
※ ダイアログに対するアサーションについては、ページのアサーションとして定義することはなく、**テスト編集画面で直接 設定** します。
詳しくは III. 操作の流れ/機能説明 » 6.ダイアログアサーションの作成 を参照してください。

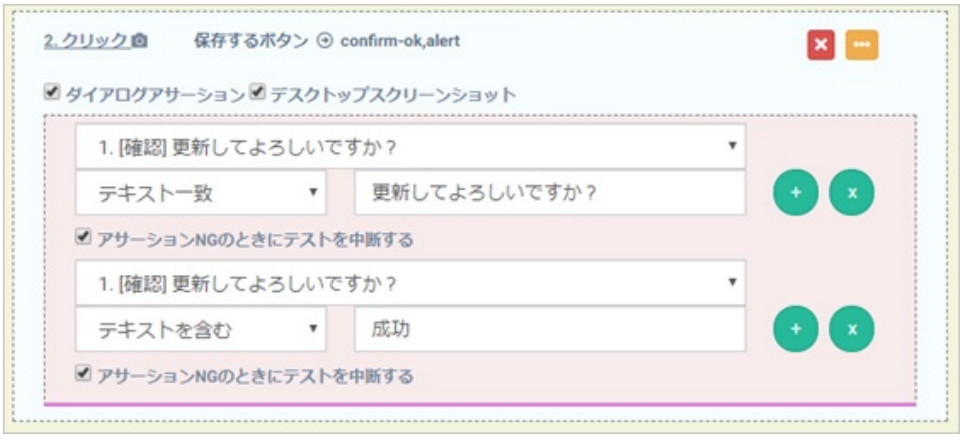


Fig. 5.2.4.2-3 ダイアログアサーション

5.2.4.3 ブラウザ操作追加

操作追加プルダウンメニューから、**ブラウザ操作追加** を選択してください。
共通項目選択イアログが表示されます。

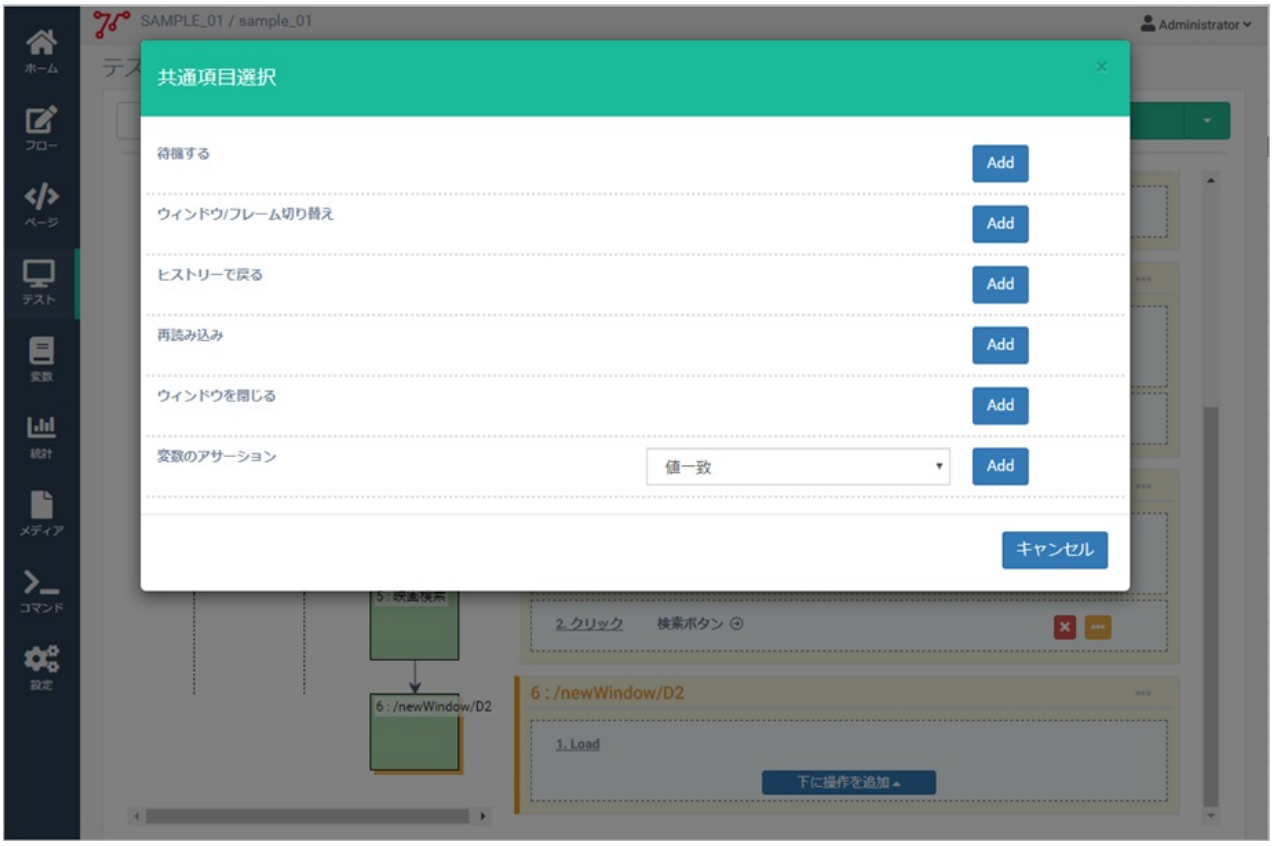


Fig. 5.2.4.3-1 共通項目選択イアログ

このダイアログでは、ブラウザ操作に関する共通の項目が表示されます。
これらの項目は、Testablishに事前に登録されているもので、ブラウザ操作を追加するページの遷移の設定状態によって表示される項目が変動します。
表示される項目の種類には以下のものがあります。

- ・待機する
- ・ウィンドウ/フレームの切り替え
- ・ヒストリーで戻る
- ・ヒストリーで進む
- ・再読み込み
- ・ウィンドウを閉じる
- ・変数のアサーション

選択する項目によっては追加の設定ダイアログが表示されることがありますが、ダイアログの指示に従って設定を行ってください。

追加するブラウザ操作を選び、Add ボタンで操作を追加します。
テスト編集画面にブラウザ操作が追加されます。

◆変数のアサーション

テストで使用している変数に対してアサーションを行う場合は、 **ブラウザ操作**として**共通項目選択イアログ**で選択・追加します。

変数のアサーションを追加する際には、右側のプルダウンから検証の種類を選択し、Add ボタンで追加します。
キャンセルボタンをクリックして共通項目選択イアログを閉じると、テスト編集画面に変数のアサーションが追加されています。

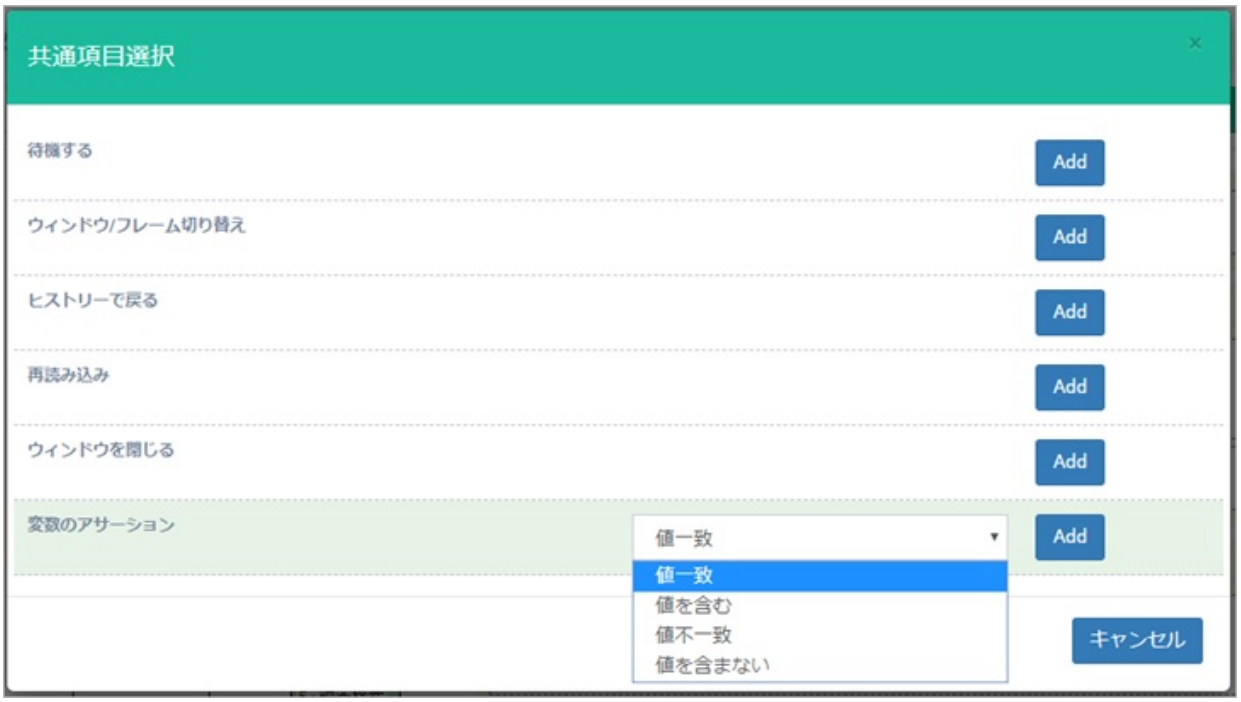


Fig. 5.2.4.3-2 共通項目選択イアログ：変数のアサーション

追加された変数のアサーションのラベルは、共通項目選択イアログで選択した検証の種類になっています。
ここでは検証する変数名とその値を入力します。
その他の実行オプション等は通常のアサーション設定と同じです。

Fig. 5.2.4.3-2 共通項目選択イアログ：変数のアサーション

5.2.4.4 ステップ操作コピー

テストケースの中に 同じ画面 のステップがあり、同じ操作やアサーション定義を行いたい場合、別のステップから操作をコピーすることができます。
コピーを挿入したいステップの 操作追加プルダウンメニューから、**ステップ操作コピー** を選択してください。
ステップ操作コピーダイアログが表示されます。

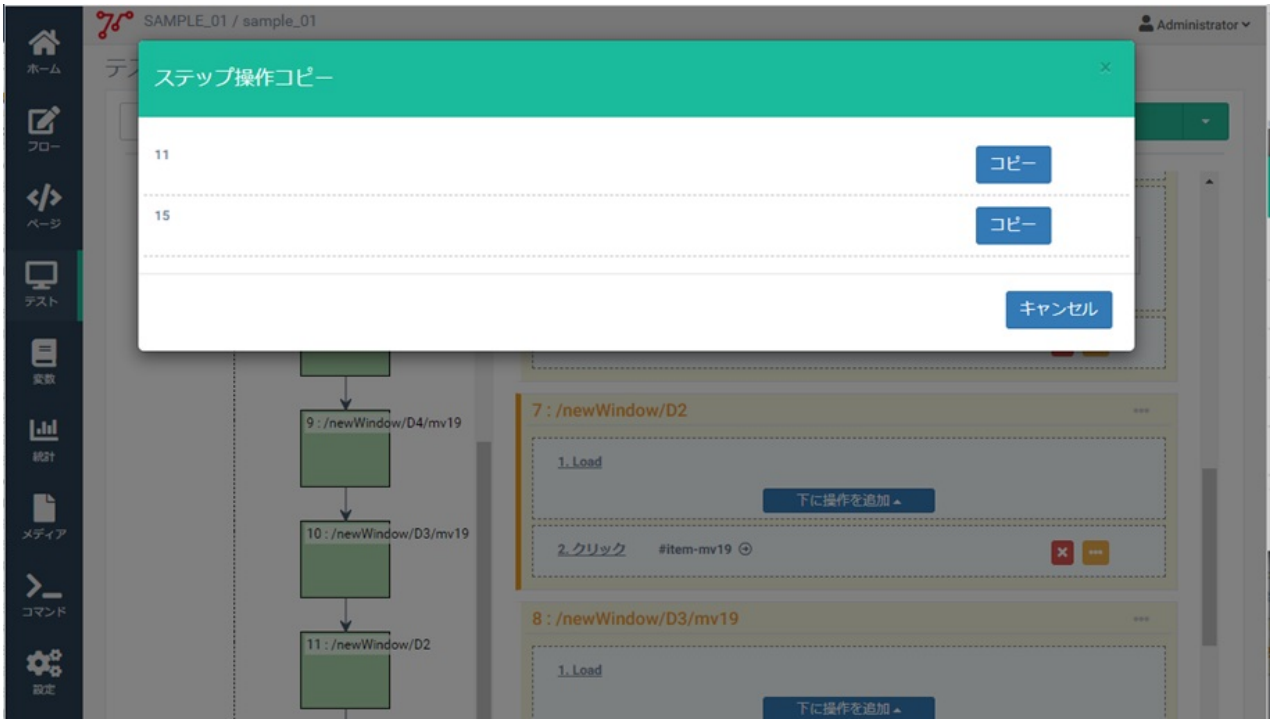


Fig. 5.2.4.4 ステップ操作コピーダイアログ

このダイアログでは、コピーを挿入したいステップと同じ画面のステップ番号が表示されます。
追加するステップ番号を選び、コピー ボタンをクリックするとテスト編集画面にステップがコピーされます。

5.2.4.5 コマンド追加

コマンドを起動したいステップの 操作追加プルダウンメニューから、**コマンド追加** を選択してください。
コマンド選択ダイアログが表示されます。
このダイアログでは、**コマンドメニュー**で登録している**外部コマンド**が表示されます。

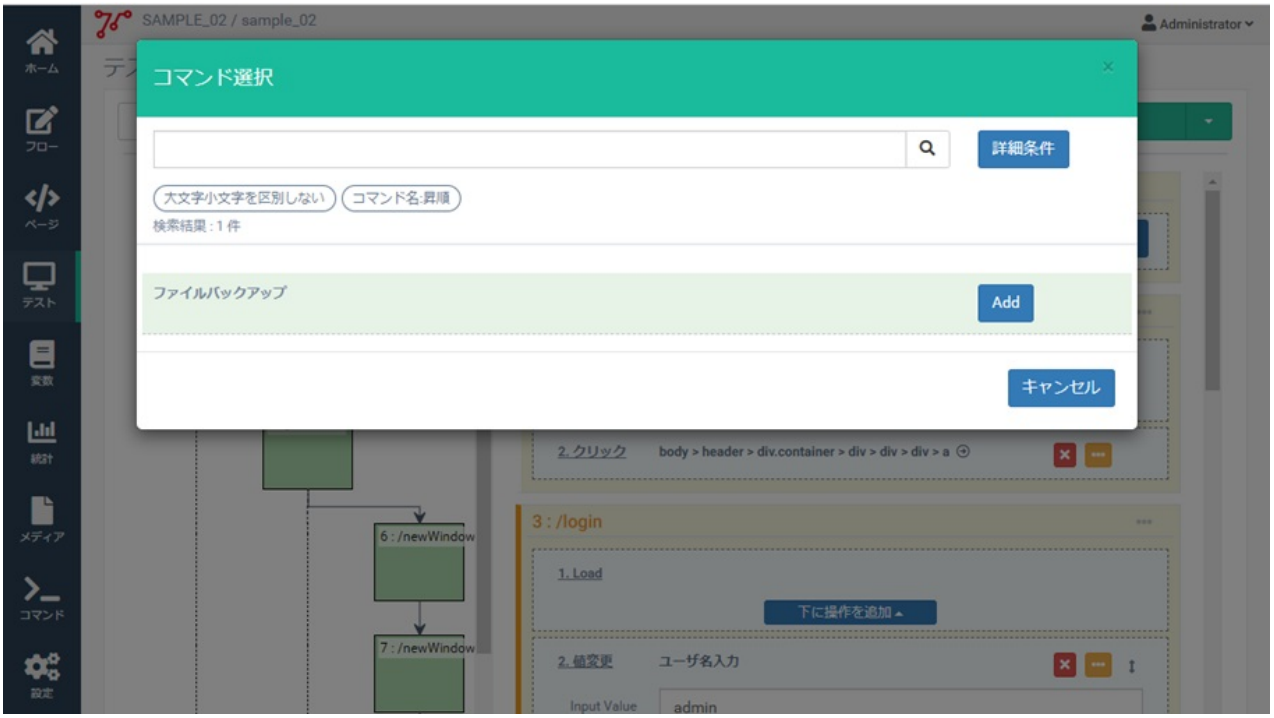


Fig. 5.2.4.5 コマンド選択ダイアログ

Addボタンをクリックするとテスト編集画面に外部コマンドが追加されます。
外部コマンドは続けて追加できます。ダイアログを閉じるには キャンセル ボタンをクリックしてください。

外部コマンドについては、9. コマンドメニューを参照してください。

5.2.4.6 スクリプト実行追加

操作追加プルダウンメニューから、**スクリプト実行追加** を選択してください。
スクリプト選択ダイアログが表示されます。
このダイアログでは、スクリプト実行追加を選択した**ページに定義されている** Javascript が表示されます。

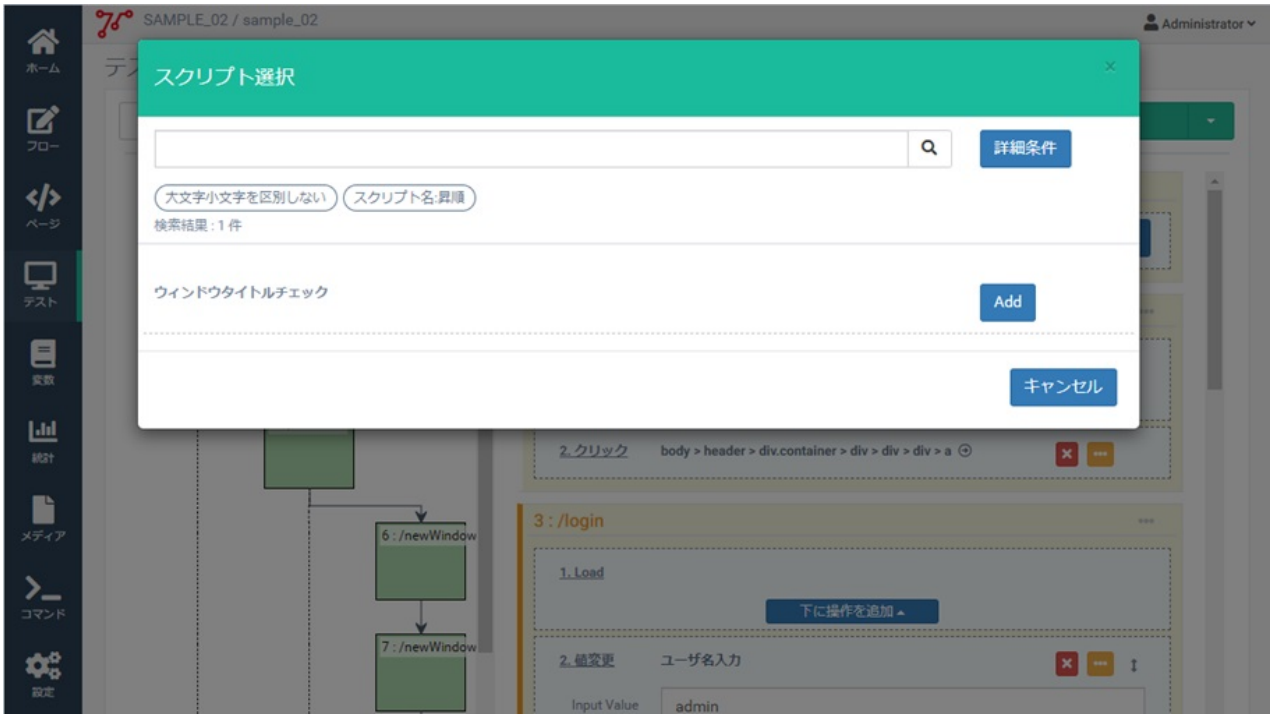


Fig. 5.2.4.6 スクリプト選択ダイアログ

Addボタンをクリックするとテスト編集画面に スクリプト が追加されます。
スクリプトは続けて追加できます。ダイアログを閉じるには キャンセル ボタンをクリックしてください。

スクリプトについては、ページメニュー » 4.2.4 スクリプト タブを参照してください。

5.2.4.7 テンプレートから追加

操作追加プルダウンメニューから、**テンプレートから追加** を選択してください。
テンプレートの読み込みダイアログが表示されます。
このダイアログからは、
・ファイルから読み込む
・テンプレートから読み込む
二つの方法でテンプレートを読み込むことができます。

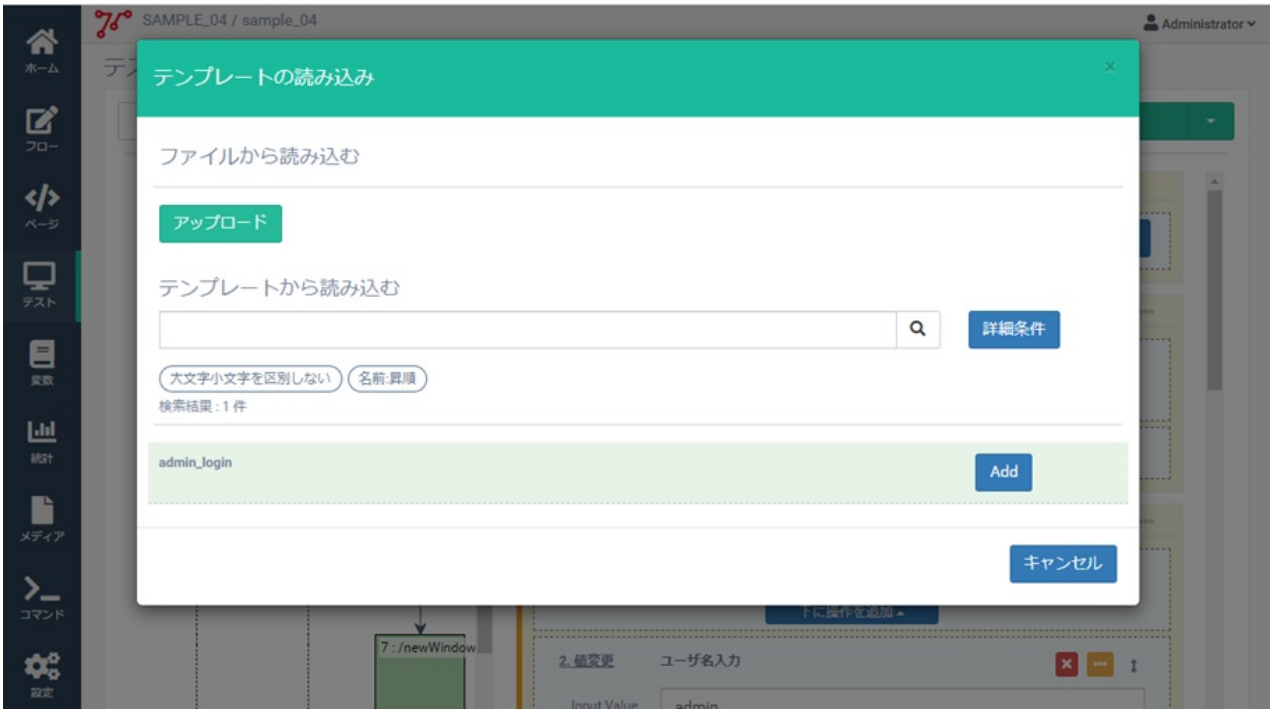


Fig. 5.2.4.7 テンプレートの読み込みダイアログ

ファイルから読み込むには、 **アップロードボタン** をクリックし、ページのテンプレートタブでエクスポートしたExcelファイル(※)を指定します。
(※) Excelファイルをインポートする際には、エクスポートしたファイルの**基本情報シートを削除し、操作シートのみにしておく必要があります。**

アップロードボタンの横に「読み込みに成功しました。」と表示されたら、キャンセルボタンをクリックしてテスト編集画面に戻ります。

テンプレートから読み込むには、ダイアログに表示されている **ページに定義されている** ページテストテンプレート から選択します。
Addボタンをクリックするとテスト編集画面に スクリプト が追加されます。
スクリプトは続けて追加できます。ダイアログを閉じるには キャンセル ボタンをクリックしてください。

ページテストテンプレートについては、ページメニュー » 4.2.5 テンプレート タブを参照してください。

5.2 テスト編集 画面 目次 / 5. テストメニュー 目次 に戻る

5.2.5 ステップ操作 プルダウンメニュー

メニューに表示される項目は、ステップの状況によって変化します。

5.2.5.1 一般操作のクリア

Loadや画面遷移を伴う操作などの特殊な操作を除く一般的な操作を削除することができます。

5.2.5.2 ページテンプレート作成

テンプレートに含めることができる操作がない場合は、このメニューは表示されません。
(注意)ページ遷移を伴う操作はテンプレートに含めることはできません。

ページテンプレート作成 を選択すると、新規テンプレート作成ダイアログが表示され、該当のステップをページテストテンプレートとして保存することができます。

Fig. 5.2.5.2 新規テンプレート作成ダイアログ

作成されたページテストテンプレートは、該当ページの テンプレート一覧画面で確認することができます。

5.2.5.3 ステップの削除

該当のステップを削除します。
ステップが**遷移で別画面につながっている場合**はこのメニュー項目は表示されません。

5.2 テスト編集 画面 目次 / 5. テストメニュー 目次 に戻る

5.2.6 項目操作プルダウンメニュー

メニューに表示される項目は、ステップの状況によって変化します。

5.2.6.1 これ以降をすべて削除

項目操作ボタンをクリックした操作とそれ以降のステップをすべて削除します。

5.2.6.2 項目をコピー

操作をコピーして追加します。
画面遷移を伴う操作の場合、このメニュー項目は表示されません。

5.2 テスト編集 画面 目次 / 5. テストメニュー 目次 に戻る

5.3 テストスイート一覧 画面

複数のテストをまとめたテストスイートを一覧表示する画面です。
各テストスイート名はリンクになっており、スイートごとの設定画面に移動できます。

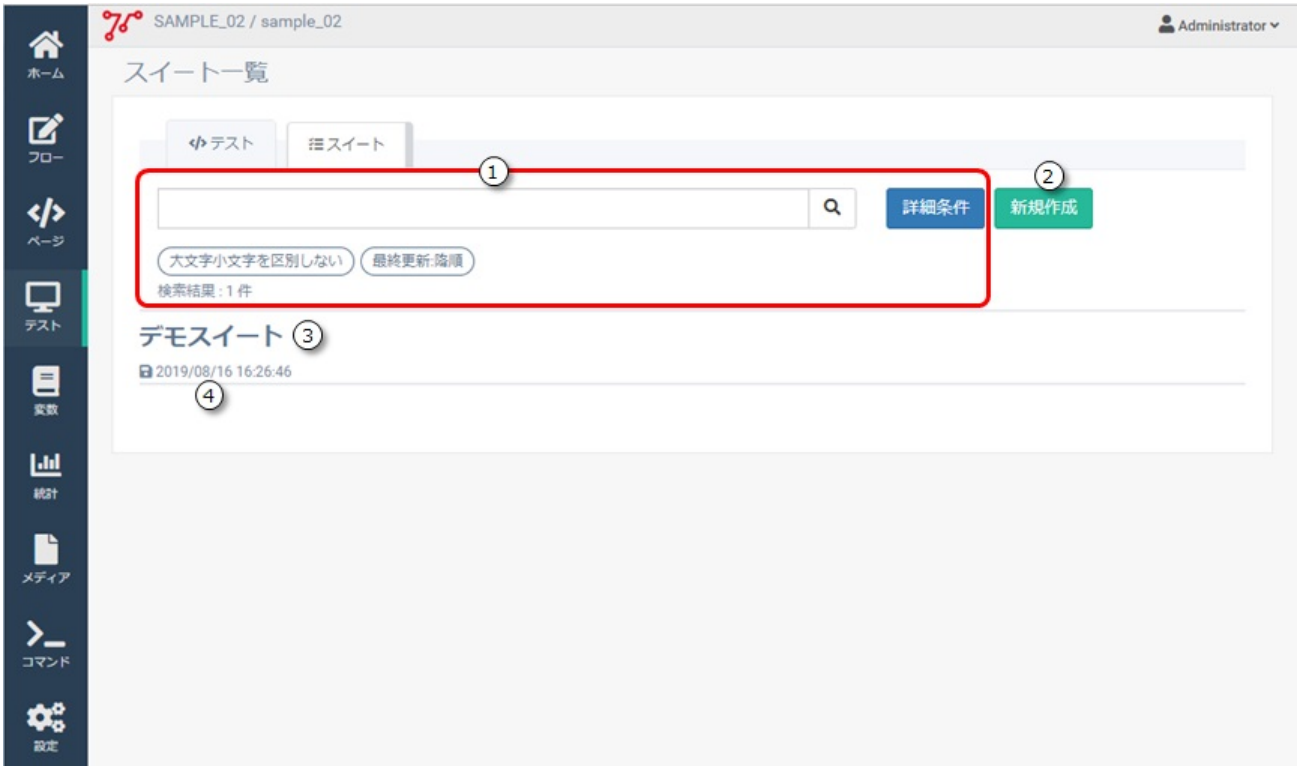


Fig. 5.3 テストスイート一覧 画面

No.	名称	説明
1	一覧検索・ソート	一覧の表示内容の絞り込みと検索とその結果表示です。 詳しくは、1.各画面共通»1.3 一覧表示での検索・ソート機能 を参照してください。
2	新規作成ボタン	新しいスイート作成ページに移動して、新しいスイートの編集画面を行います。
3	テストスイート名	クリックすると、そのスイート編集画面に移動します。
4	最終更新日時	

5. テストメニュー 目次 に戻る

5.4 テストスイート編集 画面

複数のテストをまとめたスイートを作成・編集する画面です。
この画面からは、作成するスイートの内容を設定します。スイートにテストを追加・削除・変更ができます。
作成したスイートのテストコードのダウンロードができます。

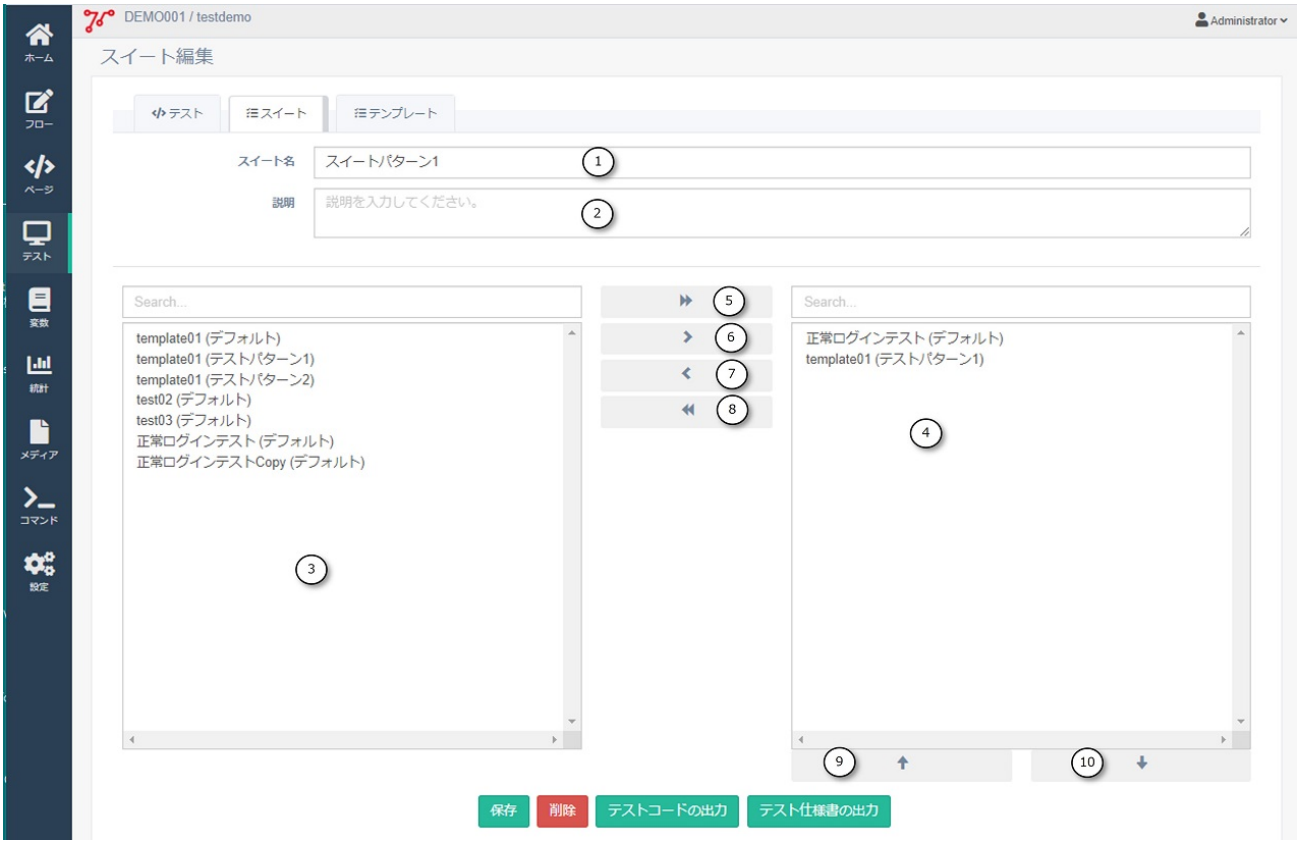


Fig. 5.4 テストスイート編集 画面

No.	名称	説明
1	スイート名	スイートの名前を入力します。
2	説明	スイートの説明を入力します。
3	テスト一覧	スイートに追加することができるテストの一覧です。 テストパターンがあるテストは全てのテストが表示されます。
4	スイートに追加されたテスト一覧	設定しているスイートに追加されているテストの一覧です。
5	スイートへ追加（全部）	表示されているテストをすべてスイートへ追加します。
6	スイートへ追加（選択）	選択しているテストをスイートへ追加します。
7	スイートから削除（選択）	選択しているテストをスイートから削除します。
8	スイートから削除（全部）	スイートに追加されているテストをスイートから削除します。

9	上へ移動	スイートに追加されたテスト一覧で選択しているテストを一つ上に移動します
10	下へ移動	スイートに追加されたテスト一覧で選択しているテストを一つ下に移動します
11	保存ボタン	クリックすると、作成したスイートの設定を保存します。
12	削除ボタン	クリックすると、設定しているスイートを削除します。
13	テストコードの出力	クリックすると、設定しているスイートの**テストコード**をダウンロードします
14	テスト仕様書の出力	クリックすると、設定しているスイートの**テスト仕様書**をダウンロードします

5. テストメニュー 目次 に戻る

5.5 テスト仕様書へのテスト実行結果のマージ

v1.4.0 より テスト仕様書にテスト実行結果をマージすることが可能になりました。
この機能を利用するためにはいくつかの設定を追加する必要があります。

なお、テストの**実行時**に結果をマージした仕様書が作成されるため、テスト編集画面 5.2.3 テストコード出力プルダウンメニュー の **"テスト仕様書の出力"、"テスト仕様書の出力(全パターン)" で出力したテスト仕様書**はマージ機能の **対象外**です。

テスト実行結果をマージしたテスト仕様書を作成するためには、testablish-test.ini ファイルに下記のパラメータを追加します。

key	説明
[testablish]	
testSpecificationMerge	テスト仕様書マージを行うかどうかを true/false で設定します。
testSpecificationMergeType	テスト仕様書マージをするファイルタイプ "excel" または "word"で指定します。 両方の場合は"excel,word"と指定します。
[各種ブラウザセクション]	
tester	テスト実施者 テスト仕様書の「実施者」フィールドに設定されます。 設定がない場合は空白になります。

ini ファイルについての詳細は III. 操作の流れ/機能説明 » 2.テストの自動実行 » 2.1.2 testablish-test.ini の項目詳細 を参照してください。

テスト仕様書マージ設定 (testSpecificationMerge) が true になっている場合、テスト仕様書の「実施者」「実施日」「結果」にそれぞれ値が設定されます。

テスト実行後、テスト結果がマージされたテスト仕様書は、以下のディレクトリに出力されます。
<展開したテストコードフォルダ>**/testDocument/result/**

出力される仕様書のファイル名は以下の形式となります。(testablish-test.ini ファイルに指定されていない値は付加されません。)
[browser.name]_[browser.version]_[browser.platform]_<テスト名>_<出力日時>.docx
[browser.name]_[browser.version]_[browser.platform]_<テスト名>_<出力日時>.xlsx

テスト結果がマージされたテスト仕様書の詳細は III. 操作の流れ/機能説明 » 2.テストの自動実行 » 2.2 テストレポートを確認する を参照してください。

5. テスト メニュー 目次 | II. 各画面の説明 目次 | 使い方マニュアル 目次 に戻る

6. 変数 メニュー

6.1 変数機能 概要

Testablishでは、変数を扱うことができます。
変数とは、箱のように「値を入れておくもの」で、名前を付けて作成・利用します。
変数は「**変数**」メニューで定義を作成し、テストシナリオ内で使用します。

変数を使用すれば、自由に値を保持・書き換え・参照することができます。

Testablishでは、値を入れる(代入)と参照ができる「**変数**」と、 参照のみができる「**定数**」を作成・利用することができます。
テストシナリオ内で、変数は何度でも値の代入が可能ですが、定数は定義時に指定した値を参照することしかできません。

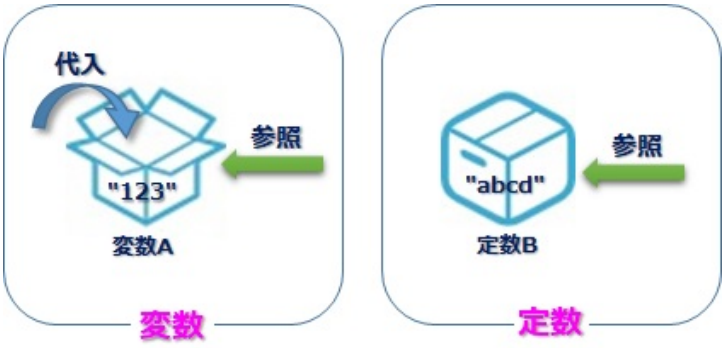


Fig. 6.1-1 変数と定数

変数にはスコープ（有効範囲：使用できる範囲）を指定します。
Testablishでは、以下の3種類のスコープがあります。

- **ページ**　テストシナリオ内のステップ内のみで有効になります。
- **テスト**　テストシナリオ内で有効になります。
- **スイート**　スイートとして設定している複数のテストシナリオで有効になります。

変数の場合は有効な範囲の中であればどこからでも値の代入・参照ができます。
定数の場合はスコープの限定はなくどこからでも値の参照が可能ですが、値の代入はできません。

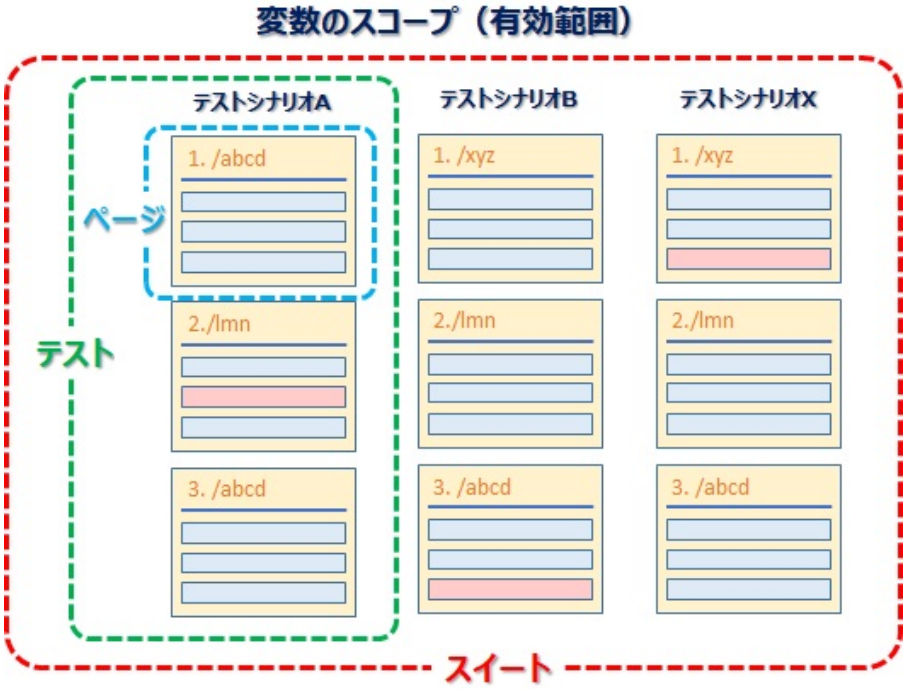


Fig. 6.1-2 変数のスコープ

6.2 変数一覧画面

変数一覧画面は、作成した変数の一覧を表示する画面です。
各変数名はリンクになっており、変数ごとの設定画面に移動できます。
この画面から、新たに変数を追加することができます。

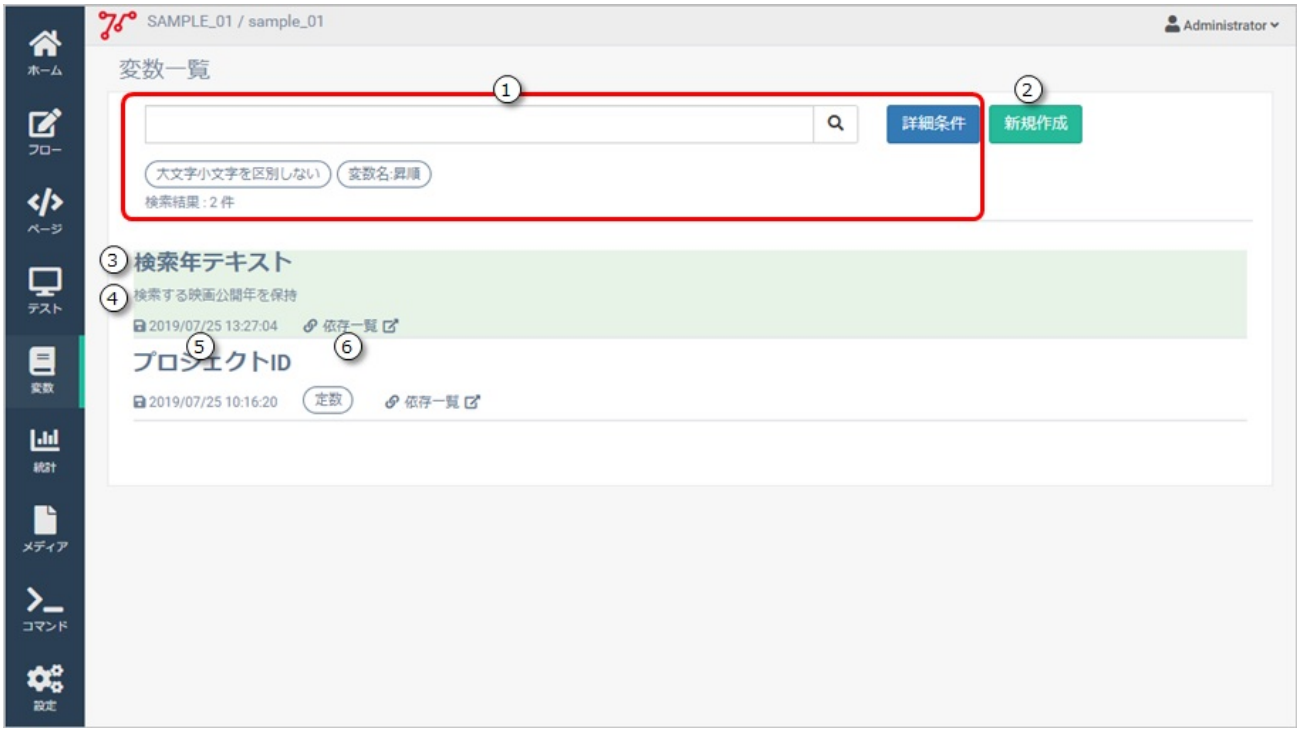


Fig. 6.2 変数一覧画面

No.	名称	説明
1	一覧検索・ソート	一覧の表示内容の絞り込みと検索とその結果表示です。 詳しくは、1.各画面共通»1.3 一覧表示での検索・ソート機能 を参照してください。
2	新規作成ボタン	変数編集画面に移動し、新しい変数を定義することができます。
3	変数名	クリックすると、その変数の変数編集画面に移動します。
4	説明	編集画面で設定されている説明を表示します。
5	最終更新日時	
6	依存一覧リンク	このページを使用しているテストの一覧画面を別ウィンドウで開きます。 詳しくは、1.各画面共通»1.4 依存一覧リンク を参照してください。

6.3 変数編集画面

変数編集画面は、選択した変数を編集する画面です。
変数 または **定数** を作成することができます。

変数 は 設定したスコープ（有効範囲）内において、**参照** と **代入** を行うことができます。
定数 は どこからでも**参照**を行うことができます。
定数を作成して、値を外部ファイル化することも可能です。詳しい方法については、 III. 操作の流れ/機能説明 » 3.変数の外部ファイル化とデータのマスク を参照してください。

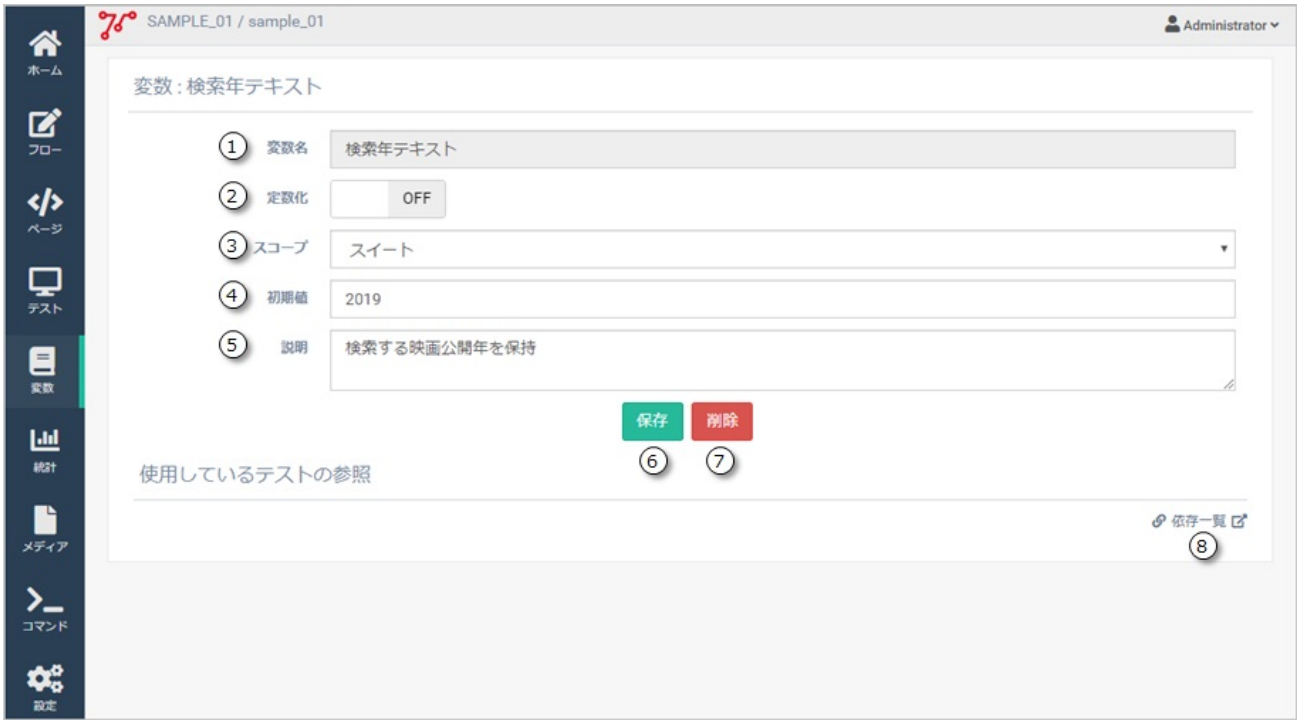


Fig. 6.3 変数編集画面

No.	名称	説明
1	変数名	作成する変数の名称を設定します。 変数名には、以下の文字を含むことはできません。 「" (ダブルクォーテーション)」、「\ (バックスラッシュ)」、「/ (スラッシュ)」、「\b (バックスペース)」、「\f (改ページ)」、「\n (改行)」、「\r (キャリッジリターン)」、「\t (タブ)」
2	定数化	ON に設定すると**定数**として作成されます。初期値は OFFです。
3	スコープ	変数のスコープ(有効範囲)を設定することができます。 スコープは「**ページ**」「**テスト**」「**スイート**」の3種類から選択できます。 定数の場合、スコープは設定できません。
4	初期値	変数の初期値を定義することができます。 定数化がON の場合は、ここに設定した値が定数値になります。
5	説明	作成した変数の説明を設定します。
6	保存ボタン	作成した変数の設定を保存します。
7	削除ボタン	作成した変数を削除します。

