DevTest Solutions

CA Application Test の使用

バージョン 8.0

ca technologies

CA への連絡先

テクニカル サポートの詳細については、弊社テクニカル サポートの Web サイト（http://www.ca.com/jp/support/）をご覧ください。

目次

第 1 章: Application Test の概要

このセクションには、以下のトピックが含まれています。

Registry

レジストリは、すべての DevTest サーバおよび DevTest ワークステーション コンポーネントの登録を一元的に行うための場所です。

レジストリは、すべての DevTest ランタイム コンポーネントの場所を追跡し、登録済みの各コンポーネントの場所を検索できるようにします。 また、レジストリは、サーバ管理、レポート、CVS、および CA Continuous Application Insight のための Web コンソールを提供します。 コンポーネント間の通信に使用される共通の JMS プロバイダは、レジストリ プロセスで起動および実行されます。 DevTest によって展開された Java エージェント用のブローカもレジストリ内に存在します。

レジストリの完全修飾名は、**tcp://ホスト名または IP アドレス:2010/レジストリ名** です。 以下に例を示します。

- tcp://localhost:2010/Registry
- tcp://myserver:2010/Registry
- tcp://myserver.example.com:2010/Registry
- tcp://172.24.255.255:2010/Registry

DevTest ワークステーション には、レジストリ モニタが含まれています。 レジストリ モニタを使用すると、テスト スイートのテスト ケース、シミュレータ、コーディネータ、および仮想環境をモニタできます。

レジストリは少なくとも 1 つのラボ (P. 394)と関連付けられ、これには Default ラボという名前が付けられます。 ラボを指定せずにコーディネータ サーバ、シミュレータ サーバ、または VSE サーバを作成すると、サーバは Default ラボに属します。

レジストリの起動

レジストリがローカルにインストールされていない場合は、レジストリがインストールされているサーバにログインします。 レジストリを起動するには、以下のいずれかの手順に従います。

Windows で有効

- コマンド プロンプトを開き、**LISA_HOME¥bin** ディレクトリに移動して、以下のコマンドを入力します。

  ```
  Registry
  ```

- ［スタート］メニューをクリックし、［すべてのプログラム］-［DevTest］-［レジストリ］をクリックします。
- **LISA_HOME¥bin** ディレクトリにある **Registry.exe** ファイルをダブルクリックします。

UNIX で有効

- ターミナル ウィンドウを開き、**LISA_HOME/bin** ディレクトリに移動して、以下のコマンドを入力します。

  ```
  ./Registry
  ```

以下のメッセージが表示されるまで待ちます。

```
レジストリは使用可能です。
```

名前付きレジストリの作成

名前付きレジストリを作成するには、**-n** オプションを指定してレジストリ実行可能ファイルを実行します。

```
LISA_HOME¥bin¥Registry -n RegistryName
```

以下の例では、**registry1** という名前のレジストリを作成します。

Windows で有効

```
cd C:¥Lisa¥bin
Registry -n registry1
```

UNIX で有効

```
cd Lisa/bin
./Registry -n registry1
```

レジストリの変更

DevTest ワークステーション で作業している場合は、別のレジストリに切り替えることができます。

次の手順に従ってください:

1. メイン メニューから［システム］-［レジストリ］-［DevTest レジストリの変更］を選択します。

   ［DevTest レジストリの設定］ダイアログ ボックスが表示されます。

2. レジストリ名を入力するか、ドロップダウン リストから以前に使用したレジストリを選択します。

3. チェック ボックスをオンまたはオフにします。

   オン： DevTest ワークステーション の起動時に［DevTest レジストリの設定］ダイアログ ボックスが表示されます。

   オフ： DevTest ワークステーション は起動時に、最後に接続したレジストリに接続します。

4. ［OK］をクリックします。

コーディネータ サーバ

コーディネータ サーバはテスト ランの情報をドキュメントとして受け取り、1 つ以上のシミュレータ サーバ (P. 17)で実行されるテストをコーディネートします。 テストをステージングする際、テストをコーディネイトするコーディネータ サーバを選択します。 また、テストのインスタンスを実行するときに使用するシミュレータ サーバを指定するステージング ドキュメントも指定します。どのコーディネータ サーバも、任意のシミュレータ サーバで（ステージング ドキュメントに基づき）インスタンスを起動できます。

コーディネータ サーバはメトリック収集およびレポートを管理し、DevTest ワークステーション にモニタ用のテスト データを渡します。DevTest サーバ 環境には、複数のコーディネータ サーバを配置できます（通常は複数）。

コーディネータ サーバは、ステージング ドキュメント、テスト ケース、および設定が提供されたときにテストを実行します。

lisa.coordName プロパティは、コーディネータ サーバのデフォルト名を設定します。

```
lisa.coordName=Coordinator
```

コーディネータ サーバの完全修飾名は、**tcp://ホスト名または IP アドレス:2011/コーディネータ名** です。

コーディネータ サーバの作成

コーディネータ サーバを作成するには、以下のコマンドを実行します。

```
LISA_HOME¥bin¥CoordinatorServer -n CoordinatorServerName -m RegistryName
```

以下の例では、**coordinator1** という名前のコーディネータ サーバを作成します。 関連するレジストリの名前は **registry1** です。

Windows で有効

```
cd C:¥DevTest¥bin
CoordinatorServer -n coordinator1 -m tcp://localhost:2010/registry1
```

UNIX で有効

```
cd DevTest/bin
./CoordinatorServer -n coordinator1 -m tcp://localhost:2010/registry1
```

-l オプションを使用すると、指定したラボ (P. 394)にコーディネータを追加できます。 **-l** オプションを指定しないと、コーディネータは Default ラボに追加されます。

--version オプションを使用するとバージョン番号を表示できます。

詳細については、「*管理*」の「サービスとしてのサーバ コンポーネントの実行」を参照してください。

コーディネータ サーバのモニタ

DevTest ワークステーション が関連するレジストリに対応付けられている場合、実行中のコーディネータ サーバが［レジストリ モニタ］に表示されます。

レジストリ モニタからコーディネータ サーバをモニタする方法

1. DevTest ワークステーション で［レジストリ モニタの切替］アイコンをクリックします。
2. ［コーディネータ サーバ］タブをクリックします。

コーディネータ サーバが、サーバ コンソールのネットワーク グラフに表示されます。

サーバ コンソールからコーディネータ サーバをモニタする方法

1. DevTest ワークステーション のメイン メニューから［表示］-［サーバ コンソール］を選択します。
2. ネットワーク グラフでコーディネータ サーバをクリックします。

   詳細ウィンドウが表示されます。

シミュレータ サーバ

シミュレータ サーバは、コーディネータ サーバ (P. 14)の管理下でテストを実行します。

仮想ユーザまたはテスト インスタンスは、シミュレータ サーバで作成および実行されます。 仮想ユーザの数、そして展開するシミュレータ サーバの数は、実行されるテストの性質によって決まります。 仮想ユーザはそれぞれがクライアント システムと通信します。

多数の仮想ユーザによる大規模なテストの場合は、仮想ユーザを複数のシミュレータ サーバに配分できます。

lisa.simulatorName プロパティは、シミュレータ サーバのデフォルト名を設定します。

```
lisa.simulatorName=Simulator
```

シミュレータ サーバの完全修飾名は、**tcp://ホスト名または IP アドレス:2014/シミュレータ名** です。

シミュレータ サーバの作成

シミュレータを作成するには、以下のコマンドを実行します。

```
LISA_HOME¥bin¥Simulator -n SimulatorName -m RegistryName
```

以下の例では、**simulator1** という名前のシミュレータを作成します。 関連するレジストリの名前は **registry1** です。

Windows で有効

```
cd C:¥DevTest¥bin
Simulator -n simulator1 -m tcp://localhost:2010/registry1
```

UNIX で有効

```
cd DevTest/bin
./Simulator -n simulator1 -m tcp://localhost:2010/registry1
```

-l オプションを使用すると、指定したラボ (P. 394)にシミュレータを追加できます。 -l オプションを指定しないと、シミュレータは Default ラボに追加されます。

-i オプションを使用すると、このシミュレータの仮想ユーザの数を指定できます。 以下に例を示します。

```
Simulator -n simulator1 -m tcp://localhost:2010/registry1 -i 100
```

--version オプションを使用するとバージョン番号を表示できます。

シミュレータを作成するとき、-n オプションを使用してコロンとデフォルト以外のポート番号を指定すると、デフォルト ポート番号を上書きできます。 以下に例を示します。

```
Simulator -n testSim1:35001
```

複数のシミュレータを実行するには、上記のように、コマンド プロンプトを使用してシミュレータを作成します。

その名前においてポート番号が指定されていない場合、シミュレータはまずポート 2014 を使用しようとします。2014 が既に使用されている場合は、2015、2016 の順に 2024 まで試します。

ポートの使用の詳細については、「*管理*」の「デフォルト ポート番号」を参照してください。

シミュレータをサービスとして実行する方法については、「*管理*」の「サービスとしてのサーバ コンポーネントの実行」を参照してください。

シミュレータ サーバのモニタ

DevTest ワークステーション が関連するレジストリに対応付けられている場合、実行中のコーディネータ サーバが［レジストリ モニタ］に表示されます。

レジストリ モニタからシミュレータ サーバをモニタする方法

1. DevTest ワークステーション で［レジストリ モニタの切替］アイコンをクリックします。
2. ［シミュレータ］タブをクリックします。

シミュレータ サーバが、サーバ コンソールのネットワーク グラフに表示されます。

サーバ コンソールからシミュレータ サーバをモニタする方法

1. DevTest ワークステーション のメイン メニューから［表示］-［サーバ コンソール］を選択します。
2. ネットワーク グラフでシミュレータ サーバをクリックします。

   詳細ウィンドウが表示されます。

コマンドライン ユーティリティ

LISA_HOME¥bin ディレクトリには、以下のコマンドライン ユーティリティが含まれています。

CAI コマンドライン ツール

PFCmdLineTool コマンドでは、コマンドラインからさまざまな CA Continuous Application Insight タスクを実行できます。詳細については、「*CA Continuous Application Insight の使用*」の「CAI コマンド ライン ツール」を参照してください。

CVS Manager

CVS マネージャでは、コマンドライン オプションを使用して CVS ダッシュボードに対してモニタを追加および削除できます。詳細については、「*CA Application Test の使用*」の「CVS マネージャ (P. 432)」を参照してください。

Make Mar

Make Mar では、MAR 情報ファイル（スタンドアロンまたはアーカイブ内）の内容を表示したり、MAR 情報ファイルからモデル アーカイブ ファイルを作成したりすることができます。 詳細については、「*CA Application Test の使用*」の「Make Mar ユーティリティ (P. 313)」を参照してください。

サービス イメージ マネージャ

サービス イメージ マネージャは、サービス イメージ（新規または既存）へトランザクションをインポートしたり、2 つ以上のサービス イメージを結合したりするために使用します。 詳細については、「*CA Service Virtualization の使用*」の「ServiceImageManager」を参照してください。

Service Manager

サービス マネージャは、実行中のサーバ プロセスのステータスの確認、リセット、または停止のために使用します。詳細については、「*管理*」の「サービス マネージャ」を参照してください。

テスト ランナー

テスト ランナーは、DevTest ワークステーション の「ヘッドレス」バージョンです。機能は同じですが、ユーザ インターフェースがありません。詳細については、「*CA Application Test の使用*」の「テスト ランナー (P. 366)」を参照してください。

VSE マネージャ

VSE マネージャは、仮想サービス環境を管理するために使用します。詳細については、「*CA Service Virtualization の使用*」の「VSE マネージャ コマンド」を参照してください。

第 2 章: CA Application Test による DevTest ポータルの使用

このセクションには、以下のトピックが含まれています。

DevTest ポータルを開く

Web ブラウザから DevTest ポータルを開きます。

注: 実行している必要のあるサーバ コンポーネントについては、「*インストール*」の「DevTest プロセスまたはサービスの起動」を参照してください。

次の手順に従ってください:

1. 以下のアクションのいずれかを完了します。
   - Web ブラウザに「**http://localhost:1507/devtest**」と入力します。レジストリがリモート コンピュータ上にある場合は、**localhost** をそのコンピュータの名前または IP アドレスに置き換えます。
   - DevTest ワークステーション から［表示］-［DevTest ポータル］を選択します。
2. ユーザ名とパスワードを入力します。
3. ［ログイン］をクリックします。

第 3 章: API テストの作成

［Create API Test］ウィンドウでは、手動で、または既存の API テストもしくはファイル システムの要求/応答ペアを使用して、テストを構築できます。

次の手順に従ってください：

1. ［プロジェクト］ドロップダウン リストからプロジェクトを選択するか、またはプロジェクト名を入力します。 プロジェクトが存在する場合、この API テストはそのプロジェクトに追加されます。 プロジェクトが存在しない場合は、作成されます。
2. ［テスト名］フィールドに、テストの名前を入力します。 テスト名が一意であるようにするため、テスト ランの後、入力した名前にデート スタンプおよびタイム スタンプが追加されます。
3. ［Base URL］フィールドに、要求/応答ペアの URL のベース部を入力します。ベース URL は、実行時に要求が接続される URL です。たとえば、以下のように入力します。

   ```
   http://localhost:8080
   ```

4. ［説明］フィールドに、テストの簡単な説明を入力します。

このセクションには、以下のトピックが含まれています。

テストからの R/R ペアのインポートによるテストの作成

既存の API テストをテスト エディタにロードすることで、テストを作成できます。

次の手順に従ってください：

1. R/R ペア エディタ ツールバーで、［既存のテストからの R/R ペアのインポート］ をクリックします。

   ［Load Existing API Tests］ウィンドウが表示されます。

2. リソース サーバで利用可能なすべてのプロジェクトと、各プロジェクトで利用可能な API テストがリストで表示されます。

   DevTest コンソールに API テストを作成していない場合、ドロップダウン リストには項目が表示されません。

3. プロジェクト名をフィルタするには、［フィルタ］フィールドに文字列を入力します。

4. API テストをインポートするには、テストを選択して［ロード］をクリックします。

ファイルからの R/R ペアのインポートによるテストの作成

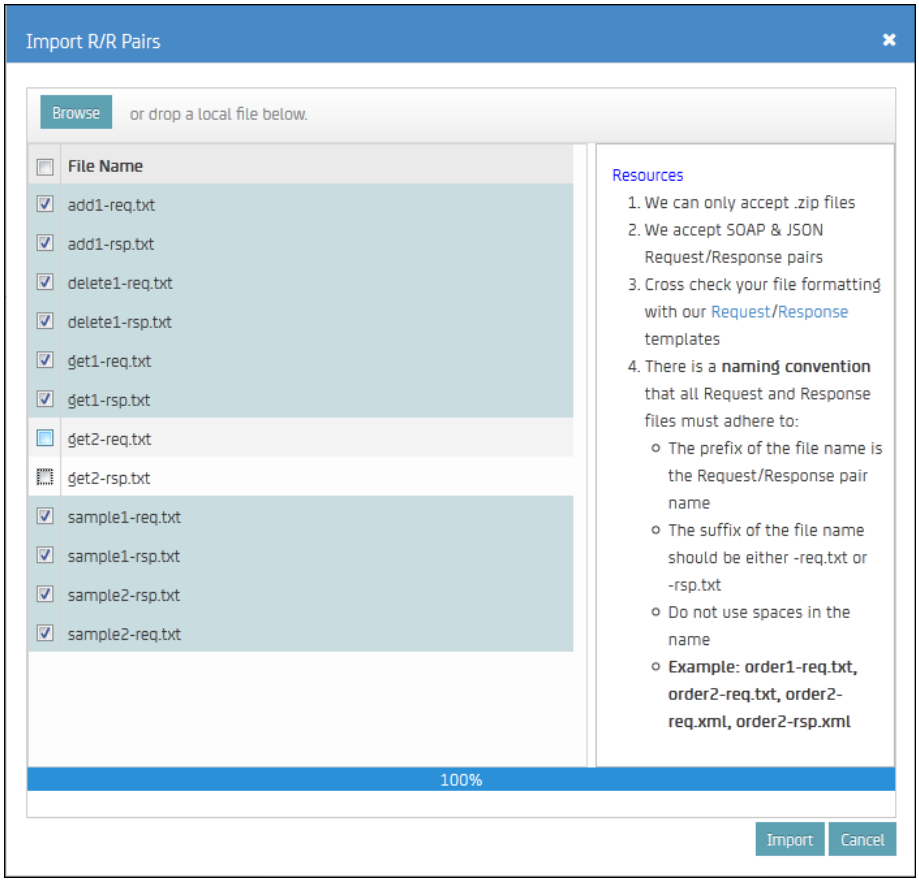
要求/応答ペアを ZIP ファイルからテスト エディタにロードすることで、テストを作成できます。

次の手順に従ってください：

1. R/R ペア エディタ ツールバーで、［ローカル ZIP ファイルからの R/R ペアのインポート］ をクリックします。

   ［Import R/R Pairs］ウィンドウが表示されます。

2. R/R ペア ファイルが含まれる ZIP ファイルを指定するには、ファイル システムを参照するか、または ZIP ファイルをウィンドウにドラッグ アンド ドロップします。



ウィンドウの［リソース］セクションに記載された、R/R ペア ファイルの要件に注意します。

3. インポートする R/R ペア ファイルを選択します。R/R ペア ファイルをすべて選択するには、［ファイル名］チェック ボックスをオンにします。
4. ［インポート］をクリックします。

手動でのテスト スイートの作成

テストを手動で作成するには、Test Editor を使用します。

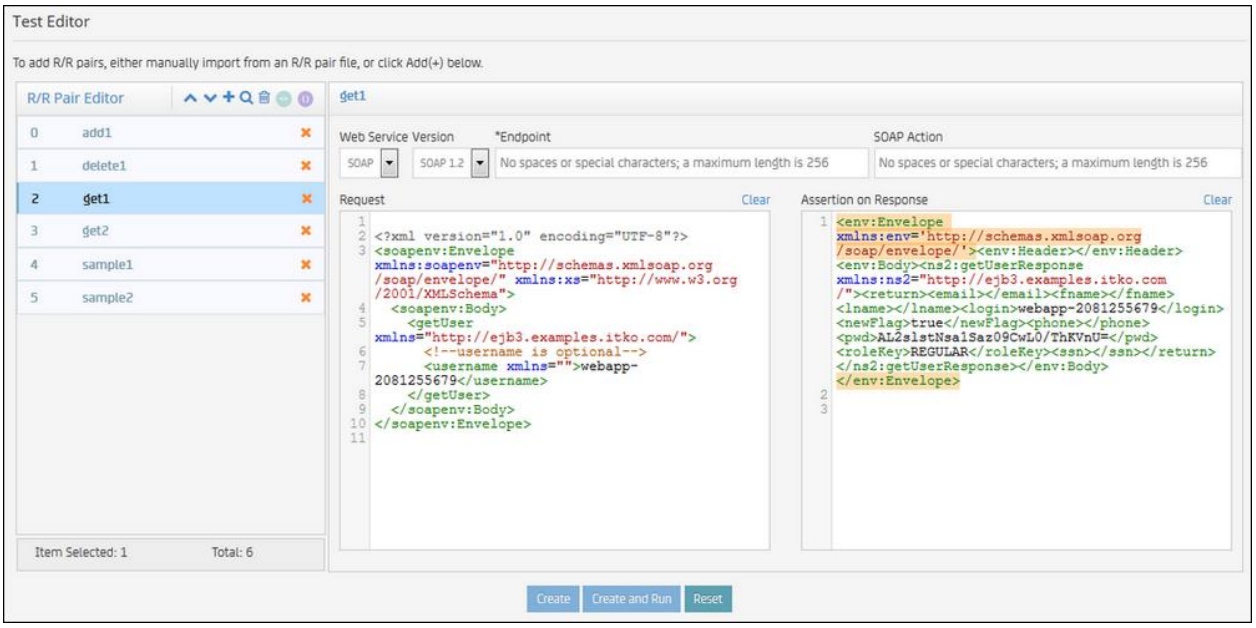
次の手順に従ってください:

1. ［R/R ペアの追加］ をクリックします。

   0 という番号の付いた、最初の R/R ペアが表示されます。 デフォルト名は、**Baseline-0** です。 名前を変更するには、名前をダブルクリックします。 編集を終了したら、Enter キーを押して編集モードを終了します。
2. 右側ペインで、R/R ペア名の下にあるドロップダウン リストから［Web サービス］を選択します。 有効なオプションは、REST および SOAP です。
3. 以下のいずれかの操作を実行します。
   - REST の場合は、［タイプ］（GET、POST、PUT、DELETE、または HEAD）および［Resource Path］（Web サービスへのパス）を入力します。
   - SOAP の場合は、［バージョン］（SOAP 1.2 または SOAP 1.1）、［エンドポイント］（この要求の特定のエンドポイント）、および［SOAP アクション］（SOAP 1.2 の場合はオプション）を入力します。
4. ［要求］フィールドに、要求のテキストを入力します。
5. ［Assertion on Response］フィールドに、応答のテキストを入力します。

API テストの編集

テストを編集するには、テスト エディタを使用します。



テストの名前を変更するには、名前をダブルクリックし、新しい名前を入力して、［Enter］をクリックします。

API テストを作成するには、［作成］ボタンをクリックします。 テストが正常に作成されると、新しいテストが［Test Edit］ページで自動的に開きます。

API テストを作成して実行するには、［Create and Run］ボタンをクリックします。 テストが正常に作成されると、モニタリング ポートレットが表示されます。

重要: DevTest でテスト ファイルの名前に「.invalid」が追加される、最も可能性が高くてよく見られる原因としては、（たとえば、動的 Java 実行ステップにおいて）テストに必要な Java クラスが DevTest で使用できないことが挙げられます。そのようなテスト ファイルが保存されるか、または閉じられると、DevTest によってテストのコピーが作成され、拡張子が「.invalid」となるようにファイル名が変更されます。

第 4 章: テストの監視

［Monitor Test］ウィンドウでは、CA Application Test で実行された API テスト、テスト ケース、およびテスト スイートを参照できます。

このセクションには、以下のトピックが含まれています。

テストおよびスイートのフィルタ処理

データベースから返されるテスト ケースおよびテスト スイートをフィルタするには、［検索］ をクリックします。

以下のパラメータを入力します。

送信元

テスト ケースおよびテスト スイートの開始日時を定義します。

終了

テスト ケースおよびテスト スイートの終了日時を定義します。

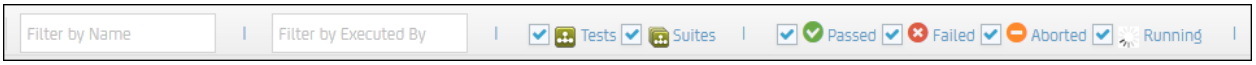
Executed By

テスト ケースまたはテスト スイートをサブミットしたユーザを指定します。

先週実行されたテスト ケースおよびテスト スイートを表示するには、デフォルト値の［過去 7 日間］のままにします。 ドロップダウン リストを使用して、さまざまな日時範囲を選択できます。

フィルタを適用するには、［適用］をクリックします。

結果が表示されたら、ペインの一番上のフィルタ オプションを使用して、さらにフィルタすることができます。



Filter by Name

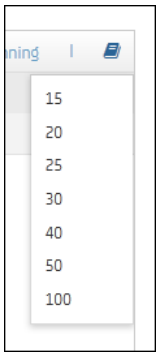
特定のテスト ケースまたはスイートの結果のみを表示するには、テストまたはスイートの名前またはその一部を入力します。

Filter by Executed By

特定のユーザがサブミットしたテスト ケースおよびスイートのみを表示するには、ユーザ ID またはその一部を入力します。

また、［テスト］または［スイート］のいずれかを選択することで、タイプによってフィルタすることもできます。 結果によってフィルタするには、［成功］、［失敗］、［中止］、または［実行中］を選択します。

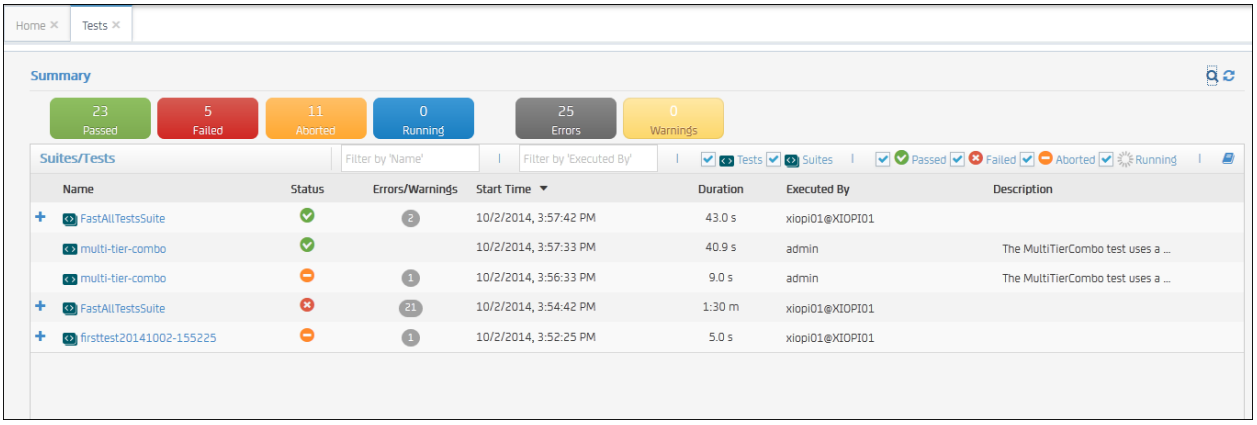
表示する行数を選択するには、［Page Size］をクリックします。



テストおよびスイートのモニタ

［Suites/Tests］ペインには、入力したフィルタ基準に一致する、テスト ケースおよびスイートが表示されます。

サマリ アイコンでは、成功、失敗、中止、または実行中のサイクルの数が表示されます。



以下のフィールドが表示されます。

name

テスト ケースまたはスイートの名前が表示されます。スイート名の前にプラス記号またはマイナス記号があります。スイートのテスト ケースを展開するか、または折りたたむには、その記号をクリックします。DevTest Solutions ポータルで作成されたテスト スイートでは、スイート名にデート スタンプおよびタイム スタンプが追加されます。

名前にマウス カーソルを置くと、情報アイコンが名前の右側に表示されます。テストまたはスイートの詳細については、そのアイコンをクリックしてください。

ステータス

テスト ケースまたはテスト スイートのステータス（成功、失敗、中止、または実行中）が示されます。

Errors/Warnings

各テスト ケースおよびテスト スイートのエラー（灰色の円に表示）および警告（黄色の円に表示）の数が表示されます。

Start Time

各テスト ケースおよびテスト スイートの開始日時が表示されます。

期間

テスト ケースまたはテスト スイートの実行の期間（秒単位）が表示されます。

Executed By

テスト ケースまたはテスト スイートのサブミット実行者の ユーザ ID が表示されます。

説明

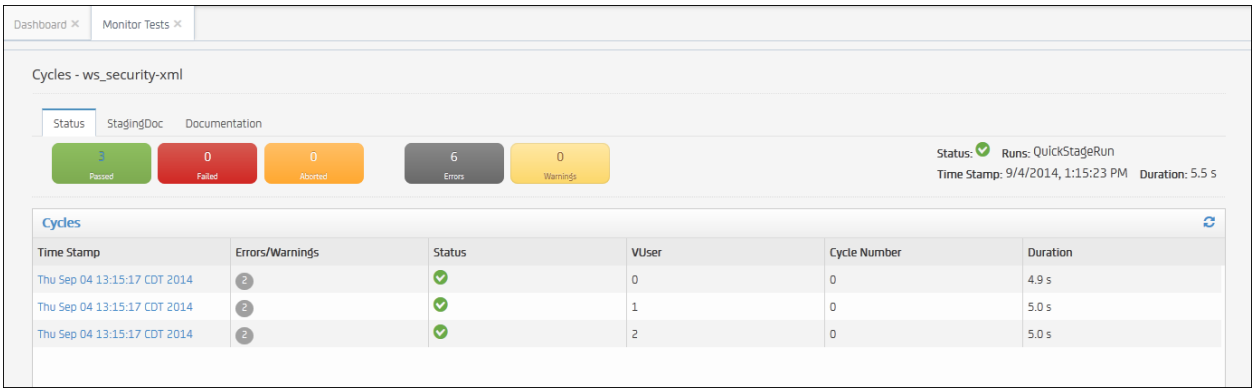
テスト ケースまたはテスト スイートの説明が表示されます。

テストおよびスイートの詳細の表示

テスト ケースの実行に関する詳細を表示するには、［Suites/Tests］ペインでテスト ケース名をクリックします。 ［ステータス］タブが表示されます。

サイクルが複数あるテスト ケース

サイクルが複数あるテスト ケースの場合は、各サイクルのステータスを示す、サイクル結果が表示されます。



各サイクルについて、以下のフィールドが表示されます。

タイム スタンプ

テスト ケースの各サイクルについて、開始日時が表示されます。

Errors/Warnings

各テスト ケースおよびテスト スイートのエラー（灰色の円に表示）および警告（黄色の円に表示）の数が表示されます。

ステータス

テスト ケースまたはテスト スイートのステータス（成功  、失敗  、中止  、または実行中  ）が示されます。

仮想ユーザ

各サイクルの仮想ユーザの数が表示されます。

Cycle Number

サイクル実行のインデックスが表示されます。テスト ランのサイクルが 1 つのみの場合、Cycle Number は 0 です。

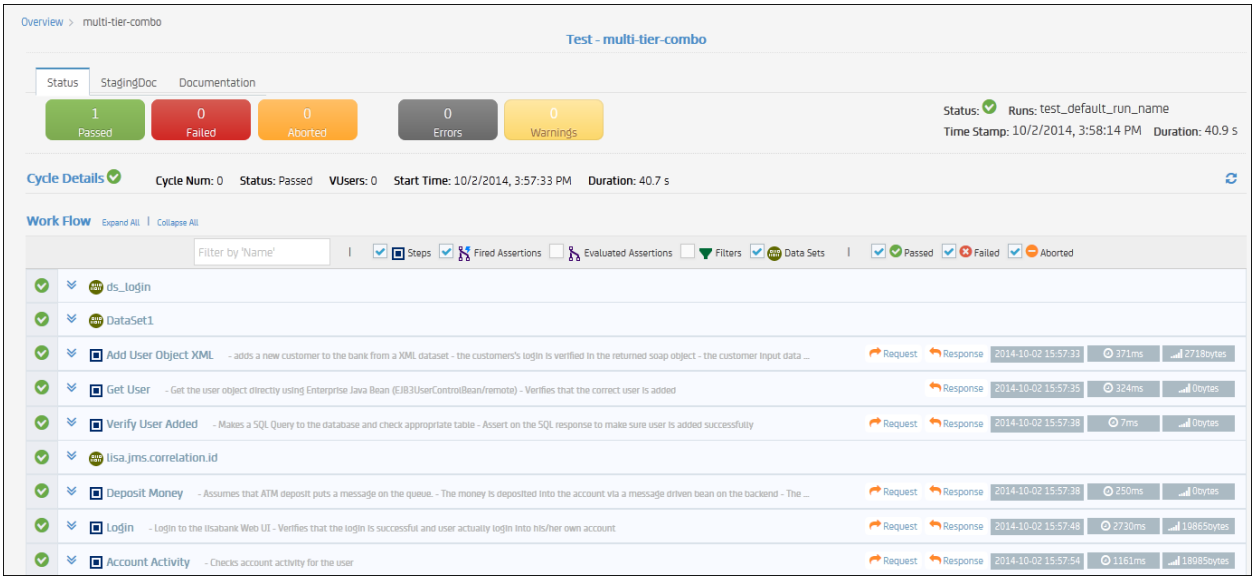
期間

テスト ケースまたはテスト スイートの実行の期間（秒単位）が表示されます。

Cycle Details

サイクル詳細を表示するには、以下のいずれかのアクションを実行します。

- サイクルが複数あるテスト ケースの場合は、サイクルのタイム スタンプをダブルクリックする
- サイクルが 1 つのみのテスト ケースの場合は、テスト ケース名をダブルクリックする