8	依存一覧リンク	このページを使用しているテストの一覧画面を別ウィンドウで開きます。 詳しくは、1.各画面共通»1.4 依存一覧リンク を参照してください。

6.4 変数機能の利用

変数機能を利用する場合、

1. 変数メニューで、変数を定義する
2. テスト編集画面でテストシナリオに操作を追加して、変数を利用する

といった手順になります。

以下の映画検索画面と変数を例に、変数の利用方法を説明していきます。
対象の映画検索画面には、「映画公開年」入力フィールドと「検索」ボタンがあります。

Fig. 6.4.1-1 対象の画面：映画検索画面

変数メニューで、「検索年テキスト」という名称で、検索する映画公開年を保持する目的の変数を作成しておきます。

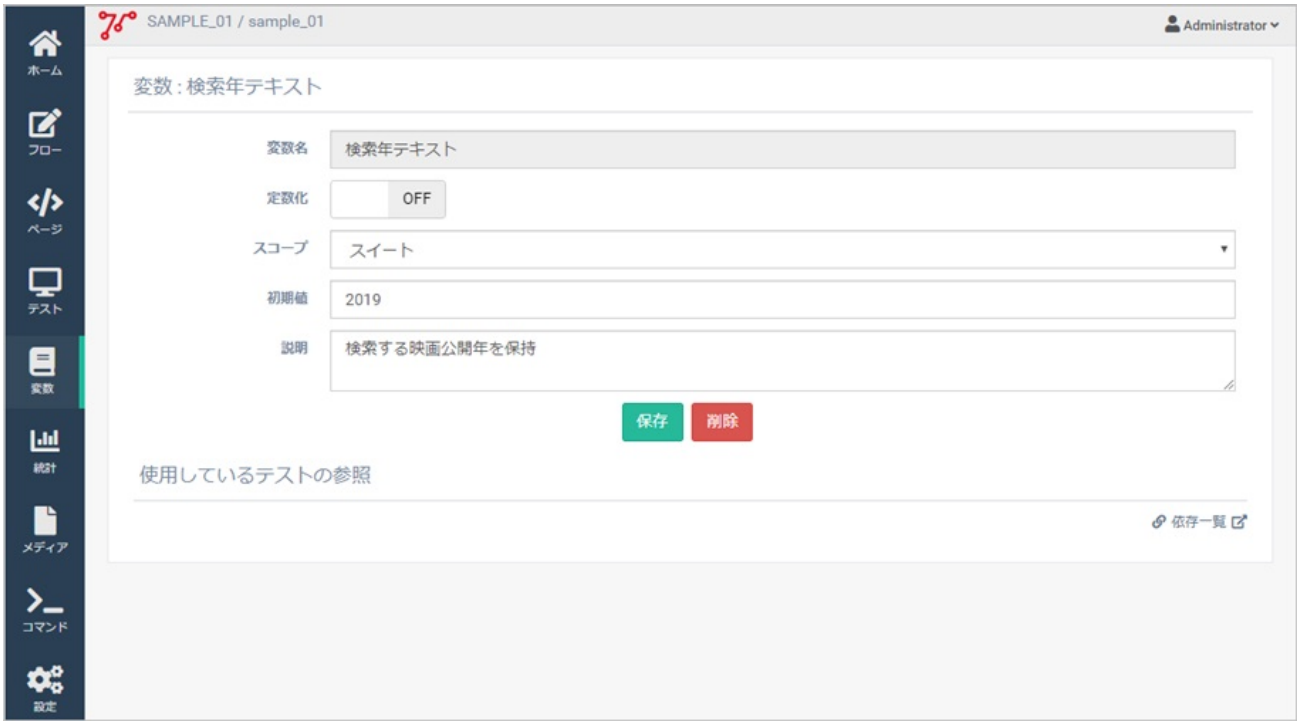


Fig. 6.4.1-2 変数サンプル

6.4.1 変数への代入と値の参照

変数への値の ”代入” と ”参照” を確認するためのテストを作成していきます。
※ 説明文中の(1)~(4)は、説明後半の「変数サンプル」テストを実行した際の画面表示画像の番号に対応しています。

この例では、映画検索画面の 「映画公開年」という操作(入力フィールド)で変数を利用します。
下に操作を追加 のプルダウンから **操作追加** をクリックしてください。

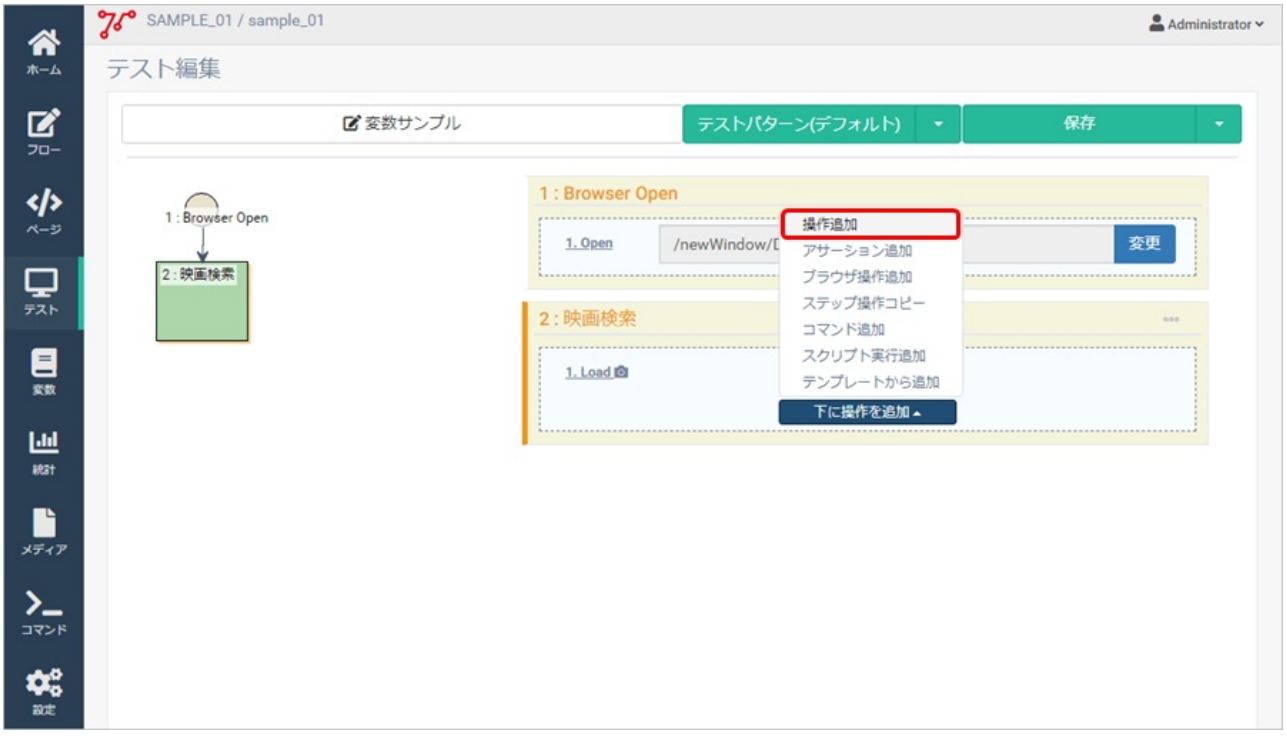


Fig. 6.4.1-1 操作追加

表示される追加項目選択ダイアログで、「映画公開年」の「値変更」を選択し、Add ボタンでテストシ

ナリオに操作を追加します。
Input Value の値 に "2018" を設定しておきます。 ...(1)

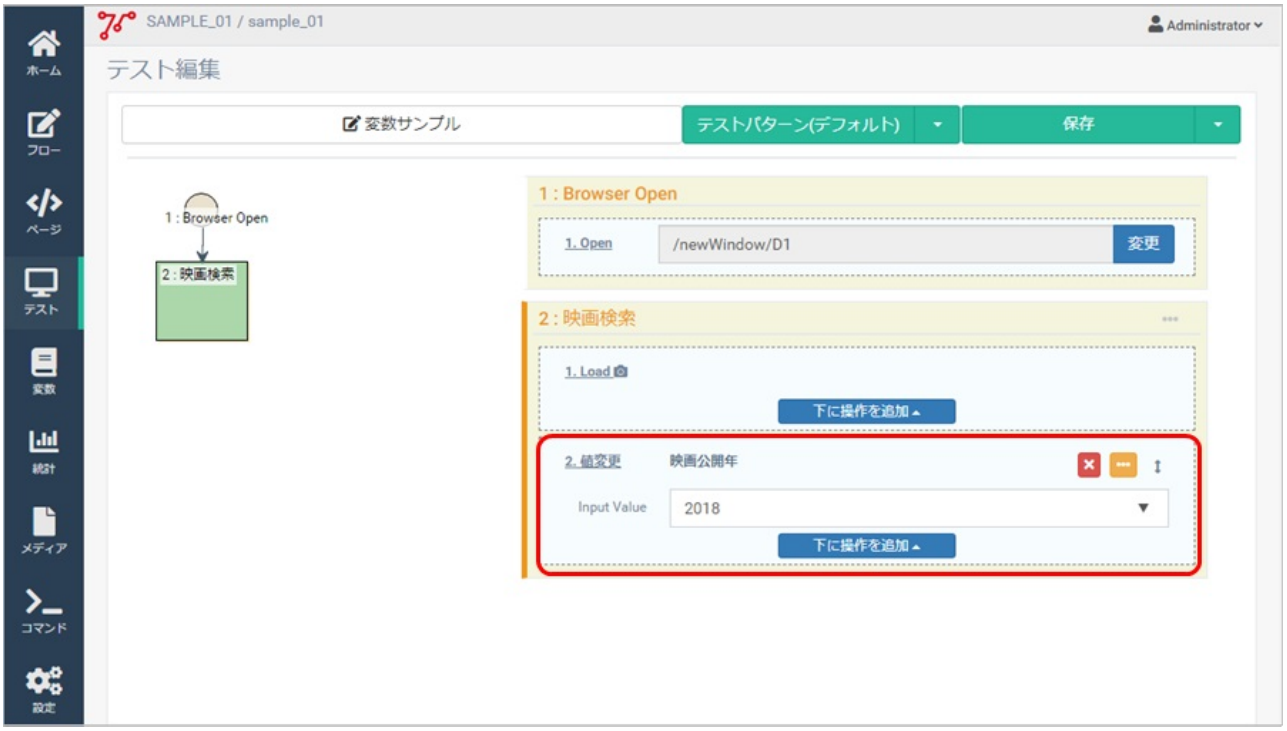


Fig. 6.4.1-2 操作追加：値変更

次に、「映画公開年」フィールドに設定されている値を変数に代入していきます。

下に操作を追加 のプルダウンから **操作追加** をクリックし、 表示される追加項目選択ダイアログで、「映画公開年」の「**変数代入**」を選択し、Add ボタンでテストシナリオに操作を追加します。

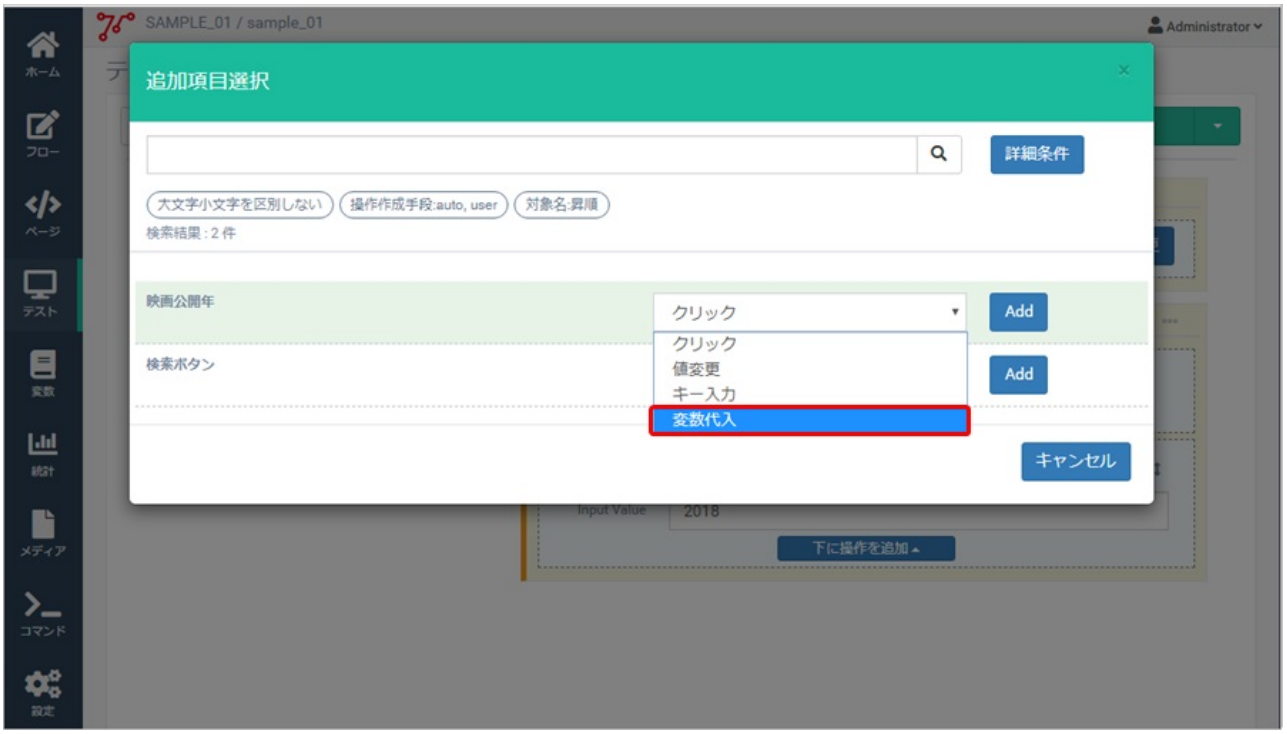


Fig. 6.4.1-3 操作追加：変数代入

「変数代入」として操作が追加されています。
ここで、「変数名」に 事前に変数メニューで定義しておいた**変数**「検索年テキスト」を設定します。

ここで、**変数**「検索年テキスト」に「映画公開年」フィールドの値が**代入**されています。 **...(2)**

Fig. 6.4.1-4 変数代入

変数の参照が分かりやすいように、操作を追加し、一旦、画面の「映画公開年」フィールドの値を "２０１９" に変更しておきます。 **...(3)**

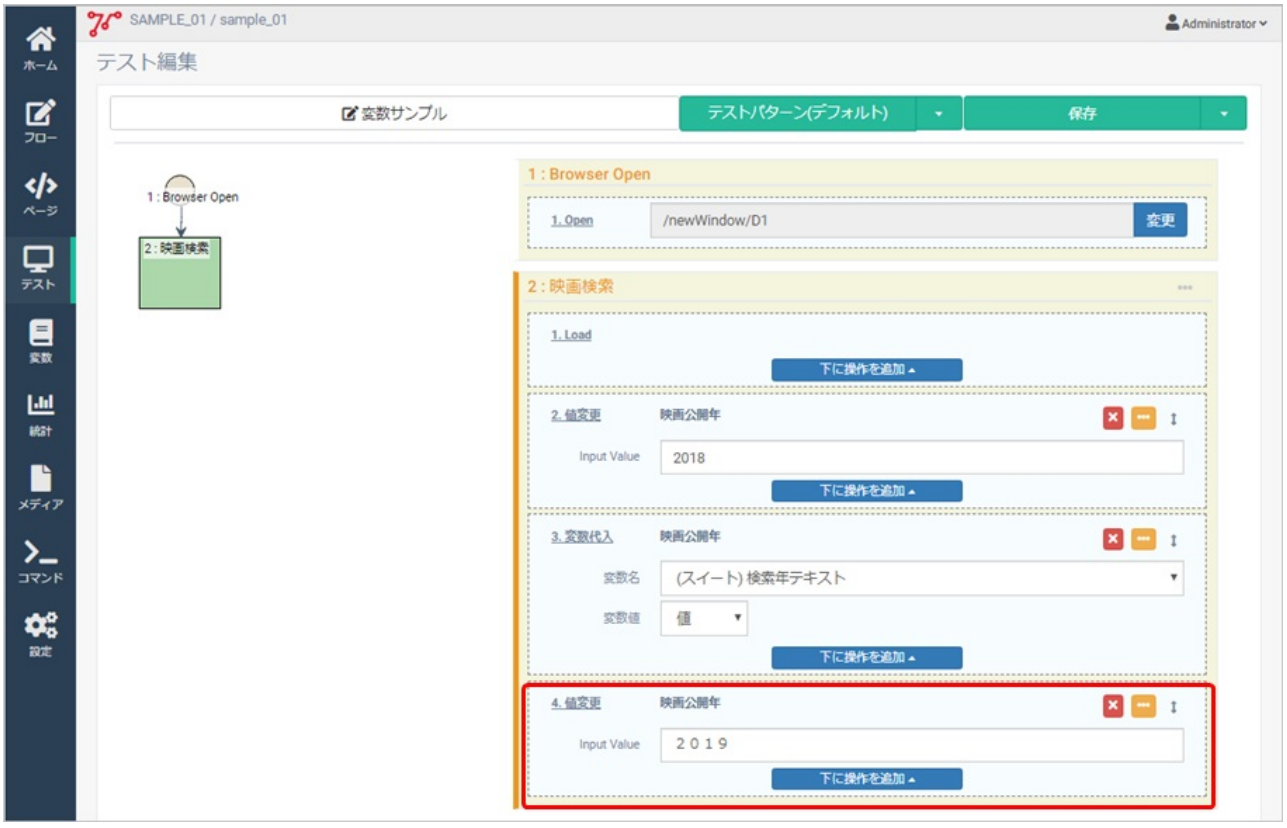


Fig. 6.4.1-5 値変更

次に、**変数**「検索年テキスト」の値を参照し、その値を画面の「映画公開年」フィールドに設定していき

ます。
操作を追加し、画面の「映画公開年」フィールドに "**${検索年テキスト}**" と設定します。
画面のフィールドには、**変数**「検索年テキスト」の値である"2018"が表示されます。 ...(4)

このように、変数に保持されている値を**参照**するには、

${<変数名>**}**

というように、**${}** で変数名を囲んで記述します。

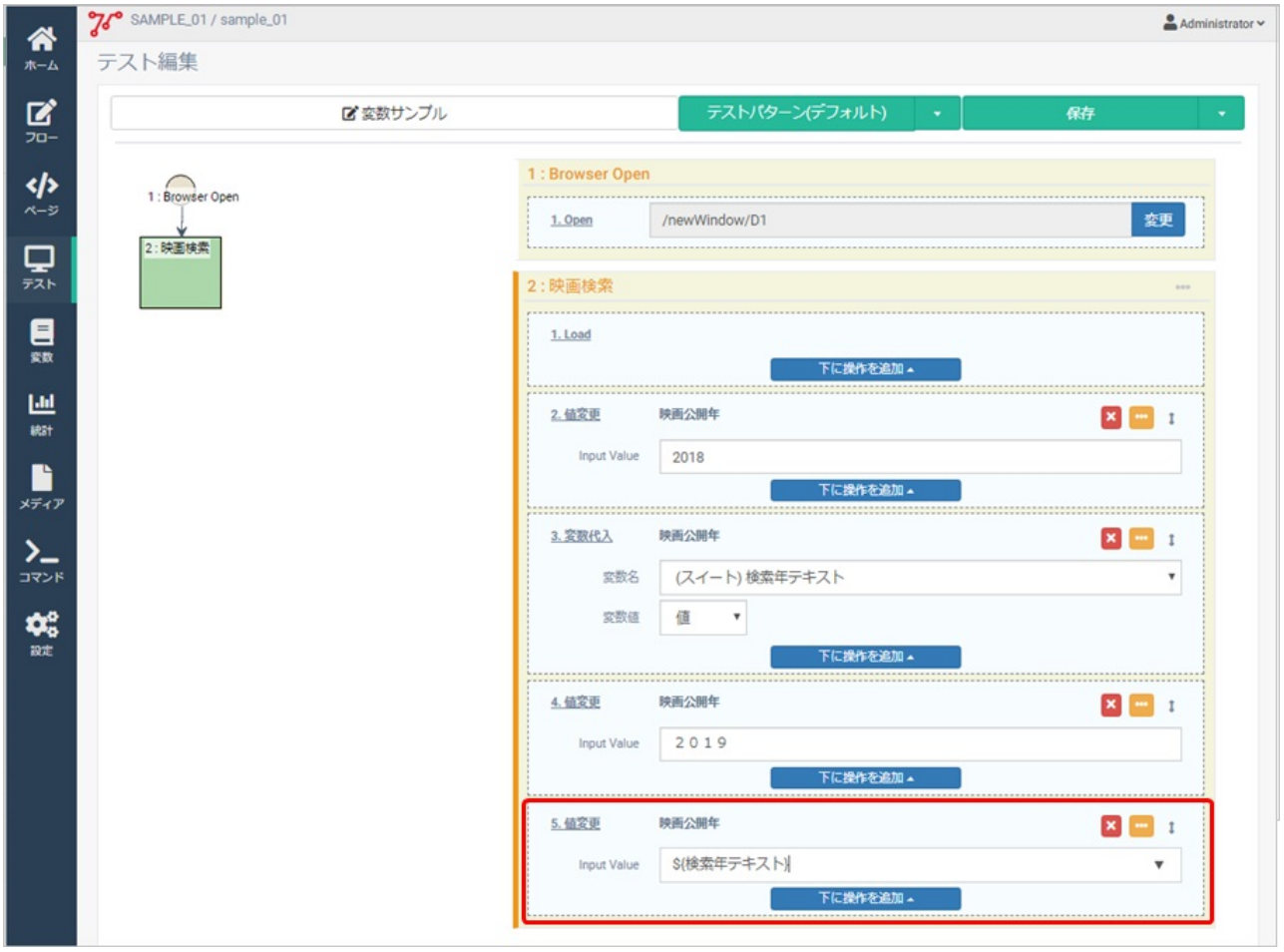


Fig. 6.4.1-6 値の参照

この「変数サンプル」というテストを実行すると、画面は以下のように表示されていきます。

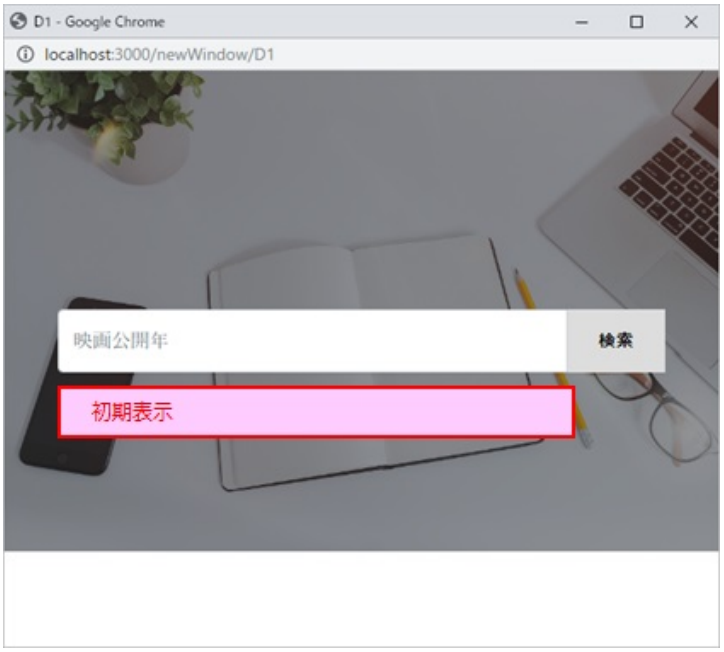


Fig. 6.4.1-7 初期表示

(1)

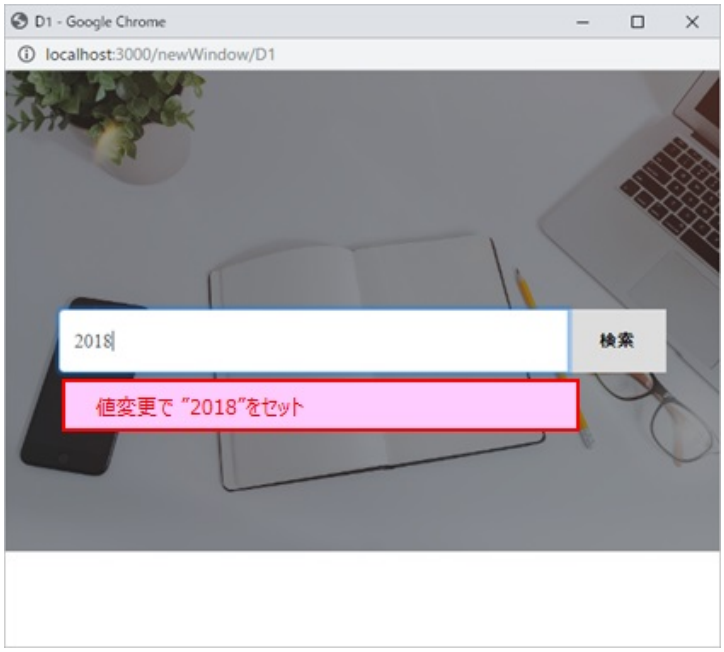


Fig. 6.4.1-8 値変更

(2) 操作「変数代入」

(3)

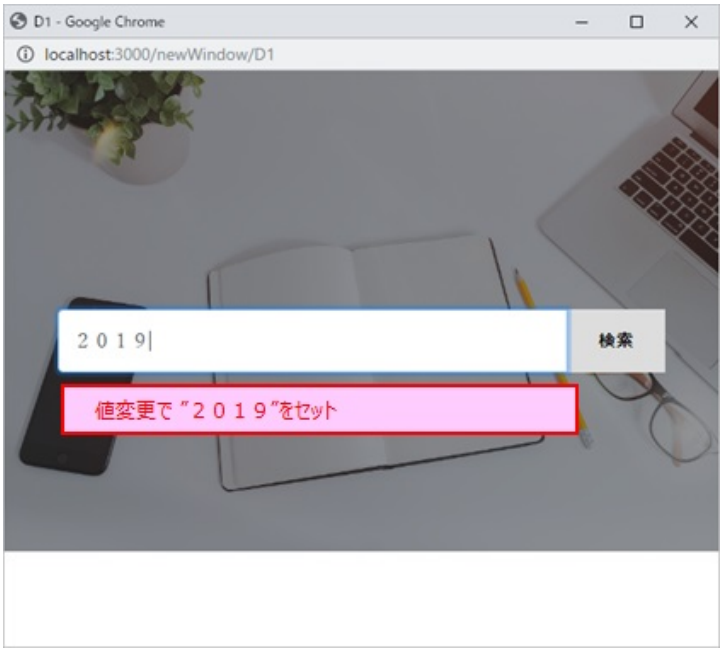


Fig. 6.4.1-9 値変更

(4)

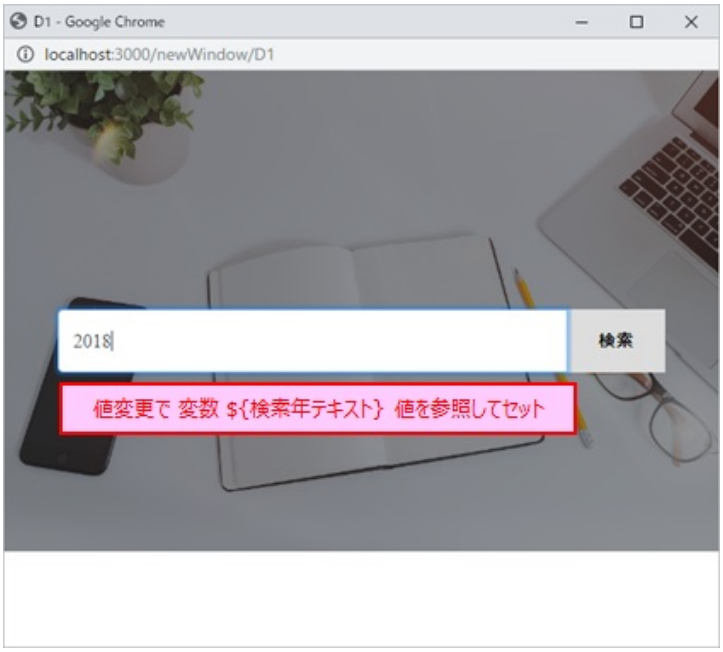


Fig. 6.4.1-10 変数の値をセット

6. 変数 目次に戻る ｜ II. 各画面の説明 目次 ｜ 使い方マニュアル 目次 に戻る

7. 統計 メニュー

- 7.1 統計情報 画面

7.1 統計情報 画面

テスト統計情報画面は、プロジェクトのテストの統計情報を表示する画面です。

Webアプリに対するテストの、画面使用率・操作使用率・遷移使用率・操作後処理使用率が確認できます。
各使用率の数値は、Testablishの **プロジェクトに登録された操作**のうち、実際に**テストシナリオに使用されている割合**を表します。
(テスト対象Webアプリケーションに存在していても、Testablishの プロジェクトに登録されていない画面(ページ)や操作は、統計情報の対象にはなりません。)

フローグラフから画面をクリックすると、ページ右側にその画面ごとの各使用率を見ることができます。

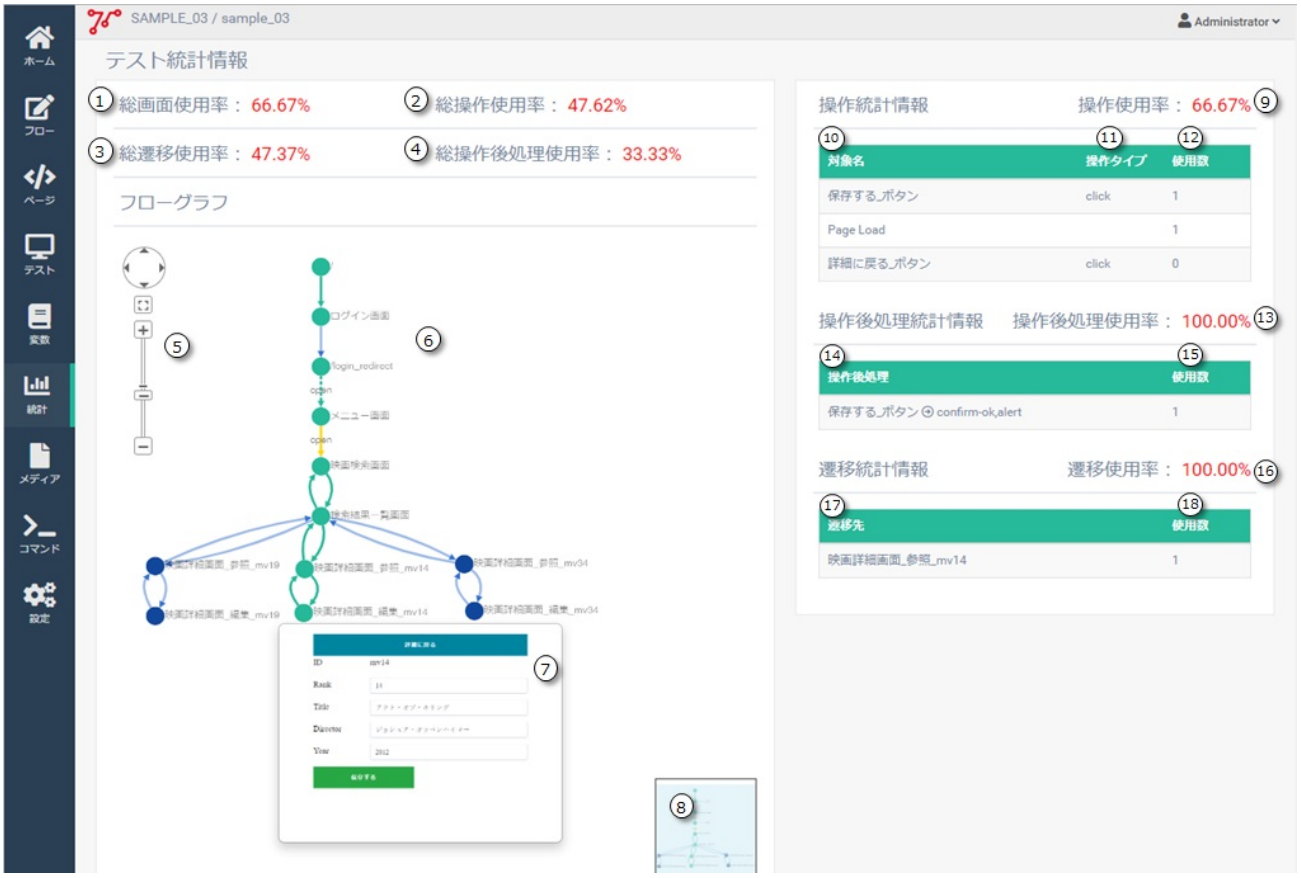


Fig. 7.1 統計情報画面

No.	名称	説明
【テスト統計情報】		
1	総画面使用率	プロジェクトに登録されている画面の総使用率を表示します。
2	総操作使用率	プロジェクトに登録されている操作の総使用率を表示します。
3	総遷移使用率	プロジェクトに登録されている画面に設定されている遷移の総使用率を表示しま
4	総操作後処理 使用率	プロジェクトに登録されている画面に設定されている、 操作後の処理の総使用率を表示します。
【フローグラフ】		
	●	登録されている画面のうち、テストで未使用の画面を表します。

		登録されている画面のうち、テストで使用されている画面を表します。
		画面に登録されている(複数の）遷移が、テストですべて未使用の状態です。
		画面に登録されている(複数の）遷移が、テストで一部使用されている状態です。
		画面に登録されている(複数の）遷移が、テストですべて使用されている状態で
5	操作スライダー	フローグラフの表示を操作します。
6	フローグラフ	プロジェクトに登録されているすべての操作を繋げた 全体フローグラフを表示します。
7	画面サムネイル画像	フローグラフの画面(●)をクリックすると、 キャプチャされた画面画像がある場合は、そのサムネイル画像を表示します。
8	俯瞰図	現在のフローグラフの表示が、全体のフローグラフのどの部分かを表示します。
【操作統計情報】		
9	操作使用率	
10	対象名	その画面の操作部品の名前を表示します。
11	操作タイプ	その画面の操作部品の操作タイプを表示します。
12	使用数	その画面の操作部品の使用回数を表示します。
【操作後処理統計情報】		
13	操作後処理使用率	その画面から行われる操作後の処理の使用率を表示します。
14	操作後処理	その画面から行われる操作後の処理の名称を表示します。
15	使用数	その画面から行われる操作後の処理の使用数を表示します。
【遷移統計情報】		
16	遷移使用率	その画面から行われる画面遷移の使用率を表示します。
17	遷移先	その画面から遷移する画面を表示します。
18	使用数	その画面から遷移する画面の使用数を表示します。

7. 統計 メニュー 目次 | II. 各画面の説明 目次 | 使い方マニュアル 目次 に戻る

8. メディア メニュー

8.1 メディア 画面

メディア画面は、**ファイルのアップロード操作を含むテストで使用するファイルを事前に登録しておく**画面です。
実施したいテストのシナリオに「ファイルをアップロードする」という操作が含まれる場合、 この画面にあらかじめ登録しておくことでファイルをテストのアップロード操作に使用できるようになります。

Fig. 8.1 メディア画面

No.	名称	説明
1	ファイルアップロード欄	クリックするとファイル選択ダイアログを表示します。 この欄にファイルを直接ドラッグ＆ドロップすることもできます。 登録したファイルは、アイコンとファイル名が表示されます。 登録したファイルは、 テスト編集画面でアップロード操作の対象ファイルとして 使用できるようになります。
2	登録ファイル サムネイル	登録したファイルのサムネイルが表示されます。
3	登録ファイル名	登録したファイルの名前です。ファイル名を変更する場合、 このテキストボックスから変更してください。
4	登録ファイル操作プルダウン	登録したファイルに対する操作を行うプルダウンメニューを表示します
5	名前を変更	(3)に入力した名前にファイル名を変更します。
6	削除	ファイルを削除します。
7	ダウンロード	ファイルをダウンロードします。

8.2 メディア機能の利用

実施したいテストのシナリオに「ファイルをアップロードする」という操作が含まれる場合、

1. アップロードするファイルをメディアメニューで登録しておく
2. ファイルアップロードをおこなう操作をページに定義する
3. テストシナリオのファイルアップロード操作を追加する

といった手順が必要です。

8.2.1 メディアファイルの登録

メディア画面で、アップロードフォーム にファイルをドラッグ＆ドロップすることで、ファイルが登録されます。
または アップロードフォーム をクリックして表示されるファイル選択ウィンドウからファイルを選択して登録することもできます。

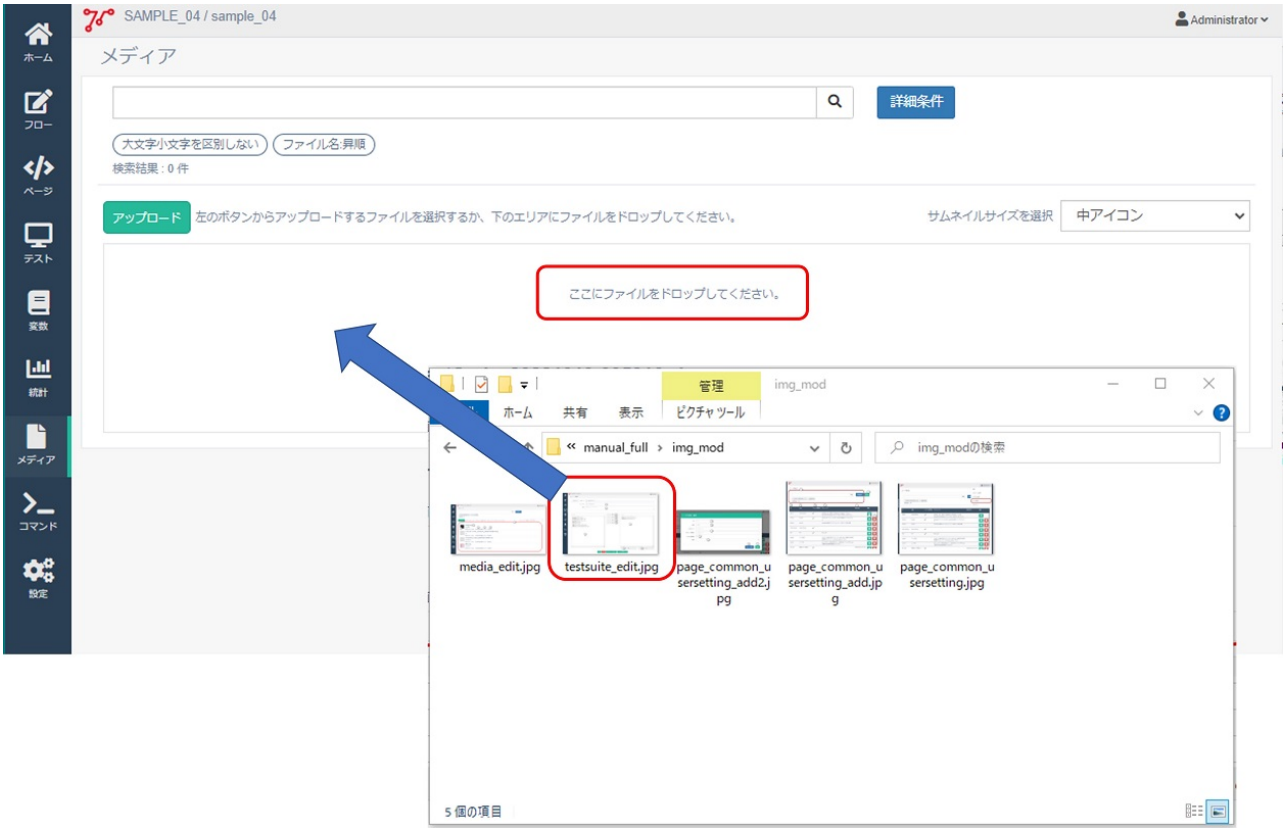


Fig. 8.2.1-1 メディア画面でファイルをドラッグ＆ドロップ

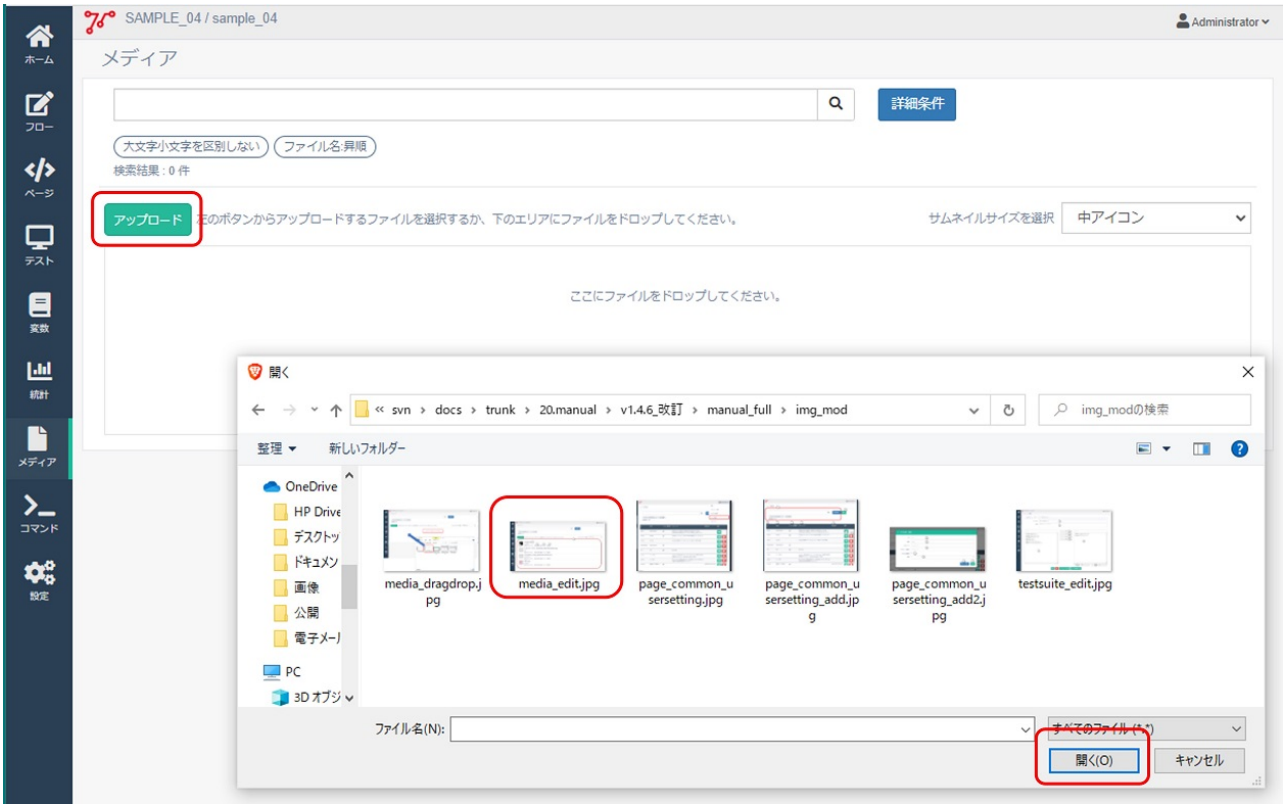


Fig. 8.2.1-2 メディア画面でファイルを選択

アップロードしたファイルは画面の下部に表示されます。
アップロード後には名前の変更、削除、ダウンロードが可能です。

Fig. 8.2.1-3 メディア画面で登録されたファイル

8.2.2 ページでのファイルアップロード操作の定義

ページの操作を定義する際に、**操作タイプを「値変更」、入力タイプを「file」** にすることで、サーバーにファイルを送信する操作を設定することができます。

Fig. 8.2.2 ファイルアップロード操作を定義

8.2.3 テストへのアップロード操作の追加

テスト編集画面では、ファイルアップロード操作が登録されているページのステップで、「下に操作を追加」ボタンから該当の操作を追加します。
参照 ボタンが表示され、テスト時にアップロードするファイルを選べるようになります。