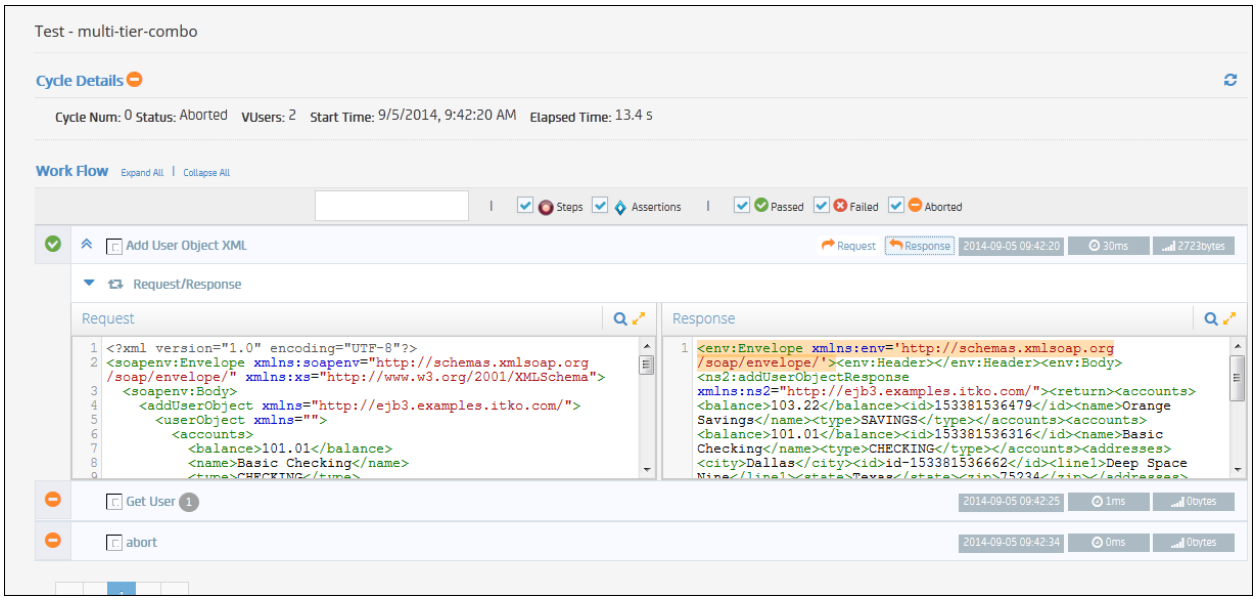
テスト ケースの各テスト ステップが表示されます。 各ステップと関連付けられている、データ セット、フィルタ（グローバルおよび非グローバル）、およびアサーション（グローバルおよび非グローバル）もすべて表示されます。

テスト ケースのステップをすべて展開するには、［すべて展開］をクリックします。 ステップをすべて折りたたむには、［すべて折りたたむ］をクリックします。

各エレメント名の左側に、矢印  が表示されます。 要求/応答、イベント、エラーまたは警告、プロパティなどの、エレメントに関する詳細情報を表示するには、矢印をクリックします。

ステップでエラーまたは警告が発生した場合、ステップ名の右側にある灰色の円（エラーの場合）またはオレンジの円（警告の場合）に、エラーおよび警告の数が表示されます。この数にマウス カーソルを置くと、エラーまたは警告のテキストが表示されます。

ステップに要求および応答が含まれる場合は、［要求］ または［応答］ をクリックすると、その詳細が表示されます。



要求または応答のテキストを検索するには、［検索］ をクリックします。 クエリ文字列を入力し、［Enter］をクリックします。

注: イベントまたはプロパティが含まれたステップのイベントまたはプロパティが表示されない場合は、そのテストのステージング ドキュメントを確認して、設定または参照された、イベント、要求、応答、スクリーンショット、およびプロパティが記録されるように設定されていることを確認してください。 **Run1User1CycleShowAll** ステージング ドキュメントを使用して、すべてがキャプチャされることを確認できます。

要求または応答のビューを展開するには、［展開］ をクリックします。

テストが使用するステージング ドキュメントを表示するには、［ステージング ドキュメント］タブをクリックします。

テストに関連付けられたドキュメントを表示するには、［ドキュメント］タブをクリックします。

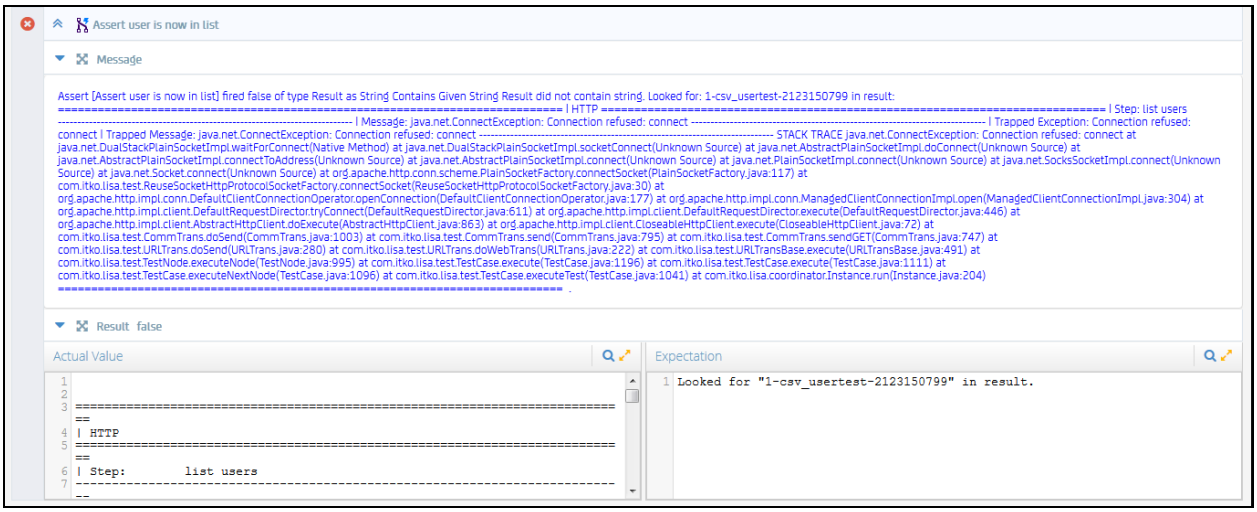
アサーション、フィルタ、データ セット、およびコンパニオンの詳細の表示

アサーション、フィルタ、データ セット、およびコンパニオンの結果の詳細を表示するには、エレメント名の左側の矢印  をクリックします。

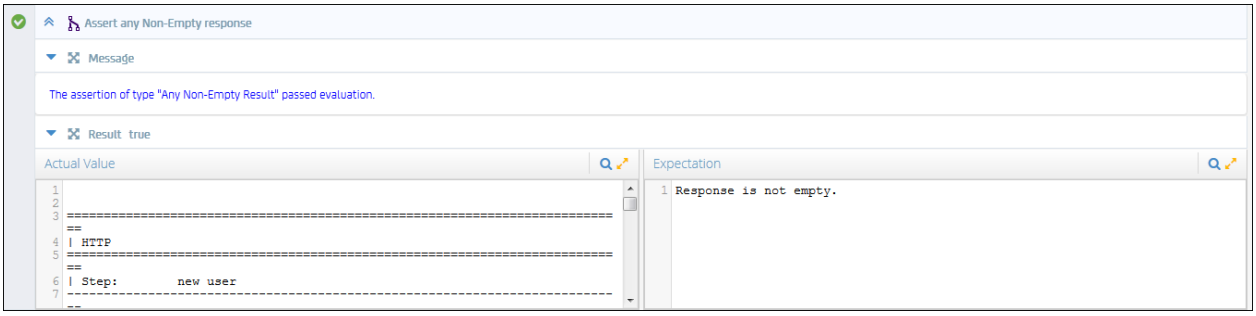
アサーションの詳細

フィルタ バーの適切なチェック ボックスをオンにすることで、起動されたアサーションまたは評価されたアサーションの詳細を選択して表示することができます。

起動されたアサーションの詳細では、アサーションによって作成されたメッセージ、およびアサーションの結果が表示されます。

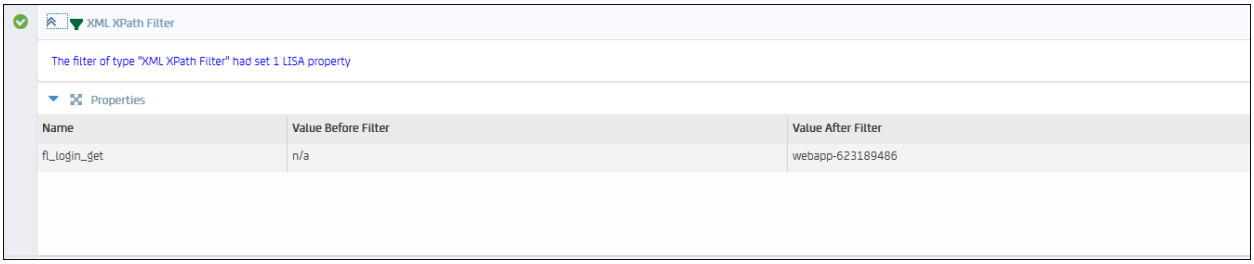


評価されたアサーションの詳細では、アサーションによって作成されたメッセージ、およびアサーションの結果が表示されます。



フィルタの詳細

フィルタの詳細では、フィルタによって設定されたプロパティが表示されます。

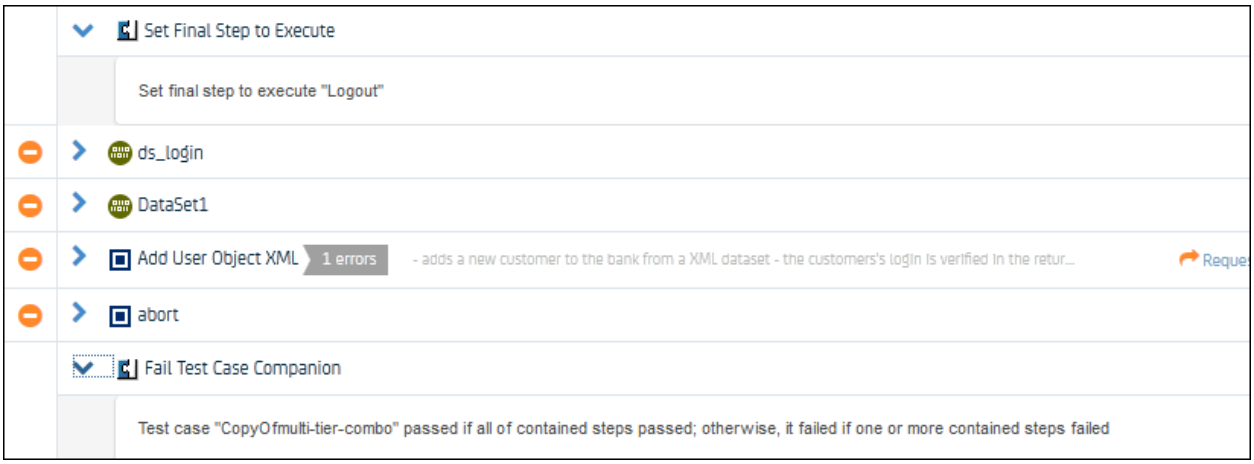


データ セットの詳細

データ セットの詳細では、データ セット コンテンツ、およびデータ セットによって変更されたプロパティが表示されます。

コンパニオンの詳細

コンパニオンの詳細では、コンパニオンによって実行されたアクションが表示されます。



第 5 章: テスト アーティファクトの管理

テスト アーティファクトを管理するには、左側のナビゲーション バーの［Manage］オプションを使用するか、またはポータルの［Quick Links］ポートレットで［Manage Test Artifacts］を選択します。

DevTest ポータルで作成されたテストを管理するには、［Manage API Test］を選択します。

DevTest ワークステーション によって作成されたテスト ケースを管理するには、［Manage Tests］を選択します。

DevTest ワークステーション によって作成されたテスト スイートを管理するには、［Manage Test Suites］を選択します。

プロジェクトを選択するには、［プロジェクト］ドロップダウンを使用します。 ドロップダウンで利用可能なプロジェクトは、LISA_HOME ディレクトリの Projects ディレクトリにあるプロジェクトです。

このセクションには、以下のトピックが含まれています。

第 6 章: API テストの管理

［Manage API Tests］オプションでは、DevTest ポータルで作成されたテストを編集できます。

次の手順に従ってください：

1. テストのリストから、テストを選択してクリックします。

   テスト エディタが表示されます。

2. 「API テストの編集 (P. 29)」で説明した、テスト エディタを使用します。

第 7 章: テストの管理

テストを実行するには、テスト名を右クリックして［実行］を選択します。

テストを削除するには、テスト名を右クリックして［削除］を選択します。

テストが含まれる Mar (P. 300) ファイルをダウンロードするには、テスト名を右クリックして［Download Mar］を選択します。

テストに関する、情報およびドキュメントを表示するには、テスト名をクリックします。

第 8 章: テスト スイートの管理

テスト スイートを実行するには、スイート名を右クリックして［実行］を選択します。

テスト スイートを削除するには、スイート名を右クリックして［削除］を選択します。

テスト スイートの Mar (P. 300) ファイルをダウンロードするには、スイート名を右クリックして［Download Mar］を選択します。

テスト スイートに関する、情報およびドキュメントを表示するには、スイート名をクリックします。

第 9 章: CA Application Test でのワークステーションおよびコンソールの使用

このセクションには、以下のトピックが含まれています。

ワークステーションおよびコンソールの概要

このセクションには、以下のトピックが含まれます。

DevTest ワークステーション

DevTest ワークステーション は、テストの開発、ステージング、およびモニタのための統合環境です。

DevTest ワークステーション のバージョンまたは DevTest サーバ 環境で作業を行えます。

DevTest ワークステーション では、テストはワークステーション環境で管理および実行されます。 DevTest ワークステーション は、以下のコンポーネントをテストするために QA/QE、開発、ビジネス分析チームが使用するテスト クライアントです。

- さまざまなブラウザや Web ユーザ インターフェース
- ユーザ インターフェースの基盤

DevTest ワークステーション は、ユニット テスト、機能テスト、統合テスト、レグレッション テスト、およびビジネス プロセス テストをコードを記述せずに構築およびステージングするために使用します。 DevTest ワークステーション は、レジストリが実行されていることを必要とします。テストは DevTest ワークステーション 環境（テスト オーサリング IDE）で管理および実行されます。この環境では、組み込みのコーディネータ サーバおよびシミュレータ サーバが実行されます。

DevTest サーバ では、テストも管理され、ワークステーション環境で実行されます。 ワークステーションはサーバに接続し、DevTest ワークステーション で開発されたテストを展開およびモニタします。

DevTest テスト ケースおよび仮想サービス用のサーバ サイド エンジン（テストおよび仮想サービス モデル オーサリング IDE）は、ユニット、機能、負荷、パフォーマンスの各テストのために DevTest テスト ケースを継続的に管理、スケジュール、調整します。

DevTest ワークステーション を開く

DevTest ワークステーション を開くときには、レジストリ (P. 11)を指定するよう要求されます。

コンピュータに DevTest サーバ がインストールされている場合は、以下のレジストリを使用できます。

- ローカル コンピュータで実行されているレジストリ
- リモート コンピュータで実行されているレジストリ

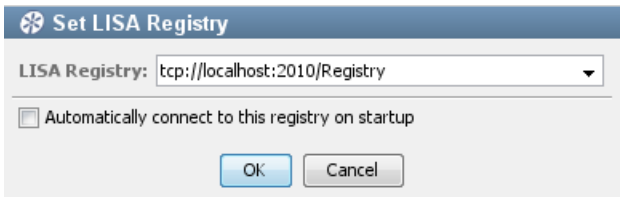
コンピュータに DevTest ワークステーション がインストールされている場合は、リモート コンピュータで実行されているレジストリを使用します。

SSL 対応のレジストリを指定する方法の詳細については、「*管理*」の「通信を保護するための SSL の使用」を参照してください。

DevTest ワークステーション を開く方法

1. 以下のいずれかを実行します。
   - コマンド プロンプトを開き、**LISA_HOME¥bin** ディレクトリに移動し、DevTest ワークステーション 実行可能ファイルを実行します。
   - デスクトップに DevTest アプリケーション アイコン  がある場合は、アプリケーション アイコンをダブルクリックします。
   - ［スタート］メニューをクリックし、［すべてのプログラム］-［DevTest］-［DevTest ワークステーション］をクリックします。