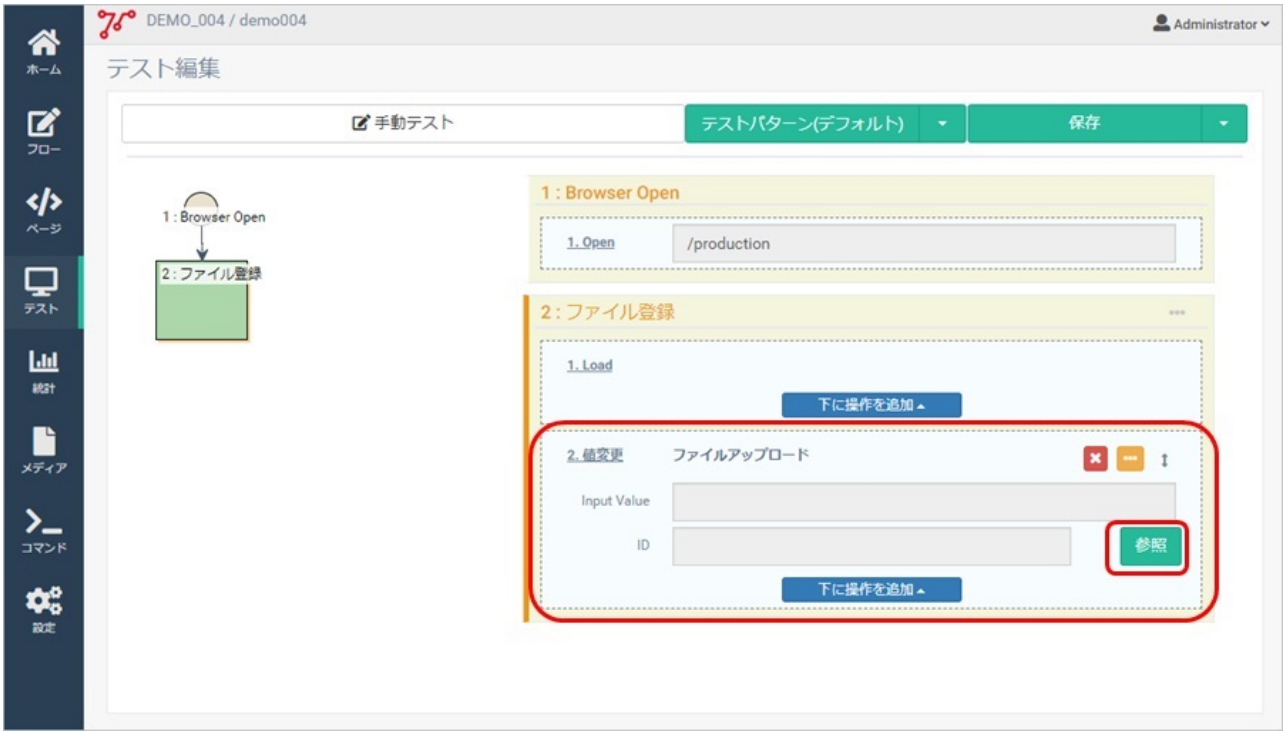


Fig. 8.2.3-1 テストにファイルアップロード操作を追加

参照ボタンをクリックすると、メディア画面で登録したファイルが一覧表示されます。

利用するファイルを **チェック** し、**選択** ボタンをクリックしてください。

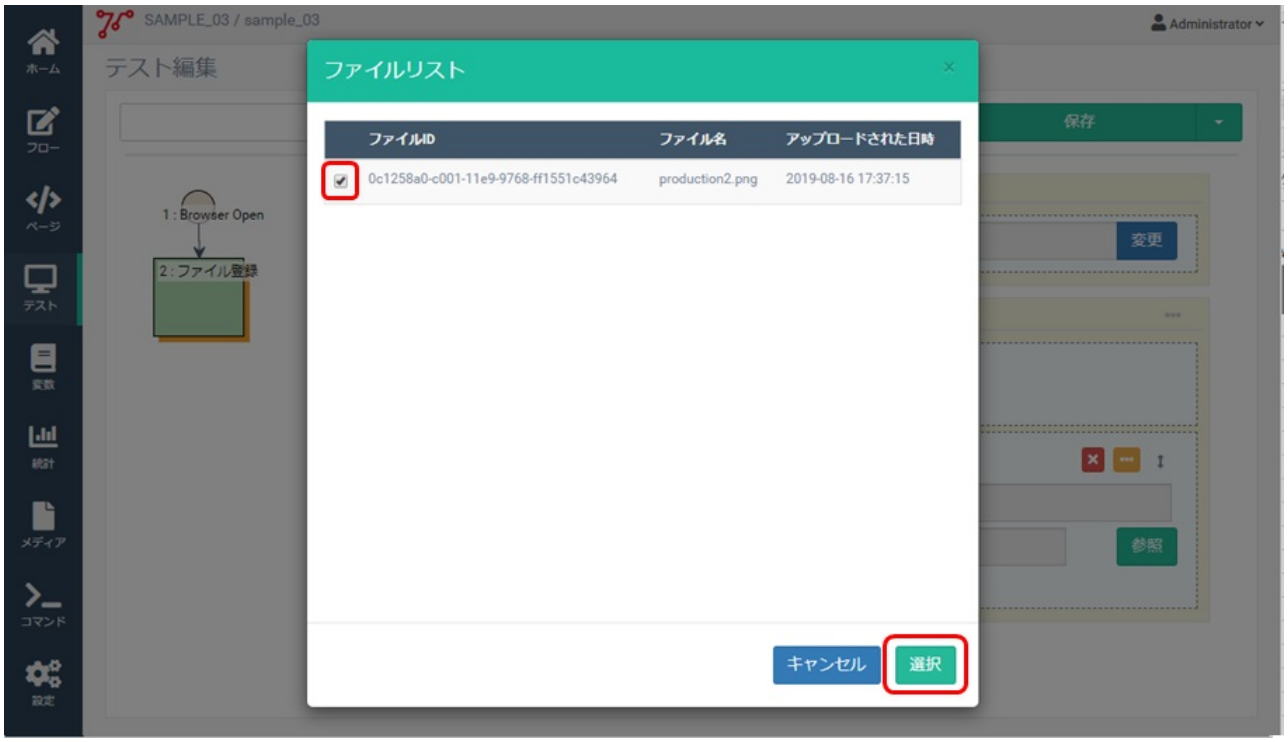


Fig. 8.2.3-2 アップロードするファイルを選択

テスト編集画面に選択したファイル名が表示されます。

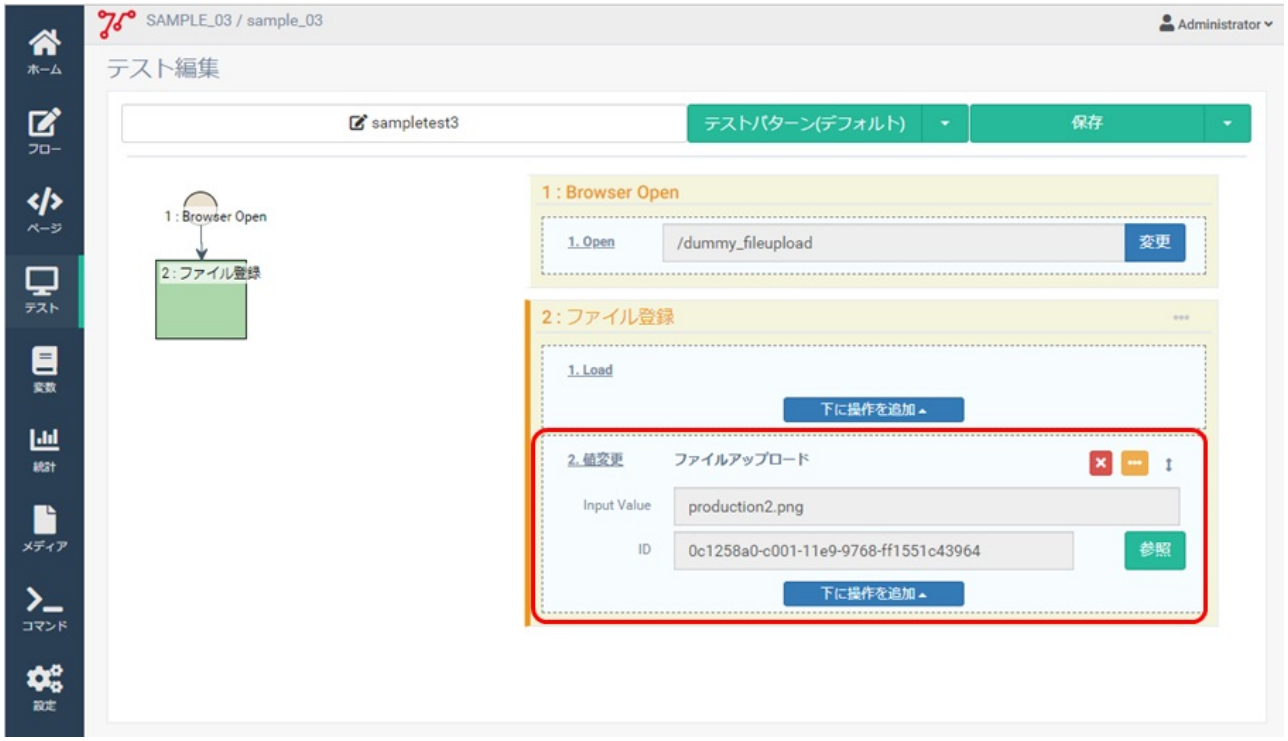


Fig. 8.2.3-3 アップロードファイル選択完了

ファイルアップロード操作を含むテストを作成してテストコードを出力した場合、 出力したテストコードのzipファイルを展開すると、mediaFolder の中にアップロードしたファイルが配置されています。

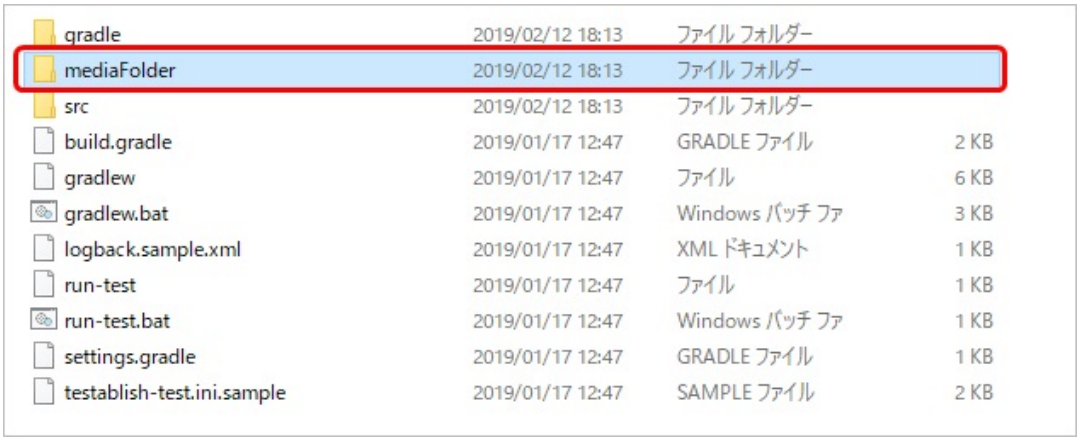


Fig. 8.2.3-4 アップロードファイル出力確認

8. メディア メニュー 目次 | II. 各画面の説明 目次 | 使い方マニュアル 目次 に戻る

9. コマンド メニュー

9.1 外部コマンド機能 概要

外部コマンド機能は、あらかじめ作成しておいたコマンドファイルを登録し、テスト実行時にその外部コマンドを起動するための機能です。
例えば、WebアプリケーションのDBのレコードの値を初期化したり、操作の前にバックアップファイルを作成するなど、 外部コマンドの記述次第でさまざまに利用できます。

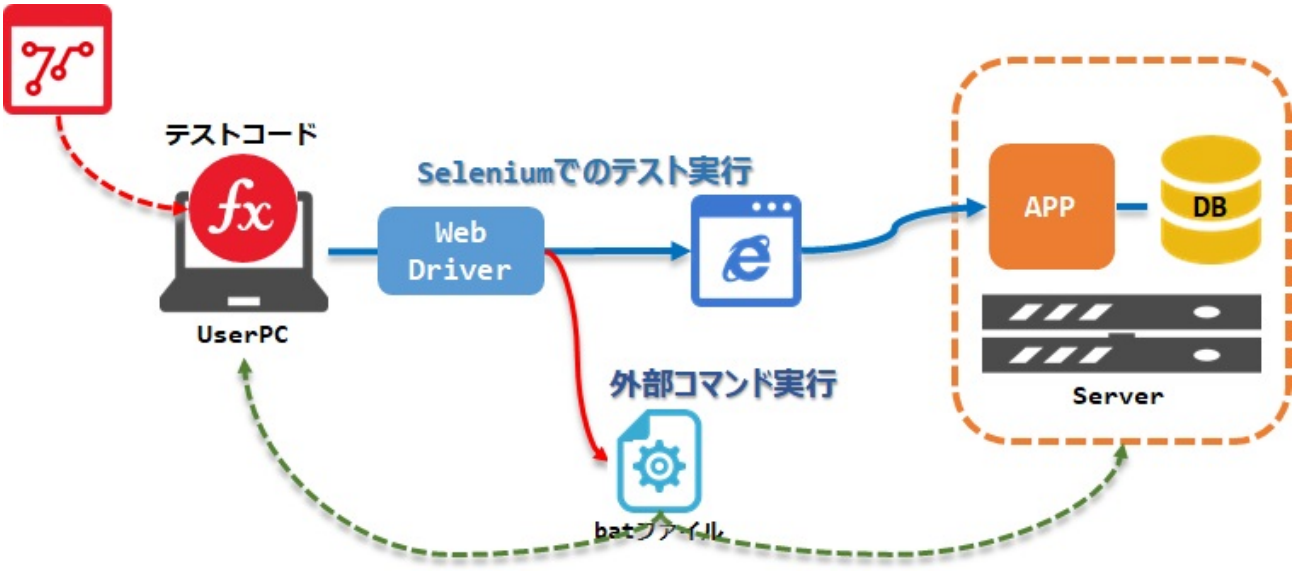


Fig. 9.1 外部コマンド機能概要

9.2 コマンド一覧画面

コマンド一覧画面は、作成したコマンドの一覧を表示する画面です。
各コマンド名はリンクになっており、コマンドごとの設定画面に移動できます。
この画面から、新たにコマンドを追加することができます。

Fig. 9.2 コマンド一覧画面

No.	名称	説明
1	一覧検索・ソート	一覧の表示内容の絞り込みと検索とその結果表示です。 詳しくは、1.各画面共通»1.3 一覧表示での検索・ソート機能 を参照してください。
2	新規作成ボタン	コマンド編集画面に移動し、新規コマンドを作成することができます。
3	コマンド名	クリックすると、そのコマンドのコマンド編集画面に移動します。
4	説明	編集画面で設定されている説明を表示します。
5	最終更新日時	
6	依存一覧リンク	このページを使用しているテストの一覧画面を別ウィンドウで開きます。 詳しくは、1.各画面共通»1.4 依存一覧リンク を参照してください。

9.3 コマンド編集画面

コマンド編集画面は、選択したコマンドを編集する画面です。

Fig. 9.3 コマンド編集画面

No.	名称	説明
1	コマンド名	作成するコマンドの名称を設定します。
2	実行するコマンド	実行したい外部コマンドを設定します。 OSのシェルコマンドを指定することもできます。
3	コマンドファイル	コマンドファイルをアップロードして、Testablishに登録してお 詳細は 9.3.1 コマンドファイルの取り扱いを参照してください。
4	引数(テストで指定)	引数の設定をテストごとで行う場合に選択してください。
5	引数の説明	テストごとに引数の設定を行うときの説明を入力してください。 テストに外部コマンドを挿入したときの入力フィールドのタイト
6	引数(固定)	固定した引数の設定を行う場合に選択してください。
7	引数の値	固定で行う引数の値を入力してください。
8	引数の追加	引数の選択・入力欄を追加します。
9	引数の削除	追加した引数を削除します。
10	アサーション	作成したコマンドをアサーションの色で表示します。
11	説明	作成したコマンドの説明を設定します。
12	保存ボタン	作成したコマンドの設定を保存します。
13	コピーボタン	作成したコマンドの設定をコピーします。
14	削除ボタン	作成したコマンドを削除します。
15	テスト依存一覧リンク	このページを使用しているテストの一覧画面を別ウィンドウで開

		詳しくは、1.各画面共通»1.4 依存一覧リンク を参照してくださ
16	テストテンプレート依存一覧リンク	このページを使用しているテストテンプレートの一覧画面を別ウ 詳しくは、1.各画面共通»1.4 依存一覧リンク を参照してくださ

9.3.1 コマンドファイルの取り扱い

Testablishで、作成した外部コマンドを登録する方法には二通りあります。

1. コマンド編集画面でコマンドの定義情報だけを登録しておき、アップロードしないでおく方法
2. コマンド編集画面でコマンドの定義情報とともに、利用するコマンドファイルをアップロードして登録しておく方法

9.3.1.1 コマンドファイルをアップロードしない場合

コマンド編集画面で、外部コマンドの定義情報のみを登録しておきます。
この時、実際のコマンド実行ファイルはこの画面では登録(アップロード)しません。

テスト編集画面で、外部コマンドを利用するシナリオを作成し、テストコードを出力します。
出力されたテストコードフォルダの直下に、実際のコマンド実行ファイルを手動で配置します。
配置の方法は 9.4.2 コマンドファイルの配置を参照してください。
この時、外部コマンドが実行されるワーキングディレクトリはテストコードフォルダ直下になります。

下記の例では、コマンドは powershell(windowsに標準で搭載されているシェル) を利用し、引数としてシェルスクリプトファイルを与えています。
引数で渡しているスクリプトファイルはワーキングディレクトリからみたパスを指定しています。
この例では コマンド実行ファイルそのものは配置する必要はありませんが、コマンドファイル(powershell)が引数として使用するシェルスクリプトファイルをワーキングディレクトリ以下に手動で配置する必要があります。

Fig. 9.3.1.1 （例）コマンドファイルをアップロードしない場合

9.3.1.2 コマンドファイルのアップロード

コマンド編集画面で、外部コマンドの定義情報と、実際のコマンドファイルを登録します。
実際のコマンドファイルを登録するには、**参照...** ボタンをクリックして該当のコマンドファイルを選択します。
アップロードするファイルは、コマンドバッチファイルの他に、圧縮ファイル(**zipのみ**)も選択できます。

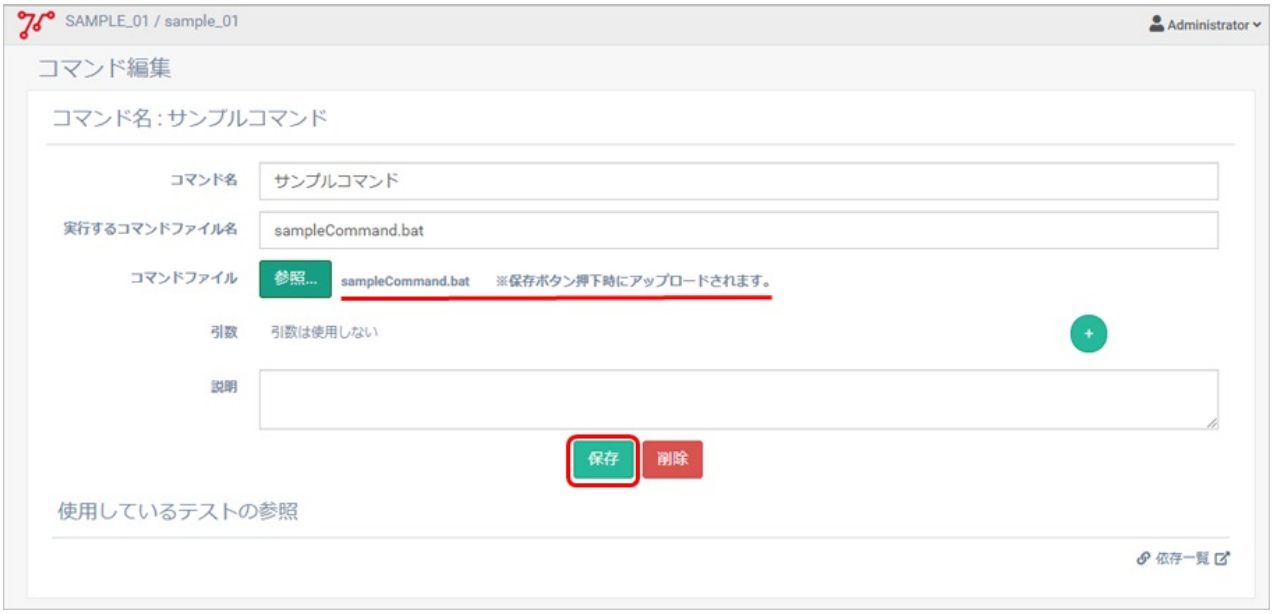


Fig. 9.3.1.2-1 コマンドファイルアップロード

選択したコマンドファイル名が **参照...** ボタンの右に表示されますが、この状態ではまだアップロードは完了していません。
選択したコマンドファイルは、下部の **保存** ボタンをクリックした時点でアップロードが完了し、**ダウンロード** ボタンの右に アップロードされているコマンドファイル名と アップロード日時が表示されます。

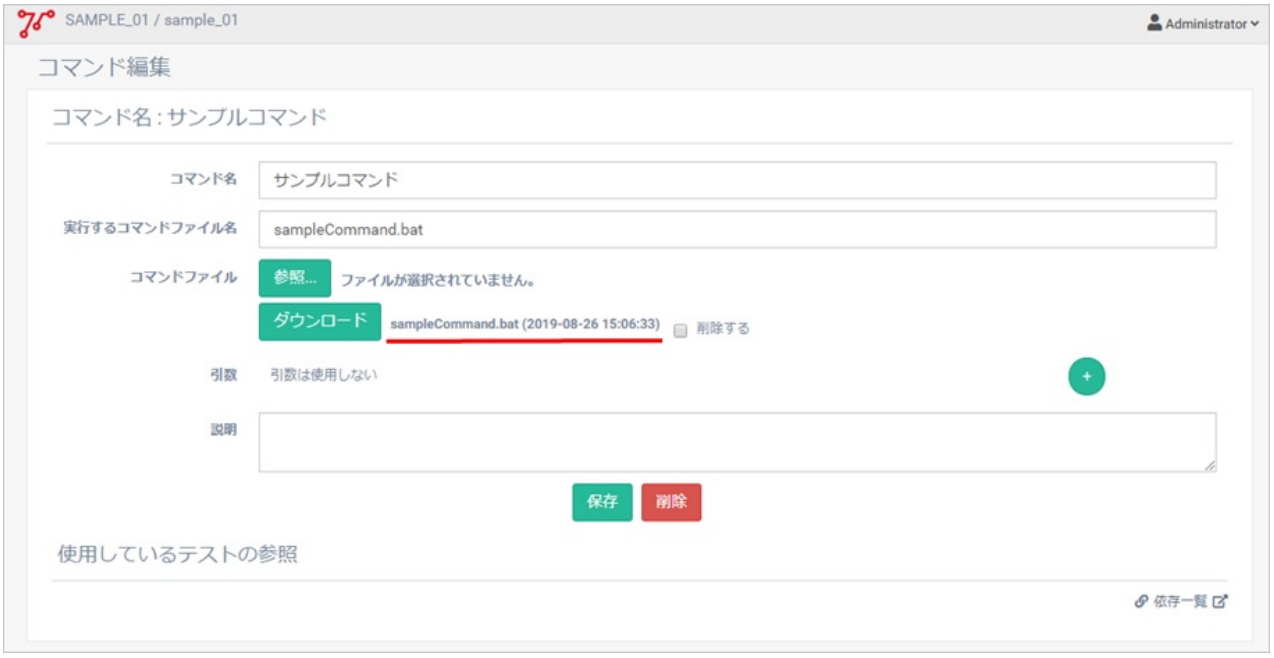


Fig. 9.3.1.2-2 コマンドファイルアップロード完了

コマンド編集画面で、コマンドファイルがアップロードされている状態で、再度 **参照...** ボタンからフ

ァイルを指定してアップロードすることができます。アップロード後、**保存**ボタンをクリックすると、
先にアップロードされていたコマンドファイルに**上書き**されます。
ひとつのコマンドに登録できるコマンドファイルは一つだけです。
コマンドファイルが複数の場合は zipで圧縮して登録してください。

Fig. 9.3.1.2-3 コマンドファイル 置き換えアップロード

Fig. 9.3.1.2-4 コマンドファイル 置き換えアップロード完了

9.3.1.3 コマンドファイルのダウンロード

外部コマンド定義にアップロードされているコマンドファイルは、ダウンロードすることができます。
ダウンロードボタンをクリックして、保存してください。

9.3.1.4 コマンドファイルの削除

外部コマンド定義にアップロードされているコマンドファイルを削除することができます。
アップロードされているコマンドファイル名の右の **削除する**チェックボックスにチェックを入れます。

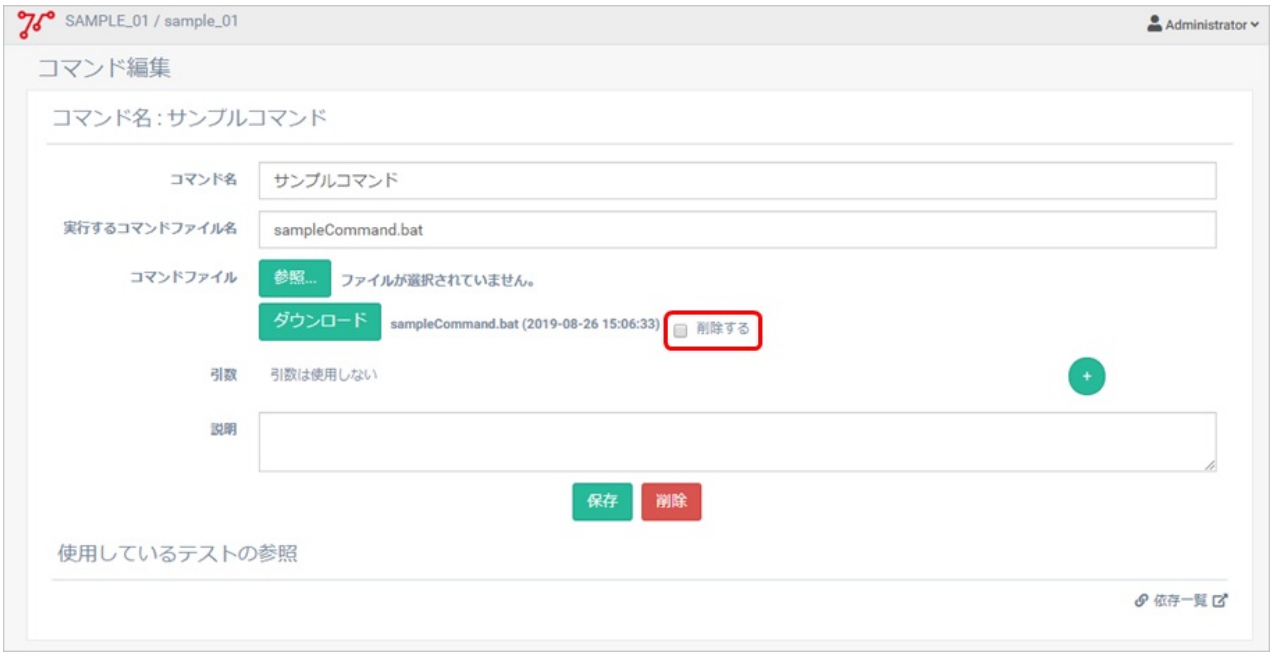


Fig. 9.3.1.4-1 削除チェックボックス

削除するチェックボックスにチェックを入れると、コマンドファイル名が赤字になり、取り消し線が表示されますが、この状態ではまだ削除は完了していません。
下部の **保存** ボタンをクリックした時点で削除されます。

Fig. 9.3.1.4-2 コマンドファイル削除

9.4 外部コマンド機能の利用

テストでコマンド機能を利用する場合、以下のような手順が必要です。

1. 外部コマンドとして、実現したい機能を持つバッチファイル・シェルスクリプト等を作成する
2. コマンドメニューから、作成した外部コマンド定義を登録する
3. テストシナリオに、登録したコマンドを追加する
4. テストコードを出力する
5. 外部コマンドの実行ファイルをアップロードしていない場合、コマンドファイルをテストコードフォルダに配置する
6. テストを実行する

以下のコマンドを例に、利用方法を説明していきます。
この例では、コマンド編集画面でコマンドの定義情報だけを登録しておき、実行するコマンドファイルは**アップロードされていません**。

Fig. 9.4.1 コマンドサンプル

9.4.1 テストへのコマンドの登録

コマンドの設定はテスト編集画面から行います。
コマンド機能を利用するテストを選択し、コマンドを起動したいステップに設定を行います。
下に操作を追加 のプルダウンから **コマンド追加** をクリックしてください。

Fig. 9.4.1-1 コマンド追加

コマンド選択のダイアログに選択可能なコマンドが表示されます。
利用したいコマンドの Add ボタンをクリックしてください。

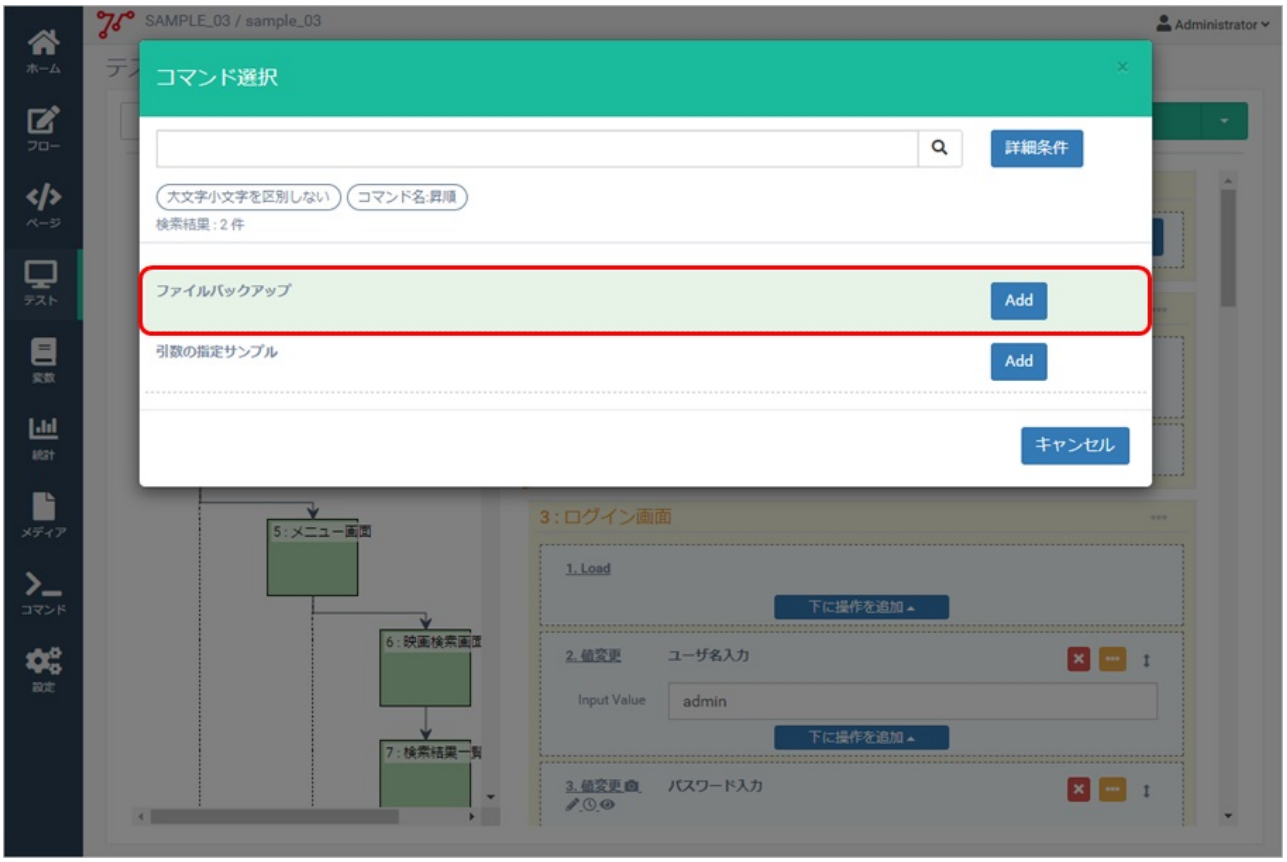


Fig. 9.4.1-2 コマンド選択

ダイアログで選択したコマンドがテストに追加されています。

引数を **テストで指定** を選択して作成した場合は、コマンド引数を指定してください。

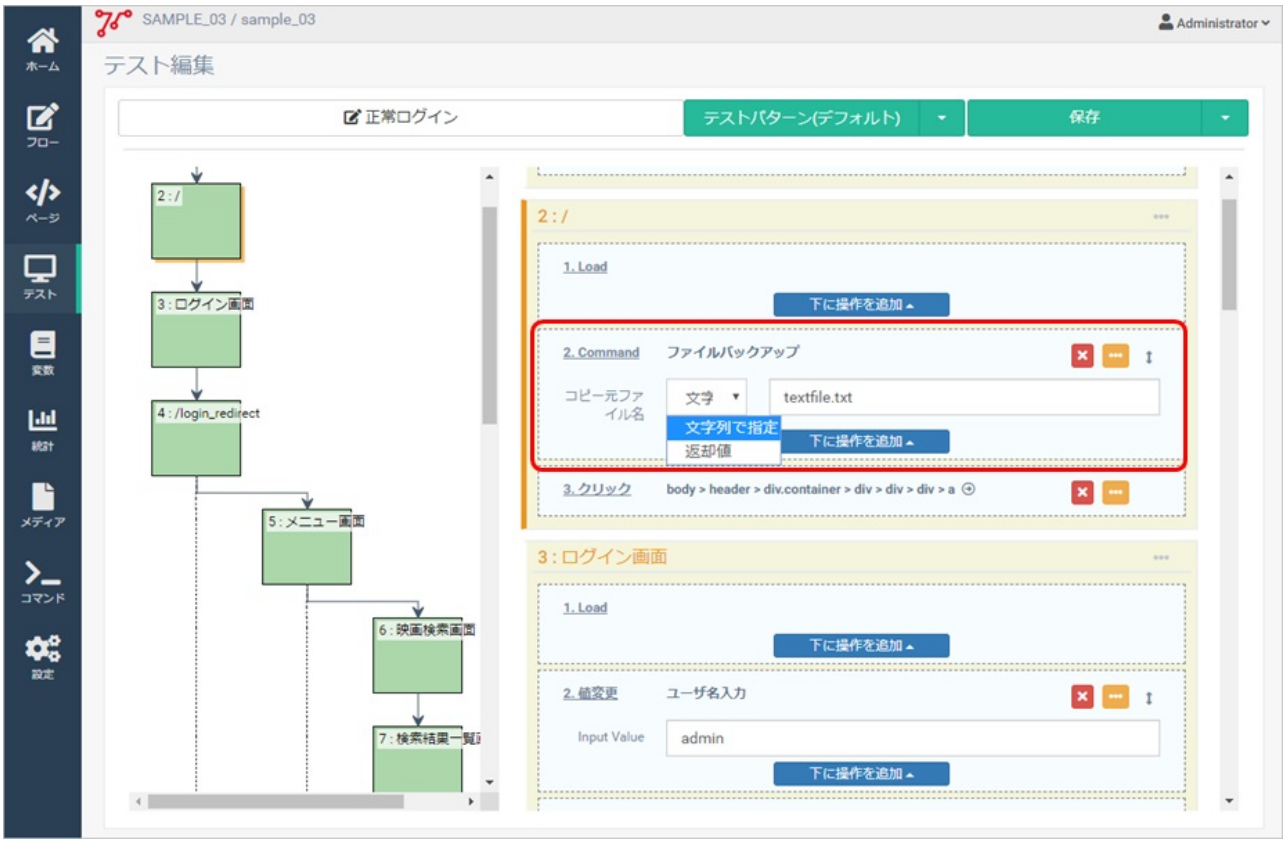


Fig. 9.4.1-3 コマンド確認

テストの編集が終わったら、一旦 **保存** し、保存右のプルダウンから **テストコードの出力** をクリックしてください。

Fig. 9.4.1-4 テストコード出力

9.4.2 コマンドファイルの配置

外部コマンドとして作成したコマンドファイルは、展開した **テストコードのフォルダ直下** に配置してください。 (テストを実行するためのバッチファイル **run-test.bat** と同じ階層に置きます。)
※ コマンドファイルが正しく配置されていない場合、自動テストが失敗します。

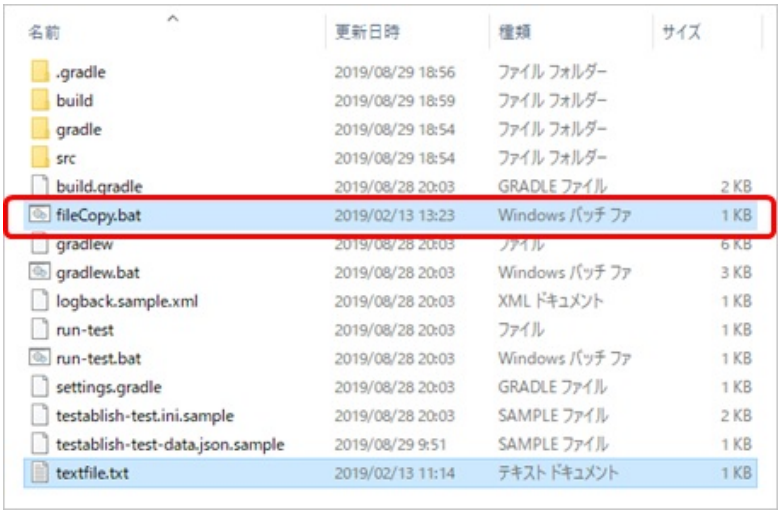


Fig. 9.4.2-1 コマンドファイル設置

なお、外部コマンドとして作成したコマンドファイルをコマンド編集画面で**アップロードしてある**場合、**手動でのコマンドファイルの配置は不要**です。 テストコードの出力時に **commandFile** フォルダが生成され、その中に展開・配置されます。
commandFile フォルダの生成について、詳細は 9.4.4 コマンドファイルのワーキングディレクトリ を参照してください。

出力したテストコードの実行は　**III 操作の流れ/機能説明 » 3 テストの自動実行** を参照してください。

9.4.3 結果確認

自動テストを実行(**run-test.bat**をダブルクリック)し、コマンド機能の結果を確認します。
コマンド機能もテストも成功した場合は **BUILD SUCCESSFUL** と表示されます。

Fig. 9.4.3-1 BUILD SUCCESSFUL

自動テスト後、以下のフォルダに コマンドの実行ログ **<ステップ番号>-<シーケンス番号>-command.log** が作成されています。
　⇒ <展開したテストコードフォルダ>**/reports/<ブラウザ名>/tests/<実行したテスト名>**

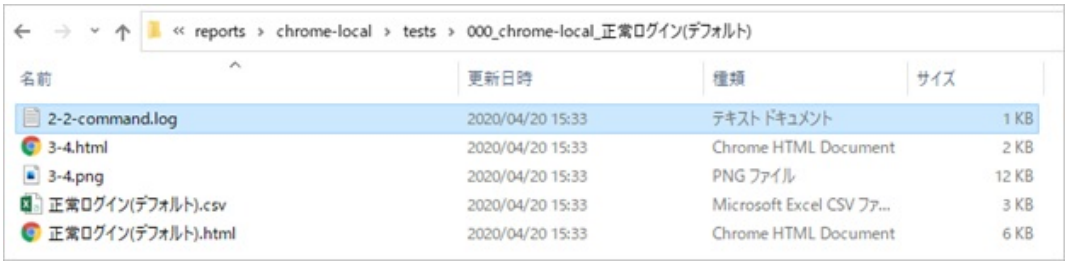


Fig. 9.4.3-2 コマンド実行ログ

同じフォルダに **HTMLファイル(<実行したテスト名>.html)** がありますので、これをクリックしてください。
テスト結果が成功と表示され、commandを実行した行の状態はOKになっています。

ステップ	画面名	シーケンス	ターゲット	アクション	状態	エラー	設定値	変数値	実行日時	画面
1	/	1		open	OK		["/"]		2020/04/20 15:33:21.713	
2	/	1	Page Load	load	OK				2020/04/20 15:33:22.885	
2	/	2	ファイルバックアップ	command	OK		["textfile.txt"]		2020/04/20 15:33:22.955	
2	/	3	link_ログインページへ進む	click	OK				2020/04/20 15:33:24.016	
3	ログイン画面	1	Page Load	load	OK				2020/04/20 15:33:24.420	
3	ログイン画面	2	ユーザ名入力	click	OK				2020/04/20 15:33:24.439	
3	ログイン画面	3	ユーザ名入力	change	OK		["admin"]		2020/04/20 15:33:25.517	
3	ログイン画面	4	パスワード入力	change	OK		XXXXX	XXXXX	2020/04/20 15:33:26.635	
3	ログイン画面	5	ログインボタン	click	OK				2020/04/20 15:33:30.030	
4	/login_redirect	1	Page Load	load	OK				2020/04/20 15:33:30.172	
5	メニュー画面	1	Page Load	load	OK				2020/04/20 15:33:30.173	

Fig. 9.4.3-3 レポート*SUCCESS*

実行ファイルがテスト時に成功する条件は **終了ステータス が 0** の場合で、それ以外のステータスは **エラー** となります。
読み込む外部コマンドファイルが未作成だったり、コマンドが間違っていたりする場合には、例外が発生して エラー となります。

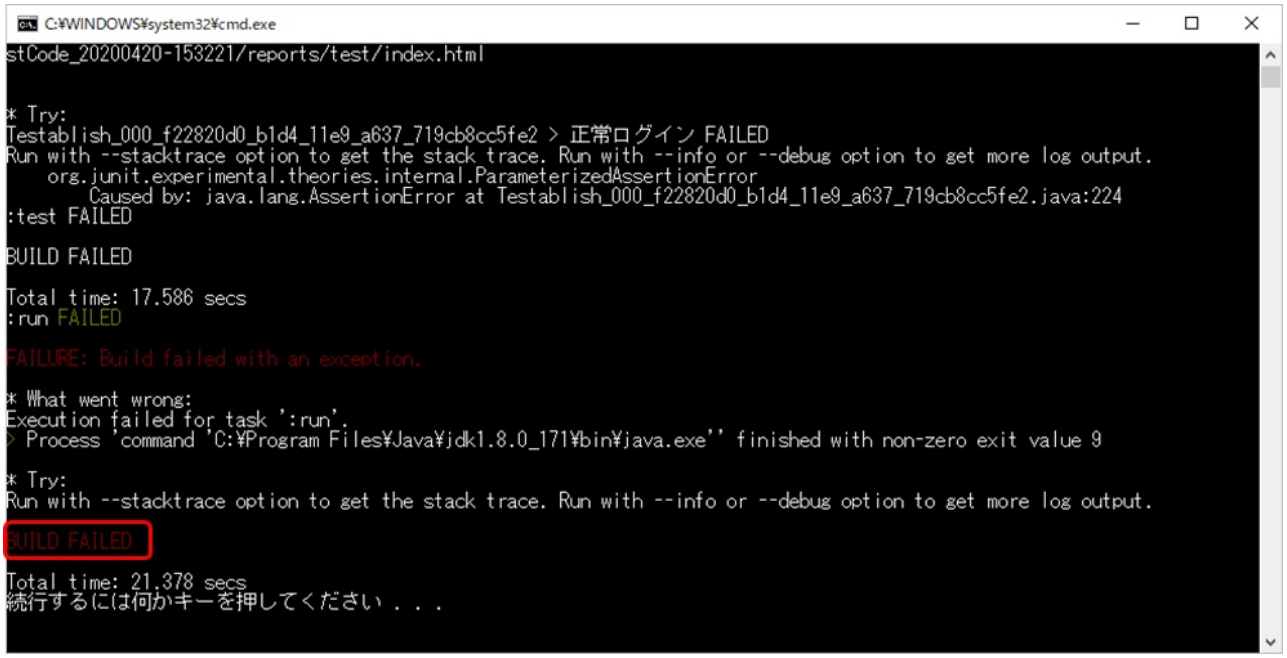


Fig. 9.4.3-4 BUILD FAILED

Fig. 9.4.3-5 レポートFAILED

9.4.4 コマンドファイルのワーキングディレクトリ

外部コマンド実行用のファイルを作成する際に、そのコマンドのワーキングディレクトリがどこになるかを把握しておくことが重要になります。

コマンドファイルをアップロードしていないコマンドの場合と、 コマンドファイルをアップロードしてある場合で、ワーキングディレクトリは異な ります。
また、コマンドファイルをアップロードした際のコマンドファイルの形式(単一ファイル・圧縮ファイル(階層なし)・圧縮ファイル(階層あり))によっても異なります。

9.4.4.1 アップロードしていない場合

コマンドファイルをアップロードしていないコマンドのテストを実施する場合（9.4.2 コマンドファイルの配置）、コマンドファイルを**テストコードのフォルダ直下** に配置するため、**テストコードのフォルダ直下** がコマンドのワーキングディレクトリになります。

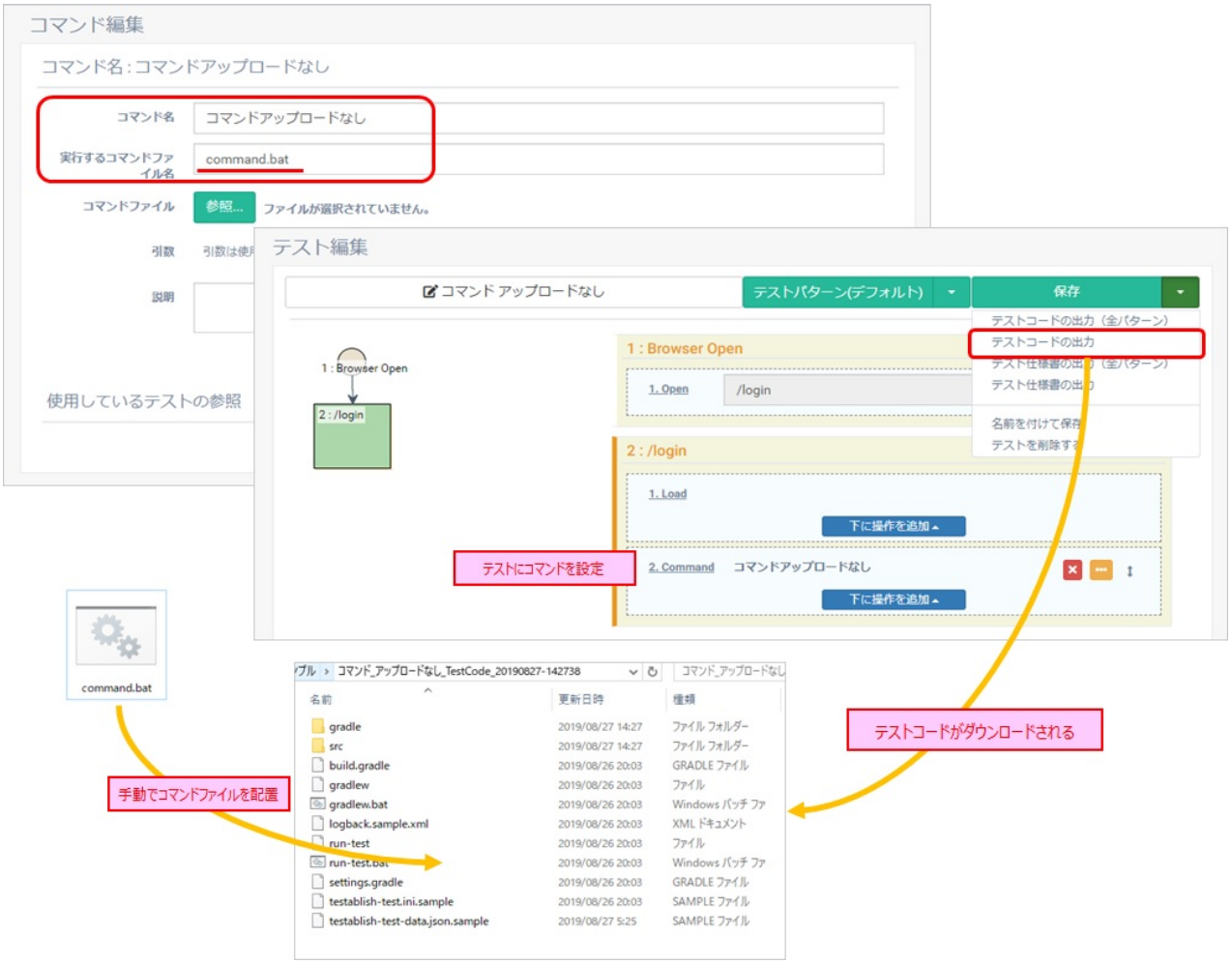


Fig. 9.4.4.1 アップロードしていない場合

9.4.4.2 単一ファイルの場合

展開されたテストコードファイルの直下に生成された**commandFile** フォルダの中に **コマンドID名**のフォルダが生成され、その中に 単一の コマンド実行ファイルが配置されます。
このため、 **テストコードのフォルダ直下/commandFile/<コマンドID>/** 直下がワーキングディレクトリとなります。

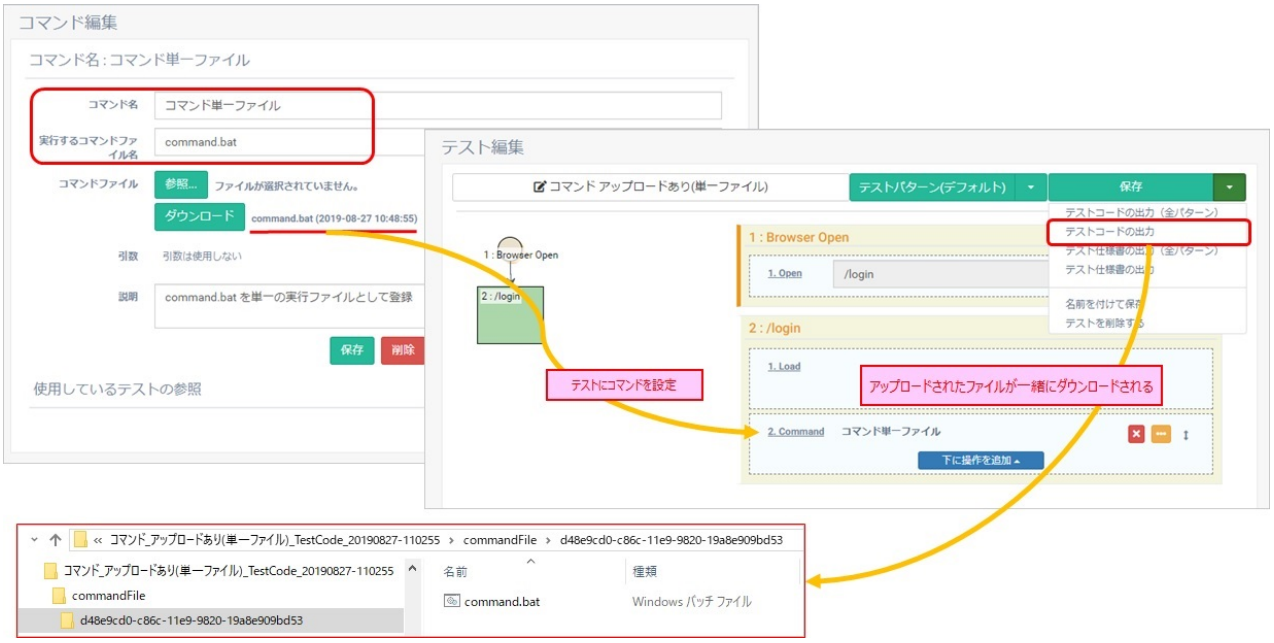


Fig. 9.4.4.2 単一ファイルの場合

9.4.4.3 圧縮ファイル(階層なし)の場合

単一ファイルと同様に、 展開されたテストコードファイルの直下に生成された**commandFile** フォルダの中に **コマンドID名**のフォルダが生成され、その中に 単一のコマンド実行ファイルが配置されます。
このため、 **テストコードのフォルダ直下/commandFile/<コマンドID>/** 直下がワーキングディレクトリとなります。

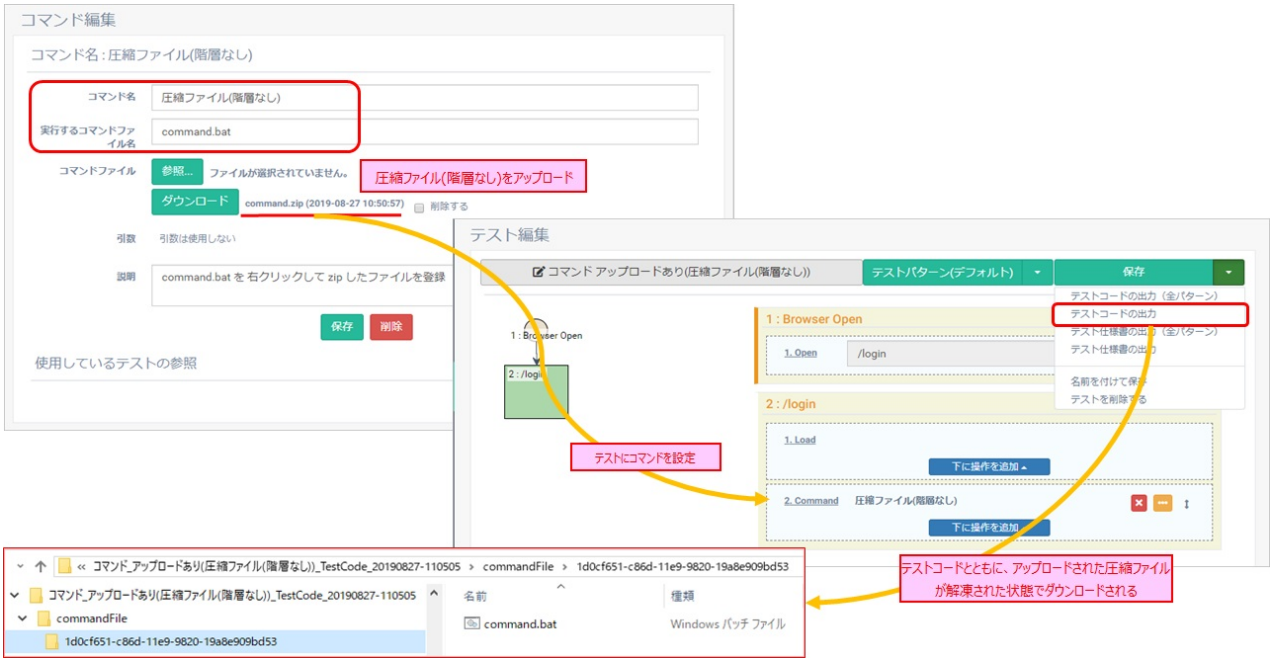


Fig. 9.4.4.3 圧縮ファイル(階層なし)の場合

9.4.4.4 圧縮ファイル(階層あり)の場合

圧縮ファイル（階層あり）とは、例えば、以下のようにメインとなるバッチからサブバッチを呼び出すような構成のコマンドを作成し、圧縮してアップロードした場合です。

```
sampleFolder/      ← バッチファイルをまとめたフォルダ
  │ main.bat       ← メインのバッチファイル
  └subBatch/
     │ sub1.bat    ← メインから呼び出されるサブバッチファイル
     │ sub2.bat    ← メインから呼び出されるサブバッチファイル
```

展開されたテストコードファイルの直下に生成された**commandFile** フォルダの中に **コマンドID名**のフォルダが生成され、その中に **展開されたフォルダ**が配置されます。
このため、 **テストコードのフォルダ直下/commandFile/<コマンドID>/zipを展開したフォルダ** 直下がワーキングディレクトリとなります。

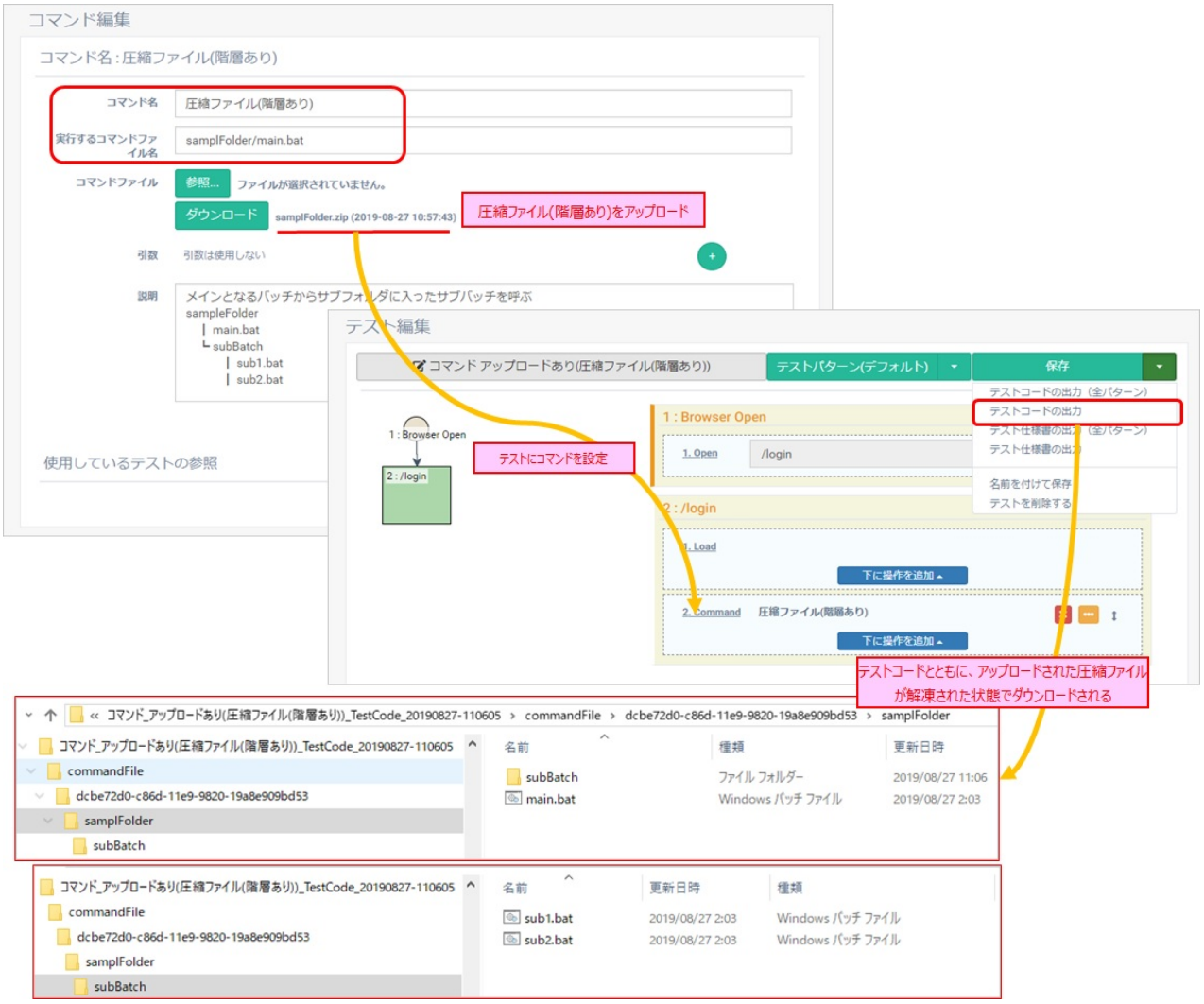


Fig. 9.4.4.4 圧縮ファイル(階層あり)の場合

9.4.5 外部コマンドと返却値（変数の利用）

外部コマンドを利用する際に、その返却値を受け取って後続のテスト実行処理に反映したい場合があります。
このようなとき、コマンドの返却値を受け取る変数を作成してコマンドの引数に与えることで、外部コマンドからの返却値を受け取ることができるようになります。

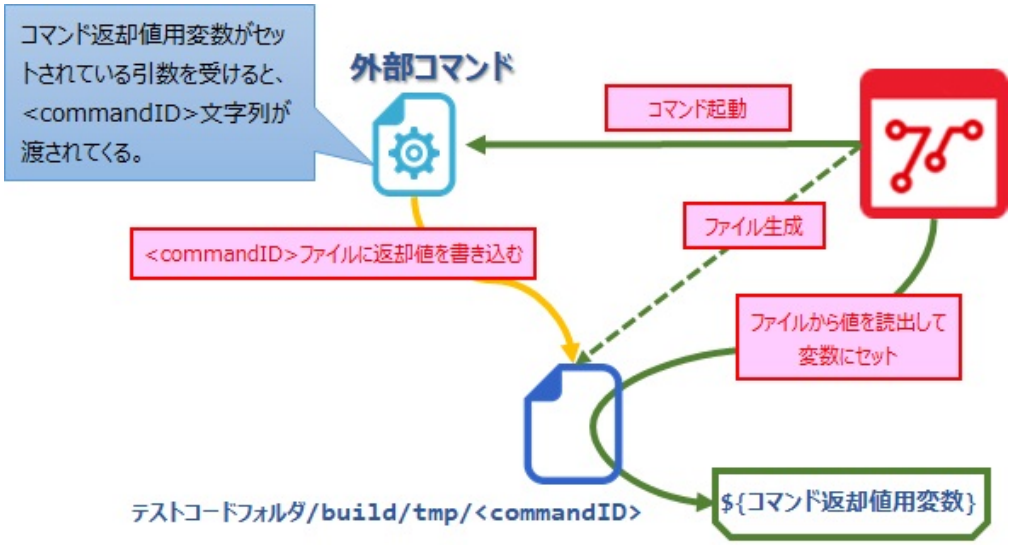


Fig. 9.4.5-1 コマンドの返却値の受け渡しの仕組み

コマンドの返却値の受け渡しの仕組み：

【Testablish側】

1. 返却値を受け取るための変数を用意しておきます。
2. テストでコマンドを呼び出す際に、外部コマンドに引数として返却値用の変数を渡します。
   実際には返却値用の変数の値が引数で渡るわけではなく、コマンドには**コマンドID文字列が渡されます**。
   同時に、**コマンドID文字列**を名称とするファイルを**テストコードフォルダ/build/tmp/** 以下に作成します。
3. Testablishは **テストコードフォルダ/build/tmp/** に生成されている **<コマンドID文字列>**ファイルの内容を読み出し、返却値用の変数にセットします。

【外部コマンド側】

1. コマンドの構造として、**コマンドID文字列**を受け取るための引数を用意しておきます。
2. 引数として渡されてくる**コマンドID文字列**を名称とするファイルに**返却値の内容を書き込み**ます。
   この時、コマンドのワーキングディレクトリに注意してください。
   コマンドの定義によってコマンドファイルが展開される場所が異なるため、build/tmp/ の相対的な参照位置が変わります。（9.4.4 コマンドファイルのワーキングディレクトリ）

つまり、Testablish側のテストシナリオ内でコマンドを呼び出す際に、「返却値」タイプの変数をコマンド側に渡し、 外部コマンド側では必ず引数で**コマンドID文字列**を受け取って、返却値は**テストコードフォルダ/build/tmp/<コマンドID文字列>**ファイルに書き込むといった構造にしておかなければなりません。

以下は、外部コマンドの呼び出しと返却値を確認するテストのサンプルです。

ここで使用している 「管理者名取得」という外部コマンドは **二つの引数を受け取り、"admin" という文字列を返します**。

事前に、変数とコマンド定義およびフィールドに対するアサーションの定義をしておきます。
管理者名取得コマンドが受け取る引数のうち一つは返却値用の引数です。

まず、変数を用意しておきます。

Fig. 9.4.5-2 変数の用意

次に、コマンドを定義しておきます。
「管理者名取得」コマンドは引数を二つ受け取るつくりなので、定義でも引数を二つ用意しています。一つは返却値用の引数です。
また、この例では「管理者名取得」コマンドで実際に実行されるバッチファイル (sampleCommand.bat) をアップロードしています。

Fig. 9.4.5-3 コマンドの定義

sampleCommand.bat の内容は以下のようになっています。

```
echo %1
echo %2
echo admin > ../../build/tmp/%2
exit /b
```

テストで使う login画面の ユーザID フィールドに対してアサーションを定義しておきます。

Fig. 9.4.5-4 アサーションの定義

ログイン画面のステップで外部コマンドの呼び出しと返却値を確認するテストシーケンスを作成していきます。

1. ログイン画面がロードされています。(自動的に作成されます)

2. 画面の入力フィールドの値を変更します。 **下に操作を追加** ボタン から **操作追加**を選択し、 画面の入力フィールド「ユーザID」に対して **値変更** の操作を追加します。
   テストに追加された操作ボックスで "user1"をセットしています。

3. コマンドを実行する操作を追加します。
   下に操作を追加 ボタン から **コマンド追加**を選択し、 ダイアログから**管理者名取得**コマンドを追加します。
   テストに追加された操作ボックスで引数の設定を行います。
   一つ目の引数には、"user1"という文字列を与えています。
   二つ目の引数には、コマンドの返却値を格納するための**変数**「コマンド返却値(ユーザID)」を与えています。
   返却値を受け取るためには、この引数のタイプを**返却値**としなければなりません。
   なお、この操作の操作番号と操作種類は、コマンド追加した直後は "3.Command"と表記されますが、返却値で変数代入が行われているため保存後画面をリロードすると、”3.変数代入”に変わります。

4. 返却値をフィールドに設定します。
   下に操作を追加 ボタン から **操作追加**を選択し、 画面の入力フィールド「ユーザID」に対して **値変更** の操作を追加します。

前のコマンド実行の操作での返却値を受けた**変数**「コマンド返却値(ユーザID)」の値を画面の入力フィールド「ユーザID」にセットしています。

5. 画面の入力フィールド「ユーザID」に対してアサーションを設定します。
   下に操作を追加 ボタン から **アサーション追加**を選択し、
   画面の入力フィールド「ユーザID」に対して **値一致** のアサーションを追加します。
   前の操作でコマンド返却値を設定したィールド「ユーザID」の値と 値："admin" が一致するかを検証します。

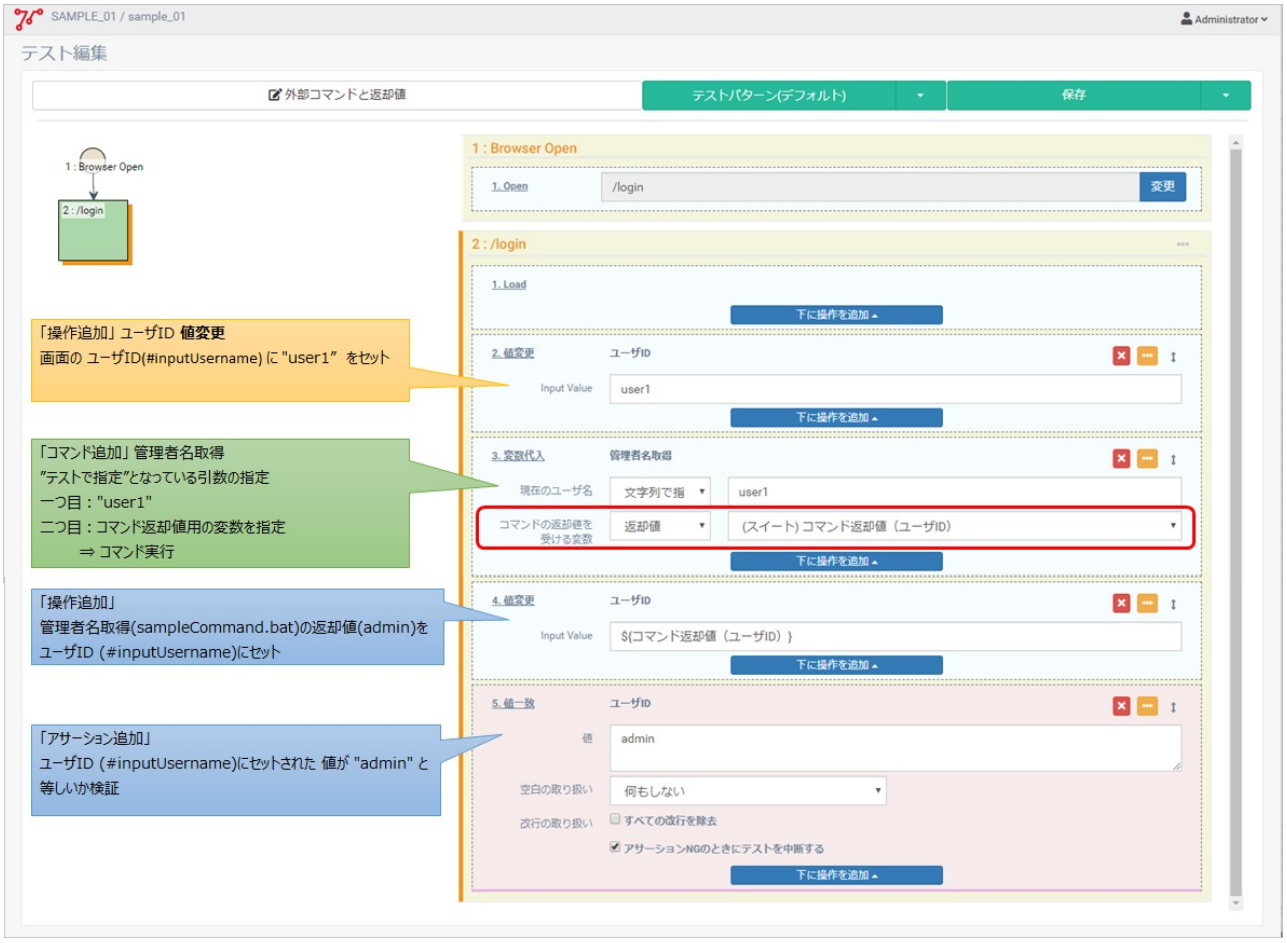


Fig. 9.4.5-5 テスト内容

上記のテストを自動実行した際の結果は **<展開したテストコードフォルダ>/reports/summary.html** からブラウザ毎にリンクを辿って確認することができます。


外部コマンドと返却値(デフォルト) テスト結果

成功

テスト開始日時: 2020/04/20 16:34:13

画像サイズ 10 %

summry > chrome-local > 外部コマンドと返却値(デフォルト)

ステップ	画面名	シーケンス	ターゲット	アクション	状態	エラー	設定値	変数値	実行日時	画面
1	/login	1		open	OK		["/login"]		2020/04/20 16:34:20.672	
2	/login	1	Page Load	load	OK				2020/04/20 16:34:22.192	
2	/login	2	ユーザID	change	OK		["user1"]		2020/04/20 16:34:22.272	
2	/login	3	管理者名取得	command	OK		["user1","コマンド返却値（ユーザID）"]		2020/04/20 16:34:23.513	
2	/login	4	ユーザID	change	OK		["${コマンド返却値（ユーザID）}"]	[admin]	2020/04/20 16:34:24.694	
2	/login	5	ユーザID	value-equal	OK		["admin"]		2020/04/20 16:34:25.919	

Fig. 9.4.5-6 テスト実行結果

実行したコマンドのログは、<展開したテストコードフォルダ>/reports/<ブラウザ名>/tests/<実行したテスト名> のフォルダに **<ステップ番号>-<シーケンス番号>-command.log** として作成されています。

```
C:¥...¥外部コマンドと返却値_TestCode_20200420-163206¥commandFile¥31daf0c0-c8a2-11e9-9820-19a8e909bd53>echo user1
user1

C:¥...¥外部コマンドと返却値_TestCode_20200420-163206¥commandFile¥31daf0c0-c8a2-11e9-9820-19a8e909bd53>echo 17d65088-1e10-
4746-b209-5da8c81f7598
17d65088-1e10-4746-b209-5da8c81f7598

C:¥...¥外部コマンドと返却値_TestCode_20200420-163206¥commandFile¥31daf0c0-c8a2-11e9-9820-19a8e909bd53>echo admin
1>../../build/tmp/17d65088-1e10-4746-b209-5da8c81f7598

C:¥...¥外部コマンドと返却値_TestCode_20200420-163206¥commandFile¥31daf0c0-c8a2-11e9-9820-19a8e909bd53>exit /b
```

Fig. 9.4.5-7 コマンドのログ
(ファイルパスの前半を”…”に修正しています)

9. コマンド メニュー 目次 | II. 各画面の説明 目次 | 使い方マニュアル 目次 に戻る

10. 設定 メニュー

10.1 プロジェクト設定画面

プロジェクト設定画面では、テスト対象アプリの設定および、使用するユーザの割り当て等を行う画面です。
また、テスト仕様書の出力設定やDBのバックアップ/リストア等も行うことができます。

プロジェクトは、Testablishでの操作の収集、テストシナリオの作成、テストコードの出力、ドキュメントの作成などを行う大枠になります。

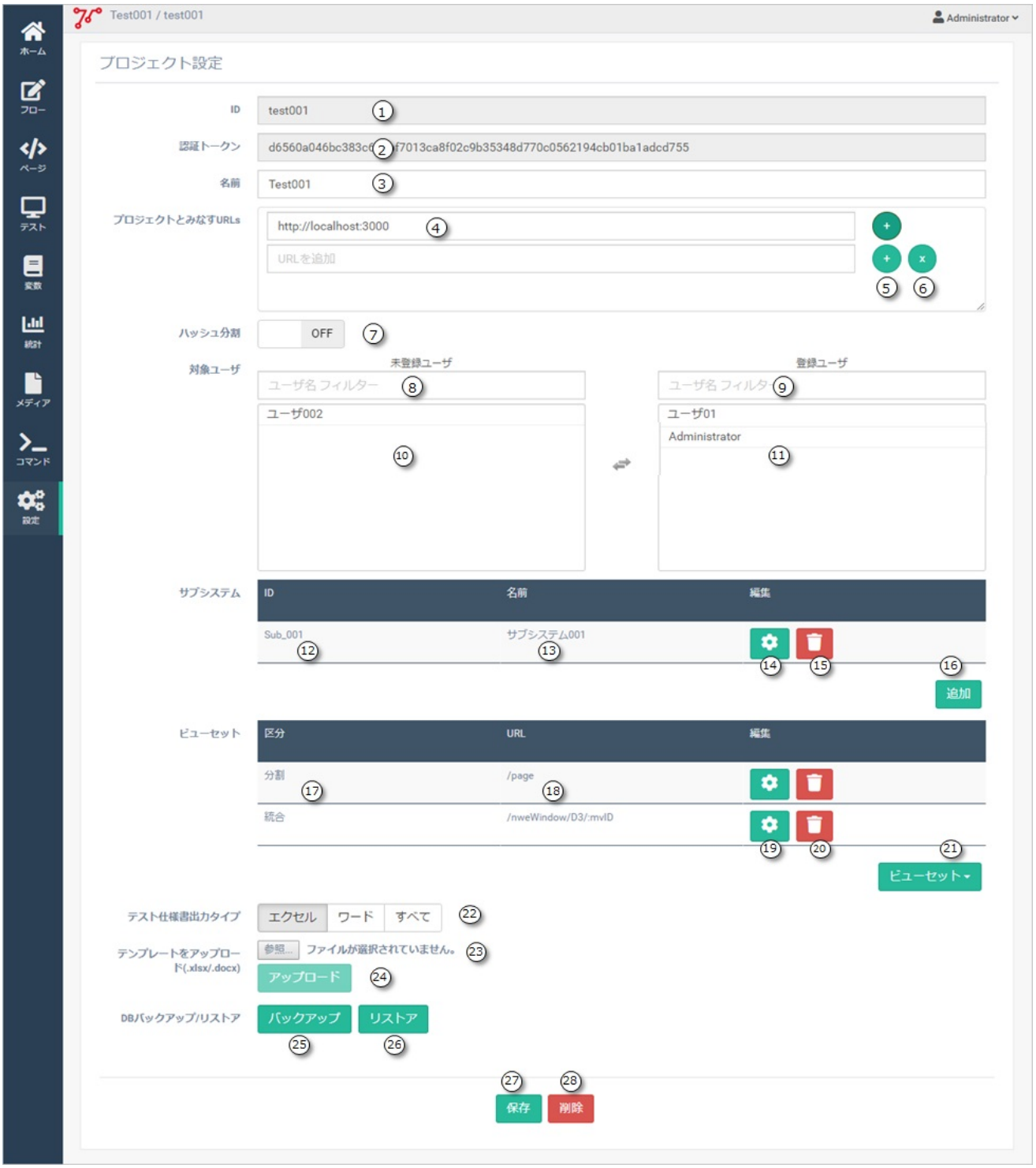


Fig. 10.1 プロジェクト設定画面

No.	名称	説明
1	プロジェクトID	新規作成時に指定したプロジェクトのIDを表示します。
2	認証トークン	プロジェクト作成時に発行されるトークンを表示します。 操作キャプチャを行う際に、この認証トークンを利用します。
3	名前	
4	プロジェクトとみなすURLs	プロジェクトのホームとなるアドレスを設定します。
5	追加ボタン	URL入力欄を追加します。
6	削除ボタン	追加したURLを削除します。

7	ハッシュ分割 スライドボタン	ハッシュ分割のON/OFF を選択します。
8 9	ユーザ名フィルター	未登録ユーザ・登録ユーザをユーザ名で絞り込むことができます。
10	未登録ユーザリスト	プロジェクトの利用が登録されていないユーザを一覧表示します。 未登録のユーザをクリックすると(11) の登録ユーザリストに追加されます。
11	登録ユーザリスト	プロジェクトの利用が許可されたユーザを一覧表示します。 登録ユーザをクリックすると(10)の未登録ユーザに戻します。
サブシステム		
12	ID	サブシステムがある場合、サブシステムのIDを表示します。
13	名前	サブシステムがある場合、サブシステムの名前を表示します。
14	編集ボタン	サブシステムダイアログを表示し、 サブシステムの設定を編集することができます。
15	削除ボタン	サブシステムを削除します。
16	追加ボタン	サブシステムダイアログを表示し、 サブシステムの設定を追加することができます。
ビューセット		
17	区分	ビューセットがある場合、ビューセットの区分（統合/分割） を表示します。
18	URL	ビューセットがある場合、 ビューセットとして設定されているURLを表示します。
19	編集ボタン	ダイアログを表示し、ビューセットの設定を編集することができます。
20	削除ボタン	ビューセットを削除します。
21	ビューセット追加 プルダウンボタン	プルダウンから、 ページ統合かページ分割を選択してビューセットを追加します。 統合または分割ダイアログを表示します。
テスト仕様書出力タイプとテンプレート		
22	テスト仕様書出力タイプ	テスト仕様書を出力するときのファイル形式を選択します。 **エクセル/ワード/すべて** から選択できます。
23	ファイル参照ボタン	テスト仕様書のテンプレートファイルを選択します。
24	アップロードボタン	(23)で選択したテンプレートファイルをプロジェクトに適用します。
DBバックアップ/リストア		
25	バックアップ ボタン	プロジェクトのデータベースをzipファイル（バックアップファイル） で出力します。 それぞれのプロジェクトごとにバックアップをおこなってください。
26	リストア ボタン	クリックするとエクスプローラーのファイル選択ウィンドウが表示され リストアするバックアップファイルを選択できます。 確認ポップアップダイアログが表示され、 OKをクリックするとリストアが実行されます。
27	保存ボタン	プロジェクト設定の編集を保存します。

28	削除ボタン	プロジェクト自体を削除します。

10.1.1 プロジェクトとみなすURLs

このプロジェクトで利用する対象WebアプリケーションのURLを設定します。
ここで設定したアドレスの下層にあるすべてのページがを対象に、Testablishで操作を収集することができるようになります。
テストを実施したいWebアプリケーションの最上位アドレスを設定してください。

ひとつのプロジェクトに複数のURLを設定することができます。
機能・画面・データなどを共有する複数のWebアプリケーションを同時にテストしたい場合、ここにURLを追加することで一つのプロジェクトとして利用できます。
全く繋がりのないWebアプリを同じプロジェクトに設定することもできますが、そのような場合はプロジェクトを新たに作成するようにしてください。

10.1.2 ハッシュ分割

ハッシュ分割のスライドボタンで、ハッシュ分割機能の ON / OFF ができます。
ハッシュ分割機能では、対象の URLにハッシュ記号(#)がある場合に、単一のURLとみなすか、別のURLとみなすかの設定ができます。

デフォルトではOFFです。
OFFのままであれば、Testablishは通常ドキュメントパスだけを見ているため、ハッシュ以降のページはすべて同じページとして認識します。

この設定がONであれば、Testablishがパスに加えてハッシュの後の文字列も画面のキーとして認識するようになります。
ハッシュのみが異なる画面を区別してテストをする必要がある場合にONにしてください。

(例) SPA （Single Page Apprication)でURLのハッシュ(#)を使って、javascript で画面を構築するようなシステムの場合

Fig. 10.1.2 ハッシュ分割の例

10.1.3 対象ユーザ

このプロジェクトを利用できるユーザを設定します。

左側の **未登録ユーザリスト** で利用を許可するユーザ名をクリックし、右側の **登録ユーザリスト**へ移動させることで、そのユーザがこのプロジェクトを利用できるようになります。

プロジェクト作成後の初期状態では、**管理者ユーザも利用が許可されていません。**
管理者ユーザもこの画面で登録ユーザに設定してください。

新たなユーザを追加する方法は、画面ヘッダエリアの右端の ユーザメニューから行います。
詳しくは、1. 各画面共通 » 1.1 ヘッダエリア » 1.1.4 ユーザ設定 をご確認ください。

10.1.4 サブシステム

対象のWebアプリケーションから別サーバへのリンク・リクエスト等のつながりがある場合、サブシステムに定義しておいてください。

・サブシステムに定義がない場合、同じパス（/index.html など）を区別しないため、
　フローが正常に取得できないことがあります。
・サブシステムに定義を追加したWebアプリケーションは、同じパスであっても別画面として判断します。

サブシステムのセクションで、**追加ボタン**をクリックするとサブシステムダイアログが表示され、サブシステムの設定を追加することができます。

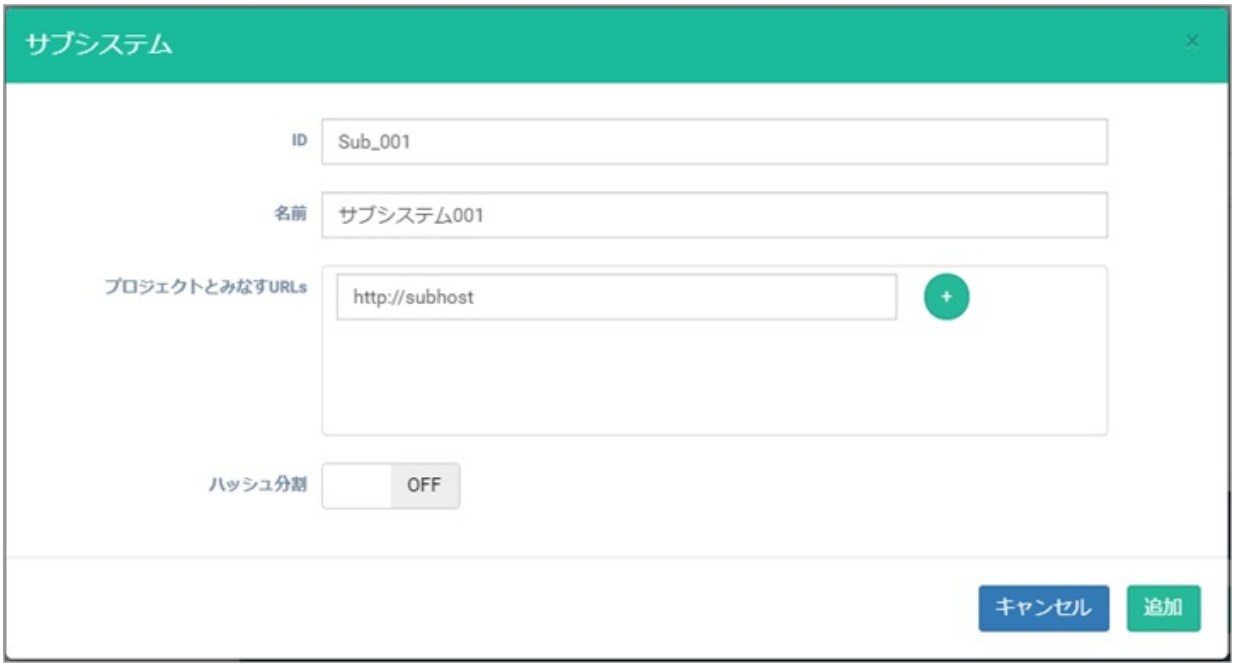


Fig. 10.1.4-1 サブシステムダイアログ

ひとつのプロジェクトには複数のサブシステムを設定することができます。
追加ボタンで必要に応じて追加してください。

以下の例のようなシステムでは、サブシステムを適用する対象になります。
(例)

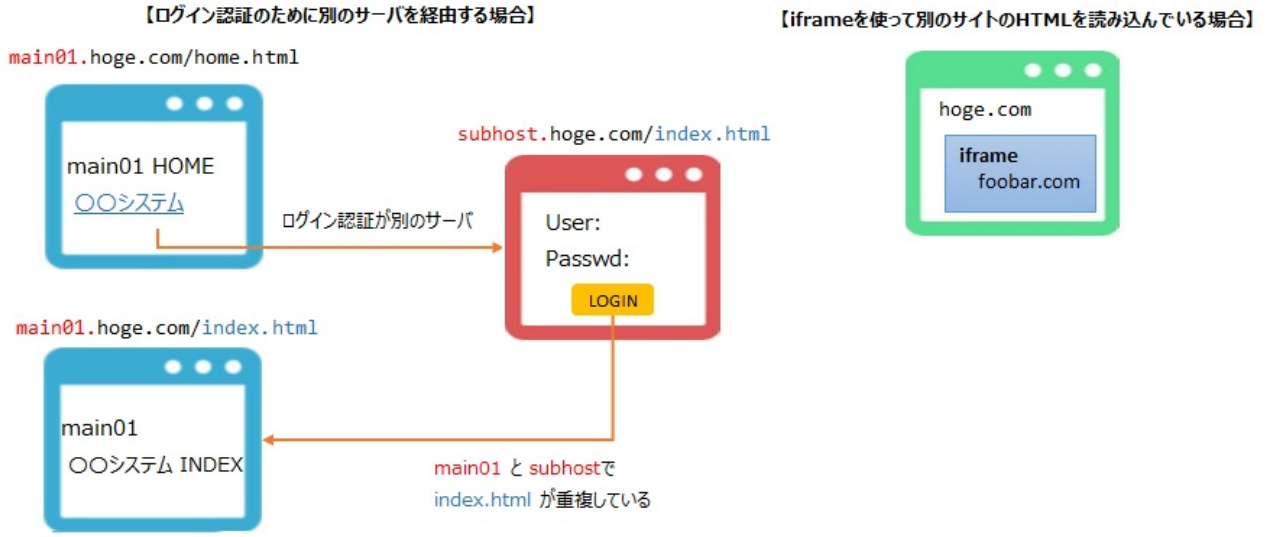


Fig. 10.1.4-2 サブシステムの例：ログイン認証のために別サーバを経由する場合・*iframe*で別サイトのページを表示している場合

10.1.5 ビューセット

10.1.5.1 ページ統合

ページ統合は、パスの一部が変わるだけの同じ構造のページが複数ある場合に、ページを同じものとして扱うことができる機能です。

ビューセット追加プルダウンボタンで **ページ統合** を選択すると、ページ統合ダイアログを表示します。

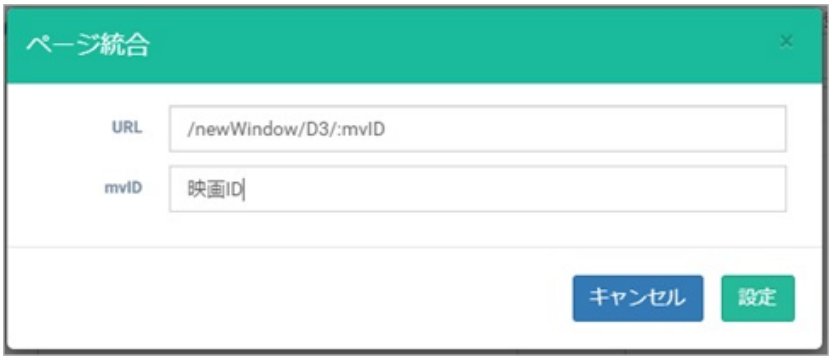


Fig. 10.1.5.1 ページ統合ダイアログ

: (コロン) を用いた以下の記述ルールに従って設定してください。
　共通URL/**:英字**

◆例1) 商品を検索して、結果から詳細を表示したときのURL（URL/products/detail/商品ID）を、ひとつのURLとして扱う
　URL/products/detail/32
　URL/products/detail/33
　URL/products/detail/34

↓

URL/products/detail/**:prdctId**

◆例2) 日記のページのURL（URL/diary/年/月/日）を、ひとつのURLとして扱う

URL/diary/2018/05/01

URL/diary/2018/05/02

URL/diary/2018/05/03

↓

URL/diary/**:year/:month/:day**

10.1.5.2 ページ分割

ページ分割は、クエリパラメータ(?～～)によって遷移ページを切り分けている場合に、ページを別のものとして扱うことができる機能です。

ビューセット追加プルダウンボタンで **ページ分割** を選択すると、ページ分割ダイアログを表示します。

Fig. 10.1.5.2 ページ分割ダイアログ

以下の記述ルールに従って設定してください。

◆例) 以下のようなクエリで切り分けたページを、別のURLとして扱う

/page?pageid=A001 ⇒ 検索画面

/page?pageid=B001 ⇒ 一覧

↓

URL：/page

分割：パラメータ：pageid　パラメータ説明：ページID

10.1.6 テスト仕様書出力タイプとテンプレート

出力するテスト仕様書のファイル形式を「エクセル」「ワード」から選択することができます。
テスト仕様書は、zipファイルで出力されます。
「すべて」を選択すると、ExcelとWordの両方がひとつのzipファイルで出力されるようになります。

テスト仕様書のテンプレートは変更することができます。
デフォルトで設定されているテスト仕様書のテンプレートファイルは以下の2つです。
これらは、インストールメディアのTestablishフォルダ内に収められています。

- testspec-template.xlsx
- testspec-template.docx

10.1.7 DBバックアップ/リストア

Testablish データベースコンテナに保存したデータを出力して、プロジェクトのバックアップをとることができます。
バックアップファイルは以下の形式の名称のzipファイルで保存されます。

<プロジェクトID>_<プロジェクト名>_<Testablishのバージョン>_<出力日時>.zip
(例) Sample01_サンプル01_v1.x.x_yyyymmdd-hhmmss.zip

リストアは、保存されているバックアップファイル(zipファイル)を指定します。
リストアすると、プロジェクトのデータはすべて削除され、指定のバックアップで置き換えられます。

なお、リストアする際にはバックアップしたプロジェクトのバージョンと、 リストア先のTestablishのバージョンが**一致**していなければなりません。

10. 設定 メニュー 目次 ｜ II. 各画面の説明 目次 ｜ 使い方マニュアル 目次 に戻る

III. 操作の流れ/機能説明

1. ページ情報および操作の収集

Testablishでは、Testablish WebExtension (以下、WebExtension) を利用して、ブラウザで行った操作を収集することができます。WebExtension は タスクトレイアプリとして提供されています。

収集に使用できるブラウザは、**Internet Explorer** と **Google Chrome** です。

1.1 WebExtension を起動する

(1) デスクトップに作成された Testablish のショートカットアイコンをダブルクリックします。
起動するとタスクトレイに常駐します。

Fig. 1.1-1 WebExtension ショートカットアイコン

(2) タスクトレイのTestablish アイコンをクリックします。

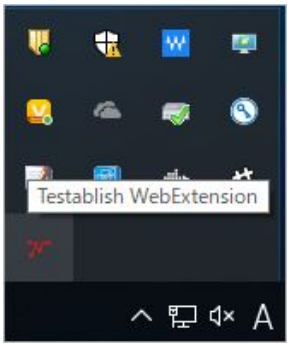


Fig. 1.1-2 WebExtension タスクトレイアイコン

(3) **Testablish WebExtension** ダイアログが表示されます。

(4) **サービス** タブを選択します。

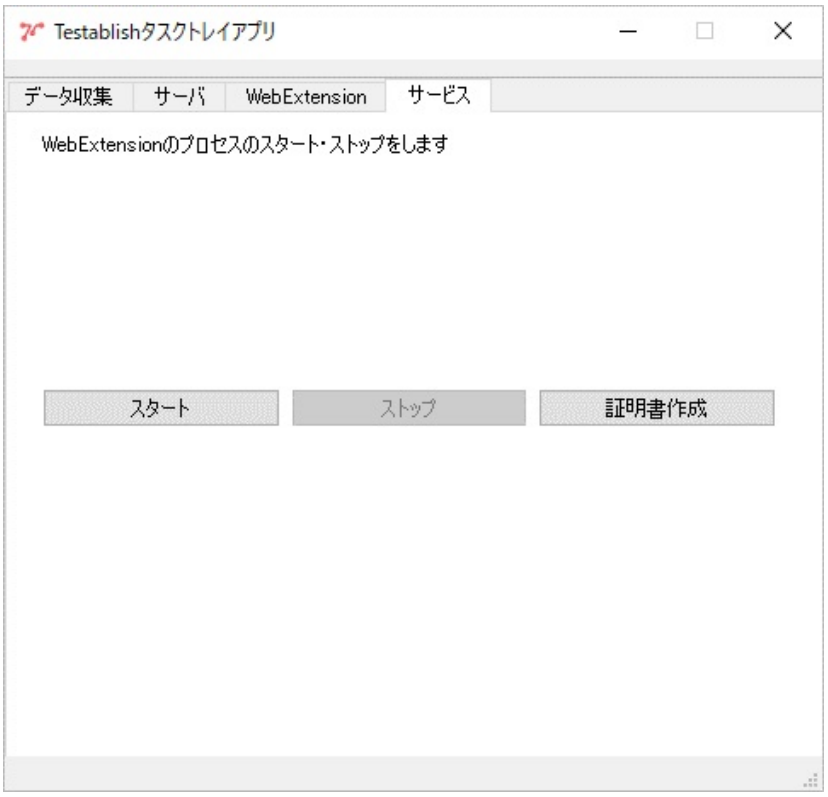


Fig. 1.1-3 WebExtension サービスタブ

(5) **スタート** ボタン でWebExtensionのサービス (Local Proxy と Back Service) を起動します。

ここまでは、操作の収集を行うブラウザが Internet Explolar と Google Chromeのどちらであっても同じ手順です。

1.2 Internet Explolar で操作を収集する

Internet Explolar で操作を収集する場合は、WebExtension サービスのスタートと同じダイアログから開始できます。

(1) タスクトレイのTestablish アイコンをクリックし、**Testablish WebExtension** ダイアログを表示します。

(2) **データ収集** タブを選択してください。

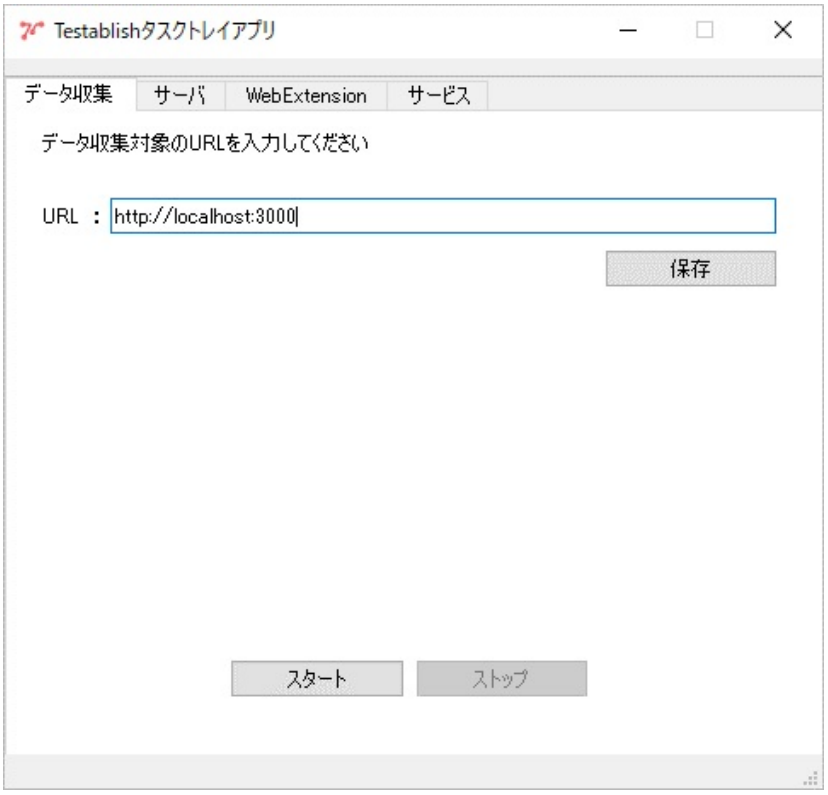


Fig. 1.2-1 WebExtension データ収集タブ

(3) **スタート** ボタンをクリックします。

(4) Internet Explolerが起動して設定したサイトが表示され、操作データの収集が開始されます。

(5) 起動したブラウザ画面で、操作を行います。

(6) **ストップ** ボタンで、データ収集を終了します。

収集した操作は、Testablishの「フロー/ビューセット編集」画面で確認してください。

1.3 Google Chrome で操作を収集する

Google Chromeで操作を収集する場合は、WebExtensionサービスを起動した後、Google Chromeのブラウザから収集を開始します。

(1) Google Chromeブラウザ右上の、Testablishアイコンをクリックします。

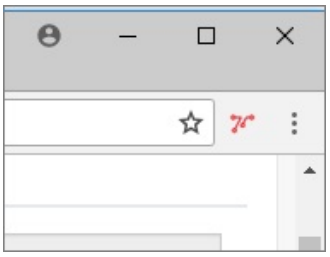


*Fig. 1.3-1 Chrome WebExtension*アイコン

(2) ダイアログが表示されますので、**スタート** ボタンをクリックします。

*Fig. 1.3-2 Chrome WebExtension*スタート

(3) Google Chrome が起動し、タスクトレイアプリでURLを設定したデータ収集対象サイトが表示され、操作データの収集が開始されます。

(4) 起動したブラウザ画面で、操作を行います。

(5) **ストップ** ボタンで、データ収集を終了します。

収集した操作は、Testablishの「フロー/ビューセット編集」画面で確認してください。

III. 操作の流れ/機能説明 目次 / 使い方マニュアル 目次 に戻る

2. テストの自動実行

テスト実行用ソフトウェア、ブラウザ、OSの設定をして、Testablishで生成したテストを自動実行します。
テスト実行環境は、Windows 10 を想定しています。
実行環境の構築は、別マニュアルの **インストールマニュアル 》III. 自動実行環境の構築手順** を参照してください。

2.1 自動テストを実行する

(1) テスト編集画面の テストコード出力 プルダウンメニュー から、Testablish のテストコードを出力します。
　　テストスイートの場合は、テストスイート編集画面の テストコードの出力ボタン から出力できます。

(2) ダウンロードされた zip ファイルを展開します。

```
[展開フォルダ]/
  │ build.gradle
  │ gradlew
  │ gradlew.bat
  │ logback.sample.xml
  │ run-test
  │ run-test.bat
  │ settings.gradle
  │ testablish-test-data.json.sample
  │ testablish-test.ini.sample
  │
  ├─gradle/
  │   :
  ├─src/
  │   :
  └─testDocument/
      <テスト名 or スイート名>_<出力日時>.docx    // テスト結果マージ用テンプレートファイル
      <テスト名 or スイート名>_<出力日時>.xlsx    // テスト結果マージ用テンプレートファイル
```

Zipファイル解凍時、テスト名に日本語が含まれている場合、解凍後のファイル名の文字化けを防ぐため、Unicode（UTF-8 / UTF8）に対応した圧縮解凍ソフト（7-Zip, Explzh 等）をご利用ください。

(3) testablish-test.ini ファイルを作成します。
　　詳細は、2.1.1 testablish-test.ini の作成 を参照してください。

(4) 展開したテストフォルダ内にある **run-test.bat** ファイルをダブルクリックします。

(5) テストの自動実行が開始されます。
※ ブラウザが起動するまでに多少の時間がかかる場合があります。

2.1.1 testablish-test.ini の作成

testablish-test.ini ファイルは、**展開したテストフォルダ内**または、その**一つ上の階層**に作成します。

自動テスト実行時に、展開したテストフォルダ内に testablish-test.iniファイルがない場合は、一つ上の

階層まで探索し、testablish-test.ini ファイルを読み込みます。

展開されたフォルダ内に [testablish-test.ini.sample]というファイルがあります。
testablish-test.ini ファイル作成の参考にしてください。

下記は、IEで 自動テストを実行するサンプルです。
baseUrl には **Testablishでテストを実施する対象サイトのURL**を指定 してください。

```
[testablish]
baseUrl = https://xxx.xxx
[ie-local]
browser.name = ie
config.fastSetValue = yes
env.webdriver.ie.driver = C:\selenium\IEDriverServer_x64_3.8.0\IEDriverServer.exe
```