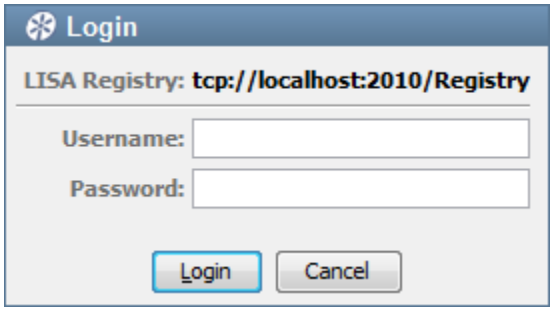
   ［DevTest レジストリの設定］ダイアログ ボックスが表示されます。



2. デフォルトのレジストリをそのまま使用するか、別のレジストリを指定します。 DevTest ワークステーション は GUI であるため、起動時にデフォルトのリモート レジストリを設定する方法はありません。このプロンプトでレジストリを指定するか、または［レジストリ モニタの切替］コマンドを使用してレジストリを変更します。

3. ［OK］をクリックします。

   ［ログイン］ダイアログ ボックスが表示されます。



4. ユーザ名とパスワードを入力して、［ログイン］をクリックします。

メイン メニュー

DevTest ワークステーション のメイン メニューには、使用可能な主な機能のオプションが表示されます。 使用可能なオプションは、選択内容に応じて動的に変化します。 一部のメニュー、ドロップダウン リスト、およびツールバーは DevTest ワークステーション の操作全体で使用できます。

注: このメニューの項目は動的です。 DevTest 管理者は、セキュリティ プロファイルの一部として使用可能な項目を制御します。

プロジェクト パネル

プロジェクトは、関連する DevTest ファイルのコレクションです。 ファイルには、テスト ケース、スイート、仮想サービス モデル、サービス イメージ、設定、監査ドキュメント、ステージング ドキュメント、データ セット、モニタ、および MAR 情報ファイルなどが含まれます。

DevTest ワークステーション で一度に開けるプロジェクトは 1 つだけです。

初めてプロジェクトを作成 (P. 70)するときには、**LISA_HOME¥Projects** ディレクトリが作成されます。

DevTest ワークステーション のプロジェクト パネル内のフォルダは、すべてファイル システムに作成されるフォルダと同じです。 **LISA_HOME** ディレクトリを確認して、フォルダ構造やファイルを参照できます。

プロジェクトにはファイルをインポートできます。

プロジェクト パネルのレイアウト

プロジェクト パネルにはプロジェクト ツリーが表示されます。

プロジェクトを新規作成する場合、プロジェクト ツリーは以下の構造になります。

- VirtualServices
  - Images
  - VRScenarios
- テスト
  - Subprocesses
  - AuditDocs
  - Suites
  - StagingDocs
- Configs
- Data

プロジェクト パネルを開閉するには、［プロジェクト］をクリックします。

プロジェクト パネル内のツールバーには、以下のアクション用のアイコンがあります。

-  プロジェクトをリフレッシュする
-  プロジェクト パネルをドッキングまたはドッキング解除する
-  プロジェクト パネルを閉じる

プロジェクトには **.settings** ディレクトリが含まれていますが、プロジェクト パネルには表示されません。**.settings** ディレクトリは、一部の設定を内部的に保存するために使用され、Projects ディレクトリ内のファイル システムで参照できます。

プロジェクト パネルの右クリック メニュー

プロジェクト パネルから、プロジェクトのさまざまなドキュメントを作成できます。 プロジェクト パネルで選択した項目を右クリックすると、表示されるメニューは選択に応じて動的に変化します。

これらの選択項目の順序は、右クリックしたときに選択していた項目によって異なります。

また、ここで説明する右クリックで選択するメニューは、メイン メニューで［ファイル］-［新規］を選択しても使用できます。

新規テスト ケースの作成

詳細については、「*CA Application Test の使用*」の「テスト ケースの作成 (P. 234)」を参照してください。

新規 VS モデルの作成

詳細については、「*CA Service Virtualization の使用*」の「VSM の使用」を参照してください。 この機能を使用するには、追加ライセンスが必要です。

新規 VS イメージの作成

詳細については、「*CA Service Virtualization の使用*」の「サービス イメージの作成」を参照してください。 この機能を使用するには、追加ライセンスが必要です。

新規ステージング ドキュメントの作成

詳細については、「*CA Application Test の使用*」の「ステージング ドキュメントの作成 (P. 251)」を参照してください。

新規スイートの作成

詳細については、「*CA Application Test の使用*」の「テスト スイートの作成 (P. 287)」を参照してください。

新規テスト監査の作成

詳細については、「*CA Application Test の使用*」の「監査ドキュメントの作成 (P. 278)」を参照してください。

以下の 1 つ以上の項目も使用できる場合があります。

- 新規フォルダの追加
- ファイルのインポート
- プロジェクトの名前変更

- 削除
- 貼り付け
- プロジェクト メタデータの表示/編集

作成したドキュメントを一致するフォルダ名のデフォルトにするには、追加する際にその選択項目を右クリックします。 フォルダ名を右クリックせずにドキュメントを追加すると、DevTest は、それをプロジェクト パネル リストの一番下に追加します。 修正するには、ルート ディレクトリに移動して、そのファイルを手動で正しいフォルダへ移動させ、プロジェクトを開き直す必要があります。

特定の種類のファイル（.tst など）を推奨されるフォルダ（Tests など）に配置しなければならないという制限はありません。 ファイルはプロジェクトの任意の場所に配置できます。 唯一の例外は .config ファイルです。 これらは Configs フォルダに配置する必要があります。

examples プロジェクト

DevTest ワークステーション を初めて開くと、プロジェクト パネルに **examples** という名前のプロジェクトが表示されます。

任意のサンプル ファイルをダブルクリックして開きます。 右側のパネルには、適切なエディタが開かれます。

実際のプロジェクト ファイルは、**LISA_HOME¥examples** ディレクトリにあります。

MARInfos (P. 302)

AllTestsSuite

テスト スイート AllTestsSuite とすべての .tst ファイル、および AllTestsSuite を実行するための付随データ ファイルが含まれるスイート MAR。 このスイートには、1User1Cycle0Think ステージング ドキュメント、DefaultAudit 監査ドキュメント、および project.config 設定ファイルも含まれます。

creditCheckValidate

creditCheckValidate テスト ケースおよび monitorRunBase ステージング ドキュメントが含まれるテスト ベース モニタ MAR。

DatabaseModel

DatabaseModel 仮想サービス モデルおよび仮想サービス イメージが含まれる仮想サービス MAR。

OnlineBankingExternalCreditCheck-local

webservices-xml-fail テスト ケース、Run1User1Cycle ステージング ドキュメント、および project.config が含まれるテスト ベース モニタ MAR。

OnlineBankingJMStest-local

async-consumer-jms テスト ケース、Run1User1Cycle ステージング ドキュメント、および project.config 設定ファイルが含まれるテスト ベース モニタ MAR。

OnlineBankingTransactionMonitor-local

以下のアイテムが含まれる テスト ベース モニタ MAR。

- multi-tier-combo テスト ケース
- Run1User1Cycle ステージング ドキュメント

- テスト ケースをサポートするすべてのデータ ファイル
- project.config 設定ファイル

OnlineBankingWebServices-local

webservices-xml テスト ケース、Run1User1Cycle ステージング ドキュメント、および project.config 設定ファイルが含まれるテストベース モニタ MAR。

rawSoap

rawSoap テスト ケース、1User0Think_RunContinuously ステージング ドキュメント、および project.config 設定ファイルが含まれるテスト MAR。

AuditDocs (P. 278)

DefaultAudit

main_all_should_fail

ws_security-xml

Configs (P. 116)

プロジェクト

project.config ファイルには、多数のプロパティのためのインテリジェント デフォルトが含まれます。

Data

Data ディレクトリには、デモ サーバの一部のサンプルを実行するために必要なデータ セット、キーストア、および WSDL が含まれています。

Monitors

creditCheckValidate.tst

CVS モニタのデモに使用するテスト ケース。特定の cid でランダムに失敗します。

monitorRunBase.stg

1 人のユーザ、1 サイクル、100% の反応時間が設定され、CAI が無効で、最長実行時間が設定されていないステージング ドキュメント。

monitorRunMultiple.stg

1 人のユーザ、継続的な実行、136% の反応時間が設定され、CAI が有効で、最長実行時間が 15 秒のステージング ドキュメント。

monitorSLARun.stg

1 人のユーザ、1 サイクル、100% の反応時間が設定され、CAI が無効で、最長実行時間が設定されていないステージング ドキュメント。 JMX および JBOSS メトリックを記録するよう選択されています。

selenium.capabilities.conf

Selenium テスト ケースのオプションをカスタマイズするために設定できる Selenium Web ドライバ パラメータを持つサンプル スケルトン パラメータ ファイル。

serviceValidator.tst

CVS モニタのデモに使用。 特定の cid でランダムに失敗します。

userAddDelete.tst

このテストは CVS のデモのモニタ設定に使用します。

Setup

Examples ディレクトリにある Setup ディレクトリには、すべての DevTest コンポーネントを起動し、すべての DevTest コンポーネントを停止し、CVS モニタをロードするためのバッチ ファイルが含まれています。

アクセス制御（ACL）を有効にしてスクリプトを使用するには、スクリプトの Service Manager コマンドにユーザ名とパスワードのオプションを追加します。 パスワードは自動的に暗号化されないため、オペレーティング システムに応じた適切な方法を使用して、必ずファイルを保護してください。

StagingDocs (P. 251)

1User0Think_RunContinuously

このステージング ドキュメントでは、反応時間がゼロの単一の仮想ユーザを実行します。 また、このステージング ドキュメントはテストを「継続的」に実行します。これは必ずしも「永久」を意味するものではありません。

このステージング ドキュメントで実行するテストが以下の条件のすべてを満たす場合、データが使い果たされると、実行が終了します。

- データ セットがテストの最初のステップにある。
- データ セットの「データの終了」ステップが「テストを終了する」である
- データ セットが「ローカル」でなく「グローバル」である

複数のデータ セットがこれらの条件に一致する場合は、期限切れになる最初のデータ セットが実行を完了します。 適切な例については、multi-tier-combo.tst テスト ケースを参照してください。

1user1cycle0think

このステージング ドキュメントは、ユーザが 1 人のテストを 0% の反応時間スケールで 1 回実行します。

ip-spoofing

DevTest インストールで IP スプーフィングのサポートをテストするには、このステージング ドキュメントを使用します。 このステージング ドキュメントの IP スプーフィングは、［IP スプーフィング］タブで有効にします。 サンプル サーバを実行している場合は、IP スプーフィング テスト用の Web ページを http://localhost:8080/ip-spoof-test で使用できます。

jboss-jmx-metrics-localhost

このステージング ドキュメントは、3 人のユーザ、継続的な実行、最長実行時間が 440 秒、および 100% の反応時間のテストを実行します。 このドキュメントでは、レポート ジェネレータのパラメータ チェック ボックスが 4 つすべてオンになっており、JBOSS JMX メトリックをすべて収集するよう指定されています。

Run1User1Cycle

このステージング ドキュメントは、ユーザが 1 人のテストを 100% の反応時間スケールで 1 回実行します。

Run1User1CyclePF

このステージング ドキュメントは、ユーザが 1 人のテストを 100% の反応時間スケールで、CAI が有効な状態で 1 回実行します。

Run1User1CycleShowAll

このステージング ドキュメントは、ユーザが 1 人のテストを 100% の反応時間スケールで 1 回実行します。また、デフォルトのレポート ジェネレータの 4 つのチェック ボックスをすべてオンにします。したがって、Web ベースのモデル実行ページにより多くの項目が表示されます。

Subprocesses

ws-sec-sub

このワンステップ テスト ケースはサブプロセスとしてマークされ、任意のサブプロセスの実行ステップから呼び出すことができます。

Suites (P. 287)

AllTestsSuite

AllTestsSuite は、1user1cycle ステージング ドキュメントおよびデフォルトの監査ドキュメント AuditDocs/DefaultAudit.aud を使用して、DevTest Tests ディレクトリにあるすべてのテストを実行します。レポート メトリックについては、要求と応答のみを記録し、デフォルト メトリックを作成します。 実行モードはシリアルです。

FastAllTestsSuite

AllTestsSuite は、1user1cycle0think ステージング ドキュメントおよびデフォルトの監査ドキュメント AuditDocs/DefaultAudit.aud を使用して、DevTest Tests ディレクトリにあるすべてのテストを実行します。レポート メトリックについては、要求と応答のみを記録し、デフォルト メトリックを作成します。 実行モードはパラレルです。

Tests (P. 85)

AccountControlMDB

LISA Bank に口座を持つユーザを追加する簡単な JMS テスト。 2 つのステップからの応答のパターンを予想し、アサートします。

async-consumer-jms

テスト ケースが応答キュー/トピックから継続的にメッセージを受信する「async consumer」キューのサンプル。 また、このキューは、メッセージを、到達した順にテスト ケースで使用可能にします。

最初のステップでキュー（テストの内部）を作成します。

2 番目のステップで、デモ サーバの JMS キューへ 3 つのメッセージを送信します。 async キューがメッセージを受信します。

3 番目のステップでは、async キューが 3 つのメッセージを受信したことを検証します。

ejb3EJBTest

LISA Bank の機能についての純粋な EJB テスト。 通常、Web ブラウザまたはその他の UI を記録することでアプリケーションをテストします。 これらのテストは「エンド ツー エンド」の統合テストです。このようなテストは「食物連鎖の下位」に相当するため、（コードを記述する必要はないものの）高度な技術を持つ作成者が必要です。

このようなテストを使用することにより、開発チームは、ユーザ インターフェースが存在しないか頻繁に変更されるためにテストに間に合わないような場合に、ユーザ インターフェースなしで継続的にテストを実行してコードを検証できます。

ejb3WSTest

このモデルは LISA Bank Web サービスを徹底的にテストします。このモデルは、ejb3EJB テストと機能的にほとんど同一で、同じ目的で使用できます（テスト ケース ドキュメントを参照）。

ip-spoofing

このサンプル テスト ケースは、DevTest の IP スプーフィングのサポートを示します。

このテストは、REST ステップを使用して「http://localhost:8080/ip-spoof-test」という URL を要求します。これは要求するクライアントの IP アドレスが含まれる Web ページです。 その後、次の URL に SOAP 要求を送信します。これは、要求するクライアントの IP アドレスを返す操作が含まれる Web サービスを識別します。

```
http://localhost:8080/itko-examples/ip-spoof-test/webservice
```

このテスト ケースは、両方の要求をループで 10 回実行します。このテストは、IP スプーフィング テストのステージング ドキュメント「ip-spoofing.stg」でステージングできます。 正しいネットワーク インターフェース設定を使用して、仮想ユーザの HTTP 要求および SOAP 応答で使用されているさまざまな IP アドレスを確認します。

jms

このテスト ケースはシンプルな JMS サンプルで、XML/テキスト メッセージおよびオブジェクトをネイティブ Java 形式で送信する方法を示しています。

Lisa_config_info

このテスト ケースは、DevTest を実行しているコンピュータに関する診断情報を取得します。 この結果は、サポートが設定の問題を解決するのに役立ちます。

load-csv-dataset-web

このテスト モデルは、Web アプリケーションをテストするためのデータ セットとしてカンマ区切り値 (CSV) ファイルを使用します。DevTest に付属しているデモ版の Web アプリケーションでは、データベースに対するユーザの追加および削除を行えます。

log-watcher

このサンプルでは、ログ ファイルの ERROR または WARNING メッセージを監視して、テストに失敗した原因を示します。

このサンプルでは、データ セットを使用して、サンプルの AntiPattern ビーンに除算のための数値を 2 つ与えています。 データセットのほぼ中間で、オペランドとして 0 を与えます。 このオペランドによって、divide-by-zero 例外がサーバで発生します。 AntiPattern ビーンはこの例外をログに記録し、結果として -1 を返します。

このサンプルは一般的なアンチパターンです。内部エラーが発生したものの、外部ではその結果が正しくないことがわからないというものです。 結果は信頼できるように見えますが、正しくありません。予期される結果は、EJB からコール元への例外の伝達です。