テストの実行を **Internet Explorer以外** で行うときは **ブラウザごと**にある **WebDriver のダウンロード**・配置 が必要です。
※ WebDriver はお使いの OS やブラウザのバージョンに合わせて選択してください。
(ブラウザのバージョンが上がった場合、そのバージョンに対応した webDriver の再取得および配置が必要になることがあります。)

Chrome でテスト実行する際の [testablish-test.ini] の記述例

```
[testablish]
baseUrl= https://xxx.xxx
  :
[chrome-local]
browser.name = chrome
browser.version = 76.0.3809.132
browser.platform = windows10
env.webdriver.chrome.driver = C:\selenium\chromedriver.exe
tester = tester01
  :
```

※ tester 等 iniファイル内に日本語を使用した場合は、ファイルの文字コードを UTF-8 に設定して保存してください。

2.1.2 testablish-test.ini の項目詳細

key	説明
[testablish]	
baseUrl	テストを実行するアプリケーションの基底となるURL
addingUrl	URLの末尾に常に追加される文字列
commandTimeoutSeconds	コマンドを実行する際にタイムアウトと認識するまでの秒数
logConfPath	テスト実行時のログの設定ファイルのパス
defaultSleepTimeMilliSeconds	操作と操作の間の時間。推奨時間は1秒(1000)。
defaultSeleniumTimeoutMilliSeconds	Seleniumの要素の検索タイムアウト時間(ミリ秒単位)。 100秒を設定した場合、あるページで要素が現れない時に 100秒間待ち続ける。推奨時間は4秒(4000)。

pageLoadTimeMilliSeconds	テスト編集画面でのloadやdummyloadの操作の時に待機する時 ミリ秒で指定する。 テストで画面を遷移する際に最初の画面が正しく動いているの 正しく目視で確認したい時などに利用できる。
remote	リモートWebドライバのURL（Selenium Gridを使用している場合）
browserSize	ブラウザ起動時の幅と高さの指定
browserPosition	ブラウザ起動時の開始位置の指定 ex) browserPosition=40x50 (x:40、y:50)
startMaximized	最大化した状態でウィンドウを起動 true/false
httpProxy	http プロキシ のドメイン
httpsProxy	https プロキシ のドメイン
noProxy	プロキシを利用しないドメイン
testSpecificationMerge	テスト仕様書マージを行うか true/false
testSpecificationMergeType	テスト仕様書マージをするファイルタイプ excel,word
interruptTestAtErrorForSuite	スイートでエラー発生時、次テストの実行を停止するか true/false　デフォルト値：true（エラー時STOP）
interruptTestAtErrorForPattern	パターンでエラー発生時、次テストの実行を停止するか true/false　デフォルト値：true（エラー時STOP）
interruptTestAtErrorForBrowser	エラー発生時、次テストの実行を停止するか true/false デフォルト値：true（エラー時STOP）
run_image_comparison	スクリーンショット画像を "path_collect_images" 以下にある画像と照合するか true/false
path_collect_images	「正しい UI レイアウト画像」フォルダのパス
path_ddt_xlsx	繰り返し実行用データファイルのパス
[firefox-local]	
tester	テスト実施者
browser.name	ブラウザ名（firefox）
browser.version	ブラウザバージョン
browser.platform	ブラウザのプラットフォーム
env.webdriver.gecko.driver	webdriver の設置場所
env.webdriver.firefox.bin	webdriver の設置場所
config.headless	ヘッドレスモードでの起動 yes/no
[chrome-local]	
tester	テスト実施者
browser.name	ブラウザ名（chrome）
browser.version	ブラウザバージョン
browser.platform	ブラウザのプラットフォーム
env.webdriver.chrome.driver	webdriver の設置場所

config.headless	ヘッドレスモードでの起動 yes/no
[safari-local]	
browser.name	ブラウザ名（safari）
env.webdriver.safari.driver	webdriver の設置場所
[ie-local]	
browser.name	ブラウザ名（ie）
config.fastSetValue	自動テスト時のキー入力の速度を向上させる設定
config.useIEEdge	Microsoft Edge IEモードで開く（true/false）
config.edgePath	Microsoft Edgeの起動パス
env.webdriver.ie.driver	webdriver の設置場所
：	：

2.2 テストレポートを確認する

テスト成功後はコマンドプロンプトに BUILD SUCCESSFUL が表示されます。
「続行するには何かキーを押してください．．．」と表示後、任意のキーを入力するとテストを終了します。

Fig. 2.2-1 BUILD SUCCESSFUL

テスト実行後、テストコードを展開したフォルダ以下は以下のような構成になっています。

```
[展開フォルダ]
  │ build.gradle
  │ gradlew
  │ gradlew.bat
  │ logback.sample.xml
  │ run-test
  │ run-test.bat
  │ settings.gradle
  │ testablish-test-data.json.sample
  │ testablish-test.ini.sample
```

```
|
├─.gradle/
├─build/
├─gradle/
├─collect_images/
|  ├─<ブラウザセクション名>/
|  |  └─tests/
|  |     ├─<実行順No>_<セクション名>_<テスト名>(<パターン名>)/
|  |     |    <テスト名>(<パターン名>).png     // (6) スクリーンショット照合用ファイル
├─reports/
|  |    summry.html                          // (1) 全テスト結果サマリファイル
|  |    testablish-test.ini                  // オリジナル iniファイル
|  |
|  ├─<ブラウザセクション名>/
|  |  |  testablish-test.ini                  // ブラウザ毎の iniファイル
|  |  |  <テスト名 or スイート名>.html         // (2) ブラウザ毎テスト結果サマリファイル
|  |  |
|  |  ├─test/
|  |  |  index.html                           // (4) gradleが生成するテストサマリ
|  |  |
|  |  └─tests/
|  |     ├─<実行順No>_<セクション名>_<テスト名>(<パターン名>)/
|  |     |    <テスト名>(<パターン名>).html     // (3) テスト結果詳細ファイル
|  |     |    <テスト名>(<パターン名>).png      // (5) スクリーンショット
|  |     |    <テスト名>(<パターン名>).expected.png     // (6) スクリーンショット照合用ファイル
|  |     |    <テスト名>(<パターン名>).compared.png     // (7) スクリーンショット照合結果ファイル
|  |     |
|  |     ├<実行順No>_<セクション名>_<テスト名>(<パターン名>)/
|  |     |    :
|  |
|  ├<ブラウザセクション名>/
|  |    :
|
└─testDocument/
   |  <テスト名 or スイート名>_<出力日時>.docx     // テスト結果マージ用テンプレートファイル
   |  <テスト名 or スイート名>_<出力日時>.xlsx     // テスト結果マージ用テンプレートファイル
   |
   └─result/
       [browser.name]_[browser.version]_[browser.platform]_<テスト名>_<出力日時>.docx
       [browser.name]_[browser.version]_[browser.platform]_<テスト名>_<出力日時>.xlsx
```

テストレポートは、[テストコード展開フォルダ]/**reports** フォルダ以下に含まれており、以下のような種類があります。

(1) 全テスト結果サマリファイル
(2) ブラウザ毎テスト結果サマリファイル。 (1)からリンクされている。
(3) テスト結果詳細ファイル。ステップ・シーケンス毎の実行結果が出力されている。(2)からリンクされている。
(4) gradle (テストプログラムを実行するミドルウェア) が生成したテストサマリファイル。
(5) スクリーンショット（※ テスト作成時に、スクリーンショット != なしに設定した場合のみ）
(6) スクリーンショット照合用ファイル
(7) スクリーンショット照合結果ファイル

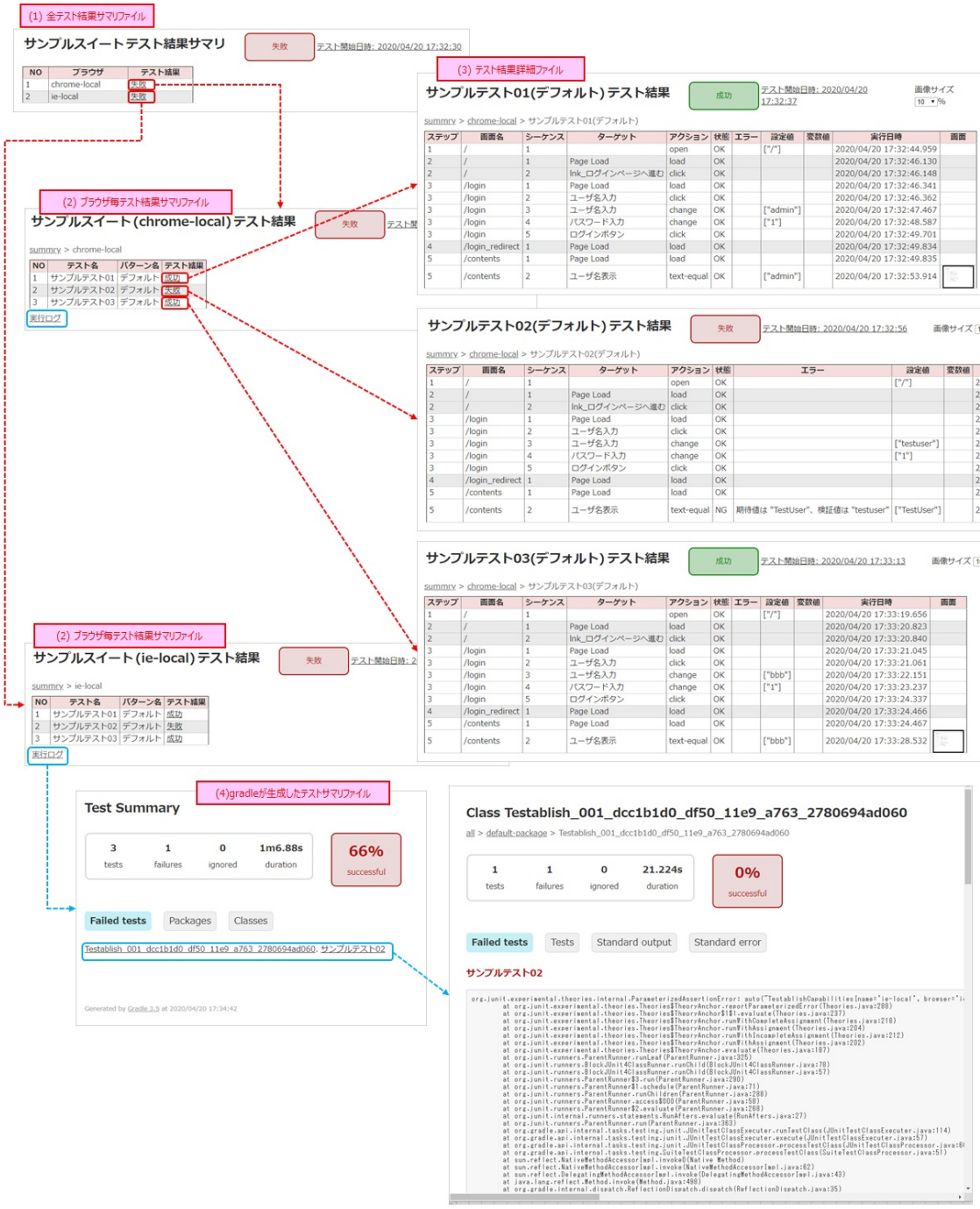


Fig. 2.2-2 テストレポート

テストの実行が失敗した場合には、コマンドプロンプトに **BUILD FAILED** が表示されます。

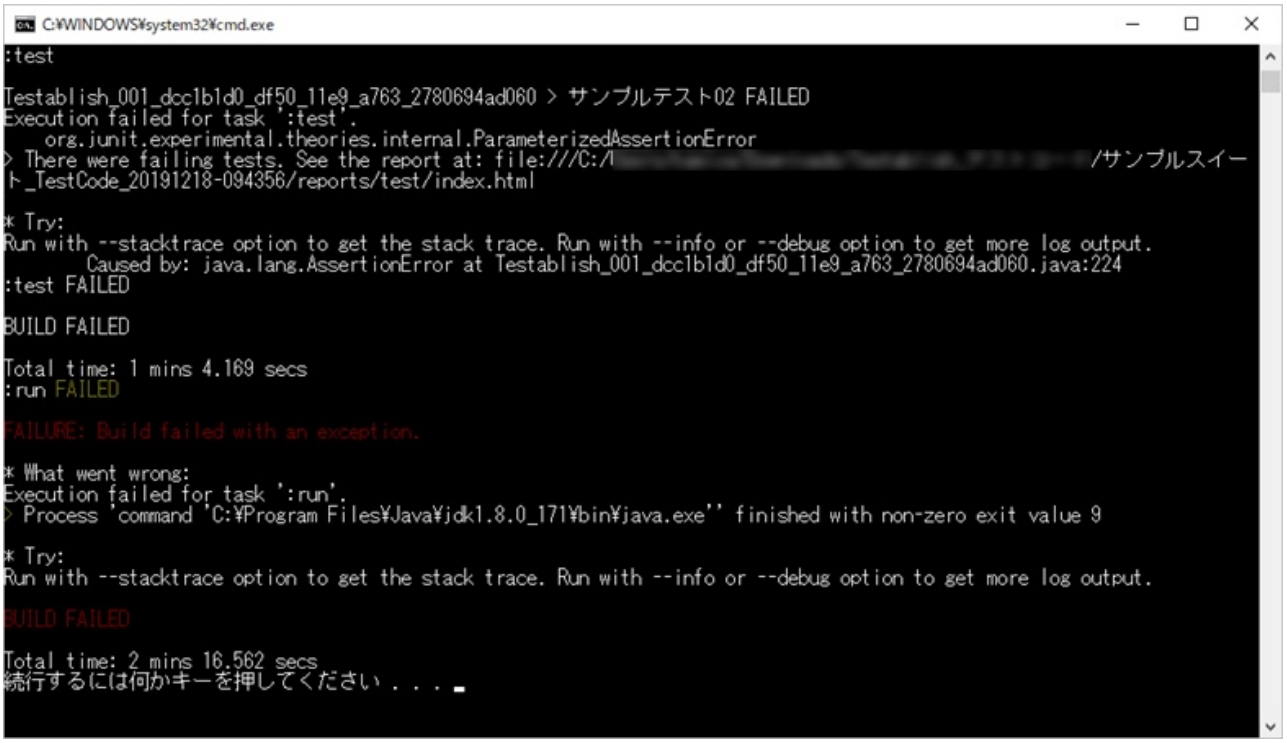


Fig. 2.2-3 BUILD FAILED

(3)のテスト結果詳細ファイルにはテストのステップ・シーケンス毎の実行結果が出力されていますので、ここでエラー内容を確認することができます。

ステップ	画面名	シーケンス	ターゲット	アクション	状態	エラー	設定値	変数値	実行日時	画面
1	/	1		open	OK		["/"]		2020/04/20 17:33:02.752	
2	/	1	Page Load	load	OK				2020/04/20 17:33:04.053	
2	/	2	lnk_ログインページへ進む	click	OK				2020/04/20 17:33:04.068	
3	/login	1	Page Load	load	OK				2020/04/20 17:33:04.391	
3	/login	2	ユーザ名入力	click	OK				2020/04/20 17:33:04.408	
3	/login	3	ユーザ名入力	change	OK		["testuser"]		2020/04/20 17:33:05.484	
3	/login	4	パスワード入力	change	OK		["1"]		2020/04/20 17:33:06.585	
3	/login	5	ログインボタン	click	OK				2020/04/20 17:33:07.694	
4	/login_redirect	1	Page Load	load	OK				2020/04/20 17:33:07.831	
5	/contents	1	Page Load	load	OK				2020/04/20 17:33:07.831	
5	/contents	2	ユーザ名表示	text-equal	NG	期待値は "TestUser"、検証値は "testuser"	["TestUser"]		2020/04/20 17:33:11.905	

Fig. 2.2-4 テスト実行結果詳細

エラーの発生状態によっては、上記(1)~(3) のテストレポートファイルおよび (5)~(7) のスクリーンショット関連ファイルが生成されていなかったり、テストレポートの記述ではエラーの原因がわからないことがあります。

テストコードを展開したフォルダの **/reports/<ブラウザセクション名>/test/** に出力されている gradle (テストプログラムを実行するミドルウェア) が生成した (4) **テストサマリファイル (index.html)** から、さらに詳しいエラーログを参照することができます。

また、エラーログ(TestSummary)は (2) ブラウザ毎テスト結果サマリファイル の下部の「実行ログ」からリンクされていますので、ここから確認することもできます。

Fig. 2.2-4-2 実行ログリンク

Testablishから始まる図の赤い枠で囲まれているところをクリックすると、詳細なエラーログが表示されますので、この ログをもとにテストが失敗した原因を探していきます。

大量のエラー行が表示されていますが、その中で ” Caused by:…” という行を確認してみてください。 この行のあたりに失敗の原因となっているメッセージが出ていることがよくあります。
(エラー状況によってはそうでない場合もあります。)

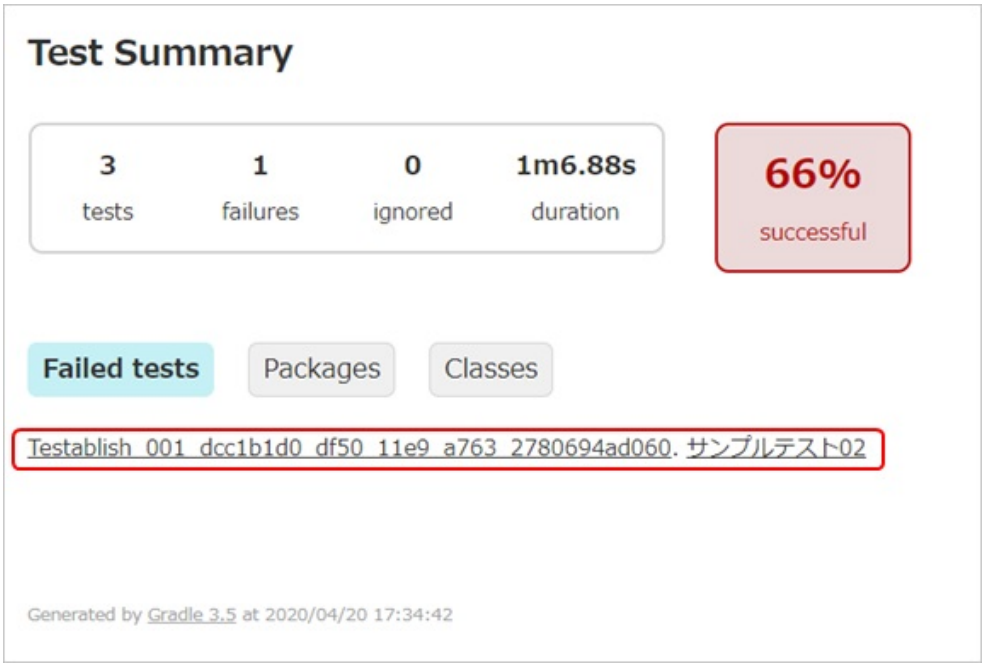


Fig. 2.2-5 Test Summary(失敗)

Class Testablish_001_dcc1b1d0_df50_11e9_a763_2780694ad060

1	1	0	16.245s
tests	failures	ignored	duration

0%
successful

Failed tests | Tests | Standard output | Standard error

サンプルテスト02

```
org.junit.experimental.theories.internal.ParameterizedAssertionError: auto("TestablishCapabilities[name='chrome-local', browser='chrome', version='null', platform='null', tester='tester_chrome', envs='[[Ljava.lang.String;@9b20748', headless='false', fastSetValue='false', successful='true', reportsFolder='reports/tests/001_chrome-local_サンプルテスト02(デフォルト)', afterReportsFolder='tests/001_chrome-local_サンプルテスト02(デフォルト)']" <from getParameters[0]>)
        at org.junit.experimental.theories.Theories$TheoryAnchor.reportParameterizedError(Theories.java:288)
        at org.junit.experimental.theories.Theories$TheoryAnchor$1$1.evaluate(Theories.java:237)
        at org.junit.experimental.theories.Theories$TheoryAnchor.runWithCompleteAssignment(Theories.java:218)
        at org.junit.experimental.theories.Theories$TheoryAnchor.runWithAssignment(Theories.java:204)
        at org.junit.experimental.theories.Theories$TheoryAnchor.runWithIncompleteAssignment(Theories.java:212)
        at org.junit.experimental.theories.Theories$TheoryAnchor.runWithAssignment(Theories.java:202)
        at org.junit.experimental.theories.Theories$TheoryAnchor.evaluate(Theories.java:187)
        at org.junit.runners.ParentRunner.runLeaf(ParentRunner.java:325)
        at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:78)
        at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:57)
        at org.junit.runners.ParentRunner$3.run(ParentRunner.java:290)
        at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:71)
        at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:288)
        at org.junit.runners.ParentRunner.access$000(ParentRunner.java:58)
        at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:268)
        at org.junit.internal.runners.statements.RunAfters.evaluate(RunAfters.java:27)
        at org.junit.runners.ParentRunner.run(ParentRunner.java:363)
        at org.gradle.api.internal.tasks.testing.junit.JUnitTestClassExecuter.runTestClass(JUnitTestClassExecuter.java:114)
        at org.gradle.api.internal.tasks.testing.junit.JUnitTestClassExecuter.execute(JUnitTestClassExecuter.java:57)
        at org.gradle.api.internal.tasks.testing.junit.JUnitTestClassProcessor.processTestClass(JUnitTestClassProcessor.java:66)
        at org.gradle.api.internal.tasks.testing.SuiteTestClassProcessor.processTestClass(SuiteTestClassProcessor.java:51)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at org.gradle.internal.dispatch.ReflectionDispatch.dispatch(ReflectionDispatch.java:35)
        at org.gradle.internal.dispatch.ReflectionDispatch.dispatch(ReflectionDispatch.java:24)
        at org.gradle.internal.dispatch.ContextClassLoaderDispatch.dispatch(ContextClassLoaderDispatch.java:32)
        at org.gradle.internal.dispatch.ProxyDispatchAdapter$DispatchingInvocationHandler.invoke(ProxyDispatchAdapter.java:93)
        at com.sun.proxy.$Proxy2.processTestClass(Unknown Source)
        at org.gradle.api.internal.tasks.testing.worker.TestWorker.processTestClass(TestWorker.java:109)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at org.gradle.internal.dispatch.ReflectionDispatch.dispatch(ReflectionDispatch.java:35)
        at org.gradle.internal.dispatch.ReflectionDispatch.dispatch(ReflectionDispatch.java:24)
        at org.gradle.internal.remote.internal.hub.MessageHubBackedObjectConnection$DispatchWrapper.dispatch(MessageHubBackedObjectConnection.java:147)
        at org.gradle.internal.remote.internal.hub.MessageHubBackedObjectConnection$DispatchWrapper.dispatch(MessageHubBackedObjectConnection.java:129)
        at org.gradle.internal.remote.internal.hub.MessageHub$Handler.run(MessageHub.java:404)
        at org.gradle.internal.concurrent.ExecutorPolicy$CatchAndRecordFailures.onExecute(ExecutorPolicy.java:63)
        at org.gradle.internal.concurrent.StoppableExecutorImpl$1.run(StoppableExecutorImpl.java:46)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
        at java.lang.Thread.run(Thread.java:748)
Caused by: java.lang.AssertionError: java.lang.Exception: java.lang.AssertionError: assertion
Expected: is "TestUser"
     but: was "testuser"
        at Testablish_001_dcc1b1d0_df50_11e9_a763_2780694ad060.step_5_2(Testablish_001_dcc1b1d0_df50_11e9_a763_2780694ad060.java:460)
        at Testablish_001_dcc1b1d0_df50_11e9_a763_2780694ad060.step_5(Testablish_001_dcc1b1d0_df50_11e9_a763_2780694ad060.java:423)
        at Testablish_001_dcc1b1d0_df50_11e9_a763_2780694ad060.auto(Testablish_001_dcc1b1d0_df50_11e9_a763_2780694ad060.java:147)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:50)
        at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:12)
        at org.junit.runners.model.FrameworkMethod.invokeExplosively(FrameworkMethod.java:47)
        at org.junit.experimental.theories.Theories$TheoryAnchor$2.evaluate(Theories.java:274)
        at org.junit.internal.runners.statements.RunAfters.evaluate(RunAfters.java:27)
        at org.junit.rules.TestWatcher$1.evaluate(TestWatcher.java:55)
        at org.junit.rules.RunRules.evaluate(RunRules.java:20)
        at org.junit.experimental.theories.Theories$TheoryAnchor$1$1.evaluate(Theories.java:232)
        at org.junit.experimental.theories.Theories$TheoryAnchor.runWithCompleteAssignment(Theories.java:218)
        at org.junit.experimental.theories.Theories$TheoryAnchor.runWithAssignment(Theories.java:204)
        at org.junit.experimental.theories.Theories$TheoryAnchor.runWithIncompleteAssignment(Theories.java:212)
        at org.junit.experimental.theories.Theories$TheoryAnchor.runWithAssignment(Theories.java:202)
        at org.junit.experimental.theories.Theories$TheoryAnchor.evaluate(Theories.java:187)
        at org.junit.runners.ParentRunner.runLeaf(ParentRunner.java:325)
        at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:78)
        at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:57)
        at org.junit.runners.ParentRunner$3.run(ParentRunner.java:290)
        at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:71)
        at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:288)
        at org.junit.runners.ParentRunner.access$000(ParentRunner.java:58)
        at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:268)
        at org.junit.internal.runners.statements.RunAfters.evaluate(RunAfters.java:27)
        at org.junit.runners.ParentRunner.run(ParentRunner.java:363)
        at org.gradle.api.internal.tasks.testing.junit.JUnitTestClassExecuter.runTestClass(JUnitTestClassExecuter.java:114)
        at org.gradle.api.internal.tasks.testing.junit.JUnitTestClassExecuter.execute(JUnitTestClassExecuter.java:57)
        at org.gradle.api.internal.tasks.testing.junit.JUnitTestClassProcessor.processTestClass(JUnitTestClassProcessor.java:66)
        at org.gradle.api.internal.tasks.testing.SuiteTestClassProcessor.processTestClass(SuiteTestClassProcessor.java:51)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at org.gradle.internal.dispatch.ReflectionDispatch.dispatch(ReflectionDispatch.java:35)
        at org.gradle.internal.dispatch.ReflectionDispatch.dispatch(ReflectionDispatch.java:24)
        at org.gradle.internal.dispatch.ContextClassLoaderDispatch.dispatch(ContextClassLoaderDispatch.java:32)
        at org.gradle.internal.dispatch.ProxyDispatchAdapter$DispatchingInvocationHandler.invoke(ProxyDispatchAdapter.java:93)
        at com.sun.proxy.$Proxy2.processTestClass(Unknown Source)
        at org.gradle.api.internal.tasks.testing.worker.TestWorker.processTestClass(TestWorker.java:109)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at org.gradle.internal.dispatch.ReflectionDispatch.dispatch(ReflectionDispatch.java:35)
        at org.gradle.internal.dispatch.ReflectionDispatch.dispatch(ReflectionDispatch.java:24)
        at org.gradle.internal.remote.internal.hub.MessageHubBackedObjectConnection$DispatchWrapper.dispatch(MessageHubBackedObjectConnection.java:147)
        at org.gradle.internal.remote.internal.hub.MessageHubBackedObjectConnection$DispatchWrapper.dispatch(MessageHubBackedObjectConnection.java:129)
        at org.gradle.internal.remote.internal.hub.MessageHub$Handler.run(MessageHub.java:404)
        at org.gradle.internal.concurrent.ExecutorPolicy$CatchAndRecordFailures.onExecute(ExecutorPolicy.java:63)
        at org.gradle.internal.concurrent.StoppableExecutorImpl$1.run(StoppableExecutorImpl.java:46)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
        at java.lang.Thread.run(Thread.java:748)
Caused by: java.lang.AssertionError: assertion
Expected: is "TestUser"
     but: was "testuser"
        at org.hamcrest.MatcherAssert.assertThat(MatcherAssert.java:20)
        at org.junit.Assert.assertThat(Assert.java:956)
        at Testablish_001_dcc1b1d0_df50_11e9_a763_2780694ad060.step_5_2(Testablish_001_dcc1b1d0_df50_11e9_a763_2780694ad060.java:452)
        ... 57 more
java.lang.AssertionError: assertion
Expected: is "TestUser"
```

Fig. 2.2-6 テストエラーログ

2.3 テストスイートを実行する

1. テストスイート編集 画面 の **テストコードの出力ボタン** からテストスイートのテストコードをダウンロードしてください。
2. zipファイルを展開します。
   Zipファイル解凍時、テスト名に日本語が含まれている場合、解凍後のファイル名の文字化けを防ぐため、Unicode（UTF-8 / UTF8）に対応した圧縮解凍ソフト（7-Zip, Explzh 等）をご利用ください。
3. 2.1 自動テストを実行する と同様に[testablish-test.ini]を作成・設置をしてください。
4. 展開したテストフォルダ内にある **run-test.bat** ファイルをダブルクリックします。
5. テストの自動実行が開始されます。
   - iniファイルで指定した各ブラウザセクション毎に実行されます。
   - テストはスイートに設定した順序で実行されます。

以上により、テストスイートが実行されます。

例えば、スイートでは test01, test02 の順で設定し、iniファイルで セクションを [chrome-local] , [ie-local] の順で指定した場合、test01(Chrome) ⇒ test02(Chrome) ⇒ test01(ie) ⇒ test02(ie) の順で実行されます。

なお、スイートに設定しているテストに複数のパターンが設定されている場合、スイートで実行されるのはデフォルトのパターンのみになります。

レポートの確認については 2.2 テストレポートを確認する を参照してください。

III. 操作の流れ/機能説明 目次 / 使い方マニュアル 目次 に戻る

3. 変数の外部ファイル化とデータのマスク

Testablishでテストを作成・実行していく中で、テストシナリオやエビデンスにIDやパスワードなど、セキュリティ上 そのままの文字列を残したくないような値を設定することがあります。
テストシナリオに直書きで設定した値は、テスト仕様書やテスト結果レポートにもそのままの文字列で出力されます。

このようなケースでは、**値の変数化(定数)** と **実行結果のマスク**を利用して対応することができます。

以下にサンプルを使って手順を紹介します。
この例では、ログイン時のユーザ名入力とパスワード入力の値をを変数化(定数)し、パスワード入力の値については、実行結果をマスクしていきます。

1. 定数の定義
   変数メニューから「ログインユーザ」「パスワード」という変数を作成します。
   この時、定数化ボタンをONにして、**定数**として作成します。

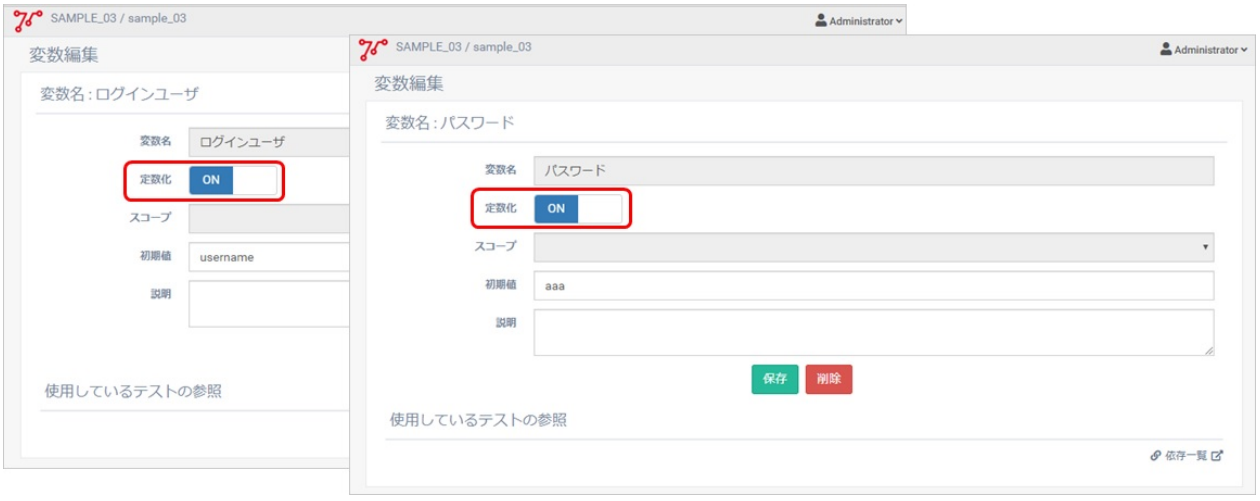


Fig. 3-1 定数の定義

2. テスト編集：定義した定数の使用(設定)
   作成した定数をテストシナリオに設定していきます。
   操作：ユーザ名入力　値設定：InputValue：**${ログインユーザ}** (プルダウンから選択できます)
   操作：パスワード入力　値設定：InputValue：**${パスワード}** (プルダウンから選択できます)
   パスワード入力のシーケンスで、「3.値変更」の文字列をクリックしてフィールドを展開し、**「実行結果をマスクする」にチェック**を入れておきます。
   テストシナリオを**保存**します。

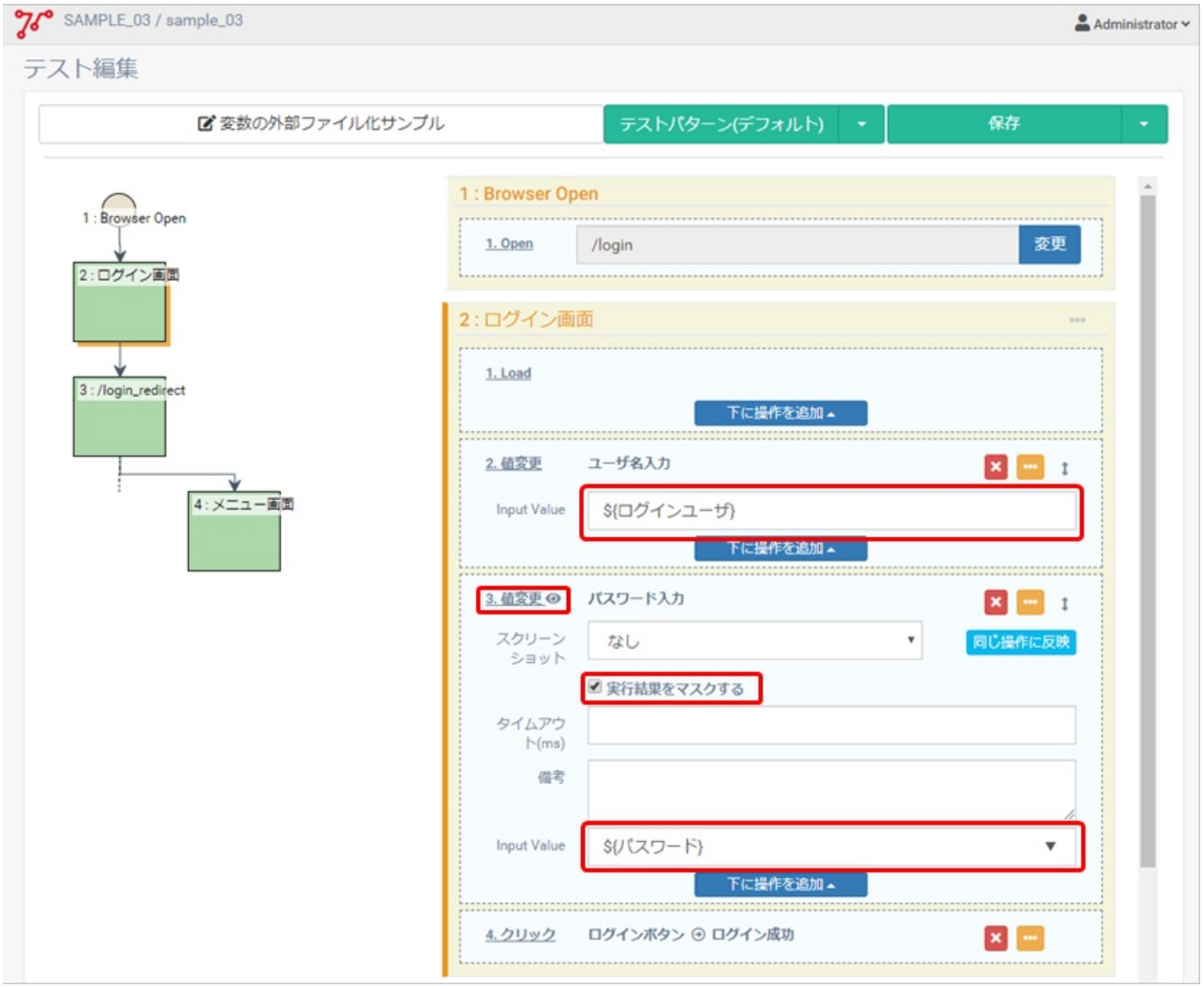


Fig. 3-2 定数の使用

3. テストコードを出力
   保存ボタン横の▼から テストコードを出力します。
   出力したテストコードのzipファイルを解凍・展開します。
   ※ Zipファイル解凍時、テスト名に日本語が含まれている場合、解凍後のファイル名の文字化けを防ぐため、Unicode（UTF-8 / UTF8）に対応した圧縮解凍ソフト（7-Zip, Explzh 等）をご利用ください。

4. jsonファイルの作成・編集
   テストシナリオで設定した **${ログインユーザ}** や **${パスワード}** に設定される値は、**testablish-test-data.json** という外部ファイルで設定します。

   出力したZipファイルを展開したフォルダ内には **testablish-test-data.json.sample** というファイルがあり、 変数メニューで定義した**定数**が json 形式で書き出されています。

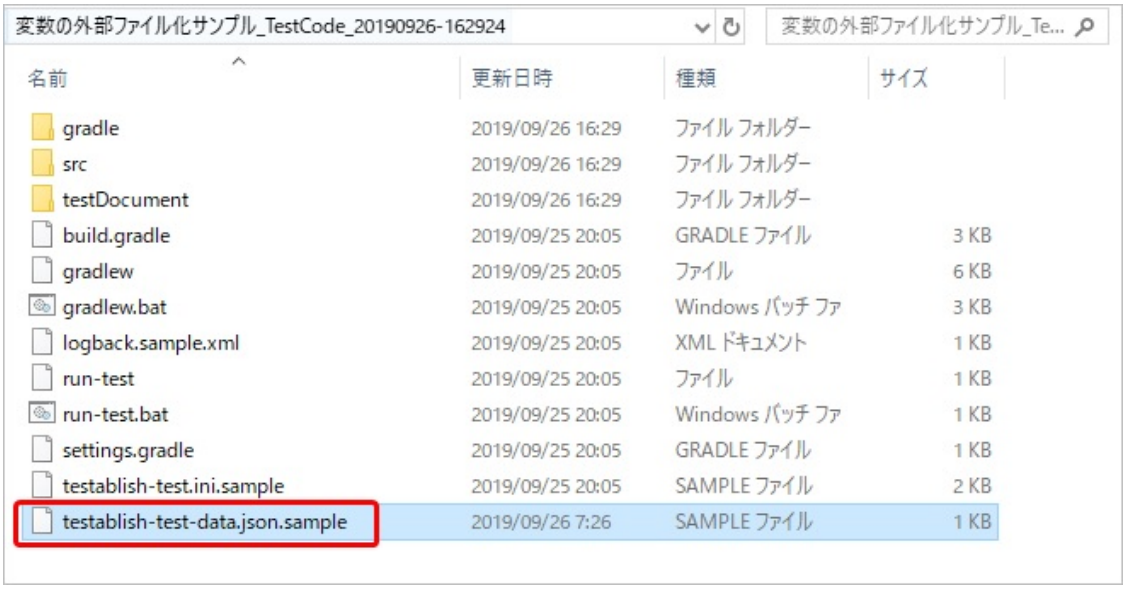


Fig. 3-3 展開したフォルダ内の *testablish-test-data.json.sample*ファイル

testablish-test-data.json.sample ファイルの値には定義時の初期値が設定されていますので、"定数名" : "値" の「値」の部分を適宜修正し、**testablish-test-data.json** というファイル名に**変更**して保存します。

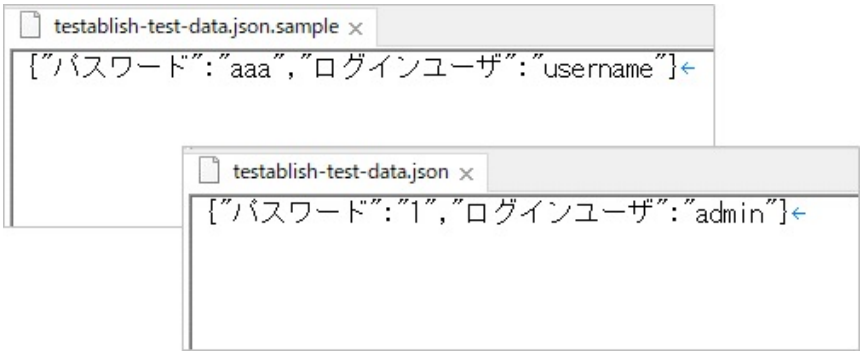


*Fig. 3-4 testablish-test-data.json*ファイル

5. テスト実行
   testablish-test-data.json ファイルを作成した後、テストを実行します。

6. 結果の確認
   テストの実行結果を確認します。

   テスト編集画面で設定した変数(定数)名について、[テストコード展開フォルダ]/reports/tests/[セクション名]_テスト名/ フォルダ以下にあるテスト結果詳細ファイルを確認してみます。

   このファイルの「設定値」「変数値」欄を見てみると、**「実行結果をマスクする」** にチェックを入れていない「ユーザ名入力」のシーケンスでは変数(定数)名と実際の変数値、 チェックを入れた「パスワード入力」のシーケンスでは両方がマスクされて「XXXXX」という値になっています。

変数の外部ファイル化サンプル(デフォルト)テスト結果

成功

テスト開始日時: 2020/04/21 16:31:58

画像サイズ 10 %

summry > chrome-local > 変数の外部ファイル化サンプル(デフォルト)

ステップ	画面名	シーケンス	ターゲット	アクション	状態	エラー	設定値	変数値	実行日時	画面
1	ログイン画面	1		open	OK		["/login"]		2020/04/21 16:32:03.991	
2	ログイン画面	1	Page Load	load	OK				2020/04/21 16:32:05.254	
2	ログイン画面	2	ユーザ名入力	change	OK		["${ログインユーザ}"]	[admin]	2020/04/21 16:32:05.275	
2	ログイン画面	3	パスワード入力	change	OK		XXXXX	XXXXX	2020/04/21 16:32:06.431	
2	ログイン画面	4	ログインボタン	click	OK				2020/04/21 16:32:07.553	
3	/login_redirect	1	Page Load	load	OK				2020/04/21 16:32:07.687	
4	メニュー画面	1	Page Load	load	OK				2020/04/21 16:32:07.688	

Fig. 3-5 テスト結果詳細ファイル

テスト編集画面で設定した変数(定数)名は、[テストコード展開フォルダ]/testDocument/result/ 以下にあるテスト仕様書の操作手順の記述でも変数(定数)名に置き換えられています。

システム名	SAMPLE_03	変数の外部ファイル化サンプル(デフォルト)	作成者	admin	2019/09/26
テストNo.	val_1		更新者	admin	2019/09/26

テストケース					スクリーンショット	操作手順	備考	テスト			結果	不具合管理No.
No.	画面名	No.	操作種別	操作				実施者	予定日	実施日		
1	ログイン画面											
		1	ブラウザオープン			/loginを開く		test_chrome		2020/4/21	OK	
2	ログイン画面											
		1	ユーザ操作	初期表示		ログイン画面が表示される		test_chrome		2020/4/21	OK	
		2	ユーザ操作	値変更		ユーザ名入力 の値を ${ログインユーザ} に変更する		test_chrome		2020/4/21	OK	
		3	ユーザ操作	値変更		パスワード入力 の値を ${パスワード} に変更する		test_chrome		2020/4/21	OK	
		4	ユーザ操作	クリック		ログインボタン をクリックする		test_chrome		2020/4/21	OK	
3	/login_redirect											
		1	ユーザ操作	初期表示		/login_redirectが表示される		test_chrome		2020/4/21	OK	
4	メニュー画面											
		1	ユーザ操作	初期表示		メニュー画面をTopWindowで開く		test_chrome		2020/4/21	OK	

Fig. 3-6 結果がマージされたテスト仕様書

このように、**定数** を使うことで値を外部ファイル(jsonファイル)化し、 このファイル内で値を管理することができるようになります。
また、**実行結果のマスク**を利用することで、出力結果からも値を隠すことができます。

テスト実行毎に定数の値を変更するには、外部ファイル(jsonファイル)の内容を書き換えればよく、変数定義そのものを編集する必要はありません。

III. 操作の流れ/機能説明 目次 / 使い方マニュアル 目次 に戻る

4. フレームセットや iframe を含むページの定義とテスト作成

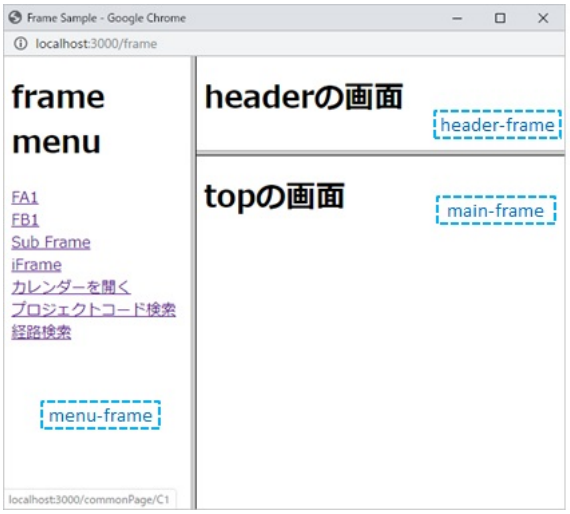
この章ではサンプルを使ってフレームセット や iframe を使っているページの作成方法やテスト作成の手順を紹介します。

4.1 フレームセット

フレームセットを使ったページの定義およびテストの作成方法を紹介します。

4.1.1 フレームセットの定義

以下の HTMLで 表わされている FrameSample 画面のページ定義を作成していきます。



【HTMLソース(抜粋)】

```
<frameset cols="200,*">
    <frame src="/frame/menu" name="menu-frame">
    <frameset rows="100,*">
        <frame src="/frame/header" name="header-frame">
        <frame src="/frame/top" name="main-frame">
    </frameset>
    <noframes>
     ...
    </noframes>
</frameset>
```

Fig. 4.1-1 FrameSample 画面

FrameSample 画面 は、frameset を使って、左右に2分割し、さらに右フレームを上下に2分割しています。
この画面を testablish で表現するためには、フレームに表示する個々のページ情報とページを表示するフレームセットを定義するページ情報の定義が必要です。

フレームセットを定義する前に、個々のページ情報を定義しておく必要があります。

Fig. 4.1-2 フレームに表示するページ情報

フレームセット定義していきます。
ページ一覧 の **新規ページ作成** ボタンをクリックして表示されるダイアログで、 **ページキー** と **ページ名** を設定し、**フレームを使用する**チェックボックスにチェックし、**作成して進む** ボタンをクリックします。

Fig. 4.1-2 フレームセットページ新規作成

新規ページ作成ダイアログで**フレームを使用する**をチェックすることで、一般のページ編集画面とは異なるフレームセット定義用のページが開きます。

初期状態では、1行x1列(分割なし)の状態です。
まず、列数のスライドバーをスライドして、左右に分割します。
すると、1行x2列 の設定行が追加されます。

次に右側を上下に2分割するため、1行x2列 の フレームタイププルダウンから **フレームセット** を選択します。
分割用のスライドバーが表示されますので、ここで行数スライドバーをスライドして上下に2分割すると、2行x1列 の設定行が追加されます。

それぞれのフレーム名、読み込むページを設定をして保存してください。

Fig. 4.1-3 フレームセット定義ページ

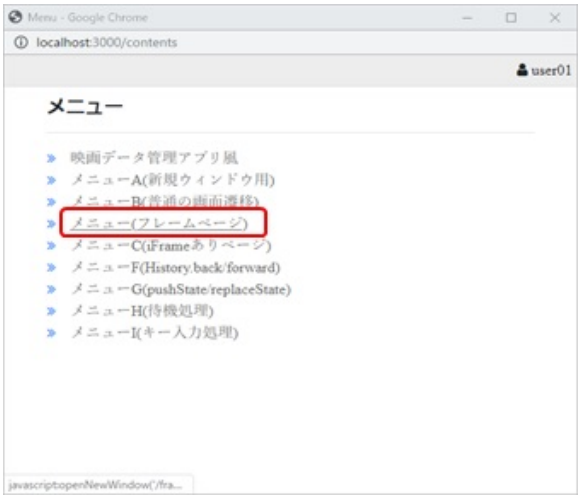
設定項目の詳細は 4.2.1.3　フレームページの設定 を参照してください。

4.1.2 フレームセットページを含むテストの作成

フレームセットページを含む画面遷移を実行するテストを作成していきます。
この例では以下のような画面遷移をします。

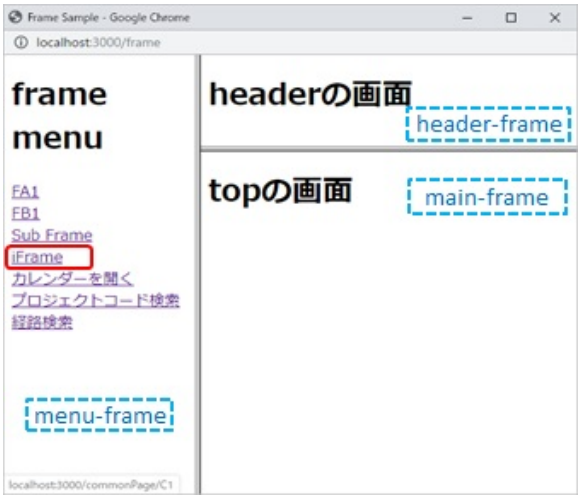
1. ブラウザオープン
   メニュー画面から操作を開始します。

2. メニュー画面
   「**メニュー(フレームページ)**」リンクをクリックします。



3. フレームセット画面の表示
   新しいウィンドウが開き、フレームセット画面が表示されます。
   フレームセット画面は、左ペイン(menu-frame)、右上ペイン(header-frame)、右下ペイン(main-frame)の3つに分割されています。

4. フレームセット画面
   左ペイン(menu-frame)に表示されている 「**iFrame**」リンクをクリックします。



5. C1画面
   右下ペイン(main-frame)に表示されている topの画面が c1の画面 に書き換わります。
   C1画面にある 「**次に進む**」ボタンをクリックします。

6. C2画面
   右下ペイン(main-frame)に表示されている C1の画面が C2の画面 に書き換わります。

まとめると、以下のような遷移になります。

- メニューからフレームセット画面に遷移
- 左ペインのリンクをクリックして右下ペインの画面をC1画面に書換え
- 右下ペインのC1画面のボタンクリックで右下ペイン内で C2画面に遷移

上記の遷移を再現するテストシナリオを作成していきます。

- 新規テスト作成
  新規テスト作成ダイアログで テスト開始ページ、テスト番号、テスト名 を指定します。

- Step1: Browser Open
  テスト編集画面が表示され、開始ページが設定されています。
  作成を開始するボタンをクリックします。
  Step2 として Menu画面のステップが追加されます。

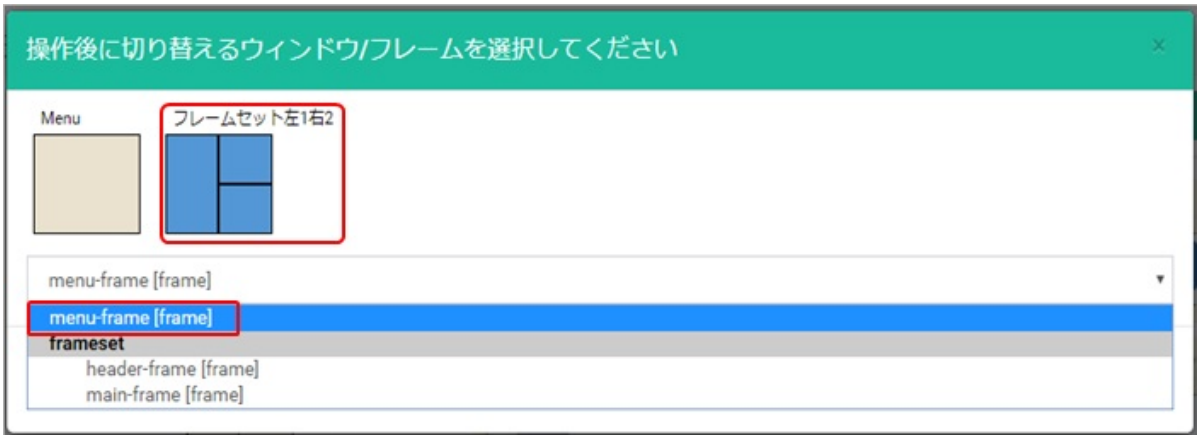
- Step2: Menu
  下に操作を追加ボタンをクリックして開くメニューから**操作追加**を選択します。

追加項目選択ダイアログで、「メニュー(フレームページ)」 を選択し、**Add**ボタンをクリックします。

対象のウェブアプリでは、このフレームセット画面は新規ウィンドウで開くページとなっていることに加えてこのフレームセット画面には複数のフレームがあるため、ウィンドウとフレームを選択する必要があります。

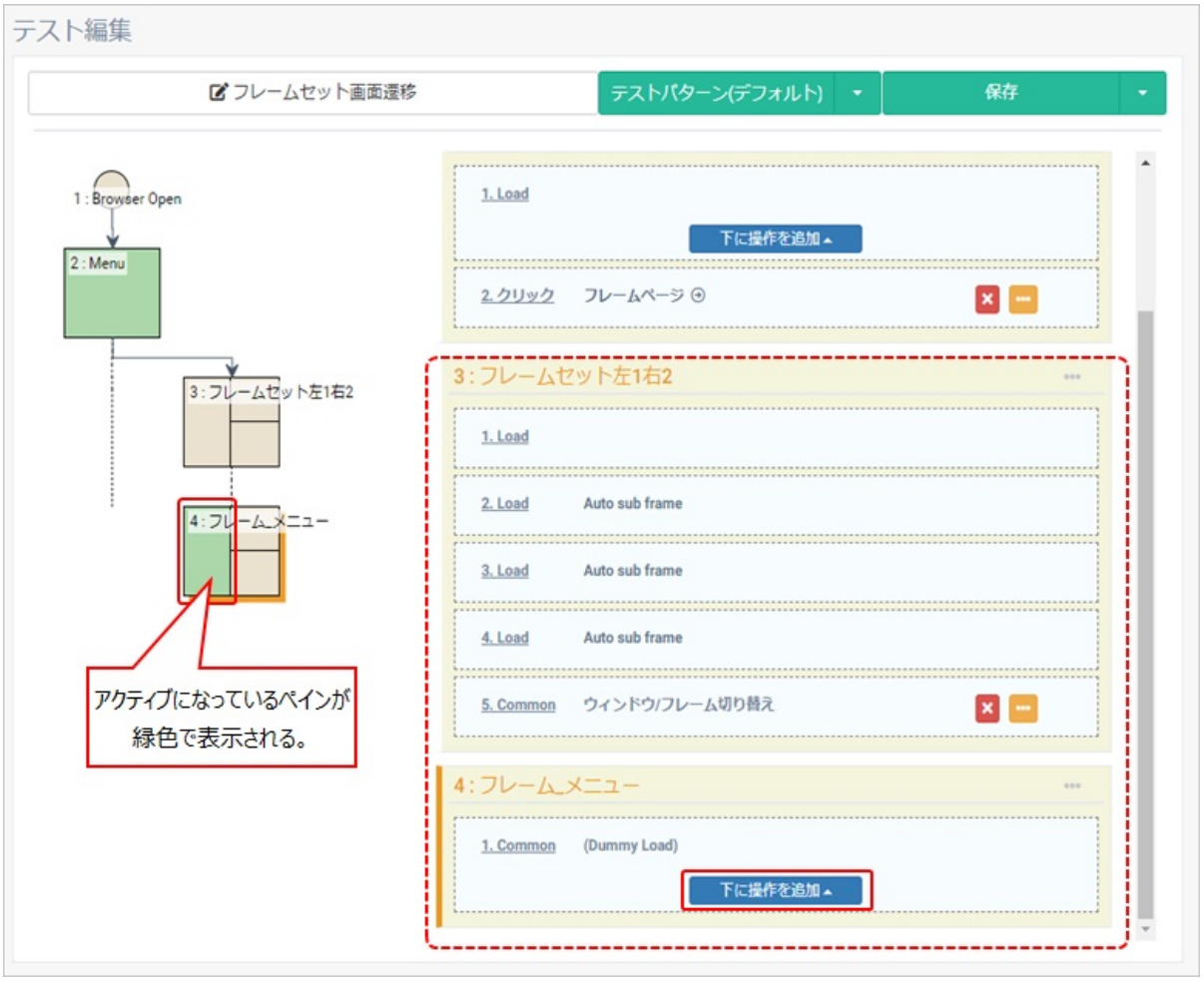
この例では、フレームセット画面の左ペイン(menu-frame)を選択します。
右側の「フレームセット左1右2」をクリックします。選択した画面図が青色になります。

ダイアログ下部にフレームを選択するためのプルダウンが表示されますので、プルダウンに表示されるリストの中から操作後に切り替えるフレームを選択します。



ダイアログ右下の**選択**ボタンをクリックしてダイアログを閉じます。

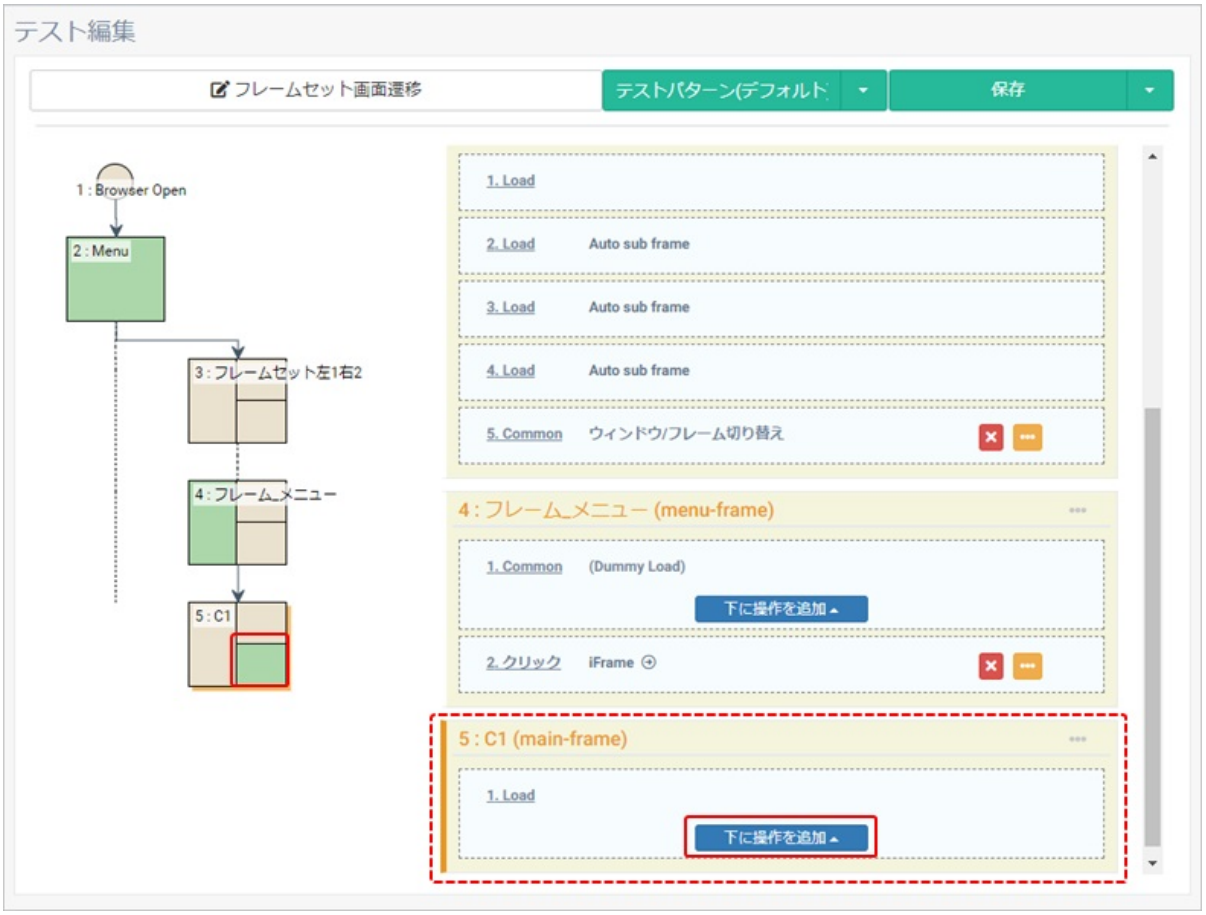
- Step3: フレームセット左1右2
- Step4: フレーム_メニュー
  フレームセット画面および各フレームの画面のロード操作と、先ほどのダイアログで指定したウィンドウフレームの切り替え操作が追加されています。
  フレームの切り替えの結果、Step4:では左ペインがアクティブになっている状態です。
  テスト編集画面の左側のテストフロー図では、そのステップでアクティブになっているペインが緑色で表示されます。



下に操作を追加ボタンをクリックして開くメニューから**操作追加**を選択します。

表示される**追加項目選択**ダイアログで、「iFrame」 を選択し、**Add**ボタンをクリックします。

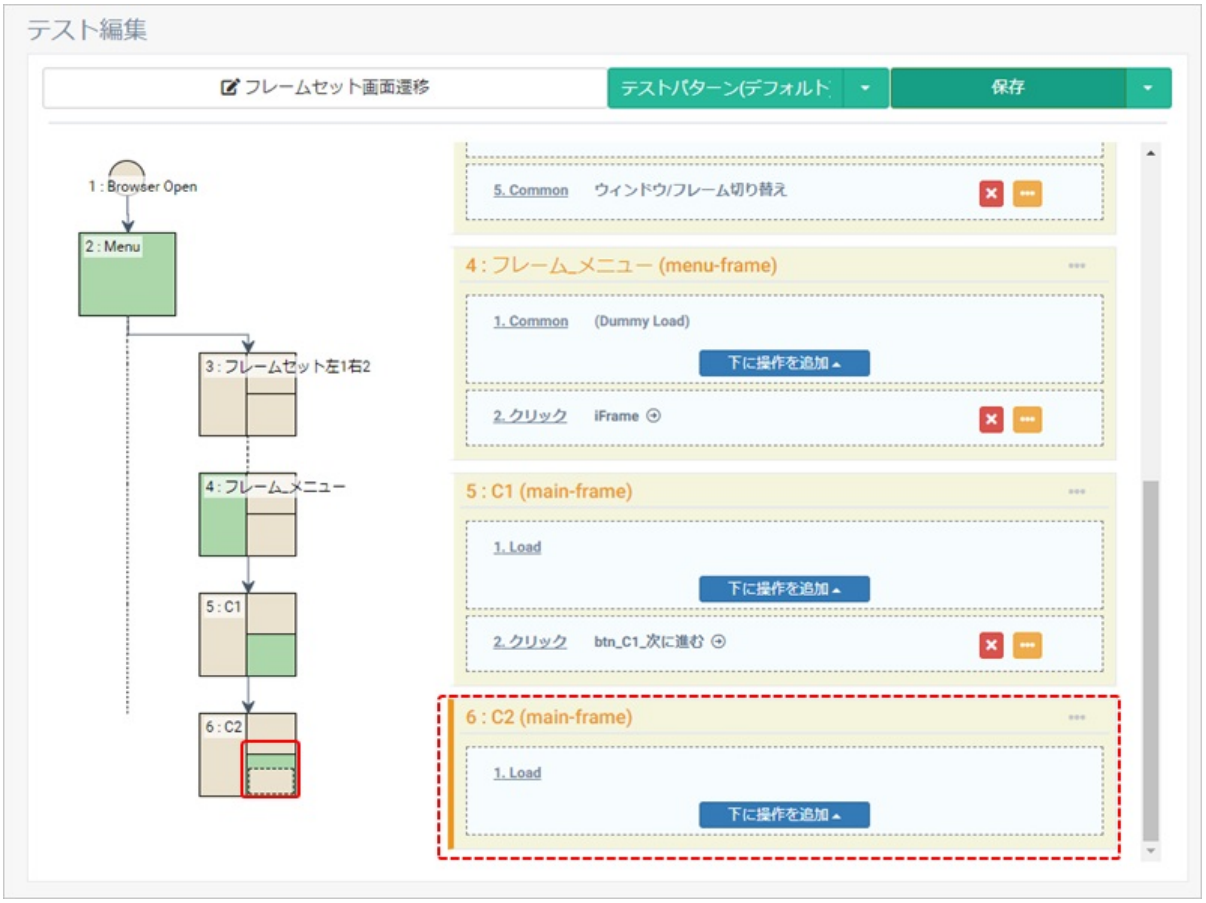
- Step5: C1画面
  前のステップで指定した「iFrame」のリンクは、右下(main-frame)のペインにC1画面を表示するように指定されているため、 アクティブなペインが右下(main-frame)に移り、C1画面のステップが追加されます。



下に操作を追加ボタンをクリックして開くメニューから**操作追加**を選択します。
表示される**追加項目選択**ダイアログで、「btn_C1_次に進む」 を選択し、**Add**ボタンをクリック

します。

- Step6: C2画面
  アクティブなペインは右下(main-frame)のまま、C2画面のステップが追加されます。



4.2 iframe

iframe を含むページの定義およびテストの作成方法を紹介します。

4.2.1 iframeの定義

iframe は常に自分自身の画面に表示されるものです。
画面描画時に生成される iframe は **PageLoad** の **操作後の流れ** 、何かの操作によって**動的** に生成される iframe は **その操作** の **操作後の流れ** として設定します。

以下の HTMLで 表わされている C2 画面のページ定義を作成していきます。

【HTMLソース(抜粋)】

```
<body>
  <h1>C2の画面</h1><br/>
  <form method="get" action="/commonPage/C3">
    <input type="submit" value="次に進む" />
  </form>
  <form method="get" action="/commonPage/C1">
    <input type="submit" value="前の画面に戻る" />
  </form>

  <iframe src="/commonPage/IF1"></iframe>

</body>
```

Fig. 4.2-1 C2 画面

C2画面では、一つの iframe を含み、その iframe には IF1画面が表示されています。

この画面を testablish で表現するためには、iframe を含む画面(この例ではC2)の定義をする前に、iframe 内に表示する個々のページ(この例ではIF1) が定義されている必要があります。

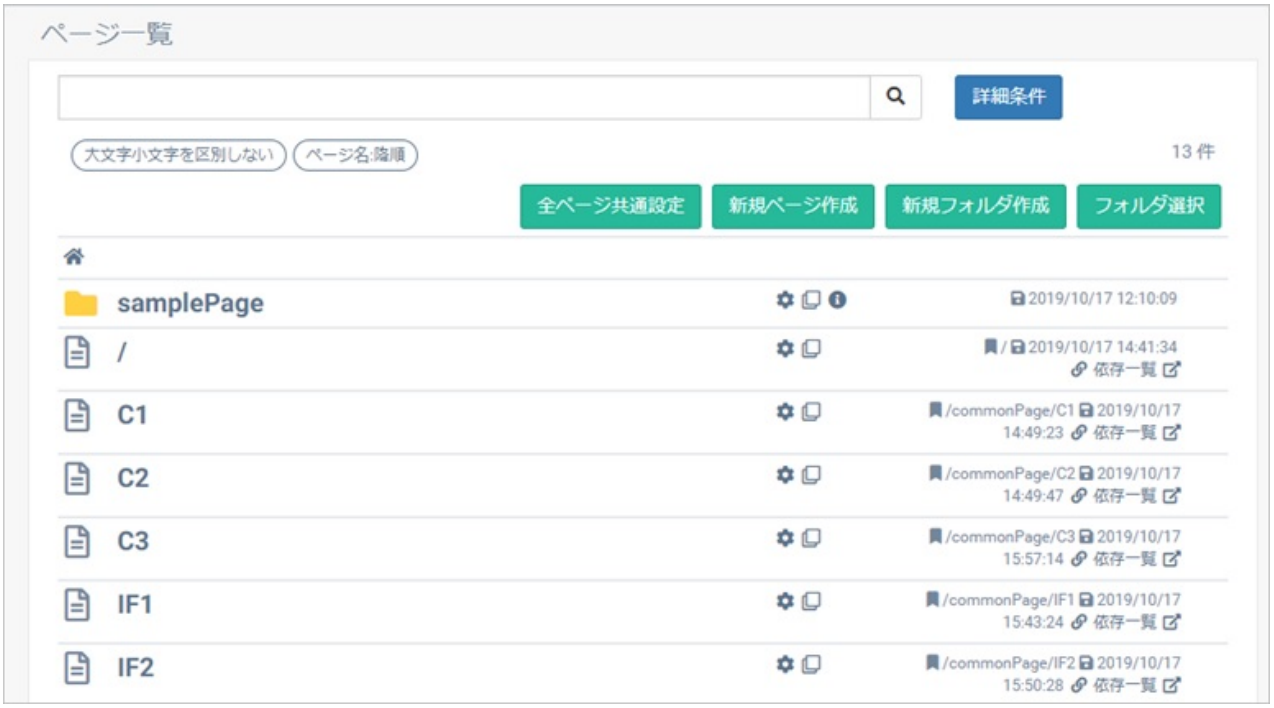


*Fig. 4.2-2 iframe*に表示されるページと*iframe*を表示するページ

C2画面の iframe は C2画面描画時に生成されているため、C2画面の **PageLoad** の **操作後の流れ** で設定します。
操作後の流れにチェックを入れると、設定行が表示されます。
右側にある **設定**ボタンをクリックすると、**操作後の処理設定ダイアログ**が表示されます。

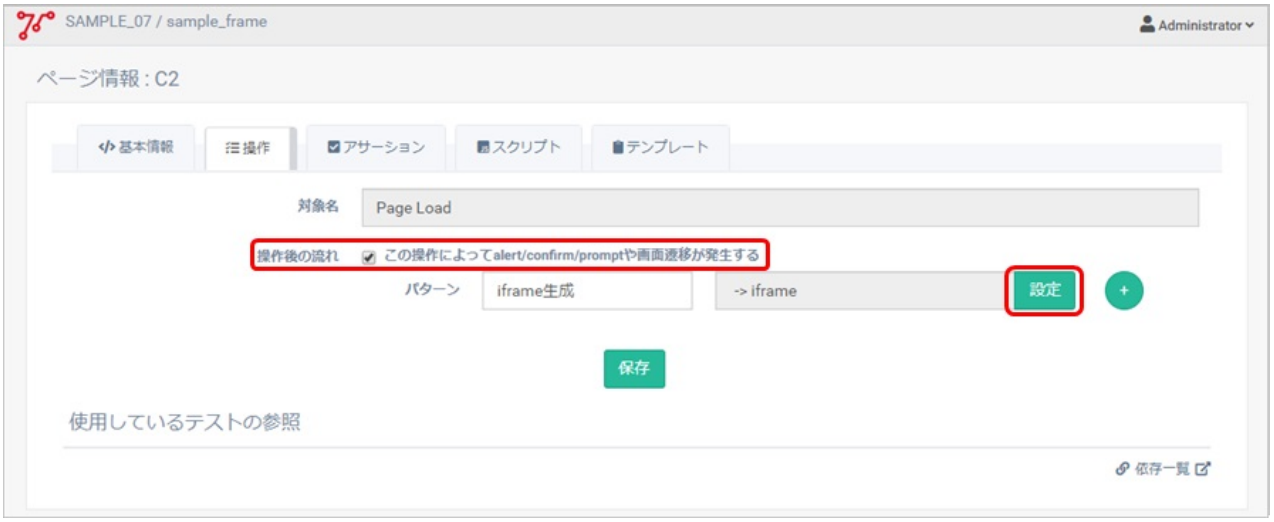


Fig. 4.2-3 PageLoad 操作編集画面

このダイアログで **iframeを生成する** にチェックを入れると、iframe 設定行が表示されます。
設定 ボタンをクリックすると、**操作後の処理設定**ダイアログ が表示されます。
ここで、**セレクタ名** または **フレーム名**、iframe 内に **表示するページ** を設定します。

セレクタ名、フレーム名については、どちらか一方は必ず指定するようにしてください。
表示するページのプルダウンには、既に定義されているページがリストされますので、その中から選択します。

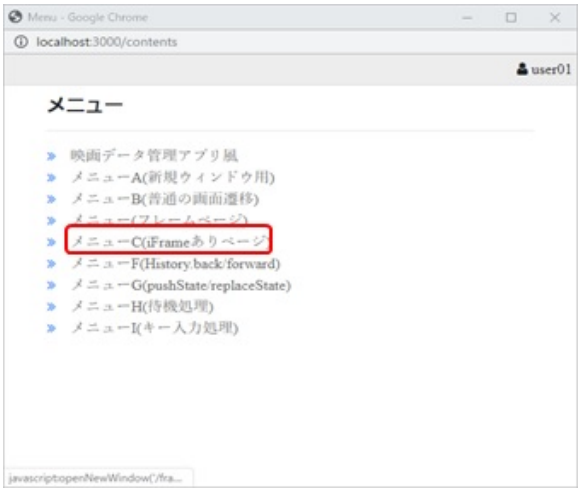
また、ページに複数の iframe がある場合は、ダイアログ内の iframe 設定行右側の **+** ボタンをクリックして行を追加し、設定を追加してください。

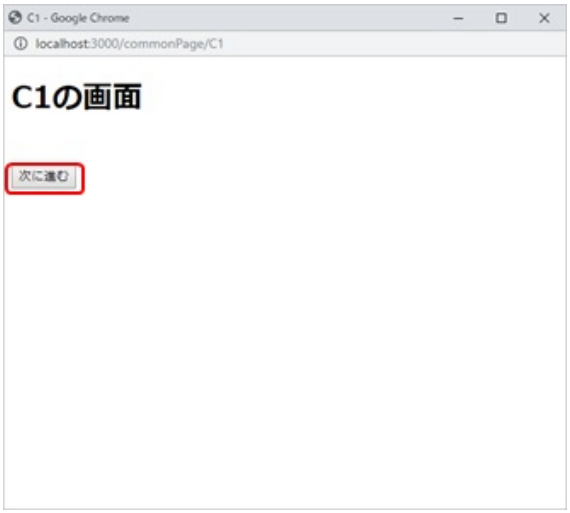
Fig. 4.2-4 操作後の処理設定ダイアログ

4.2.2 iframeを含むページのテスト作成

iframe を含む画面と iframe内での画面遷移を実行するテストを作成していきます。
この例では以下のような画面遷移をします。

1. ブラウザオープン
   メニュー画面から操作を開始します。

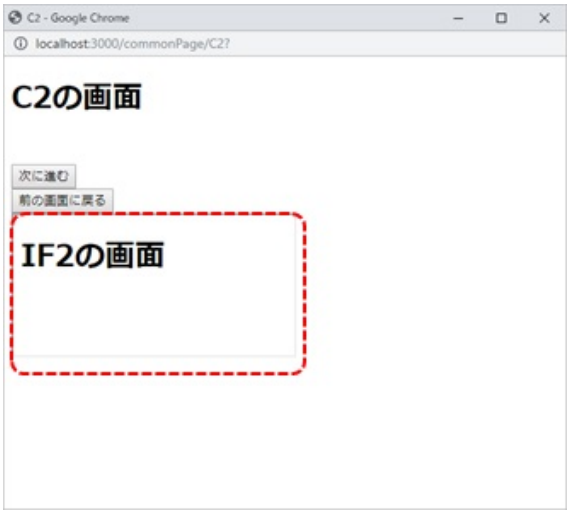
2. メニュー画面
   「**メニューC(iFrameありページ)**」リンクをクリックします。
   

3. C1画面
   新しいウィンドウが開き、C1画面が表示されます。
   次に進むボタンをクリックします。
   

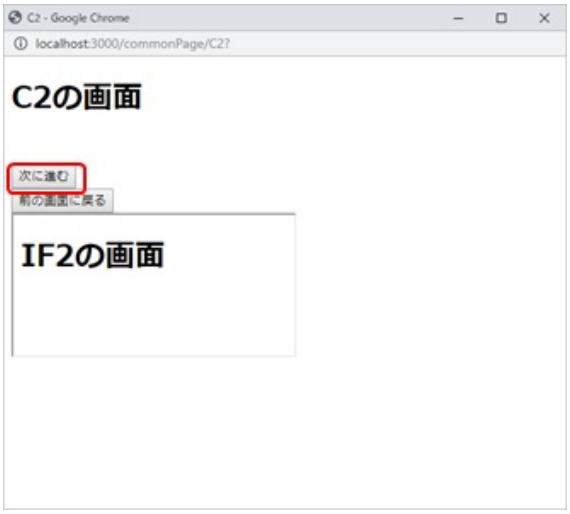
4. C2画面
   C2画面に遷移します。

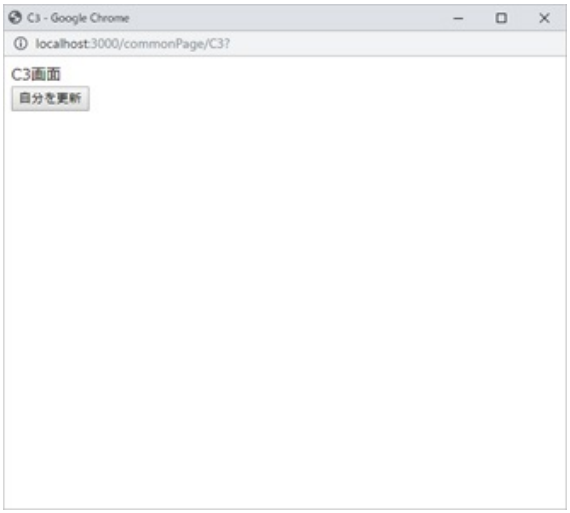
5. IF1画面
   iframe に表示されているIF1画面 内の **次に進む**ボタンをクリックします。

6. IF2画面
   iframe に表示されている画面がIF2画面に遷移します。



7. C2画面
   C2画面にある **次に進む**ボタンをクリックします。



8. C3画面
   C3画面に遷移します。


まとめると、以下のような遷移になります。

- メニューからC1画面に遷移
- C1画面からボタンクリックでC2画面に遷移
- C2画面内の iframe内の IF1画面のボタンクリックで iframe内で IF2画面に遷移
- C2画面のボタンクリックで C3画面へ遷移

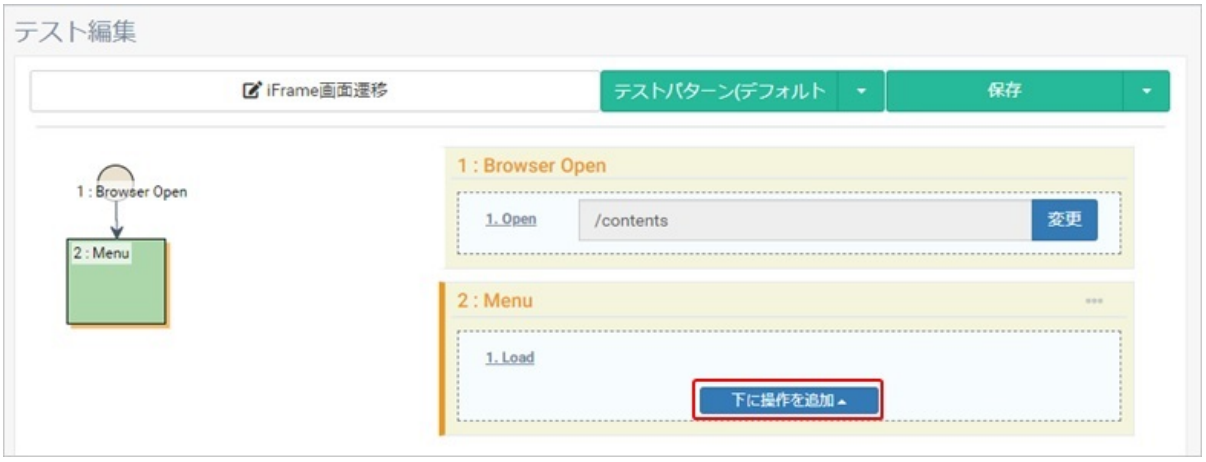
上記の遷移を再現するテストシナリオを作成していきます。

- 新規テスト作成
  新規テスト作成ダイアログで テスト開始ページ、テスト番号、テスト名 を指定します。

- Step1: Browser Open
  テスト編集画面が表示され、開始ページが設定されています。
  作成を開始するボタンをクリックします。

Step2 として Menu画面のステップが追加されます。

- Step2: Menu
  下に操作を追加ボタンをクリックして開くメニューから**操作追加**を選択します。



追加項目選択ダイアログで、「MenuC_iframeありページ」 を選択し、**Add**ボタンをクリックします。

- Step3: C1画面
  C1画面のステップが追加されます。

下に操作を追加ボタン をクリックして開くメニューから**操作追加**を選択します。

表示された**追加項目選択**ダイアログで、「btn_C1_次に進む」 を選択し、**Add**ボタンをクリックします。

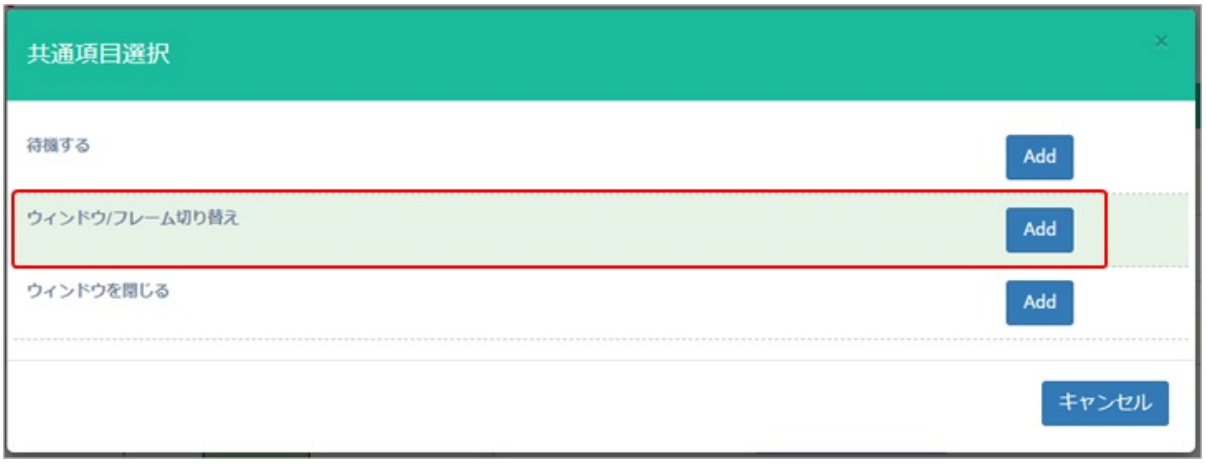
- Step4: C2画面
  C2画面のステップが追加されます。

  テスト編集画面左側のフロー図では、対象のウィンドウ内に iframe が含まれる場合、点線枠が表示されます。

次のステップで C2画面内の 「iframe に表示されている IF1画面内のボタンをクリックする」という操作を作成するためには、現在アクティブな C2画面にあるフォーカスを IF1画面に移さなければならないため、 ここでウィンドウを切替操作を追加します。

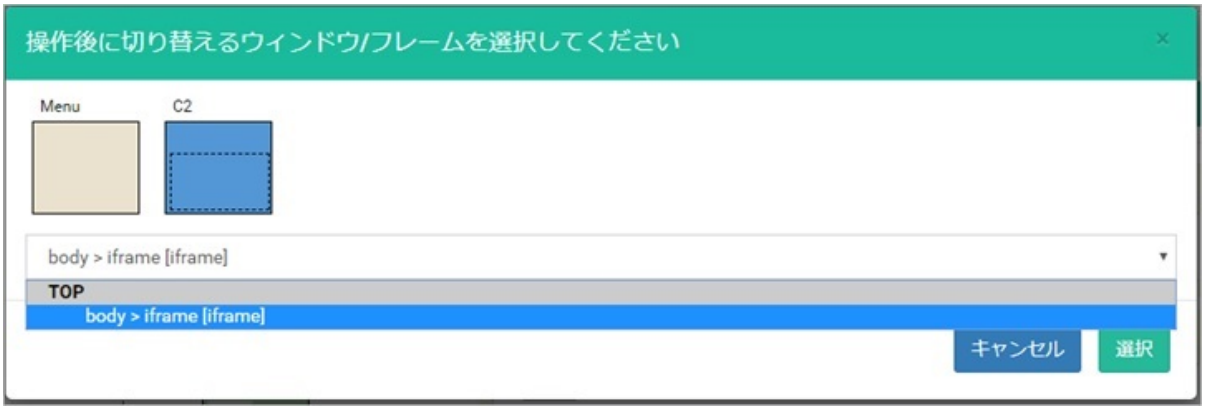
下に操作を追加ボタン をクリックして開くメニューから**ブラウザ操作追加**を選択します。

表示された**共通項目選択**ダイアログで、**ウィンドウ/フレーム切替** を選択し、**Add**ボタンをクリックします。



続けて **操作後に切り替えるウィンドウ/フレームを選択** するダイアログが表示されます。
この例では、右側の「C2」をクリックします。
選択したウィンドウ図が青色になります。

ダイアログ下部にフレームを選択するためのプルダウンが表示されます。
このプルダウンには、選択しているウィンドウの フレームや iframe の構造がリストされます。
この例では、操作後に切り替えるフレーム(body > iframe) を選択します。



ダイアログ右下の**選択**ボタンをクリックしてダイアログを閉じます。

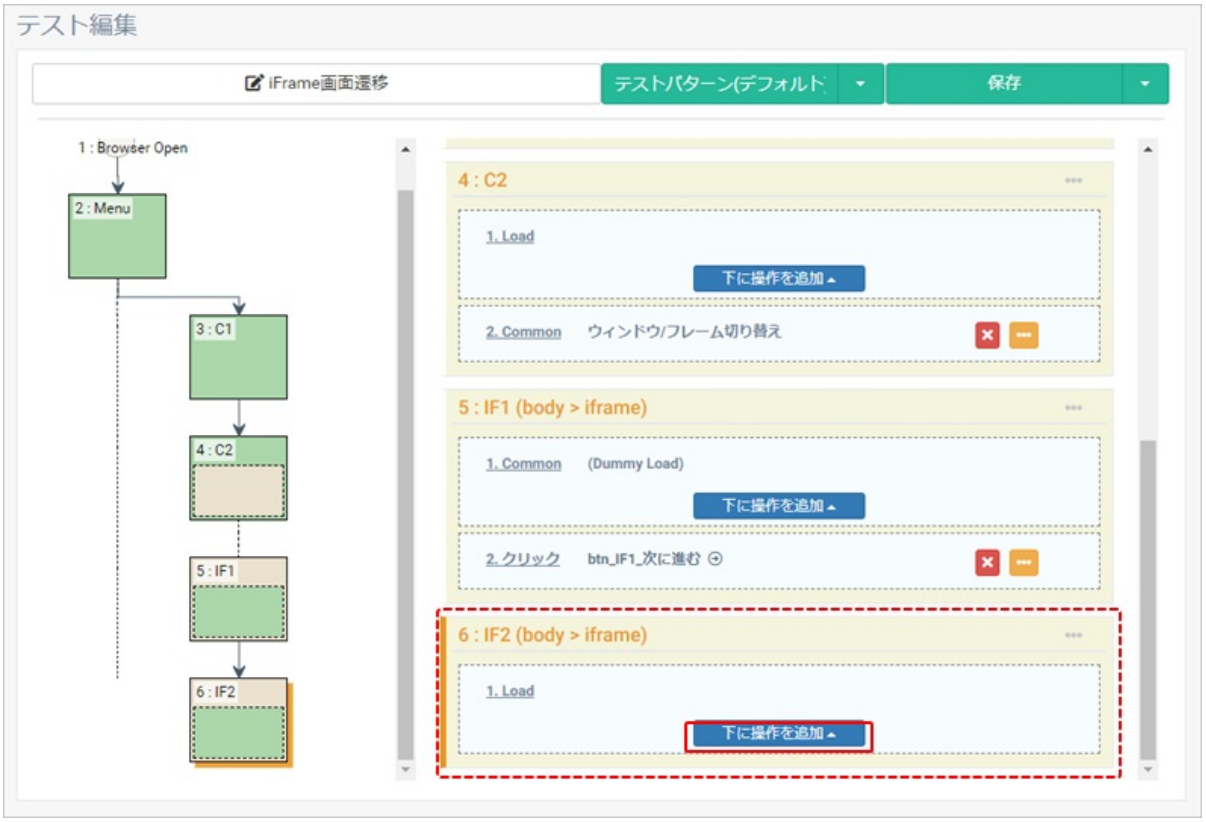
- Step5: IF1画面
  前ステップでのウィンドウ/フレーム切替操作の結果、iframe がアクティブになり、フォーカスはiframe内に表示されている IF1画面 に移っています。

  テスト編集画面左側のフロー図では、対象のウィンドウのアクティブな領域が緑色で表示され、ウィンドウ内で非アクティブな領域はベージュ色で表示されます。

IF1画面で **下に操作を追加**ボタン をクリックして開くメニューから**操作追加**を選択します。

表示された**追加項目選択**ダイアログで、「btn_IF1_次に進む」 を選択し、**Add**ボタンをクリックします。

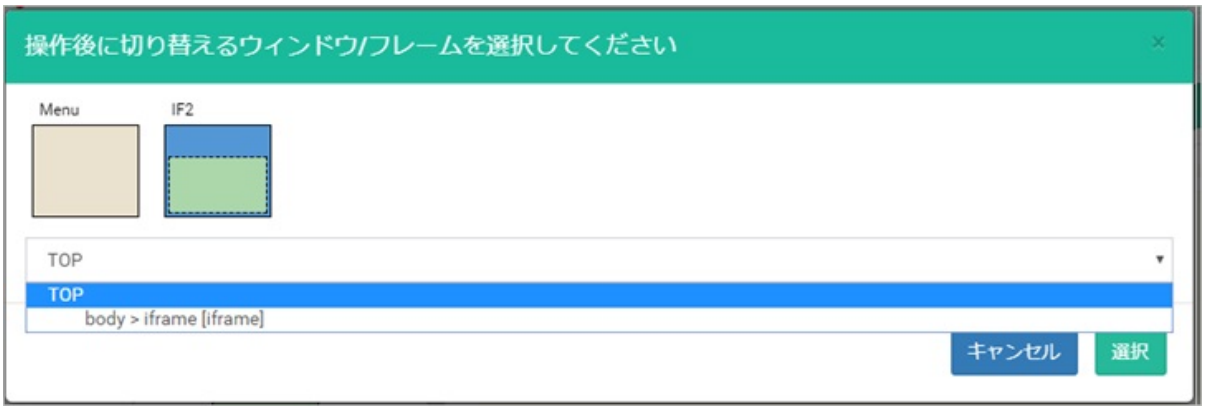
- Step6: IF2画面
  iframe内で IF2画面に遷移しています。



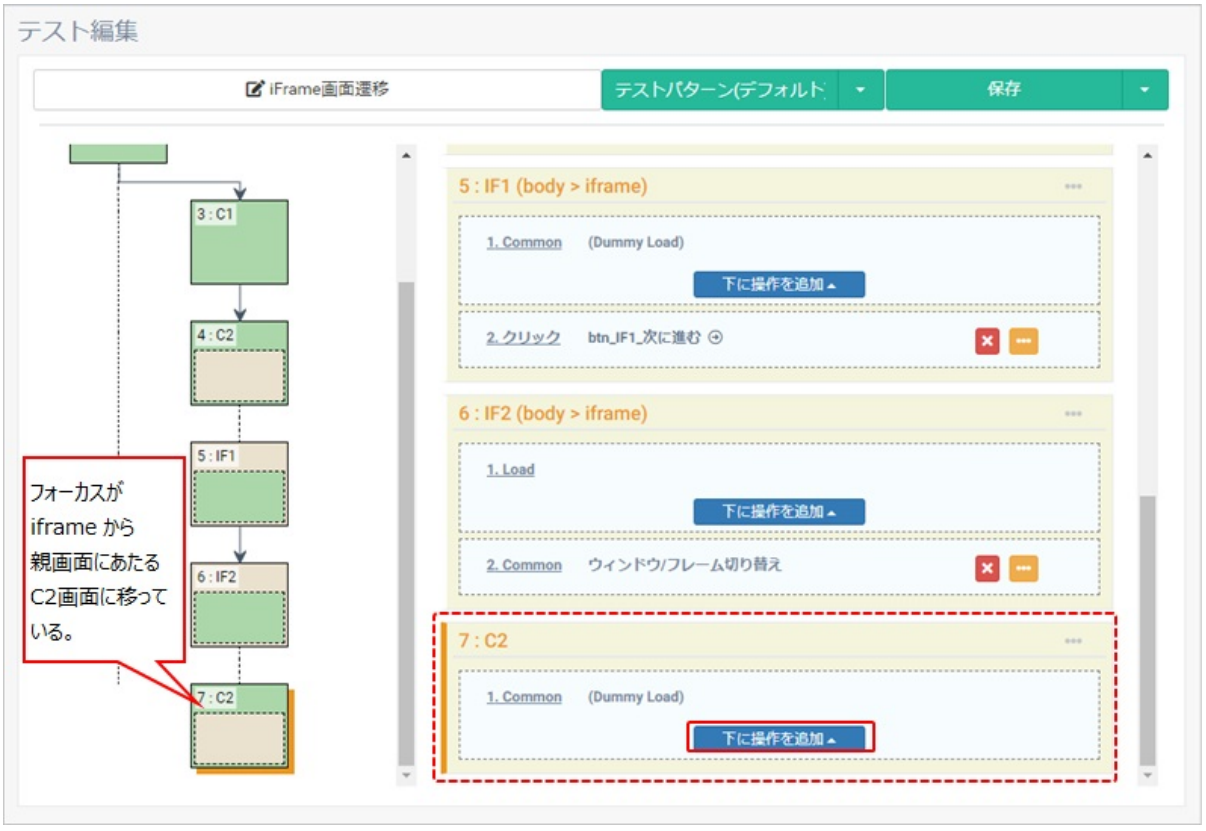
下に操作を追加ボタン をクリックして開くメニューから**ブラウザ操作追加**を選択します。

表示された**共通項目選択**ダイアログで、**ウィンドウ/フレーム切替** を選択し、**Add**ボタンをクリックします。

続けて表示されたダイアログで、IF2画面が含まれている右側を選択し、下部のプルダウンから、IF2画面を表示しているiframeがある画面(親画面C2) の TOP を選択します。



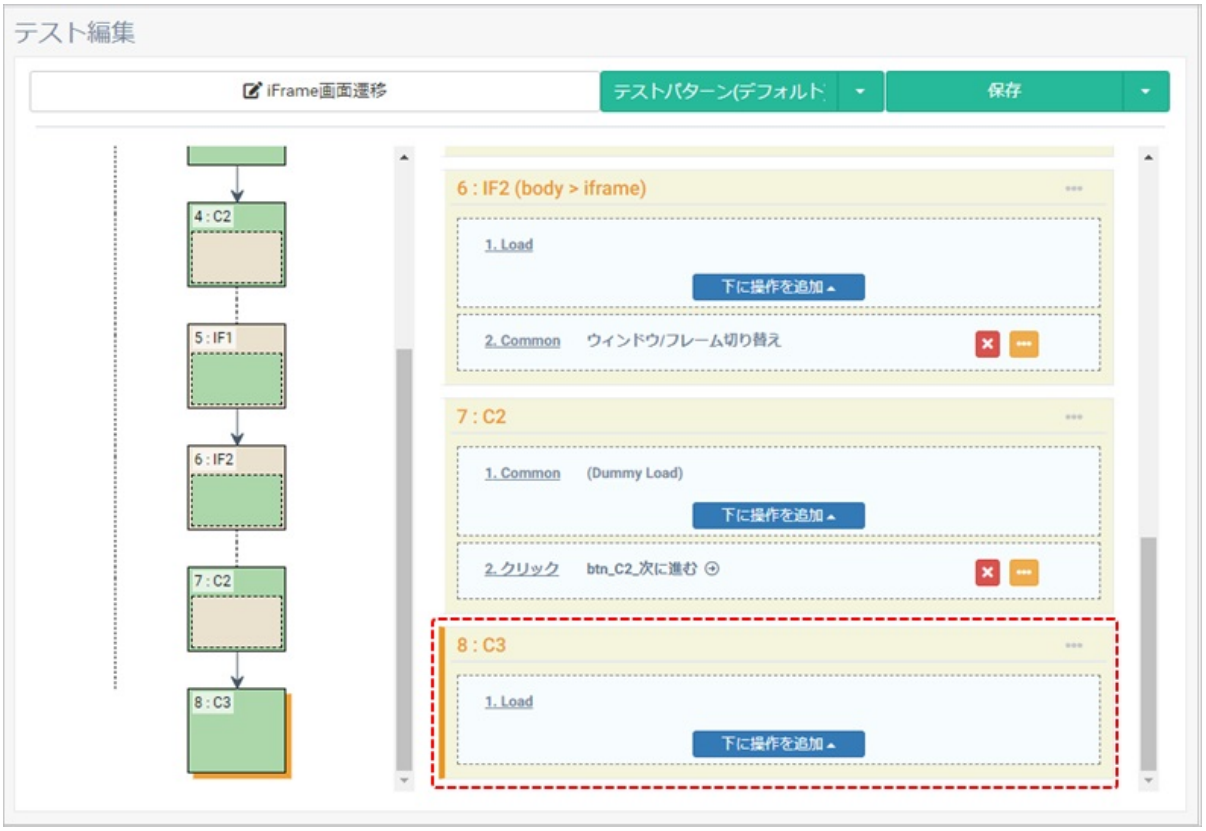
- Step7: C2画面
  前ステップでのウィンドウ/フレーム切替操作の結果、C2画面 がアクティブになっています。