CAI を使用すると、ログに例外が記録されたという事実をエージェントが記録するため、このテスト ケースは失敗します。 したがって、DevTest では何か問題があると判断します。

CAI を使用する以外の方法としては、サーバのログ ファイルを監視するグローバル アサーションを設定することです。 正規表現で構成されるものを定義します。この場合は単純に「ERROR」のテストです。正規表現は、必要に応じて単純にも複雑にも指定できます。

通常は、テストがただちに失敗するようアサーションを設定します。 この場合、「Error detected in log file」ステップに移動し、テストを正常終了させます。

DevTest では、テスト中のアプリケーションによってエラーまたは警告がログに記録される場合、それらのアプリケーションが渡されないと想定されています。 テスト ケースでは、同等のコンパニオンをデフォルトで使用することを検討してください。

main_all_should_fail

このテストは、ネガティブ テストのサンプルです。テスト ステップが失敗することを想定しているため、エラーを引き起こすことが予期されるサービス データを与えます。

このテストにはコンパニオンが 1 つ（NegativeTestingCcompanion）があり、いずれかのステップが成功するとテストが失敗します。

このケースでは、既存のデモ サーバにユーザを作成します。 このデータはデータベース自体から取得します（ユーザ名データ セットはテーブルに直接クエリ）。いずれかのステップが成功すると、テスト全体が失敗します。

成功する唯一のステップは「quietly succeed」ステップです。 このステップにより、この種のシナリオで失敗が予期されるステップにエディタで「quiet」とマークして、NegativeTestingCompanion に含まれないようにすることができます。

main_all_should_pass

このテストはサブプロセスを呼び出して、デモ サーバの USERS テーブルに一意のユーザ名を挿入します。

このデータセットは、データ シートを介して一意コード ジェネレータから値を取得します。同じことは、「カウンタ」データ セットを持つ一意コード ジェネレータも実行できました。 このサンプルは、あるデータ セットが別のデータ セットにどのように影響を及ぼすかを示します。

データ セットは指定された順に評価されます。 ステップが実行されるたびに、UniqueUser プロパティに新しい値が割り当てられます。 データ シートは、{{UniqueUser}} を 4 回参照するため、一意の値が 5 つ得られます。

いずれかのステップで失敗すると、ただちにテストが失敗します。

このテストを「main_all_should_fail」と比較してください。「main_all_should_fail」は各ステップが失敗することを想定しており、いずれかのステップが成功すると失敗する同様のテストです。このプロセスはネガティブ テストと呼ばれています。

multi-tier-combo

multi-tier-combo テストでは、さまざまなサービス エンドポイントを使用して LisaBank のサンプルを検証します。 SOAP、EJB、JMS、Selenium、および Web トランザクションをテストし、デモ サーバのデータベースを直接検証するなどのさまざまな方法でこれらのトランザクションを検証します。このテストでは、Selenium ステップを実行するために Firefox ブラウザが必要です。

multi-tier-combo テストでは、さまざまなサービス エンドポイントを使用して LISA Bank のサンプルを検証します。このテスト ケースでは、SOAP、EJB、JMS、および Web トランザクションをテストし、デモ サーバのデータベースを直接検証するなどのいくつかの方法でこれらのトランザクションを検証します。

また、このテストは、スプレッドシートから複雑な SOAP オブジェクトを作成する方法を示します。プロジェクトの Data フォルダにある「**multi-tier-users.xls**」という名前のスプレッドシートは、最初のステップにある「**User**」データ セットに値を返します。

このテストを［対話型テスト ラン （ITR）］ウィンドウで実行すると、スプレッドシートの最初の行から 1 ユーザを作成した後、テストは終了します。

サンプルの **1User0Think_RunContinuously** ステージング ドキュメントを使用してテストをステージングすると、データ セットの末尾に到達するまでテストが再起動されます。 このプロセスは大きなデータ セットを繰り返し反復する場合に適しています。 テスト ケースにループを使用することもできますが、柔軟性がなくなります。

ステージング ドキュメントでデータ セットを制御してテストを終了させると、多数の仮想ユーザにテストを広げることができます。 また、反応時間やその他のパラメータを使用して、テストのペースを制御できます。

テストの最初のステップで設定されるグローバル データ セットのみが、ステージング ドキュメントが「継続的なテスト実行を終了」する動作に影響を与えます。 データ セットがテストに対してローカルか、テスト内のどこかで宣言されている場合は、「継続的な実行」の動作は実際には「永久に実行」を意味します。

multi-tier-combo-se

multi-tier-combo-se テストでは、さまざまなサービス エンドポイントを使用して LISA Bank のサンプルを検証します。SOAP、EJB、JMS、および Web トランザクションをテストし、デモ サーバのデータベースを直接検証するなどのさまざまな方法でこれらのトランザクションを検証します。

また、このテストは、スプレッドシートから複雑な SOAP オブジェクトを作成する方法を示します。 最初のステップにある **User** データ セットは、プロジェクトの **Data** フォルダにある **multi-tier-users.xls** という名前のスプレッドシートがベースになっています。

このテストを［対話型テスト ラン （ITR）］ウィンドウで実行すると、スプレッドシートの最初の行から 1 ユーザを作成した後、テストは終了します。

サンプルの **1User0Think_RunContinuously** ステージング ドキュメントを使用してテストをステージングすると、データ セットの末尾に到達するまでテストが再起動されます。 この方法は大きなデータ セットを繰り返し反復する場合に適しています。 テスト ケースにループを使用することもできますが、柔軟性が大幅に低下します。

ステージング ドキュメントでデータ セットを制御してテストを終了させると、多数の仮想ユーザにテストを広げたり、反応時間を使用してテストのペースを制御したりすることができます。

ステージング ドキュメントが「継続的なテスト実行を終了」する動作は、テストの最初のステップで設定されるグローバル データ セットによってのみ影響を受けることに注意してください。 データ セットがテストに対してローカルか、テスト内のどこかで宣言されている場合は、「継続的な実行」の動作は実際には「永久に実行」を意味します。

rawSoap

rawSoap ステップは、「listUsers」ステップで簡単な RAW SOAP 要求を実行するワンステップのテスト ケースです。

rest-example

rest-example テストは、RESTful サービスを実行する方法を示します。デモ サーバには JAX-RS のサンプルが含まれています。 このテストの各ステップでは、XML と JSON の両方を使用するサービスと対話する方法を示します。

scripting

CA Application Test は、JSR-223 スクリプティング エンジンを利用できるため、スクリプト ステップ、アサーション、データ プロトコル ハンドラ、一致スクリプト、およびほとんどすべての場所で、{{=%language%}} 構文により、さまざまなスクリプティング エンジンを使用することが可能です。

sendJMSrr

要求/応答ペアから JMS のサービスを作成する VSEasy の機能をテストします。

service-validation

このテストは、簡単なサービス検証のサンプルです。 このテストでは、1 つの Web サービスと 1 つの EJB サービスを呼び出し、基盤となるデータベースを SQL を使用して検査して、サービスが適切に機能していることを検証します。

web-application

このテストは、Web レコーダを使用して生成された簡単な Web テストです。 このテストには、「空でない応答のアサート」などいくつかの基本的なアサーションが含まれます。これは自動的に生成されます。また、このテストには、HTML 応答の <title> タグを解析して作成される「タイトルのアサート」アサーションが含まれます。 これらのアサーションは、テストを再生する場合に、記録したページが同じページであることを確認するのに役立ちます。

webservices

このテスト ケースは、EJB3 Web サービスからユーザを追加、取得、削除します。このテストは、一意コード ジェネレータを使用して、設定ファイルから値が {{user}} であるプレフィックスを持つ数値を作成します。 パスワードは、設定ファイル内にハードコードされています。

ws_attachments

このテスト ケースでは、インラインの base64 エンコードされた BLOB データおよび XOP/MTOM 添付ファイルを送受信する機能をテストします。 ステップのフィルタおよびアサーションでは、要求および応答が正しいことを確認します。

ws_security

ws_security テスト ケースは、署名および暗号化された SOAP メッセージを使用する方法を示します。 最初の 2 つのステップは成功し、最後の 2 つのステップは失敗します。 コールは同じですが、Web サービスは暗号化または署名されていないメッセージを許可しません。

ws_security-xml

このテスト モデルは、署名および暗号化された SOAP メッセージを使用する方法を示します。 最初の 2 つのステップは成功し、最後の 2 つのステップは失敗します （コールは同じですが、Web サービスは暗号化されていないメッセージまたは署名されたメッセージを許可しません）。

このテスト プランでは、Java 環境が無制限強度の暗号化をサポートしている必要があります。 最初の 2 つのステップのどちらかが失敗する場合は、高い確率で、JCE jar を更新して無制限強度の暗号化を有効にする必要があります。 JCE jar は、www.oracle.com からダウンロードできます。 「JCE unlimited strength」というキーワードで検索し、お使いの Java バージョンに適した JCE ライブラリ（Java 7 用の JCE 7 など）を選択してください。JCE jar をインストールしたら、DevTest サービスを再起動する必要があります。

監査ドキュメントに 2 つの ステップ エラーが記録されることが予期されます。 ここで、最後の 2 つのステップの両方でステップ エラーが発生することが予期されるため、2 つのステップ エラーについて監査します。 このように、監査ドキュメントは、受信されたイベント発生の数を監査することもできます。 3 つ目のステップ エラー エントリを監査ドキュメントに追加すると、ステップ エラーが 2 つしか発生しないため、監査では、このテストが失敗します。

TestsThatFail

webservices-xml-fail

このテスト ケースは、EJB3 Web サービスからユーザを追加、取得、削除します。このテストは、一意コード ジェネレータを使用して、設定ファイルから値が {{user}} であるプレフィックスを持つ数値を作成します。 パスワードは、設定ファイル内にハードコードされています。

VServices

statefullATM.tst

statelessATM.tst

web-app-proxy.tst

DatabaseModel.vsm

kioskV4model.vsm

kioskV5.vsm

kioskV6.vsm

WebServicesModel.vsm

webservices-vs.vsm

Images

DatabaseModel.vsi

kioskV4ServiceImage.vsi

kioskV6.vsi

si-kioskV5-dynamic.vsi

si-kioskV5.vsi

WebServicesModel.vsm

プロジェクトの作成

プロジェクトは、新規に作成または既存のモデル アーカイブ（MAR）ファイルから作成できます。

注: プロジェクトは、VSE MAR ファイルからのみ作成できます。 このダイアログ ボックスでは、MAR を既存のプロジェクトに追加できません。

次の手順に従ってください:

1. メイン メニューから［ファイル］-［新規］-［プロジェクト］を選択します。

   ［新規 DevTest プロジェクトの作成］ダイアログ ボックスが表示されます。

2. ［プロジェクト名］フィールドにプロジェクトの名前を入力します。
3. **LISA_HOME¥Projects** ディレクトリ以外の場所にプロジェクトを作成するには、［プロジェクト名］フィールドの横にあるアイコンをクリックして場所を指定します。
4. 既存の MAR ファイルからプロジェクトを作成するには、以下の手順に従います。
   a. ［以下のいずれかに基づいて新しいプロジェクトを作成します］チェック ボックスをオンにします。
   b. ［MAR ファイル］オプションを選択します。
   c. MAR ファイル名およびディレクトリの場所を指定します。
5. ［Create］をクリックします。

プロジェクトを開く

プロジェクトは、DevTest ワークステーション またはコマンドラインから開くことができます。

DevTest ワークステーション からプロジェクトを開く方法

1. メイン メニューから［ファイル］-［開く］-［プロジェクト］を選択します。

   いくつかのプロジェクトをすでに開いたことがある場合は、最近開かれたプロジェクトのリストが表示され、そこから選択することができます。 選択すると、そのプロジェクトが開きます。

2. プロジェクト ファイルを参照するには、選択リストから［ファイル システム］を選択します。［プロジェクトを開く］ウィンドウが表示されます。

3. プロジェクトを選択し、［開く］をクリックします。

コマンドラインからプロジェクトを開く方法

1. **LISA_HOME¥bin** ディレクトリに移動します。

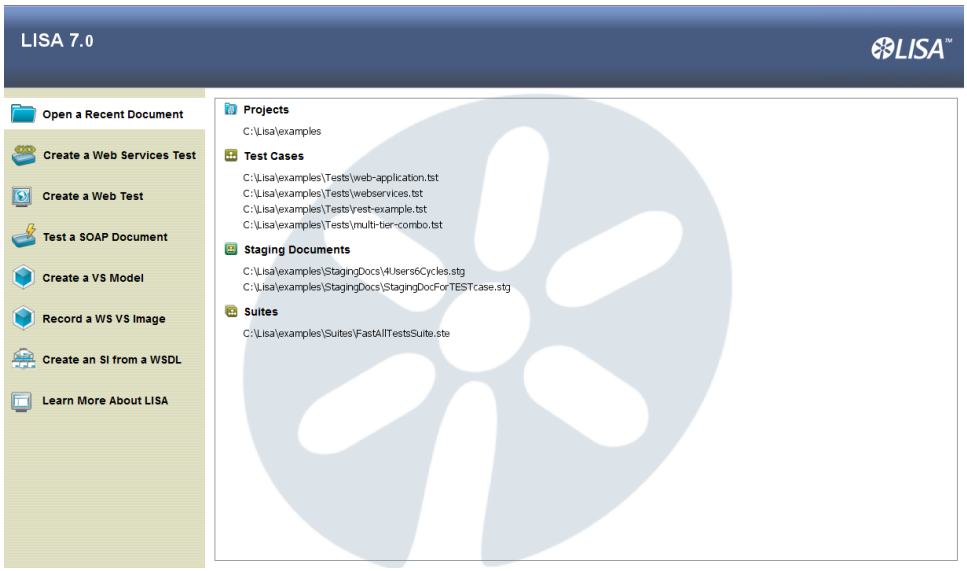
2. DevTest ワークステーション 実行可能ファイルを実行し、引数としてプロジェクト ディレクトリを指定します。 プロジェクト ディレクトリは絶対パスまたは相対パスで指定できます。 以下の例では絶対パスを使用しています。

   ```
   LISAWorkstation "C:¥Program Files¥CA¥Lisa¥Projects¥Project1"
   ```

[クイック スタート]ウィンドウ

［クイック スタート］ウィンドウは、DevTest ワークステーション を開いたときに最初に表示されるウィンドウです。

注:［クイック スタート］ウィンドウは、その他のタブが開かれた後もタブとして常に使用できます。



［クイック スタート］ウィンドウには、DevTest ワークステーション の最も有用なオプションの一部が表示されます。 オプションをクリックすると、そのパラメータがウィンドウの右側に表示されます。

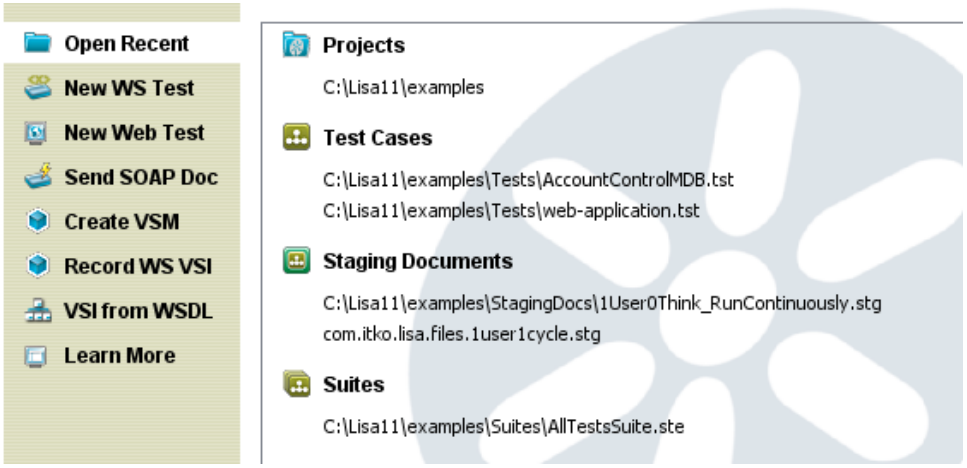
このウィンドウでは、以下のオプションを選択できます。 お使いの画面の解像度に応じて、簡潔なメニュー項目か詳細なメニュー項目のいずれかが表示されます。 すべてのメニュー オプションを表示するには、ウィンドウの下部にある下向き矢印をクリックします。

- ［最近の使用履歴］または［最近使用したドキュメントを開く］
- ［新規 WS テスト］または［Web サービス テストの作成］
- ［新規 Web テスト］または［Web テストの作成］
- ［SOAP Doc 送信］または［SOAP ドキュメントのテスト］
- ［VSM の作成］または［VS モデルの作成］
- ［WS VSI の記録］または［WS VS イメージの記録］
- ［WSDL による VSI］または［WSDL を使用した SI の作成］

- ［関連情報］または［DevTest の関連情報］

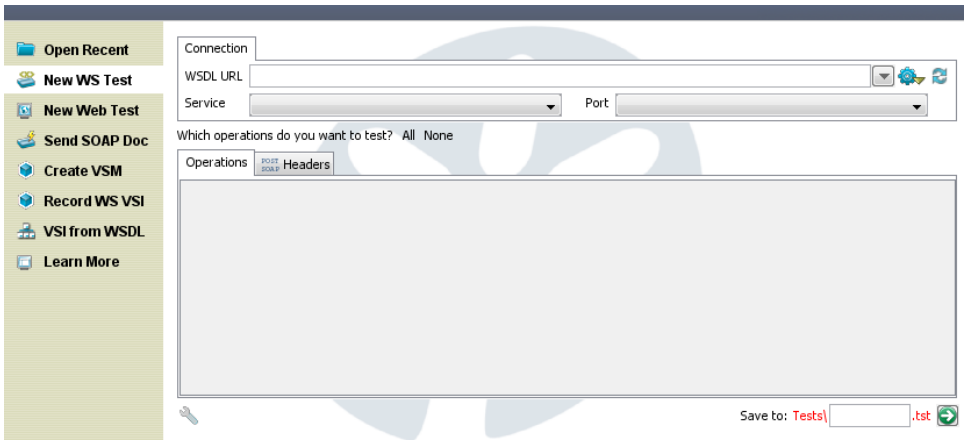
最近の使用履歴

DevTest ワークステーション が開くと、デフォルトではこのオプションが選択されます。 右ペインには、最近開いたプロジェクト、テスト ケースおよびスイート、ステージング ドキュメント、VS モデルまたは VS イメージ（該当する場合）が表示されます。



新規 WS テスト

［クイック スタート］ウィンドウの［新規 WS テスト］オプションでは、Web サービス テスト ケースを作成できます。 必要なパラメータは右ペインに表示されます。



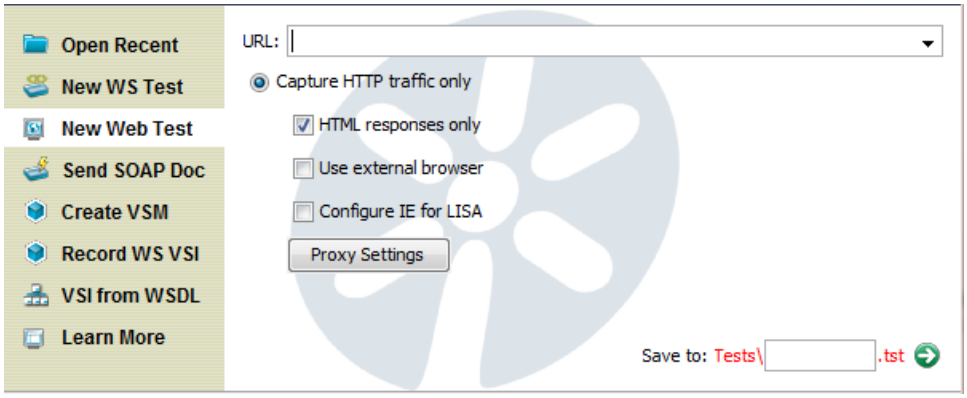
次の手順に従ってください:

1. ［WSDL URL］、［サービス］、および［ポート］の詳細を入力します。
2. ［すべて］または［なし］からテストする操作を選択するか、または各項目を個別に選択します。
3. 右ペインで、テストの名前を入力し、テストを保存したいプロジェクト フォルダ内の場所のパスを選択します。パスを選択すると、パスが［パス］フィールドに表示されます。 このペインではどのフォルダも右クリックでき、フォルダの作成、名前変更、削除を実行できます。
4. テスト ケースを作成するには、緑の矢印 をクリックします。

詳細については、「*CA Application Test の使用*」の「Web サービスの生成 (P. 479)」を参照してください。

新規 Web テスト

［クイック スタート］ウィンドウの［新規 Web テスト］オプションでは、Web テストを作成できます。



次の手順に従ってください:

1. Web テストの URL を入力します。
2. ［HTTP トラフィックのみキャプチャ］を選択します。
3. チェック ボックスでオプションを選択します。

   HTML 応答のみ

   HTML 応答のみをキャプチャします。

   外部ブラウザを使用

   外部ブラウザ ウィンドウを開きます。

   IE を DevTest 用に設定

   Internet Explorer を DevTest 用に設定します。
4. 右ペインで、テストの名前を入力し、テストを保存したいプロジェクト フォルダ内の場所のパスを選択します。パスを選択すると、パスが［パス］フィールドに表示されます。 このペインではどのフォルダも右クリックでき、フォルダの作成、名前変更、削除を実行できます。
5. テスト ケースを作成するには、緑の矢印 をクリックします。

詳細については、「*CA Application Test の使用*」の「Web サイトの記録 (P. 481)」を参照してください。

SOAP Doc 送信

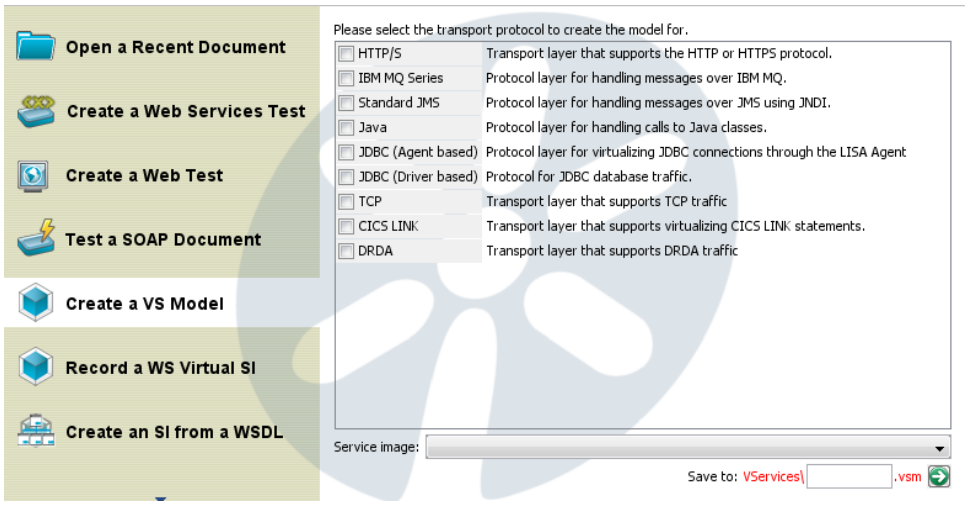
［クイック スタート］ウィンドウの［SOAP Doc 送信］オプションでは、SOAP 要求テスト ケースを作成できます。

次の手順に従ってください:

1. ［SOAP サーバ URL］および［SOAP アクション］に入力します。
2. ［SOAP サーバ URL］フィールドに Web サービス エンドポイントの URL を入力します。
3. ［SOAP アクション］フィールドに、呼び出すメソッドの WSDL の <soap: operation> タグで示されている SOAP アクションを入力します。この値は SOAP 1.1 に必要です。SOAP 1.2 では、多くの場合は空白のままにしておく必要があります。
4. エディタに SOAP 要求を入力または貼り付けます。
5. ［保存先］フィールドにテストの名前を入力します。
6. テスト ケースを作成するには、緑の矢印 をクリックします。

VSM の作成

［クイック スタート］ウィンドウの［VSM の作成］オプションでは、仮想サービス モデルおよび対応する仮想サービス イメージを作成できます。



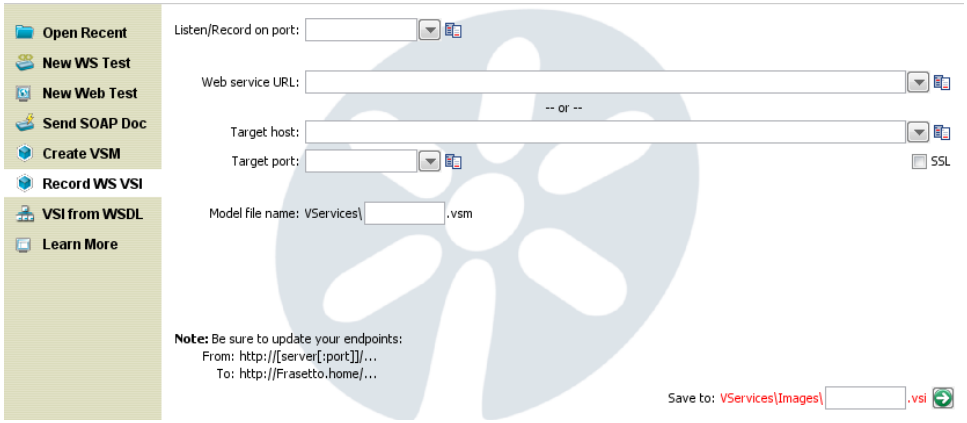
次の手順に従ってください:

1. 使用可能なプロトコルのリストからトランスポート プロトコルを選択します。 この選択内容によって、ウィンドウの次のシーケンスが変わります。
2. サービス イメージの名前を入力します
3. VSM の名前を入力し、それを保存したいプロジェクト フォルダ内の場所のパスを選択します。 パスを選択すると、パスが［パス］フィールドに表示されます。
4. テスト ケースを作成するには、緑の矢印 をクリックします。

VSM および VSI を作成するためのウィンドウおよびオプションの詳細については、「サービス イメージの作成」を参照してください。

WS VSI の記録

［クイック スタート］ウィンドウの［WS VSI の記録］オプションでは、Web サービスの仮想サービス イメージを作成できます。 必要なパラメータは右ペインに表示されます。

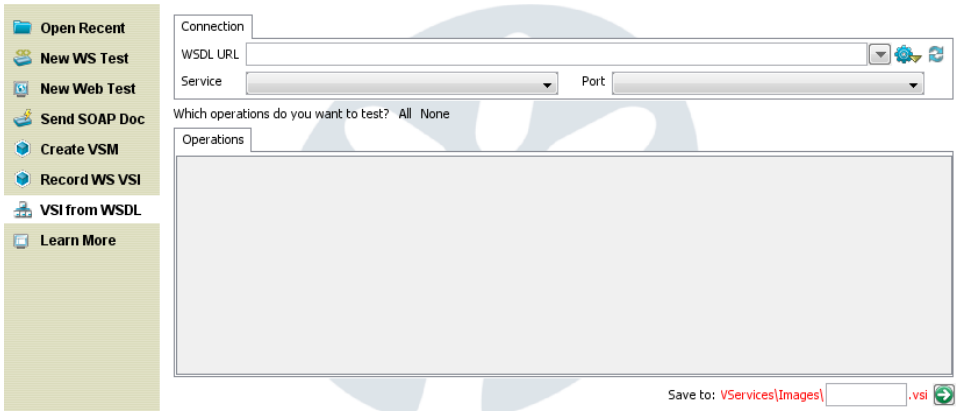


次の手順に従ってください:

1. リスンおよび記録するポートを入力します。
2. 記録する Web サービスの URL を入力します。
3. 記録するターゲット ホストおよびポートを入力します。
4. このサービス イメージで SSL を使用するかどうかを指定するには、SSL チェック ボックスを使用します。
5. VSM および SI の名前を入力します。
6. テスト ケースを作成するには、緑の矢印 をクリックします。

WSDL による VSI

［クイック スタート］ウィンドウの［WSDL による VSI］オプションでは、WSDL から仮想サービス イメージを作成できます。必要なプロパティは右ペインに表示されます。



次の手順に従ってください:

1. ［WSDL URL］、［サービス］、および［ポート］の詳細を入力します。入力する［WSDL URL］でパスにプロパティが含まれていない場合は、［サービス］および［ポート］は自動的に入力されます。
2. テストする操作を選択するには、［すべて］または［なし］をクリックするか、または各操作を個別に選択します。
3. サービス イメージの名前を入力し、それを保存するプロジェクト フォルダ内のパスを選択します。
4. テスト ケースを作成するには、緑の矢印 をクリックします。

関連情報

［クイック スタート］ウィンドウの［関連情報］オプションには、ドキュメント、オンライン サポート、および DevTest グローバル コミュニティへのリンクが表示されます。

DevTest の使用可能なすべてのユーザ ドキュメントはこのメニューからアクセスできます。

DevTest ワークステーション メモリ メータ

DevTest ワークステーション のメイン ウィンドウの右下隅にある DevTest ワークステーション メモリ メータには、メモリ使用量と空き容量に関するランタイム情報が表示されます。

- ガベージ コレクションを実行するには、メモリ メータをクリックします。
- ヒープ ダンプ ファイル（HProf）を生成するには、メモリ メータを右クリックします。［ヒープをダンプ］オプションを選択します。

  注: この機能は、CA サポートがメモリ不足状態の原因を特定するうえで役立つ診断ツールです。 メモリ不足状態が発生すると、ヒープが自動的にダンプされます。このオプションは、ヒープ ダンプを手動でトリガするためのものです。

  ダンプが作成された後、ヒープ ダンプが取得されたことを示すメッセージが表示されます。

トレイ パネル

DevTest ワークステーション には、ログ メッセージの出力ステップなどの特定の機能のためのトレイ パネルがあります。

トレイ パネルの開閉操作にアニメーションを使用するかどうかを制御できます。デフォルトでは、リモート デスクトップを使用して DevTest ワークステーション にアクセスするユーザのパフォーマンスを向上させるために、アニメーションは無効になっています。

アニメーションを有効にするには、**site.properties** または **local.properties** ファイルに以下の行を追加します。

```
lisa.ui.tray.animation=true
```

DevTest Console

Web ベースの DevTest コンソールから、以下のコンソールやダッシュボードにアクセスできます。

- レポート ダッシュボード
- 継続的検証サービス
- サーバ コンソール
- VSEasy

レポート ダッシュボード

レポート ビューアには、イベントおよびメトリック情報、テストの実行中にキャプチャされたデータから派生した情報が表示されます。 レポートの目的でキャプチャするイベントおよびメトリックを設定するには、ステージング ドキュメントを使用できます。

詳細については、「*CA Application Test の使用*」の「レポート (P. 435)」を参照してください。

継続的検証サービス

継続的検証サービス（CVS）では、長期間にわたって定期的に実行するテストおよびテスト スイートをスケジュールできます。

詳細については、「*CA Application Test の使用*」の「継続的検証サービス（CVS） (P. 417)」を参照してください。

サーバ コンソール

サーバ コンソールでは、ラボを管理し、ロール ベースのアクセス制御を設定できます。 また、サーバ コンソールは VSE ダッシュボードにアクセスする場所です。

VSEasy

VSEasy では、HTTP サービス イメージおよび VRS ファイルからのサービス イメージを作成できます。 VSEasy を使用して、SOAP、JSON、XML、または HTML ペイロードで HTTP のステートレス仮想サービスを作成できます。

詳細については、「*CA Application Test の使用*」の「クラウドの DevTest ラボ (P. 393)」、「管理」の「アクセス制御（ACL）」、および「CA Service Virtualization の使用」の「VSE ダッシュボード」、「VSEasy を使用した仮想サービスの作成と展開」を参照してください。

DevTest コンソールを開く

DevTest コンソールは、DevTest ワークステーション のメイン メニューまたは Web ブラウザから開くことができます。

LISA コンソールのホーム ページから、レポート ダッシュボード、CVS ダッシュボード、サーバ コンソール、および VSEasy にアクセスできます。 このホーム ページには、DevTest ポータルの CAI 部分へのリンクも含まれています。 ホーム ページの右下の領域には、バージョン番号が表示されます。