C2画面で **下に操作を追加**ボタン をクリックして開くメニューから**操作追加**を選択します。
表示された**追加項目選択**ダイアログで、「btn_C2_次に進む」 を選択し、**Add**ボタンをクリック

します。

- Step8: C3画面
  C3画面のステップが追加されます。



III. 操作の流れ/機能説明 目次 / 使い方マニュアル 目次 に戻る

5. 変数のアサーションの作成

v1.4.2 より、変数に格納された値を検証(アサーション)することができるようになりました。
これまでは画面上のある要素に対してのみアサーションが設定できましたが、JavaScriptの実行結果など、 どこの要素でもない変数値の検証を行えるようになります。

この章では、非常に簡単なサンプルを使って JavaScriptの実行結果を検証するテスト作成の手順を紹介します。

【概要】
ログイン画面のウィンドウタイトルが正しく設定されているか検証するテストを作成します。
ウィンドウタイトルは 画面内の要素ではないので、JavaScriptを使って取得します。
javaScriptで取得した値は JavaScriptの返却値となり、これを変数に保持して検証します。

1. **JavaScriptの作成**
   - ウィンドウタイトルを返す javaScript を作成します。
   - 今回使用するスクリプトは全ページ共通に作成しています。
     ※ 特定のページに依存しないスクリプトは、全ページ共通に作成しておくことで、どの画面からも利用できます。

   - ページメニューを開き、一覧の右上にある 「**全ページ共通設定**」ボタンをクリックします。

Fig. 5.1-1 ページ一覧画面 全ページ共通設定ボタン

   - **Common** ページの **スクリプト** タブを選択し、スクリプトを作成します。

Fig. 5.1-2 全ページ共通設定 スクリプト一覧

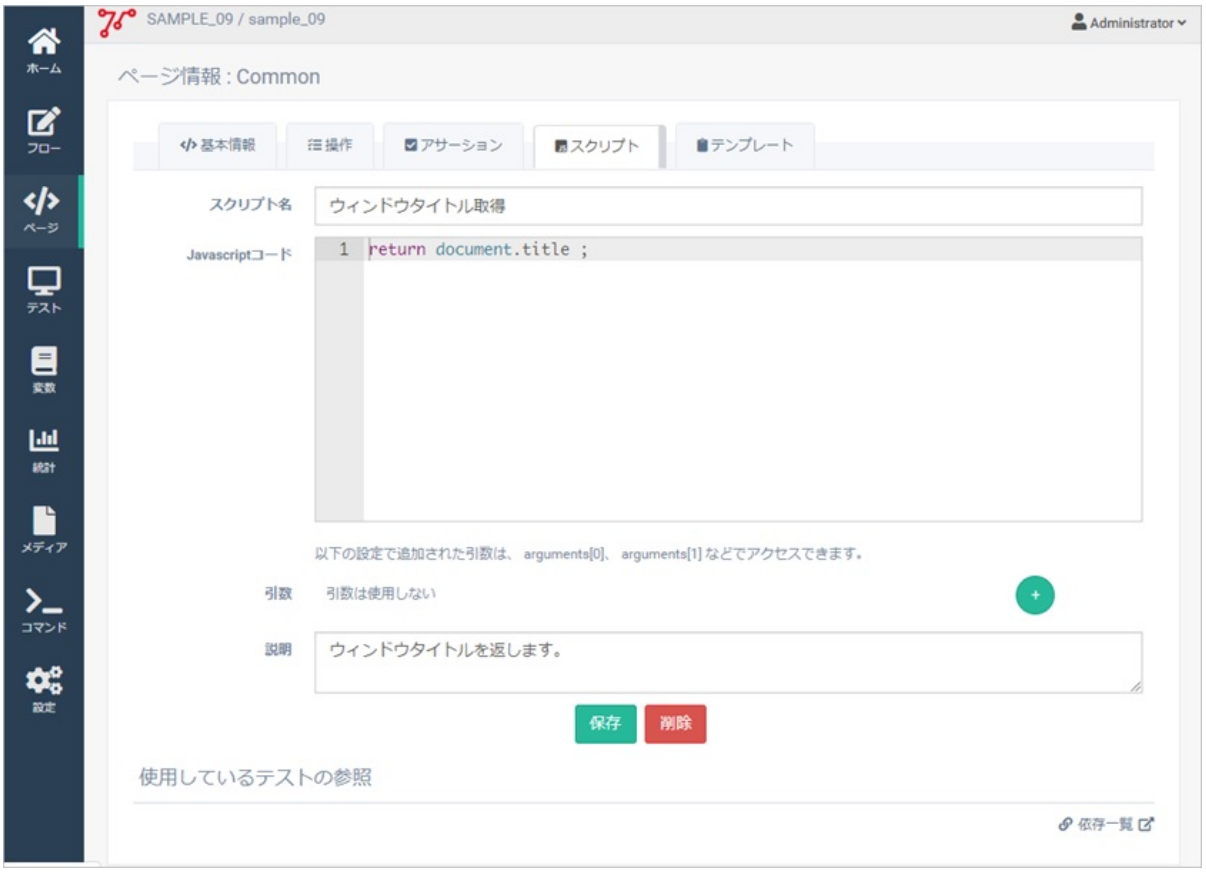


Fig. 5.1-2 全ページ共通設定 スクリプト編集

2. **変数の準備**
   - 変数メニューで、「v_ウィンドウタイトル」という名称で、ウィンドウタイトルを保持する目的の変数を作成しておきます。

Fig. 5.2-1 変数編集

3. **テストの作成**
   - テストを新規作成します。

   - ログイン画面から開始します。
     Load シーケンスの後に、「**下に操作を追加**」ボタンで 「**スクリプト実行追加**」を選択します。

Fig. 5.3-1 スクリプト実行追加

- 表示された スクリプト選択ダイアログから 追加するスクリプトを選び、**Add** ボタンをクリックします。 ダイアログを閉じるには、キャンセルボタンをクリックします。

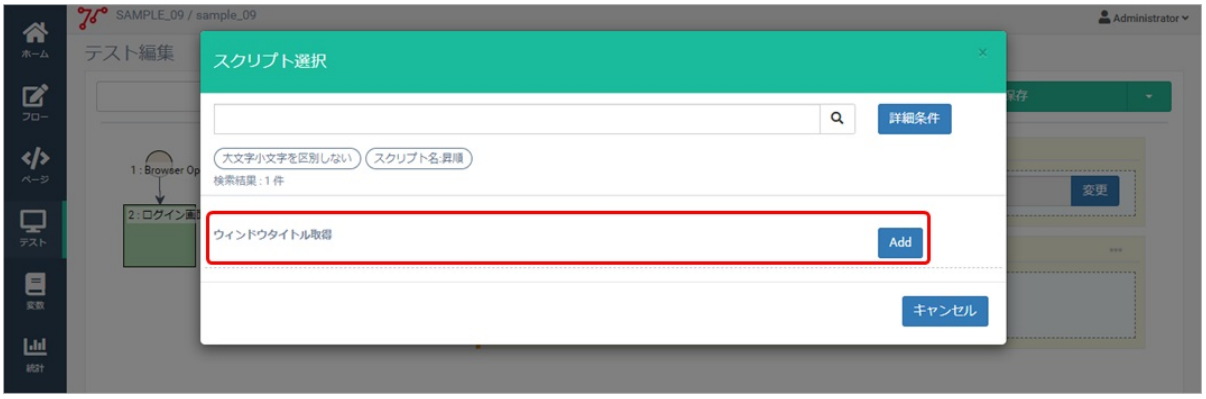


Fig. 5.3-2 スクリプト選択

- 追加されたスクリプトの **「返却値を保持」にチェックを入れ**、スクリプトから返された値を保持する **変数名を指定** します。ここでは先ほど作成しておいた **v_ウィンドウタイトル** を設定します。

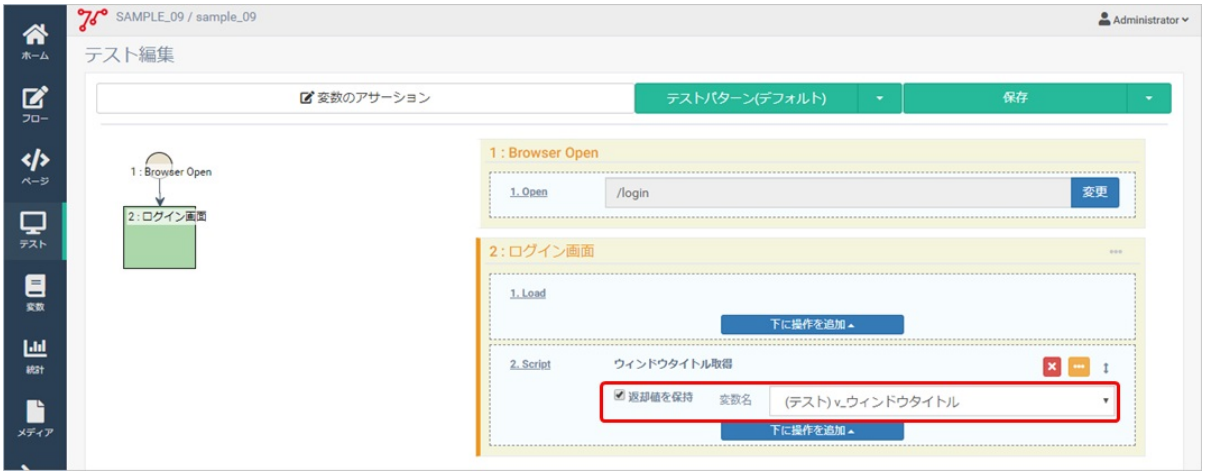


Fig. 5.3-3 変数値を保持

- 次に、変数のアサーションを設定していきます。
  「**下に操作を追加**」ボタンで 「**ブラウザ操作追加**」を選択します。
  ※ 変数のアサーションは、画面操作の「アサーション追加」とは異なります。

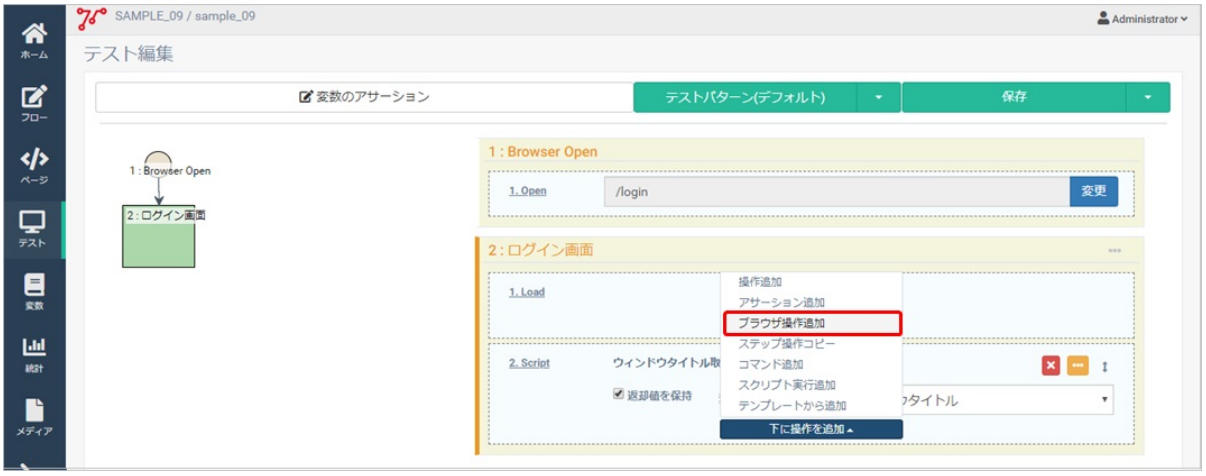


Fig. 5.3-4 ブラウザ操作追加

- 表示された 共通項目選択ダイアログから **変数のアサーション**を選び、**検証タイプを選択**してから **Add** ボタンをクリックします。 ダイアログを閉じるには、キャンセルボタンをクリックします。

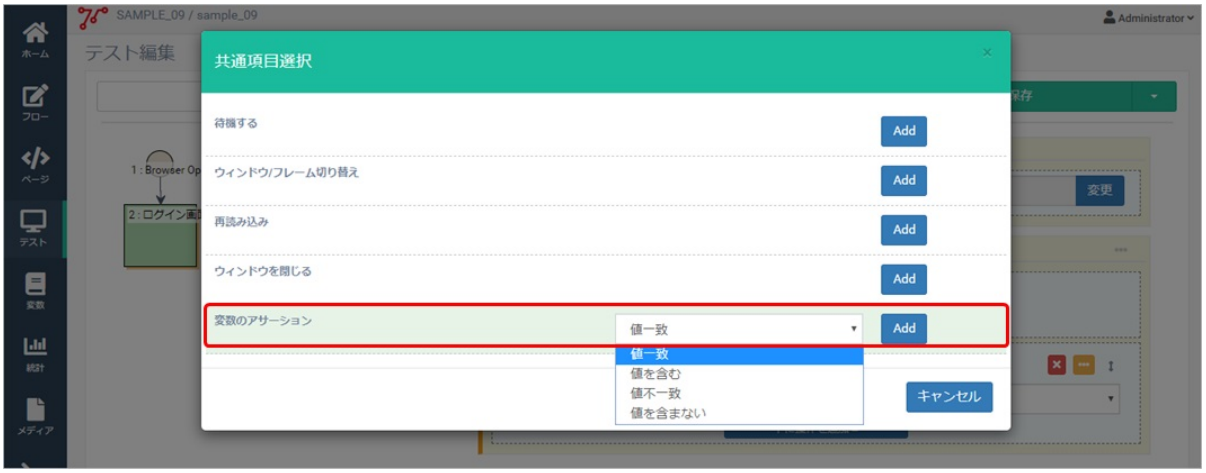


Fig. 5.3-5 共通項目選択

- 追加された変数のアサーションの詳細を設定していきます。
  変数名と検証値を設定していきます。
  ここでは、JavaScriptの返却値を保持している **v_ウィンドウタイトル** に対して、ログイン画面のタイトル文字列が **"Login"** であることを検証します。

Fig. 5.3-6 アサーション設定

- テストを保存した後、 **テストコード**を出力してテストを実行します。

4. **テスト実行結果**
   テストを実行した後のサマリファイルと、結果が反映されたテスト仕様書を確認します。

   変数のアサーションを行っているシーケンスは、 テストサマリでは アクションが「**variable-value-equal**」、 テスト仕様書では 「操作種別：**共通操作**　操作：**変数のアサーション(値一致)**」として表記されています。

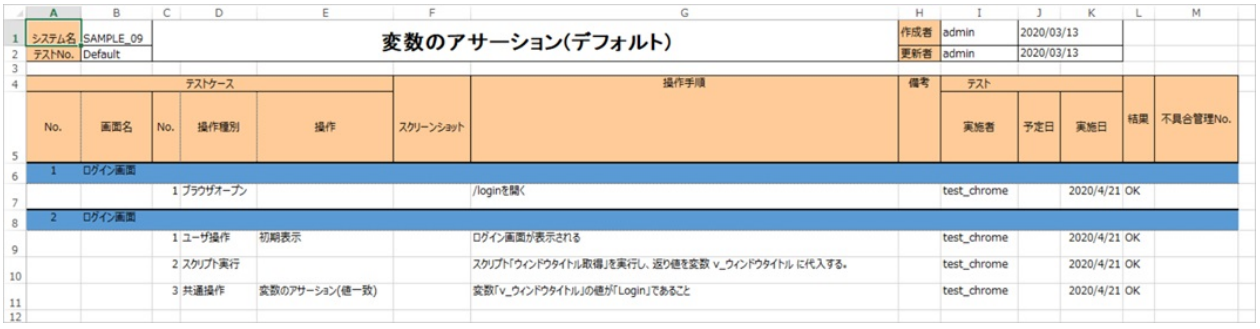
   テストの実行方法や、テスト結果の参照方法については、 III. 操作の流れ/機能説明 » 2. テストの自動実行 を参照してください。

変数のアサーション(デフォルト)テスト結果　成功　テスト開始日時: 2020/04/21 16:58:52　画像サイズ 10 %

summry > chrome-local > 変数のアサーション(デフォルト)

ステップ	画面名	シーケンス	ターゲット	アクション	状態	エラー	設定値	変数値	実行日時	画面
1	ログイン画面	1		open	OK		["/login"]		2020/04/21 16:58:58.121	
2	ログイン画面	1	Page Load	load	OK				2020/04/21 16:58:59.340	
2	ログイン画面	2	ウィンドウタイトル取得	script	OK		["v_ウィンドウタイトル"]		2020/04/21 16:58:59.374	
2	ログイン画面	3		variable-value-equal	OK		["v_ウィンドウタイトル","Login"]		2020/04/21 16:59:00.391	

Fig. 5.4-1 テストサマリ



システム名	SAMPLE_09	変数のアサーション(デフォルト)						作成者	admin	2020/03/13		
テストNo.	Default							更新者	admin	2020/03/13		

テストケース						操作手順	備考	テスト				
No.	画面名	No.	操作種別	操作	スクリーンショット			実施者	予定日	実施日	結果	不具合管理No.
1	ログイン画面											
		1	ブラウザオープン			/loginを開く		test_chrome		2020/4/21	OK	
2	ログイン画面											
		1	ユーザ操作	初期表示		ログイン画面が表示される		test_chrome		2020/4/21	OK	
		2	スクリプト実行			スクリプト「ウィンドウタイトル取得」を実行し、返り値を変数 v_ウィンドウタイトル に代入する。		test_chrome		2020/4/21	OK	
		3	共通操作	変数のアサーション(値一致)		変数「v_ウィンドウタイトル」の値が「Login」であること		test_chrome		2020/4/21	OK	

Fig. 5.4-2 結果が反映されたテスト仕様書

III. 操作の流れ/機能説明 目次 / 使い方マニュアル 目次 に戻る

6. ダイアログのアサーション作成

v1.4.3 より、ダイアログに表示された文字列を検証(アサーション)することができるようになりました。
ある操作の **操作後の流れ** で、 アラート(**alert**) / 確認(**confirm**) / プロンプト(**prompt**) ダイアログ を表示する設定になっている場合、テスト編集画面の該当するステップで直接アサーションを設定することができます。

この章では、テスト編集画面での **ダイアログのアサーションの追加** の手順を紹介します。

【概要】
　映画詳細編集画面で保存ボタンを押した際に表示されるダイアログの文言を検証するアサーションを既存のテストに追加します。
　保存ボタンの「操作後の流れ」には、確認（confirm)とアラート(alert)の2つのダイアログが設定されています。
　それぞれのダイアログに対して表示している文言を検証します。

1. **テストの構成**
   今回利用するテストの流れは以下のようになっています。
   1. 映画検索画面を表示
   2. 映画公開年を入力して「検索ボタン」をクリック
   3. 表示された検索結果一覧画面から映画名のリンクをクリック
   4. 表示された映画詳細画面で「編集するボタン」をクリック
   5. 表示された映画編集画面で「保存するボタン」をクリック
   6. 表示された映画参照画面で「一覧に戻るボタン」をクリック
   7. 検索結果一覧画面を表示

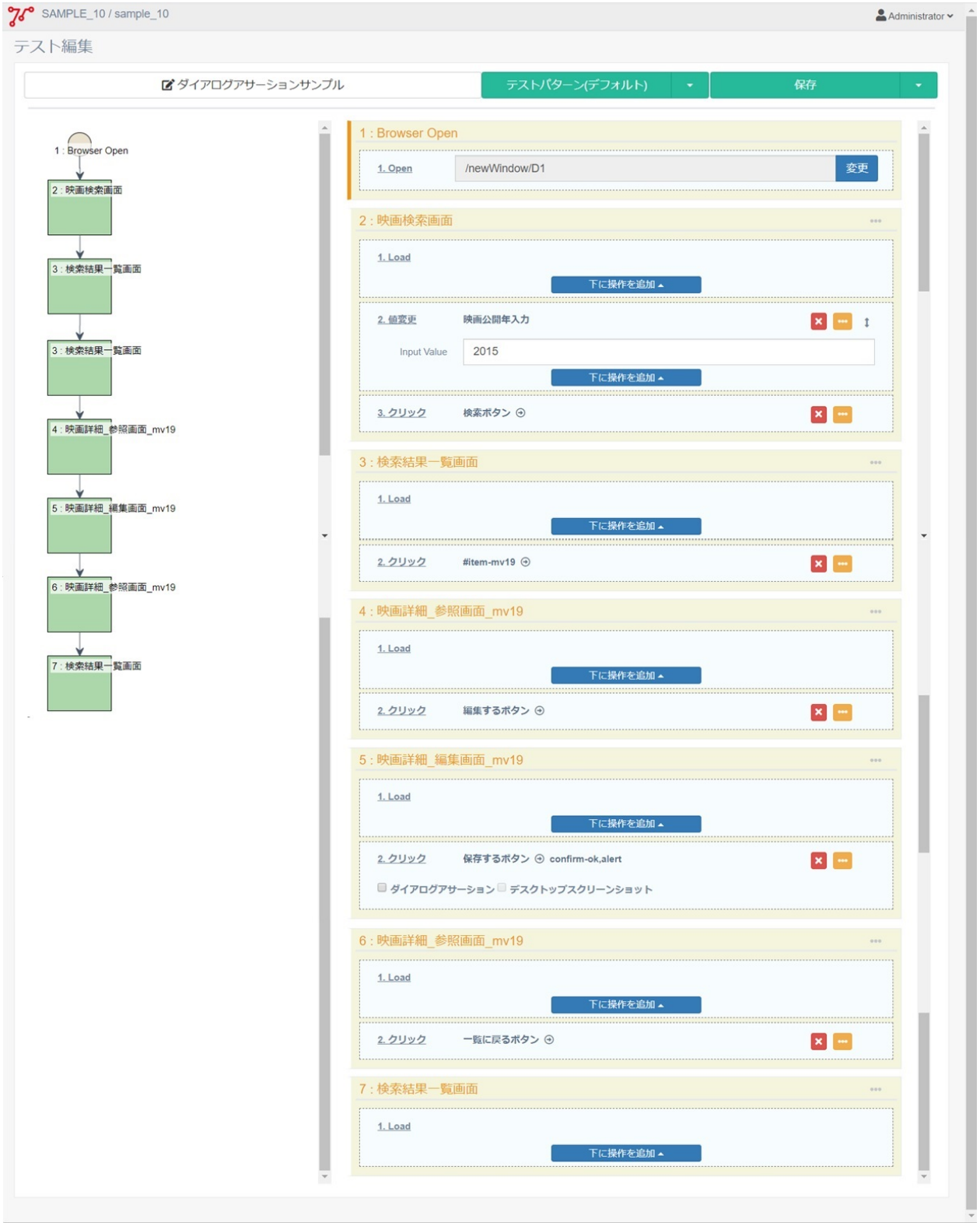


Fig. 6.1 テストの構成

2. **操作後の流れの確認**
    - 映画編集画面の操作「保存する」ボタンの設定を確認していきます。
    - 操作後の流れのセクションにパターンが一つ登録されています。

Fig. 6.2-1 「保存する」ボタンの設定

- 操作後の流れの 「設定」ボタンをクリックして「操作後の処理設定」ダイアログを開きます。

- 「alert/confirm/promptを生成する」にチェックが入っています。
  表示される順にダイアログの詳細を設定します。

- 左端のダイアログ番号はダイアログの表示順を表しています。

- この例では、「更新してよろしいですか？」と表示される確認-OK (confirmダイアログでOKボタンクリック) と 「更新に成功しました。詳細に戻ります。」アラート (alertダイアログ) が設定されています。

- その後、詳細参照画面に遷移するという流れになっています。

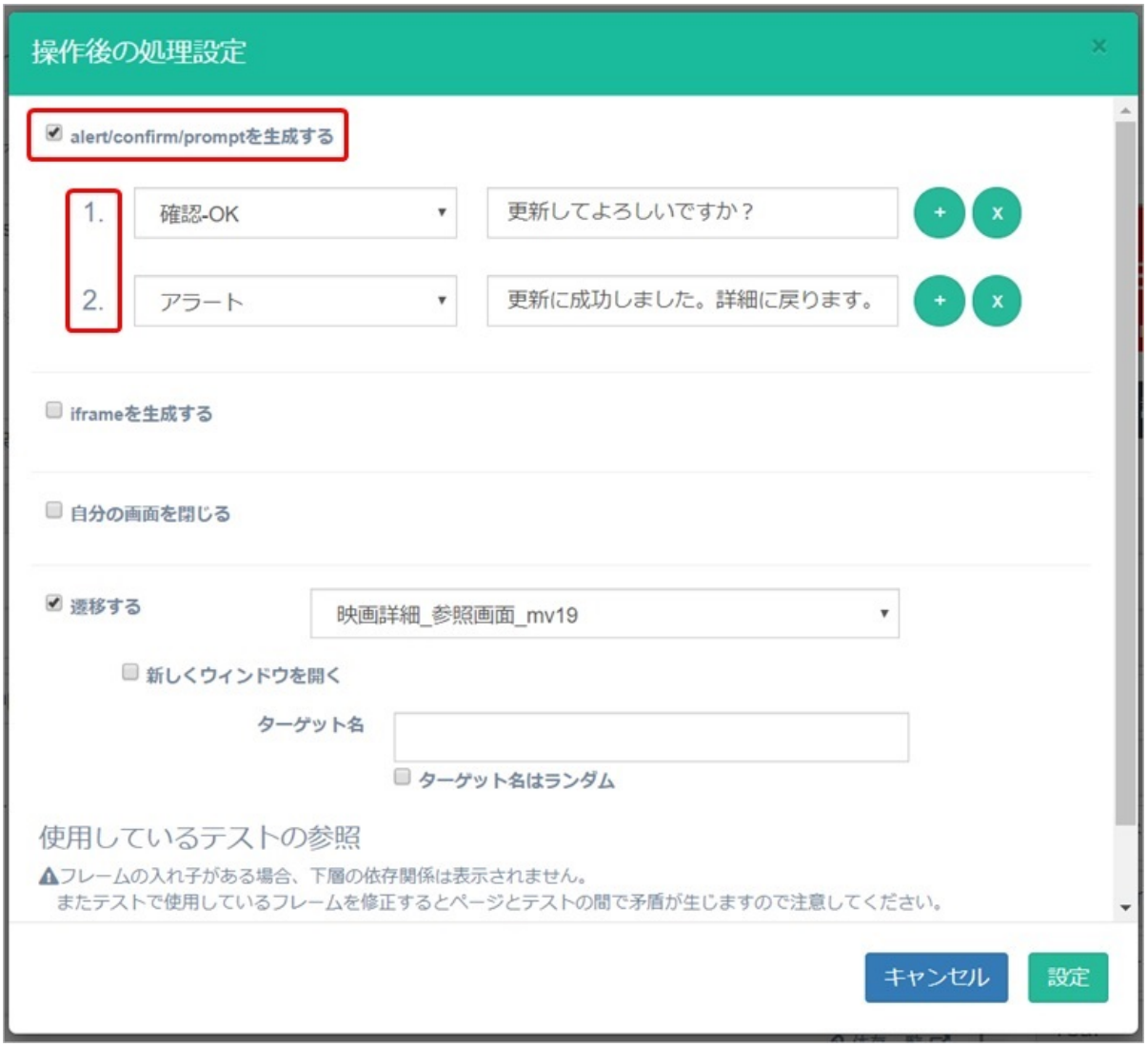


Fig. 6.2-1 「保存する」ボタンの設定

3. **テストへのアサーションの追加**
   - ダイアログアサーションを設定する操作のステップで 「ダイアログアサーション」チェックボックスにチェックを入れます。
     このチェックボックスは当該の操作に**ダイアログ表示のある「操作後の流れ」が設定されている場合**に表示されます。

   - チェックを入れると追加されるアサーション設定エリアで、検証するダイアログをリストから選択し、検証タイプと検証する値を設定します。
     ここでは、1番目の confirm ダイアログに**表示される文言**に対して、**”更新してよろしいですか？”**と一致することを検証しています。

   - アサーションを追加するには、「＋」ボタンをクリックします。

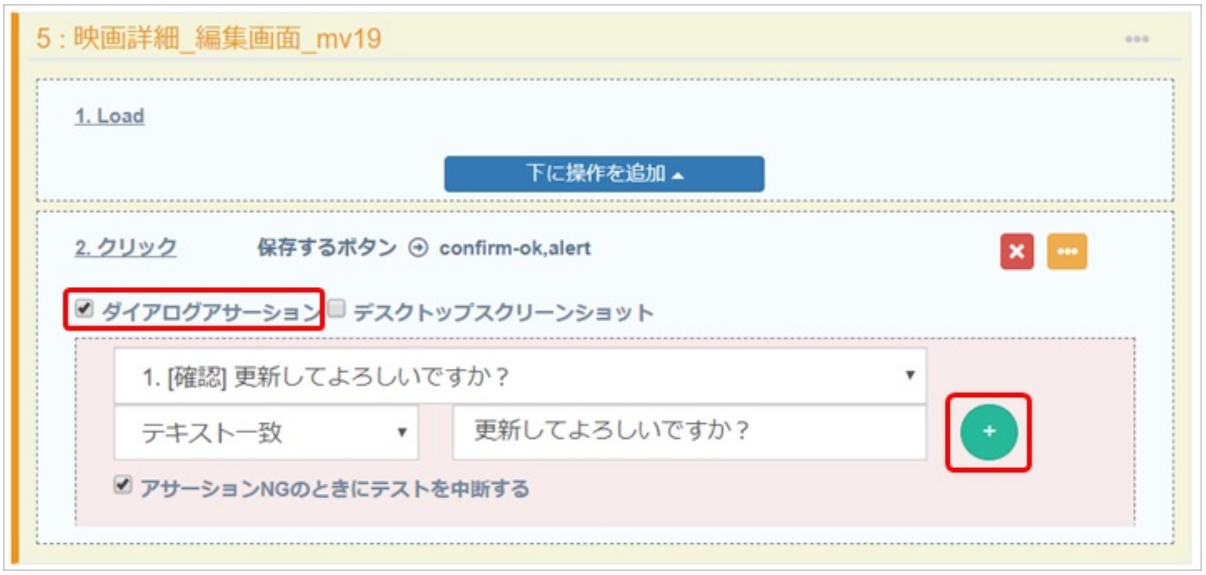


Fig. 6.3-1 ダイアログアサーションの追加*1*

- 一つ目のアサーションと同様に検証するダイアログをリストから選択し、検証タイプと検証する値を設定します。
  ここでは、2番目の alert ダイアログに**表示される文言**に対して、**”成功”** とう文字列が含まれていることを検証しています。

- アサーションはダイアログの表示順に従って作成してください。
  表示順に設定されていない場合、テストの保存時にダイアログ番号順にソートされます。

- 一つのダイアログに複数のアサーションを設定することができます。
- アサーションを削除するには、「×」ボタンをクリックします。

Fig. 6.3-2 ダイアログアサーションの追加*2*

- ダイアログが表示されている状態のスクリーンショットを取得するために、「デスクトップスクリーンショット」チェックボックスにチェックを入れておきます。

- 「デスクトップスクリーンショット」では、テスト実行時のフルスクリーンショットが取得されます。(ダイアログのみのスクリーンショットは取得できません)

- 「デスクトップスクリーンショット」にチェックを入れているので、ステップの操作名の横にカメラアイコンが表示されています。

Fig. 6.3-3 デスクトップスクリーンショット

- テストを保存した後、 テストコードを出力してテストを実行します。

4. **テスト実行結果**
   - テストを実行した後のサマリファイルと、結果が反映されたテスト仕様書を確認します。

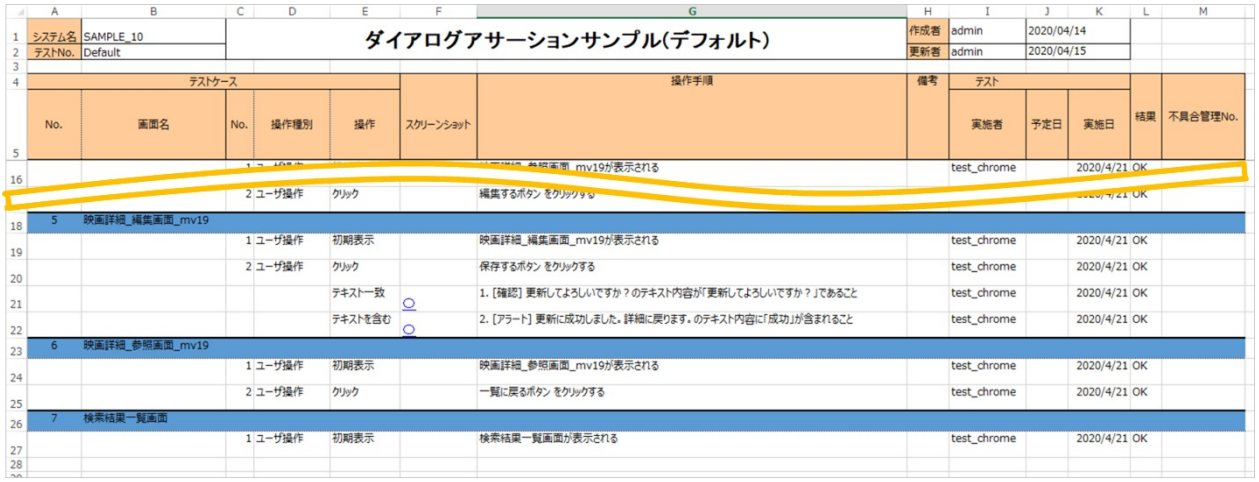
   - テストの実行方法や、テスト結果の参照方法については、 III. 操作の流れ/機能説明 » 2. テストの自動実行 を参照してください。

ダイアログアサーションサンプル(デフォルト) テスト結果　成功　テスト開始日時: 2020/04/21 17:31:51　画像サイズ 10 %

summry > chrome-local > ダイアログアサーションサンプル(デフォルト)

ステップ	画面名	シーケンス	ターゲット	アクション	状態	エラー	設定値	変数値	実行日時	画面
1	映画検索画面	1		open	OK		["/newWindow/D1"]		2020/04/21 17:31:56.665	
2	映画検索画面	1	Page Load	load	OK				2020/04/21 17:31:58.058	
2	映画検索画面	2	映画公開年入力	change	OK		["2015"]		2020/04/21 17:31:58.078	
2	映画検索画面	3	検索ボタン	click	OK				2020/04/21 17:31:59.259	
3	検索結果一覧画面	1	Page Load	load	OK				2020/04/21 17:31:59.366	
3	検索結果一覧画面	2	#item-mv19	click	OK				2020/04/21 17:31:59.388	
4	映画詳細_参照画面_mv19	1	Page Load	load	OK				2020/04/21 17:31:59.526	
4	映画詳細_参照画面_mv19	2	編集するボタン	click	OK				2020/04/21 17:31:59.550	
5	映画詳細_編集画面_mv19	1	Page Load	load	OK				2020/04/21 17:31:59.707	
5	映画詳細_編集画面_mv19	2	保存するボタン	click	OK				2020/04/21 17:31:59.730	
			1. [確認] 更新してよろしいですか？	text-equal	OK		["更新してよろしいですか？"]		2020/04/21 17:32:00.798	
			2. [アラート] 更新に成功しました。詳細に戻ります。	text-include	OK		["成功"]		2020/04/21 17:32:02.180	
6	映画詳細_参照画面_mv19	1	Page Load	load	OK				2020/04/21 17:32:03.404	
6	映画詳細_参照画面_mv19	2	一覧に戻るボタン	click	OK				2020/04/21 17:32:03.424	
7	検索結果一覧画面	1	Page Load	load	OK				2020/04/21 17:32:03.538	

Fig. 6.4-1 テストサマリ



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	システム名	SAMPLE_10	ダイアログアサーションサンプル(デフォルト)					作成者	admin	2020/04/14			
2	テストNo.	Default						更新者	admin	2020/04/15			
3													
4	テストケース						操作手順	備考	テスト				
5	No.	画面名	No.	操作種別	操作	スクリーンショット			実施者	予定日	実施日	結果	不具合管理No.
16			2	ユーザ操作	クリック		編集するボタン をクリックする		test_chrome		2020/4/21	OK	
18	5	映画詳細_編集画面_mv19											
19			1	ユーザ操作	初期表示		映画詳細_編集画面_mv19が表示される		test_chrome		2020/4/21	OK	
20			2	ユーザ操作	クリック		保存するボタン をクリックする		test_chrome		2020/4/21	OK	
21					テキスト一致	○	1. [確認] 更新してよろしいですか？のテキスト内容が「更新してよろしいですか？」であること		test_chrome		2020/4/21	OK	
22					テキストを含む	○	2. [アラート] 更新に成功しました。詳細に戻ります。のテキスト内容に「成功」が含まれること		test_chrome		2020/4/21	OK	
23	6	映画詳細_参照画面_mv19											
24			1	ユーザ操作	初期表示		映画詳細_参照画面_mv19が表示される		test_chrome		2020/4/21	OK	
25			2	ユーザ操作	クリック		一覧に戻るボタン をクリックする		test_chrome		2020/4/21	OK	
26	7	検索結果一覧画面											
27			1	ユーザ操作	初期表示		検索結果一覧画面が表示される		test_chrome		2020/4/21	OK	

Fig. 6.4-2 結果が反映されたテスト仕様書

テスト編集画面でダイアログアサーションの設定とともに「デスクトップスクリーンショット」にチェックを入れておいたため、サマリファイルでは右端の「画面」列、結果が反映されたテスト仕様書では「スクリーンショット」列のリンク先からフルスクリーンの画面画像を確認することができます。

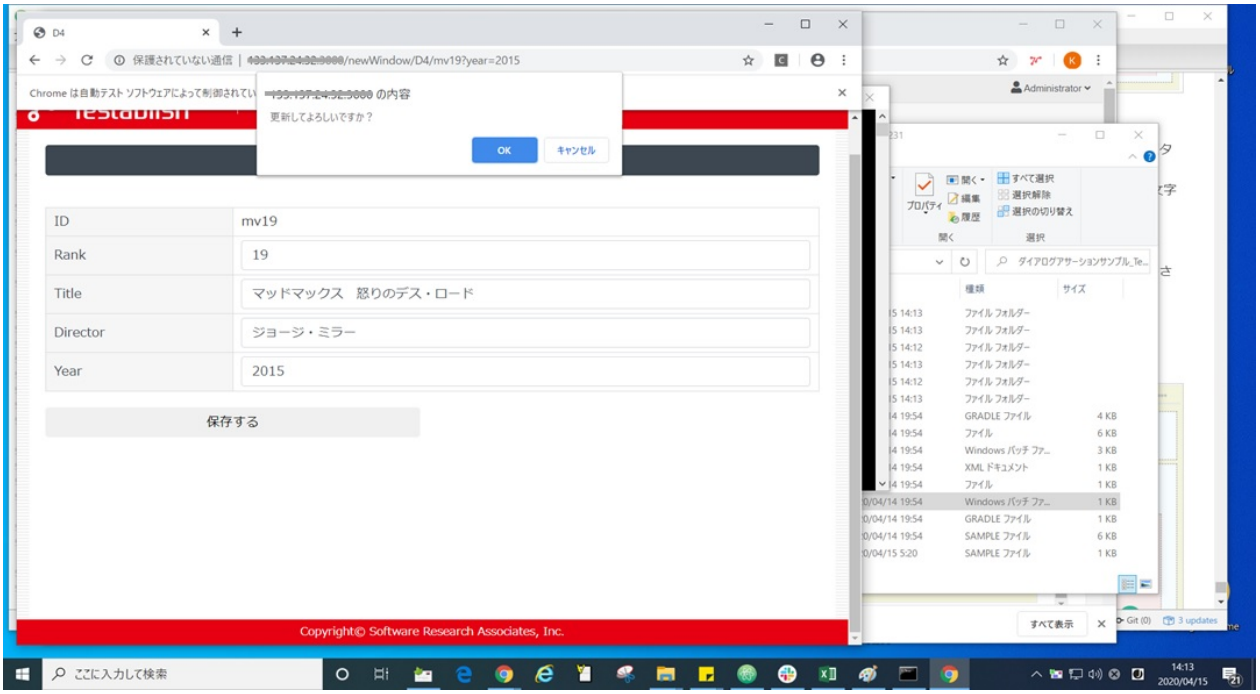


Fig. 6.4-3 デスクトップスクリーンショット

III. 操作の流れ/機能説明 目次 / 使い方マニュアル 目次 に戻る

7. UI レイアウト検証の支援機能

画面崩れの確認は**スクリーンショットの目視確認にて UI レイアウトの欠落・位置ずれ・色違い・サイズ違いを判断**する必要がありましたが、これを支援する機能です。
本支援機能により、UI レイアウト崩れ部分を**機械的に図示**できるようになります。そのため**レイアウト崩れを漏れなく検出する事が可能**になります。

この章では、**UI レイアウト検証の支援機能**の使い方を紹介します。

【概要】
　目的の UI レイアウトが正しく表示されているかを検証します。
　テストケースのスクリーンショットを利用して「正しい UI レイアウト画像」と照合することにより検証します。
　検証結果はテスト結果のhtmlで参照することができます。
　以下の図のような結果が参照できます。

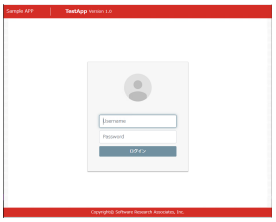


Fig. 7-1 正しい *UI* レイアウト画像

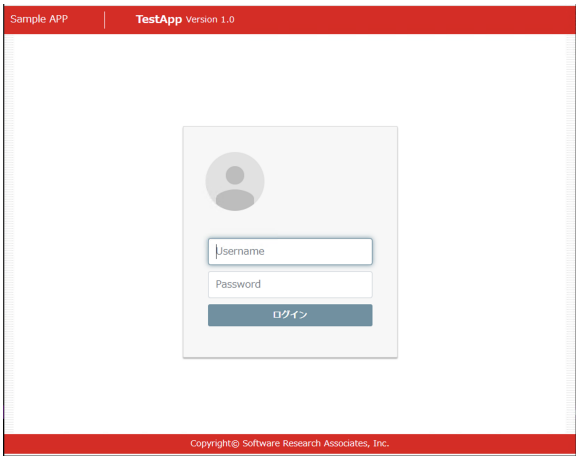


Fig. 7-2 実際の *UI* レイアウト画像
（ユーザーアイコンが左寄せになっています）

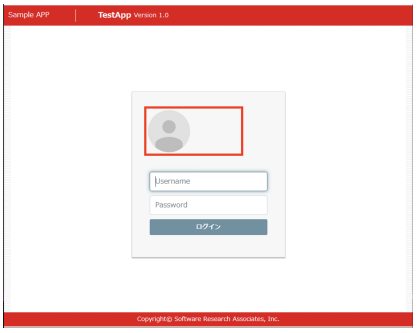


Fig. 7-3
レイアウト崩れ部分が図示された *UI* レイアウト画像

1. **テストの構成例**
   今回利用するテストの流れは以下のようになっています。
   1. テスト作成

Fig. 7.1 テストの構成

2. **スクリーンショットの設定**
   - スクリーンショットが有効になっているステップが本機能の対象となります。
     不都合がない限り「ブラウザ」を選択して下さい（「デスクトップ」でも動きますが、ブラウザ外の表示部分がレイアウト崩れと誤認される可能性があります）

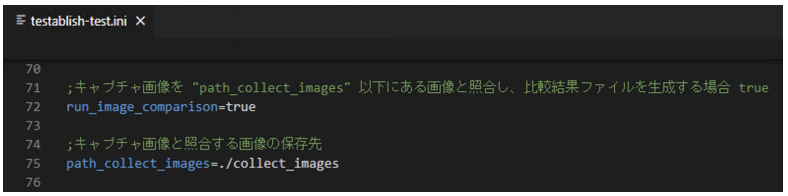
Fig. 7.2 スクリーンショットにブラウザを設定

   - テストを保存した後、 テストコードを出力します。

3. **testablish_test.ini ファイルの設定**
   - 本機能に関するいくつかの設定があります（初期値のままで動作に支障はありません）

key	初期値	説明
`[testablish]`		
`run_image_comparison`	`true`	スクリーンショット画像を `"path_collect_images"` 以下にある画像と照合するか `true/false` で設定します。 `false` に設定すると、 テスト実行時間を短縮できます （機能テスト優先時に有効です）

path_collect_images	collect_images	「正しい UI レイアウト画像」 フォルダのパスを設定します。 ネットワークフォルダを指定すると、 複数テスト環境で「正しい UI レイアウト画像」 を共有できます。



- テストを実行します。

4. **「正しい UI レイアウト画像」の配置**
   - UI レイアウトのテストに先立ち「正しい UI レイアウト画像」のセットを作成・配置する必要があります。

   - 具体的には、以下の手順を繰り返します。
     1. テスト実行＆完了する
        取得した全てのスクリーンショットが、「正しい UI レイアウト画像」の候補として testablish_test.ini ファイル内の [path_collect_images] で定義されたフォルダ以下に保存される
     2. 全てのスクリーンショットを目視確認する
     3. レイアウト崩れを発見した場合、そのスクリーンショット画像を削除し、**該当 UI を修正する**
     4. 全てのスクリーンショットが「正しい UI レイアウト画像」として問題なくなるまで、上記を繰り返す

   - 以上により、testablish_test.ini ファイル内の [path_collect_images] で定義されたフォルダ以下が、「正しい UI レイアウト画像」のセットとなります。
     UI 仕様の変更に伴い、上記の手順を繰り返す必要がある点にご注意ください。

   - 補足１：テスト実行時に「正しい UI レイアウト画像」が存在しなくても、テストが止まる事はありません
     （取得直後のスクリーンショット画像を「正しい UI レイアウト画像」として扱うため）

   - 補足２：余計な「正しい UI レイアウト画像」が存在していても、テストに支障はありません（無視されます）

5. **テスト実行結果**
   - テストを実行した後のサマリファイルと、結果が反映されたテスト仕様書を確認します。

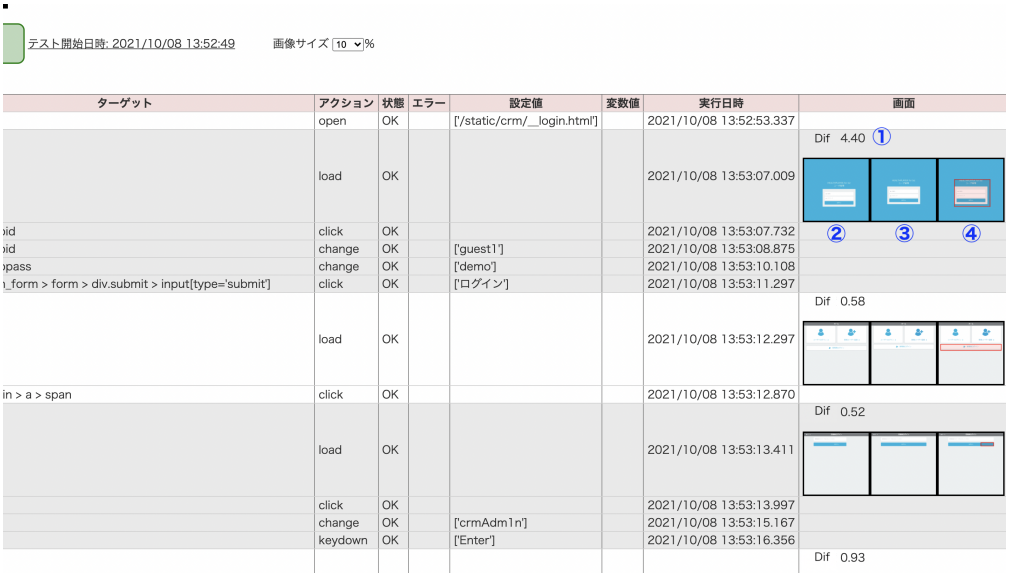


Fig. 7.5 テストサマリ

No.	名称	説明
1	Dif	「正しい UI レイアウト画像」と「スクリーンショット」との違いを数値化したものです。 完全一致する時は 0 になります。
2	正しい UI レイアウト画像	予め作成・配置した画像です。 マウス hover 操作により、Preview を画面左側に表示します。
3	スクリーンショット	今回のテスト実行時にキャプチャされた画像です。 マウス hover 操作により、Preview を画面左側に表示します。
4	レイアウト崩れ部分が図示された UI レイアウト画像	UI レイアウトの欠落・位置ずれ・色違い・サイズ違いを赤枠で示した画像です。 マウス hover 操作により、Preview を画面左側に表示します。