DevTest ワークステーション のメイン メニューから DevTest コンソールを開く方法

1. メイン メニューから［表示］-［DevTest ポータル］を選択します。

   ログイン ダイアログ ボックスが表示されます。

2. ユーザ名とパスワードを入力して、［ログイン］をクリックします。

Web ブラウザから DevTest コンソールを開く方法

1. レジストリ (P. 11)が実行されていることを確認します。
2. Web ブラウザに「**http://localhost:1505/**」と入力します。 レジストリがリモート コンピュータ上にある場合は、**localhost** をそのコンピュータの名前または IP アドレスに置き換えます。

   ログイン ダイアログ ボックスが表示されます。

3. ユーザ名とパスワードを入力して、［ログイン］をクリックします。

Web サーバのタイムアウト

デフォルトでは、DevTest コンソールが使用する Web サーバは、サーバでのプロセスの実行を 90 秒間待機します。プロセスが 90 秒より長くかかると接続は中断され、クライアント アプリケーションまたはブラウザはこの中断を適切に処理します。

デフォルトのタイムアウト値は、**local.properties** ファイルに **lisa.webserver.socket.timeout** プロパティを追加することで変更できます。値はミリ秒単位です。 以下に例を示します。

```
lisa.webserver.socket.timeout=120000
```

第 10 章: テスト ケースの作成

テスト ケースを作成するには、プロパティの設定、設定の使用とプロジェクトへの適用、フィルタの適用、アサーションの追加、データ セットの追加、およびコンパニオンの追加について理解する必要があります。 このセクションでは、複合オブジェクト エディタについても説明します。

このセクションには、以下のトピックが含まれています。

テスト ケースの構造

テスト ケースは、テスト中のシステムのビジネス コンポーネントをテストする方法の仕様です。 テスト ケースは XML ドキュメントとして格納され、特定のコンポーネントまたはシステムをテストするために必要な情報がすべて含まれています。

テスト ケースは、成功および失敗したステップの結果を表すパスで連結されたテスト ステップで記述されたワークフローです。 ステップにはアサーションを指定することができ、アサーションの起動に応じて異なるパスが提供されます。

注: テスト ケースは定期的に保存してください。

以下のトピックが含まれます。

テスト ケースのクイック スタート

テスト ケースの操作を開始する方法

1. レジストリを起動します。

   「*DevTest の使用*」の「DevTest レジストリ (P. 11)」を参照してください。

2. DevTest ワークステーション でプロジェクトを作成するか、開きます。

   「*DevTest の使用*」の「プロジェクト パネル (P. 54)」を参照してください。

3. プロジェクト内にテスト ケースを作成します。

   「*CA Application Test の使用*」の「テスト ケースの作成 (P. 234)」を参照してください。

   テスト ケースは、メイン メニューから［ファイル］-［開く］-［テスト ケース］を選択して、プロジェクトの中または外から開くことができます。

multi-tier-combo.tst テスト ケースを以下の図に示します。multi-tier-combo テスト ケースの詳細については、「*CA Application Test の使用*」の「multi-tier-combo テスト ケース (P. 92)」を参照してください。

DevTest ワークステーション は以下の主な領域（左から右に）に分かれています。

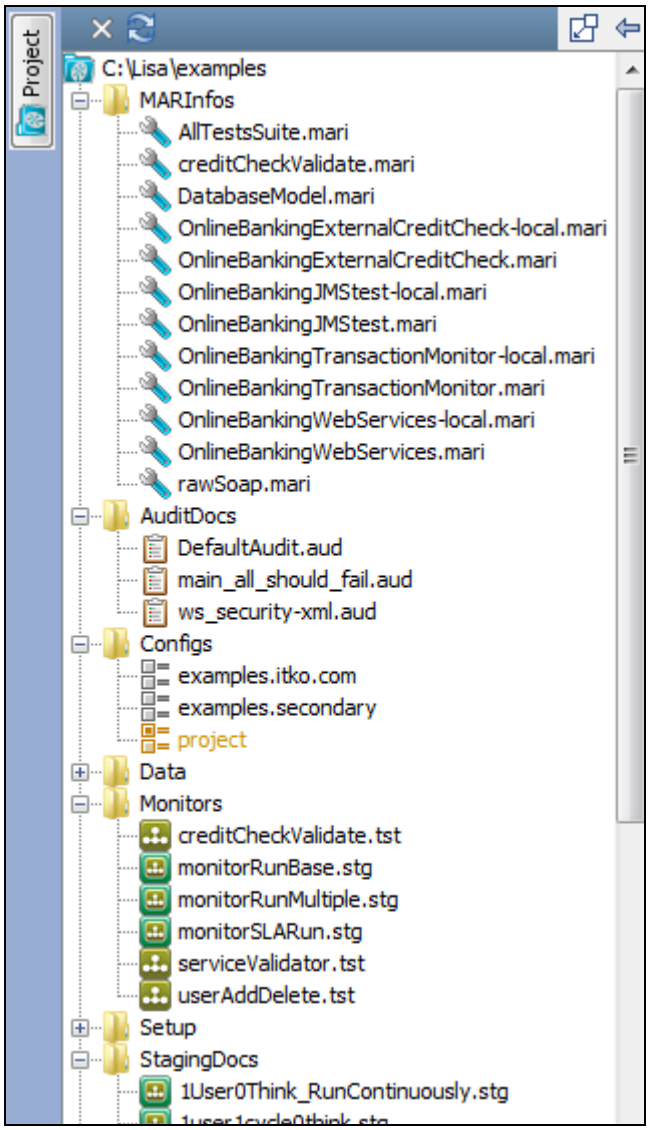
- プロジェクト パネル
- モデル エディタ
- エレメント パネル

プロジェクト パネル

プロジェクト パネルはウィンドウの左の部分にあり、ドッキングが可能です。左側の［プロジェクト］ 

ボタンを使用すると、プロジェクト パネルを開いたり閉じたりすることができます。

DevTest ワークステーション を最初に開いたときには、最後に開いていたプロジェクトがデフォルトで開かれます。

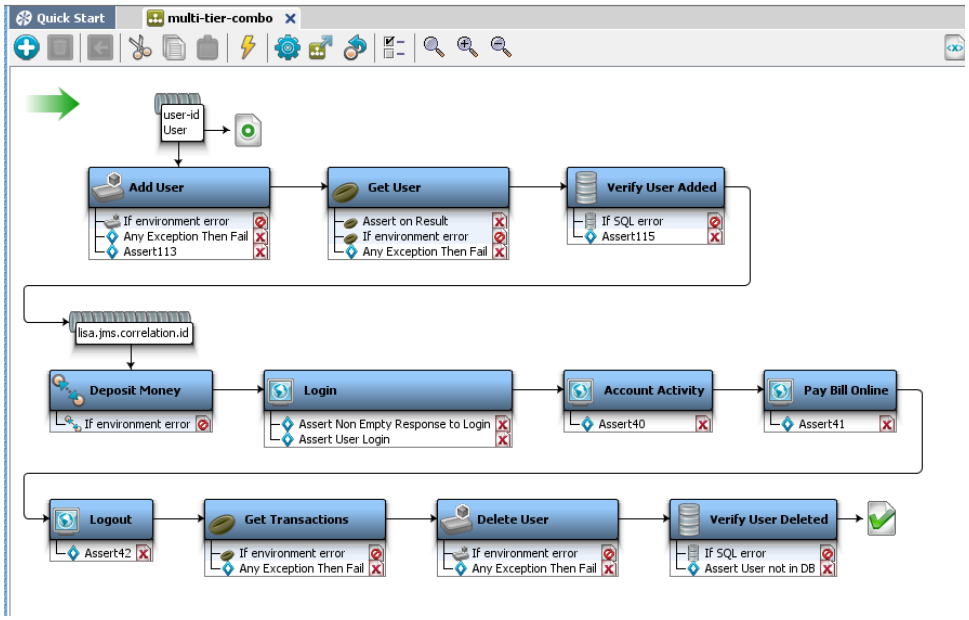


詳細については、「*CA Application Test の使用*」の「プロジェクト パネル (P. 54)」を参照してください。

モデル エディタ

モデル エディタは、テスト ケースのワークフローを作成および表示するために使用します。 テスト ケースのワークフローは、特定のテスト ケースに適用されるすべてのテスト ステップ、フィルタ、およびアサーションで構成されています。

モデル エディタはテスト ケースを作成および表示する場所です。 ウィンドウの中央の部分には、テスト ケースの名前であるタブが表示されます。



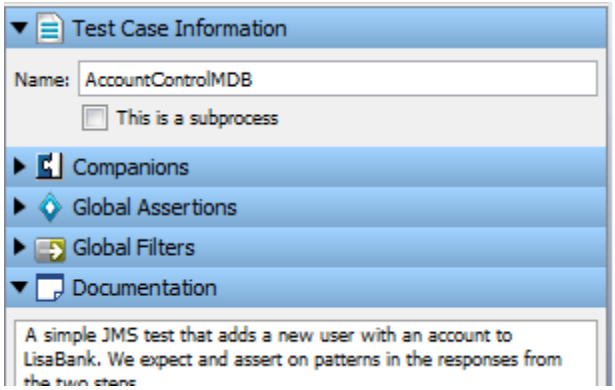
緑色の矢印  は、テスト ケースの開始点を示しています。

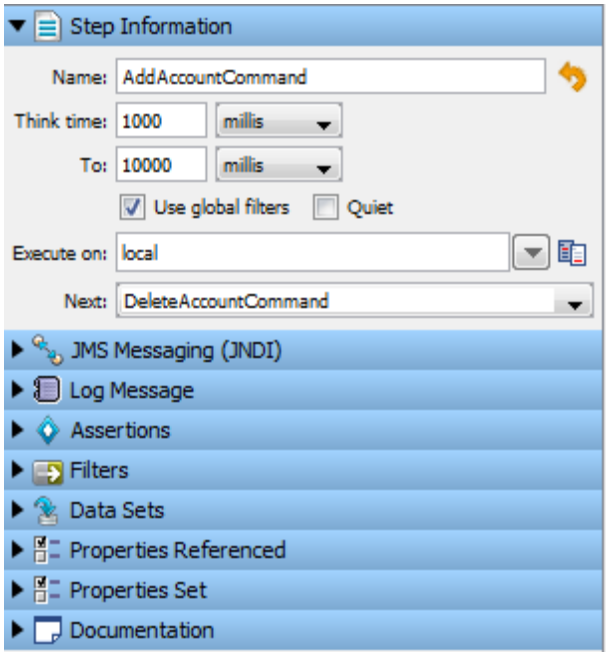
モデル エディタ内のワークフローは、テスト ケースのグラフィカルなビューを表しています。 このビューはテスト ケースのワークフローを視覚的に迅速に検証するのに役立ちます。

モデル エディタのワークフロー内のアイコンは、テスト ケースの各ステップを表しています。 アイコンはテスト ステップのタイプによって異なります。 たとえば、データベース関連のステップの場合は、データベース アイコンおよび関連するフィルタやアサーションがステップに表示されます。

エレメント パネル

エレメント パネルには、テスト ケースまたはテスト ステップに必要なエレメントが表示されます。





デフォルトのステップ名にステップ名を戻すには、名前をデフォルトに戻すボタン をクリックします。

必要なテスト ケースまたはテスト ステップのエレメントをクリックすると、エレメントの追加や削除を行えます。

一部のエレメントはグローバル レベル（テスト ケース全体）で適用できます。 一部のエレメントはステップ レベルで（特定のテスト ステップのみ）適用できます。

テスト ケースの先頭には、必要な情報を入力します。

以下に例を示します。

- テスト ケース情報： テスト ケース名を入力し、このケースがサブプロセスかどうかを選択します。
- ドキュメント： テスト ケースのドキュメントを入力します。

このテスト ケースにステップを追加すると、ステップのワークフローの形成が始まります。 エレメントの新しいセットがエレメント パネルにテスト ステップ レベルで表示されます。

multi-tier-combo テスト ケース

examples プロジェクト (P. 58)には、**multi-tier-combo** という名前のテスト ケースがあります。

このテスト ケースは、さまざまなサービス エンドポイントを使用して、デモ アプリケーションの LISA Bank を検証します。このケースでは、SOAP、EJB、JMS、および Web トランザクションをテストし、デモ サーバのデータベースを直接検証するなどのさまざまな方法でこれらのトランザクションを検証します。

また、このテスト ケースでは、複雑な SOAP オブジェクトをスプレッドシートから作成する方法を示します。 プロジェクトの Data フォルダにある **multi-tier-users.xls** スプレッドシートは、最初のステップにある User データ セットを返します。

このテストを［対話型テスト ラン（ITR） (P. 317)］ウィンドウで実行すると、スプレッドシートの最初の行から 1 ユーザを作成して終了します。

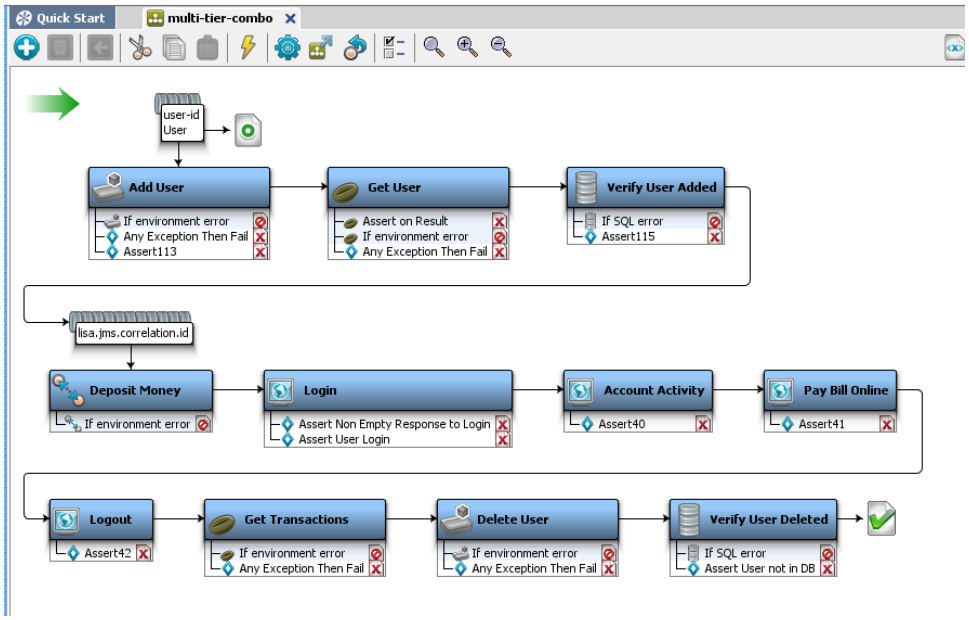
サンプルの **1User0Think_RunContinuously** ステージング ドキュメントを使用してテストをステージングする (P. 340)と、データ セットの末尾に到達するまでテストが再起動されます。 この方法は大きなデータ セットを繰り返し反復する場合に適しています。 テスト ケースにループを使用することもできますが、柔軟性がなくなります。

ステージング ドキュメントでデータ セットを制御してテストを終了させると、多数の仮想ユーザにテストを広げることができます。 また、反応時間などを使用して、テストのペースを制御できます。

テストの最初のステップで設定されるグローバル データ セットのみが、ステージング ドキュメントが「継続的なテスト実行を終了」する動作に影響を与えます。 データ セットがテストに対してローカルか、テスト内のどこかで宣言されている場合は、「継続的な実行」の動作は実際には「永久に実行」を意味します。

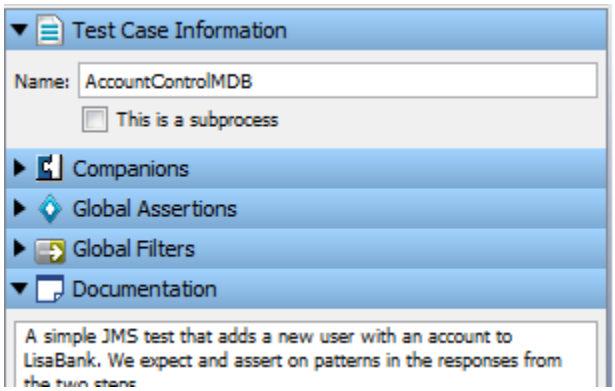
プロジェクト パネルでは **example** プロジェクト フォルダが開かれ、エレメント パネルにはテスト ケース エレメントのセットが表示されることに注目してください。