III. 操作の流れ/機能説明 目次 / 使い方マニュアル 目次 に戻る

8. 繰り返し実行機能

v1.4.6 より、Excelファイルで定義したデータをもとに「ループの開始」から「ループの終了」までを繰り返し実行できるようになりました。

基本的な使い方は以下のようになります。

1. **テストの構成**

   ここではログインユーザを変更して複数のユーザパターンでテストする例説明します。
   テストの流れは以下の繰り返しになります。

   1. ユーザ (ID) を入力
   2. パスワード (PW) を入力
   3. ログインをクリック
   4. ログアウトをクリック

2. **ひな形の作成**

   まずテスト編集画面でテストの雛形を作ります。
   上記の操作をテスト編集画面で作成します。

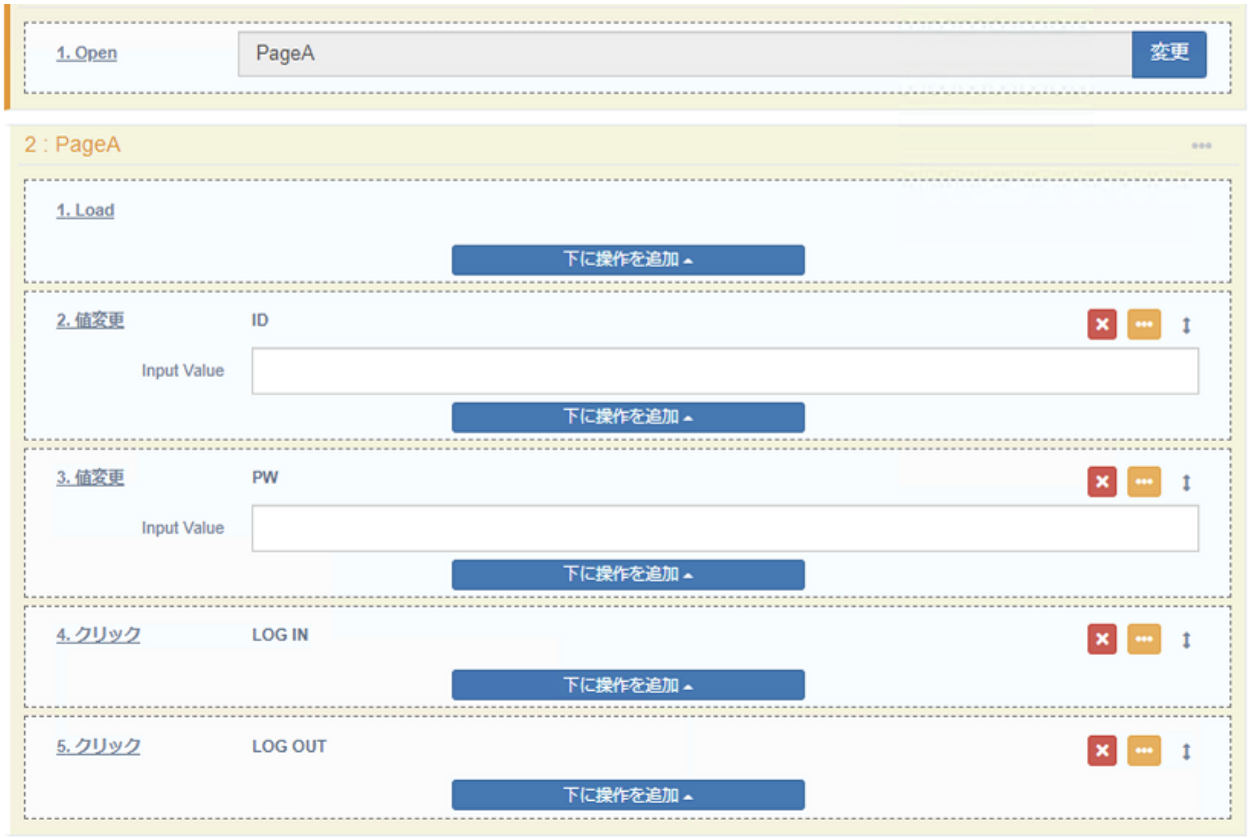


Fig. 8.1 テストの雛形

3. **繰り返し範囲の設定**

   続いて繰り返し範囲を設定します。

ID 入力の前に「ループ開始」、ログアウト後に「ループ終了」 を置きます。

ループ開始 と **ループ終了** は、**下に操作を追加** から **ブラウザ操作追加** をクリックで追加できます。

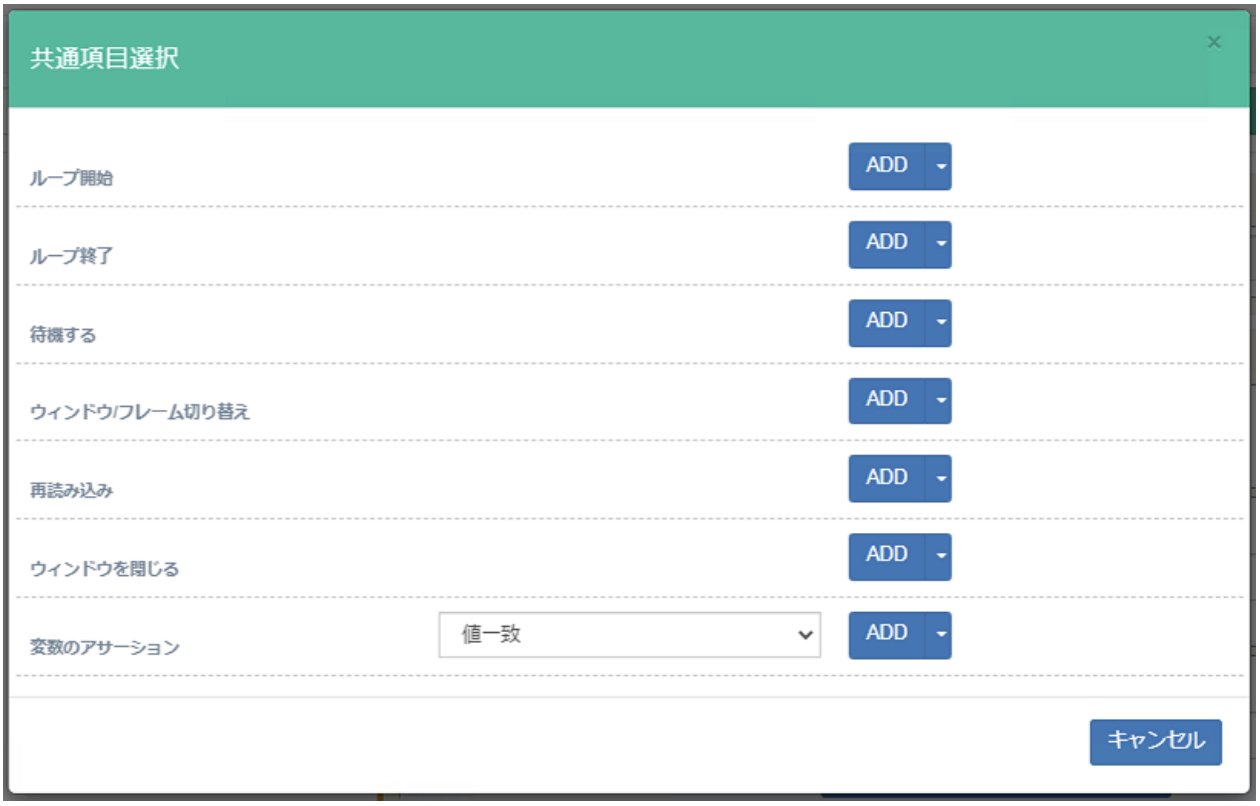


Fig. 8.2 共通項目選択ダイアログ

ループ開始、終了を配置すると繰り返し範囲が黄色のハイライトで表示されます。

Fig. 8.3 ループ開始/終了

以上が済んだら、テストコードを出力します。

Fig. 8.4 テストコードの出力

4. **繰り返しデータの整備**

テストコードのzipファイルの中に**繰り返しテストデータの雛形 ddt.xlsx**が出力されます。
ddt.xlsx ファイルの４行目以降に、繰り返し実行用の値を設定します。
ddt.xlsx ファイルには、ループ開始～ループ終了までに含まれる、全ての操作が横並びに存在します。
（１行目は操作の対象名、２行目は入力タイプ、３行目は操作タイプを示します）

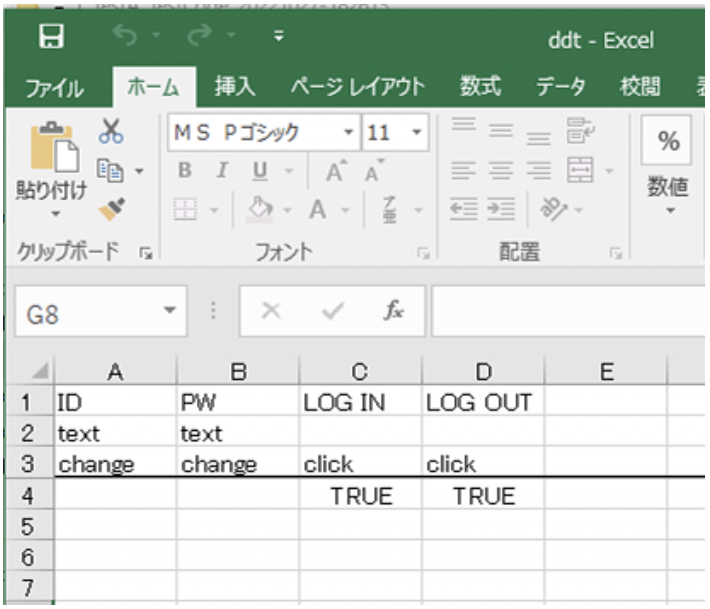
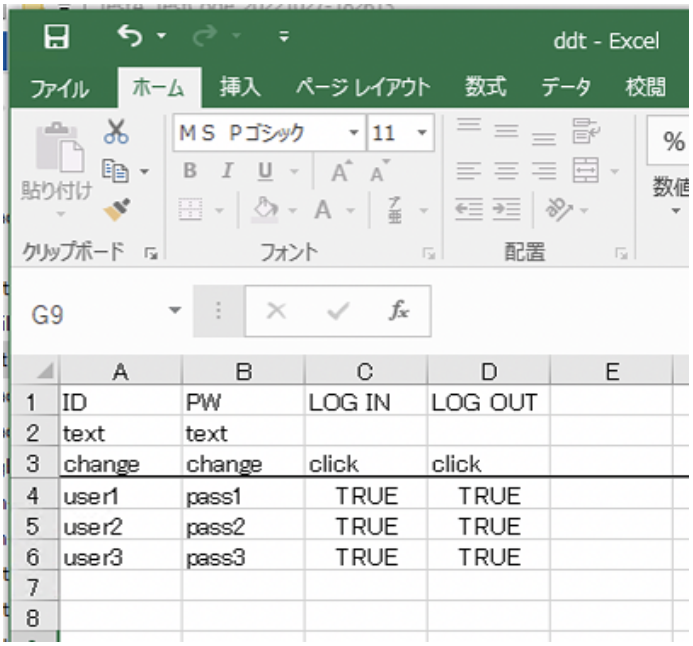


Fig. 8.5 ddt.xlsx ファイル

例えば、以下のデータで繰り返し実行したい場合は、

1. ID: user1, PW: pass1
2. ID: user2, PW: pass2
3. ID: user3, PW: pass3

以下のように記入します。



	A	B	C	D	E
1	ID	PW	LOG IN	LOG OUT	
2	text	text			
3	change	change	click	click	
4	user1	pass1	TRUE	TRUE	
5	user2	pass2	TRUE	TRUE	
6	user3	pass3	TRUE	TRUE	
7					
8					

Fig. 8.6 ddt.xlsx ファイル (2)

5. **テストの実行**

   テストを実行します。ループ開始～ループ終了までの範囲が、ddt.xlsx のデータ数だけ繰り返し実行されます。

繰り返し実行機能を使う場合、いくつかの注意点があります。

- **Radio Button の選択**

  繰り返し実行中、Radio Button の項目選択をデータパターンに応じて変えたい場合
  例えば朝・昼・夜の Radio Button を ddt.xlsx に応じて選択させたい場合、それぞれの「クリック」操作を追加します。

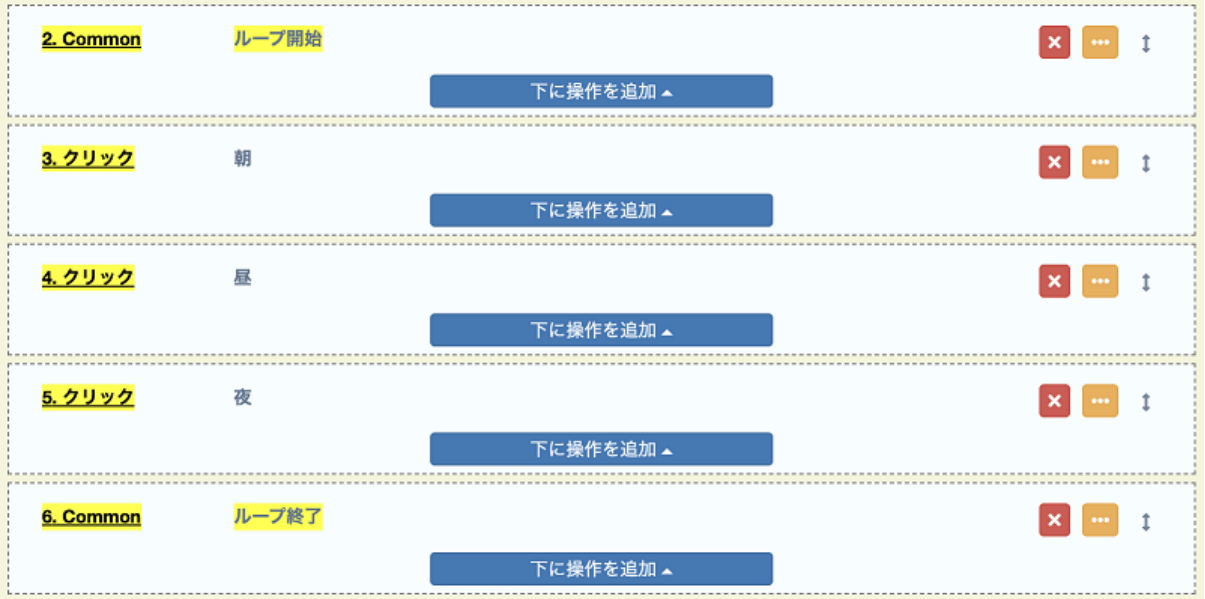


Fig. 8.7 テストの編集

この時、出力される ddt.xlsx ファイルは、以下のようになります。

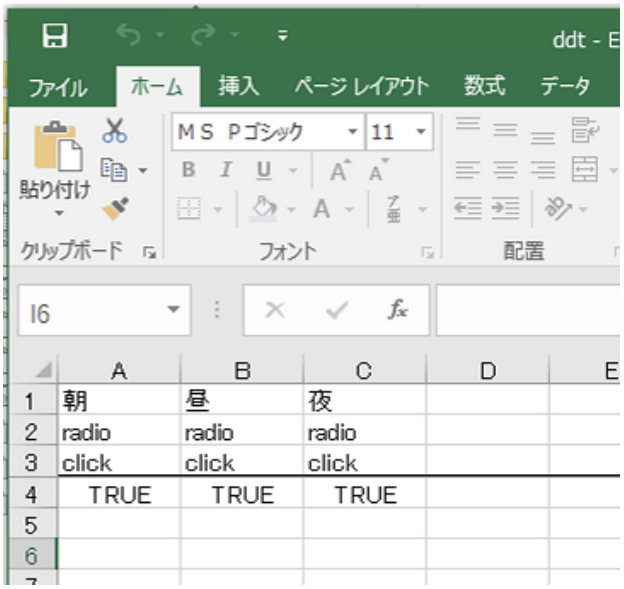


Fig. 8.8 ddt.xlsx (変更前)

このファイルを、以下のように選択させたい箇所を `"TRUE"` とし、それ以外を空にします。

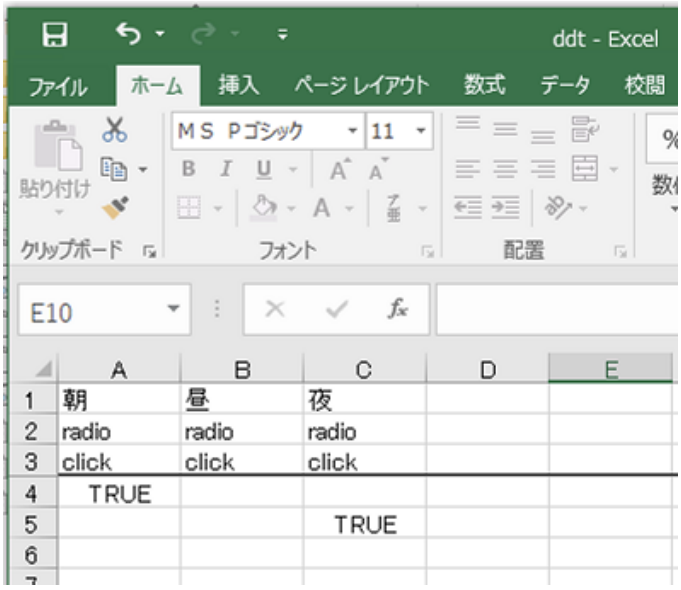


Fig. 8.9 ddt.xlsx (変更後)

- **繰り返し実行中、`CheckBox` の選択を `ddt.xlsx` に応じて変えたい場合**

  例えば平日・休日・祝日の `Checkbox` を `ddt.xlsx` に応じて選択させたい場合、それぞれの「値変更」操作を追加します。

Fig. 8.10 テストの編集

この時、出力される `ddt.xlsx` ファイルは、以下のようになります。

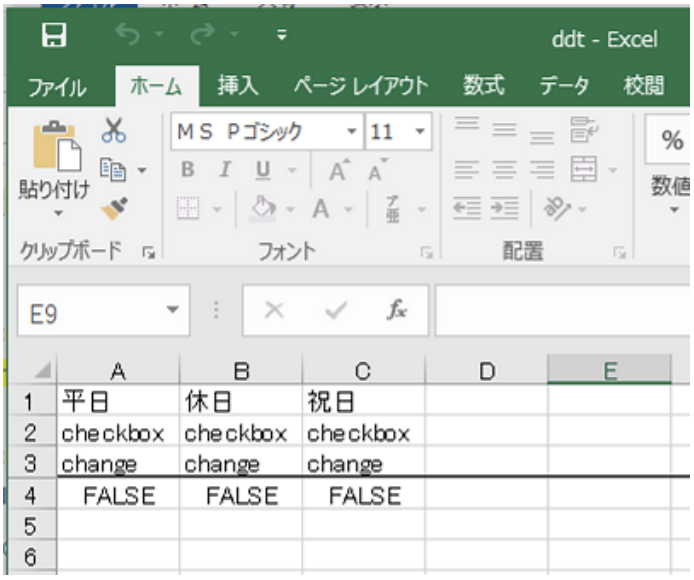


Fig. 8.11 ddt.xlsx (変更前)

このファイルを、以下のように選択させたい箇所を `"TRUE"` とし、それ以外を `"FALSE"` にします。

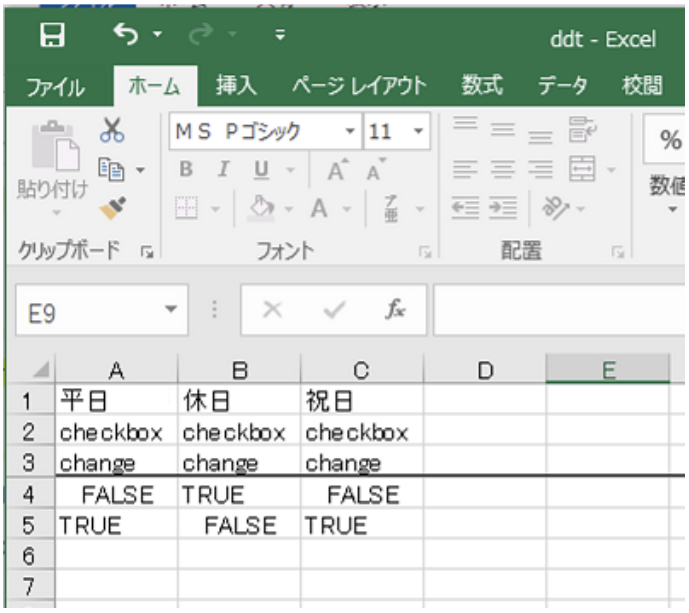


Fig. 8.12 ddt.xlsx (変更後)

- **繰り返し実行中、"select-one" の選択を ddt.xlsx に応じて変えたい場合**

  例えばメニュー１・メニュー２・メニュー３の選択肢を持つ "select-one" を ddt.xlsx に応じて選択させたい場合、「値変更」操作を追加します。

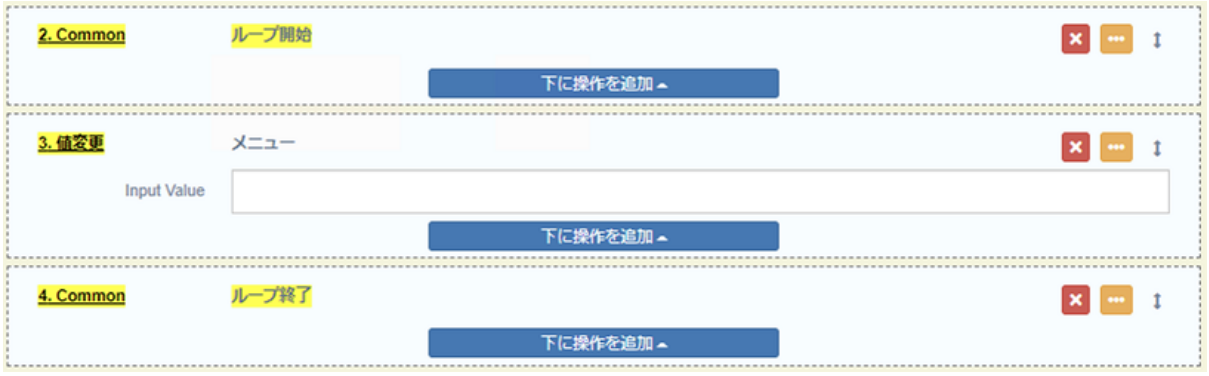


Fig. 8.13 テストの編集

この時、出力される ddt.xlsx ファイルは、以下のようになります。

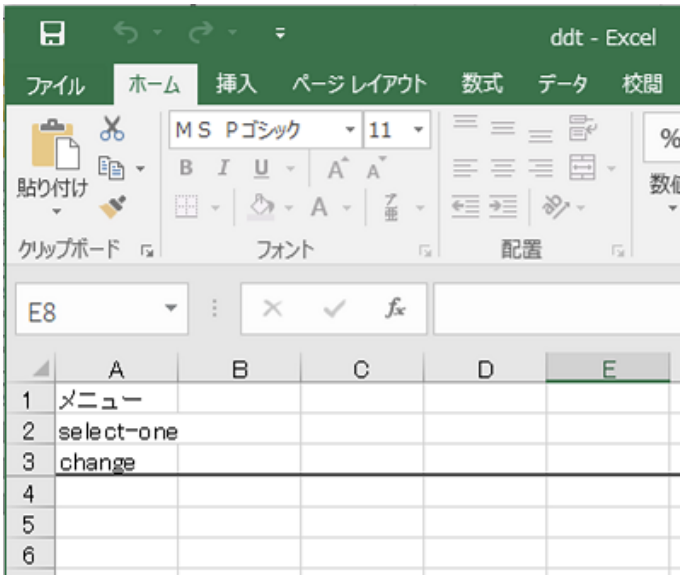


Fig. 8.14 ddt.xlsx (変更前)

このファイルに、選択させたい値を入力します。

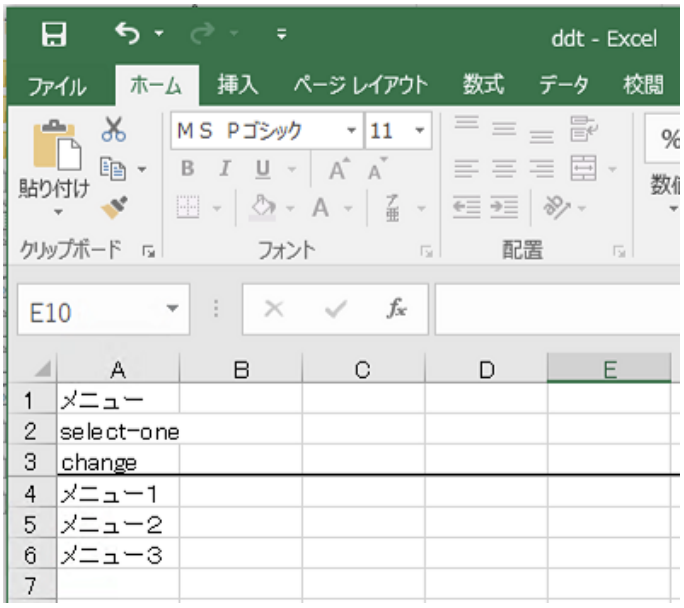


Fig. 8.15 ddt.xlsx (変更後)

- **繰り返し実行中、"select-multiple" の選択を ddt.xlsx に応じて変えたい場合**

  例えばメニュー１・メニュー２・メニュー３の選択肢を持つ "select-multiple" を ddt.xlsx に応じて選択させたい場合、「値変更」操作を追加します。

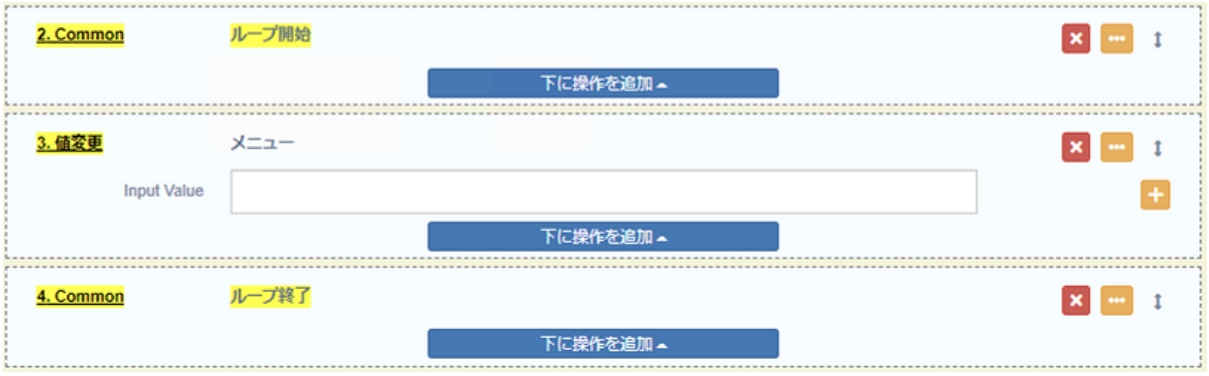


Fig. 8.16 テストの編集

この時、出力される ddt.xlsx ファイルは、以下のようになります。

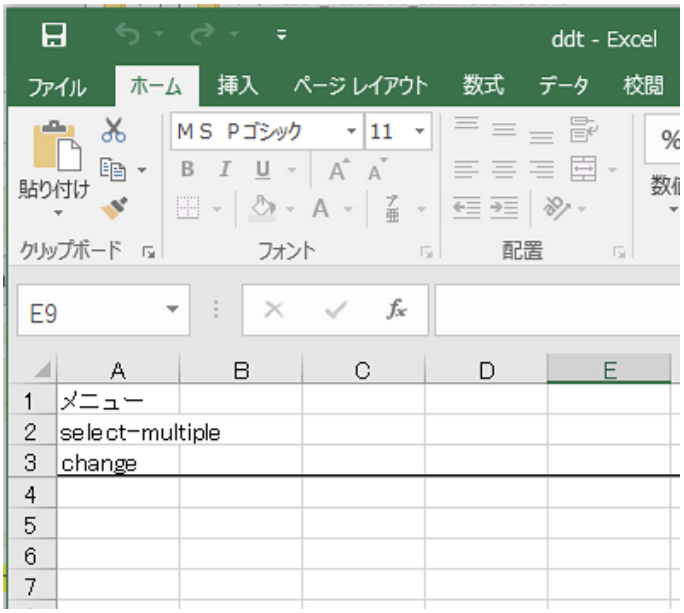


Fig. 8.17 ddt.xlsx (変更前)

このファイルに、選択させたい値を入力します（カンマ区切りで複数指定が可能です）

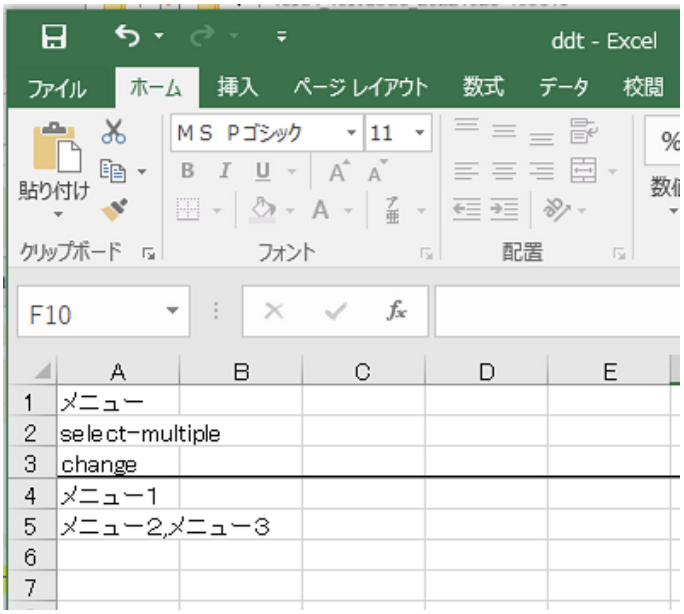


Fig. 8.18 ddt.xlsx (変更後)

- **繰り返し実行中、"file" の選択を ddt.xlsx に応じて変えたい場合**

  file に関しては対応しておりません。

III. 操作の流れ/機能説明 目次 / 使い方マニュアル 目次 に戻る

9. 操作後の流れを登録して追加

v1.4.6 より、ある操作の **操作後の流れ** を、テスト編集画面の **下に操作を追加** 時に、直接登録することができるようになりました。

この章では、テスト編集画面での **操作後の流れを登録して追加** の手順を紹介します。

1. **操作後の流れの確認**
    1. 「下に操作を追加」ボタンを押し、「操作追加」を選択します。

Fig. 9.1 「下に操作を追加」ボタン

    2. 「追加項目選択ダイアログ」が表示されます。「ADD」ボタン右側に「▼」ボタンが表示されます。

Fig. 9.2 追加項目選択ダイアログ

    3. 「ADD」ボタン右側の「▼」ボタンを押した後、「操作後の流れを登録して追加」ボタンを選択します。

Fig. 9.3 操作後の流れを登録して追加ボタン

4. 「操作後の処理設定」ダイアログが表示されます。設定完了後、「OK」ボタンを押します。

Fig. 9.4 操作後の処理設定ダイアログ

5. 「操作後の処理設定」ダイアログが閉じられ、「追加項目選択」ダイアログに戻ります。
6. この時、操作自体に「操作後の流れ」が登録され、かつ、この『「操作後の流れ」が適用された操作』がテストケースに追加されています。

Fig. 9.5 設定後

III. 操作の流れ/機能説明 目次 / 使い方マニュアル 目次 に戻る

10. ページ検索ダイアログ

v1.4.6 より、「操作後の処理設定」画面の遷移先ページを「ページ検索ダイアログ」にて検索・特定することができるようになりました。

これにより、どれが目的のページか分かりづらい、所属フォルダが異なる同名ページを区別できない、などの問題なく、目的のページを設定できます。

1. **操作例**
    - 「操作後の処理設定」ダイアログの「遷移する」欄に追加された、検索ボタンを押します。

Fig. 10.1 操作後の処理設定ダイアログ

    - このダイアログでは、最初にルートフォルダ直下のページのみ表示されます。
      フォルダ・アイコンの項目をクリックすると、そのフォルダに移動します。
      また、ページ名の検索機能を使えば、目的のページを絞り込む事もできます。

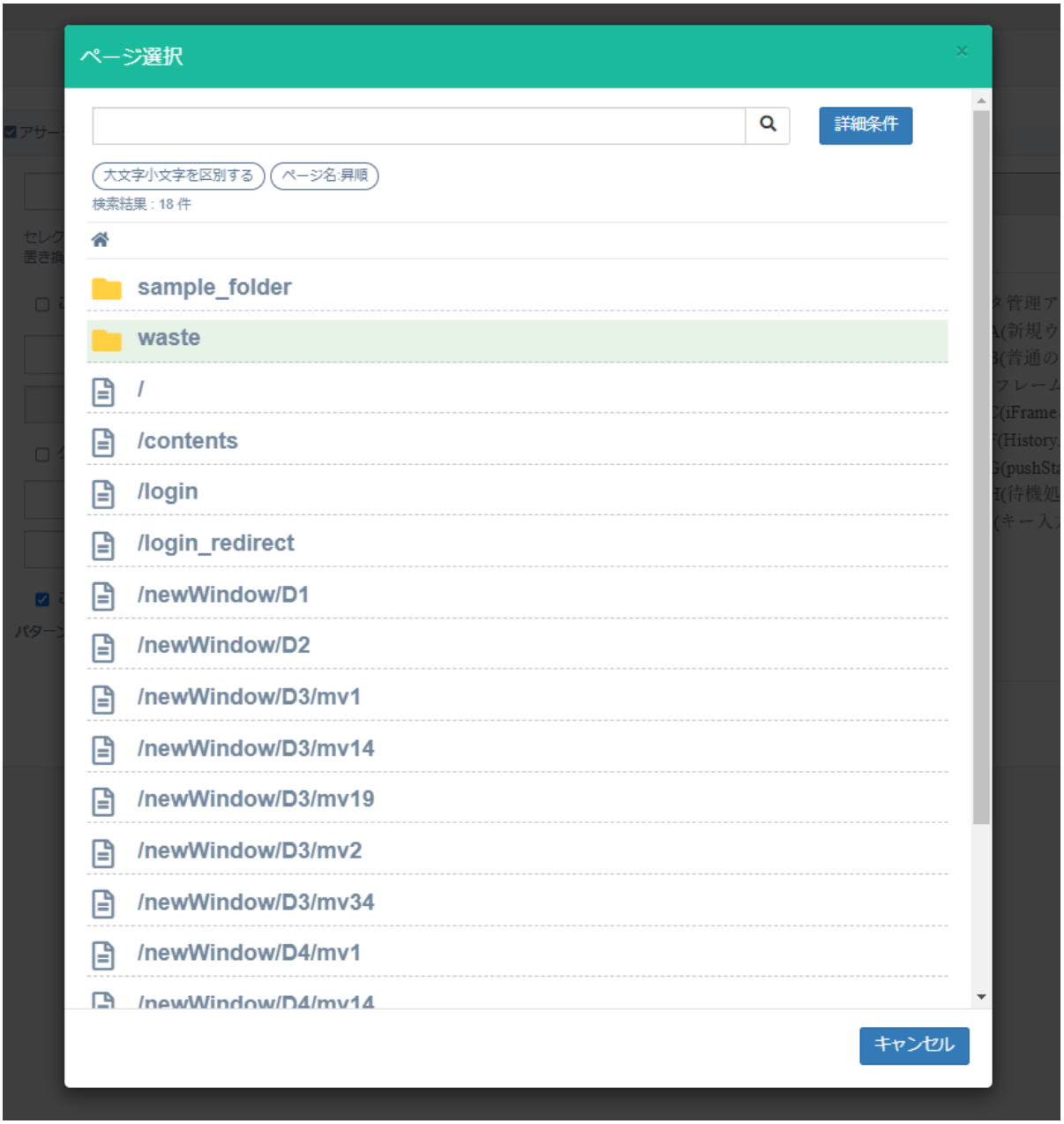


Fig. 10.2 ページ選択ダイアログ

III. 操作の流れ/機能説明 目次 / 使い方マニュアル 目次 に戻る

11. テスト・テンプレート機能

v1.4.6 より、「新しいテンプレート機能」（以下、テスト・テンプレート機能）が追加されました。

従来のテンプレート機能では、以下の問題がありましたが、これらを自動的に解決できるテンプレート機能です。

- テンプレート上の操作パターンを修正しても、各テストに反映されない
- 各テスト上のテンプレート部分を手動で差し替える必要がある
- 各テスト上のテンプレート部分が、どこからどこまでなのか判別不可

また、既に作成済のテストをテスト・テンプレート化することもできます。

使用中のテストがある状態でテスト・テンプレートを更新した場合、 ステップ番号・操作番号・操作内容が変わらなければ、テスト側の値が優先されます。

1. **テスト・テンプレート作成方法**
    1. [テスト] - [テンプレート] - [新規テンプレート作成] を選択します。

Fig. 11.1 新規テンプレート作成

    2. 起点となるページ、およびテンプレート名を設定後、[作成して進む] を選択します。

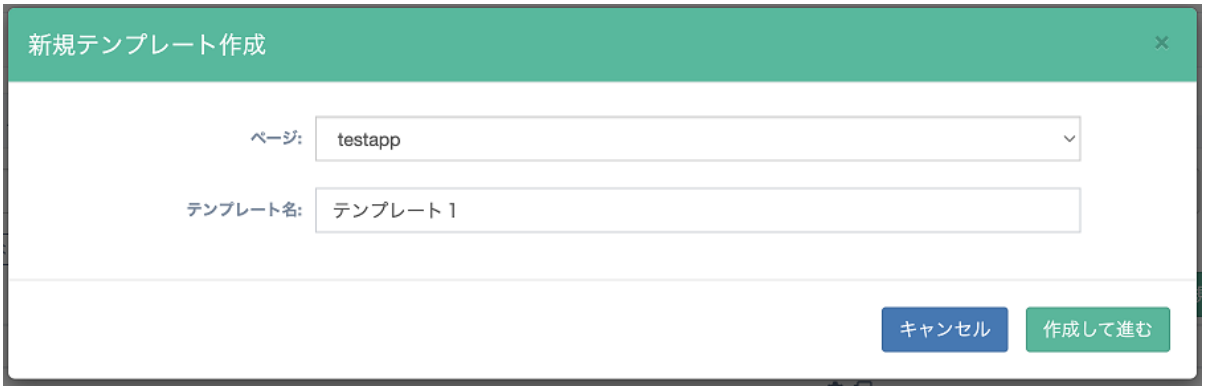


Fig. 11.2 新規テンプレート作成ダイアログ

    3. 「テストテンプレート編集画面」が表示されます。この画面では、**以下の機能を除き**「テスト編集画面」と同じ操作が可能です。
        - テストコードの出力
        - テスト仕様書の出力
        - テンプレート化
        - パターンの設定
        - ループ操作の設定
        - テストテンプレートの追加

4. **テスト・テンプレート利用方法**
   - [テスト編集] 画面にて [下に操作を追加] - [テストテンプレートを追加] メニューを選択します。

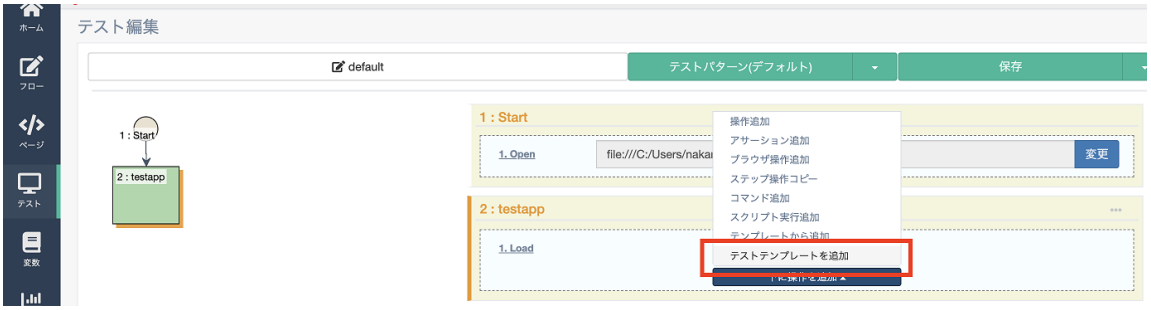


Fig. 11.3 テストテンプレートを追加ボタン

   - [テストテンプレートの読み込み] ダイアログが表示されます。
   - テンプレートを選択すると、テストケースに取り込まれます。

Fig. 11.4 テストテンプレートの読み込みダイアログ

   - 茶色の枠が、テストテンプレートを利用している箇所になります。
     このテストテンプレートに関する主な仕様は、以下の通りです。
     - テストテンプレートを編集すると、使用しているテストの該当箇所が更新されます。
     - テンプレート内の各操作のプロパティを設定できます。**テストケース内で値の編集が行われている場合はテンプレート側の編集は反映されません。**
     - テンプレート内の各操作を削除する事はできません。
     - テンプレート内に任意の操作を追加する事はできません。

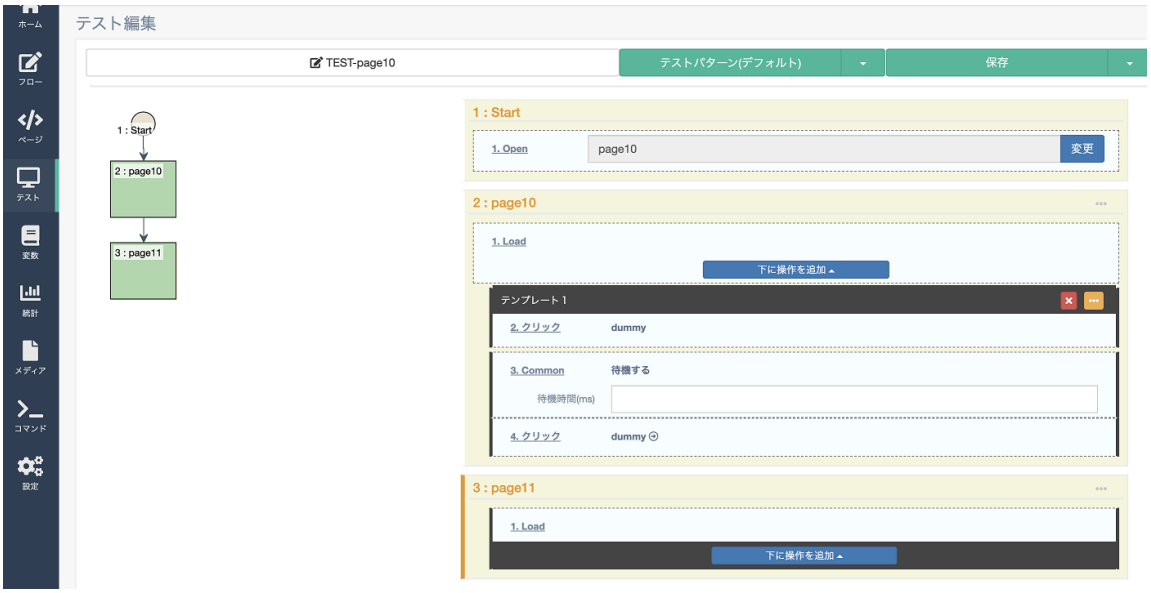


Fig. 11.5 実行結果

5. **既存のテストをテスト・テンプレート化するには**
   - 既存のテストをテンプレート化する手段として「テンプレート化する」という機能が追加されました。

Fig. 11.6 「テンプレート化する」ボタン

   - 「テンプレート化する」メニューを選択し、フォルダや名前を入力し、[OK] ボタンを押すと、テストテンプレートとして保存されます。

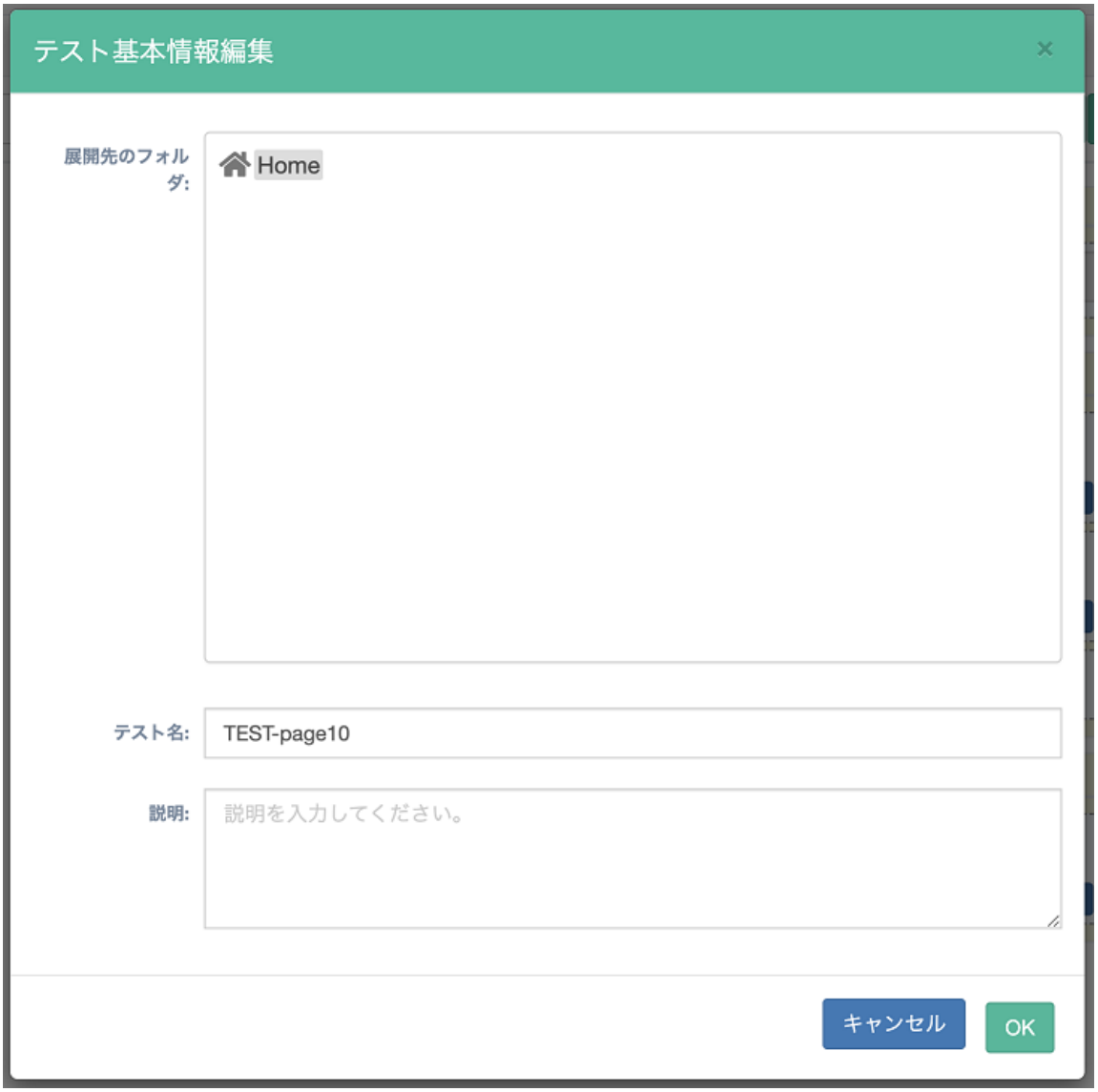


Fig. 11.7 テスト基本情報編集ダイアログ

- ただし以下の操作はできません。
  - ループを含むテストは、テンプレート化できません。
  - 同様に、既にテストテンプレートを含むテストは、テンプレート化できません。
  - １つのテストケースには、２つ以上のテスト・テンプレートは追加できません。

III. 操作の流れ/機能説明 目次 / 使い方マニュアル 目次 に戻る

Base version Testablish_v1.4.7
$Revision: 840 $ $Date:: 2023-05-02#$