このとき、モデル エディタ セクションにはテスト ケース情報が表示されます。


Quick Start
multi-tier-combo
user-id
User
Add User
If environment error
Any Exception Then Fail
Assert113
Get User
Assert on Result
If environment error
Any Exception Then Fail
Verify User Added
If SQL error
Assert115
lisa.jms.correlation.id
Deposit Money
If environment error
Login
Assert Non Empty Response to Login
Assert User Login
Account Activity
Assert40
Pay Bill Online
Assert41
Logout
Assert42
Get Transactions
If environment error
Any Exception Then Fail
Delete User
If environment error
Any Exception Then Fail
Verify User Deleted
If SQL error
Assert User not in DB

テスト ケースのエレメント

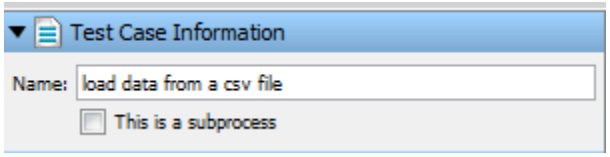
テスト ケースのエレメントは、テスト ケースを作成する際に全面的に役立ちます。以下の図は、DevTest ワークステーション の右側のテスト ケース パネルにテスト ケース エレメントがどのように表示されるかを示しています。



テスト ケース情報

［テスト ケース情報］タブでは、テスト ケースの名前を変更できます。

このタブは、サブプロセスを作成したり、テスト ケースをサブプロセスに変換したりする場合の入力場所としても使用します。 サブプロセスは、スタンドアロンのテストとして実行されるのではなく、別のテスト ケースによってコールされるテスト ケースです。

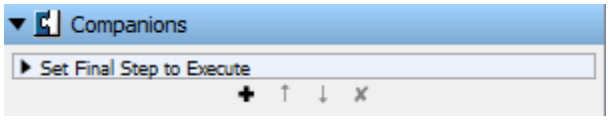


サブプロセスの詳細については、「*CA Application Test の使用*」の「サブプロセスの作成」を参照してください。

コンパニオン

コンパニオンは、すべてのテスト ケースの実行の前後に実行されるエレメントです。 コンパニオンは、テスト ケースのグローバル動作を設定するために使用します。 再起動するとコンパニオンが再度実行されます。

コンパニオン エディタを開くには、エレメント パネルでコンパニオンをダブルクリックします。

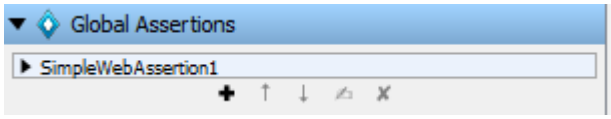


詳細については、「*CA Application Test の使用*」の「コンパニオン (P. 184)」を参照してください。

グローバル アサーション

アサーションは、1 つのステップとそのすべてのフィルタが実行された後に実行されるエレメントです。 アサーションは、ステップの実行結果がユーザの想定に一致することを検証します。 *グローバル アサーション*は、テスト ケース全体に適用されます。

アサーション エディタを開くには、［グローバル アサーション］リストでアサーションをダブルクリックします。

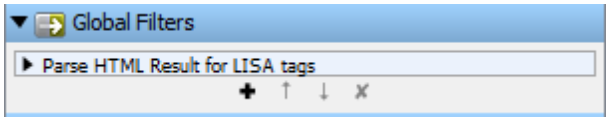


詳細については、「*CA Application Test の使用*」の「アサーション (P. 151)」を参照してください。

グローバル フィルタ

フィルタは、テスト ステップの前後に実行されるエレメントです。 フィルタを使用すると、結果のデータを変更したり、プロパティに値を格納することができます。 *グローバル フィルタ*は、テスト ケース全体に適用されます。

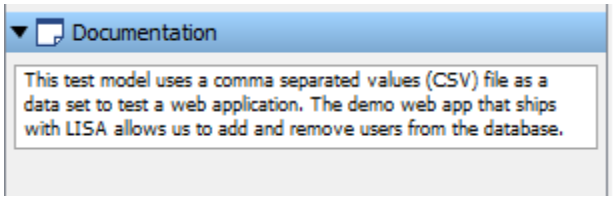
フィルタ エディタを開くには、［グローバル フィルタ］リストでフィルタをダブルクリックします。



詳細については、「*CA Application Test の使用*」の「フィルタ (P. 132)」を参照してください。

Documentation

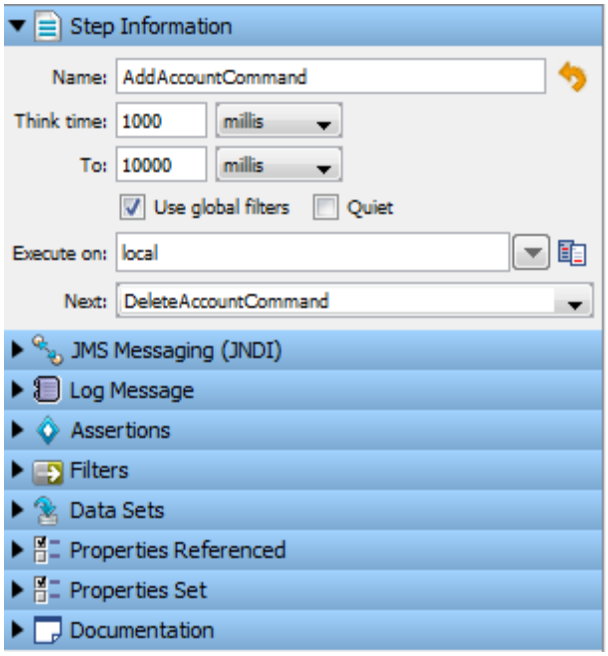
［ドキュメント］領域では、テスト ケースのドキュメントを追加できます。 このテキストはどのプロセスでも使用されませんが便利な場所です。テスト ケースの説明や、このテスト ケースを使用するその他のユーザ向けにメモを記述するのはよい習慣です。



テスト ステップのエレメント

テスト ステップはワークフロー テスト ケース エレメントであり、テスト中のシステムのビジネス機能を検証する基本的なアクションを実行します。 ステップはテスト中のシステムを部分的に呼び出すために使用できます。 通常これらのステップは、モデル エディタでテスト ケースとしてワークフローを作成するために連結されます。 各ステップでは、データを抽出するフィルタを作成したり、応答データを検証するアサーションを作成することができます。

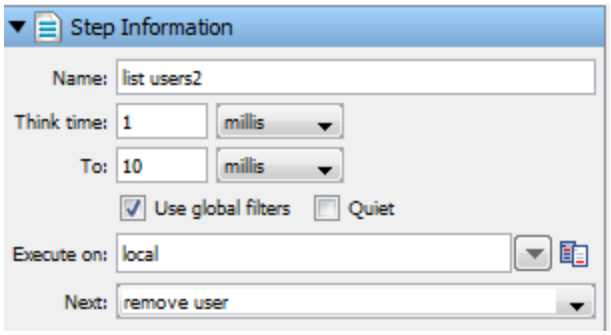
これらのエレメントの詳細については、「*CA Application Test の使用*」の「テスト ステップの作成 (P. 221)」で説明します。



ステップ情報

［ステップ情報］セクションは、テスト ステップの基本的な情報を記述する場所です。

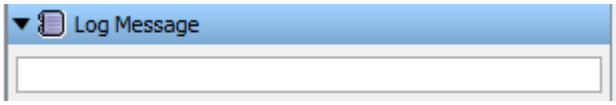
ステップ名、反応時間、実行先の詳細、および次のステップの詳細を入力できます。また、グローバル フィルタを使用してステップを実行したり、ステップをクワイエットで実行したりすることを指定できます。



詳細については、「*CA Application Test の使用*」の「テスト ステップの作成 (P. 221)」を参照してください。

ログ メッセージ

［ログ メッセージ］は、ステップに対するメッセージを入力できるテキスト フィールドです。 このメッセージは、テスト ステップまたはテスト ケースの実行時に表示されます。

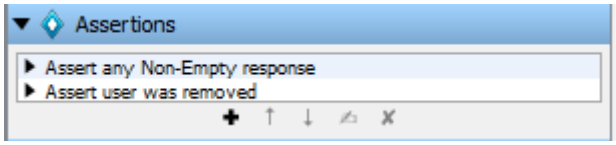


詳細については、「*管理*」の「テスト ステップ ロガー」を参照してください。

アサーション

アサーションは、1 つのステップとそのすべてのフィルタが実行された後に実行されるエレメントです。 アサーションは、ステップの実行結果がユーザの想定に一致することを検証します。 アサーションの結果は常にブール値（true または false のいずれか）です。

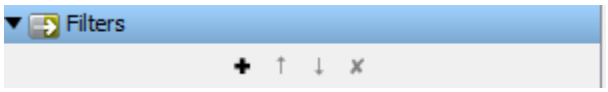
結果によってテスト ステップが成功したか失敗したか、また、テスト ケース内の次に実行するステップを判断します。つまりアサーションは、ワークフローに条件ロジック（分岐）を導入することにより、テスト ケースのワークフローを動的に変更します。



詳細については、「*CA Application Test の使用*」の「アサーション (P. 151)」を参照してください。

フィルタ

フィルタは、テスト ステップの前後に実行されるエレメントです。 フィルタを使用すると、結果のデータを変更したり、プロパティに値を格納することができます。



詳細については、「*CA Application Test の使用*」の「フィルタ (P. 132)」を参照してください。

データ セット

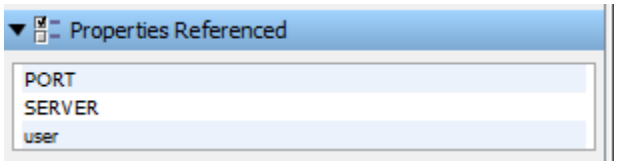
データ セットは、テストの実行時にテスト ケースにプロパティを設定するために使用できる値のコレクションです。 この機能によって、テスト ケースに外部のテスト データを使用することができます。

詳細については、「*CA Application Test の使用*」の「データ セット (P. 171)」を参照してください。

参照プロパティ

このセクションには、テスト ステップが使用または参照するプロパティのリストが表示されます。

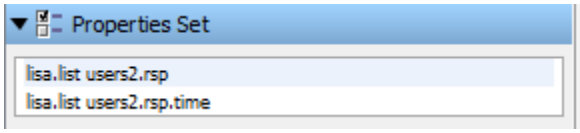
拡張ビューを開いてその変数値を取得するには、プロパティを選択して右クリックします。



詳細については、「*CA Application Test の使用*」の「プロパティ (P. 101)」を参照してください。

設定プロパティ

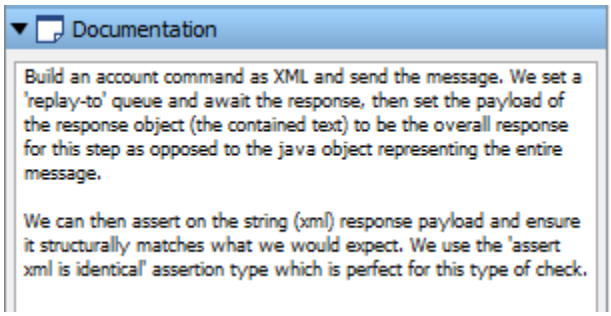
このセクションには、テスト ステップで設定するプロパティのリストが表示されます。 ［参照プロパティ］および［設定プロパティ］は特定のステップに対するものであり、別のステップが選択されたときには変化します。



詳細については、「*CA Application Test の使用*」の「プロパティ (P. 101)」を参照してください。

Documentation

［ドキュメント］領域では、テスト ステップのドキュメントを追加できます。 このテキストはどのプロセスでも使用されませんが便利な場所です。テスト ステップの説明や、このテスト ステップを使用するその他のユーザへのメモを記述するのはよい習慣です。



プロパティ

テスト プロパティは名前と値のペアであり、キー値ペアとも呼ばれます。

テスト ケースにおけるデータの独立性、再利用性、移植性の鍵となるのは、テスト ケースの抽象化された特定のデータ値を変数に置き換える機能です。 これらの変数を*プロパティ*と呼びます。 いくつかのプロパティは事前に定義されおり、アプリケーションの動作を決定します。 その他のプロパティは、テストを作成する際に作成します。

テスト ケースを作成するには、プロパティについて正しく理解することが重要です。 テスト ケースのコンテキストでは、変化する可能性があるものに対してはプロパティを使用するようにします。 このシナリオの例としては、テスト ステップ内の値や設定内の値があります。

プロパティはいくつかの方法で定義できます。定義した後は、テスト ケース内の後続のステップ、アサーション、およびフィルタで使用できます（これらはテスト ケースに対してグローバルです）。 プロパティは、いくつかの例外を除いては、テスト ケースで上書きされます。

プロパティ値が設定されると、必ずプロパティの設定イベントが記録されます。 このイベントには、プロパティ名と値が含まれます。

プロパティ値に使用できるのは、文字列だけではありません。プロパティには、テスト ステップからの文字列、数値、XML フラグメント、シリアライズされた Java オブジェクト、または完全な応答を格納できます。 プロパティの多くはテストの実行中に作成され、後続のテスト ステップで使用できます。 たとえば、**lisa.stepname.rsp** プロパティには、stepname ステップの応答が格納されます。

以下のトピックが含まれます。

プロパティの指定

プロパティの構文は {{プロパティ名}} です。

プロパティが識別され、使用されるときには、プロパティの現在の値で {{プロパティ名}} が置き換えられます。 プロパティが予測でき、唯一の選択肢となる場合があります。 また、プロパティ名を明示的に要求される場合もあります。 このような場合には、中かっこなしでプロパティ名を入力します。 プロパティをテキスト文字列に埋め込む場合には、中かっこを使用します。その他の構文としては、プロパティ式（{{=式}} または {{プロパティ名=式}}）を使用できます。

プロパティ名にはスペースを含めることができます。 ただし、スペースの使用は推奨されていません。 プロパティ構文を定義する文字（{、}、=）は使用できません。存在しないか無効なプロパティを参照すると、DevTest はそのプロパティを中かっこの状態で残します。

UNIX 式のラインフィードが含まれたプロパティ ファイルを編集する場合は、ワードパッドなどの適切なエディタを使用してください。

注: **lisa** で始まるプロパティ名は、 すべて内部使用のために予約されています。 **lisa** で始まるプロパティは、非表示にしたり、削除したりすることができます。

プロパティ式

プロパティには、さまざまなタイプのデータを格納できます。 また、式を評価して格納することもできます。 これらの式には、BeanShell で評価できる有効な Java または JavaScript の式を含めることができます。 BeanShell。 BeanShell は Java インタープリタ環境です。 さらに、これらの式には本物のように見える仮の文字列を与える文字列パターンを使用でき、大半の目的に使用できます。

BeanShell の詳細については、「DevTest での BeanShell の使用 (P. 509)」または www.beanshell.org を参照してください。

プロパティ式を使用するには、以下のいずれかの形式を使用します。

{{=expression}}

BeanShell を使用してこの式が評価され、評価の結果によって {{expression}} が置き換えられます。 たとえば、{{=Math.random()}} では静的な Java メソッドを評価して、返されたランダムな数値で {{}} の構造を置き換えます。

{{key=expression}}

{{rand=Math.random()}} を使用すると、**rand** プロパティに返されたランダムな数値を設定して、{{}} の構造をランダムな数値に置き換えます。

すでにプロパティとして定義されているため、プロパティ式において名前のみ（中かっこなし）でプロパティを参照できます。 プロパティが見つからない場合は、プロパティ式が中かっこの中に返され、式に問題があったことが示されます。

文字列パターン

*文字列パターン*は、プロパティ式の特別なタイプであり、{{ =[:patternname:] }} という構文で表します。たとえば、名前の形式を指定するときには、文字列パターン プロパティは {{ =[:First Name:] }} となります。 このプロパティは、本物の名前のように見える仮の名前に対して評価することができます。

この動作は、名前に見えないランダムな文字列を処理するよりはるかに適切です。 文字列パターン機能では、名前のほかに、姓、日付、社会保障番号、クレジットカード番号、クレジット カードの有効期限などの多くのパターンをサポートしています。 この仮のデータは、reportdb データベースの TESTDATA から取得されます。

テスト ケースで名前が必要な場合には、データ ットで {{ =[:First Name:] }} を使用することを推奨します。 「{{=[」 の部分は、文字列パターンを使用するというサインであり、これには「登録されている」項目のリストがあります。

たとえば、ログ ステップで以下の情報を使用するとします。

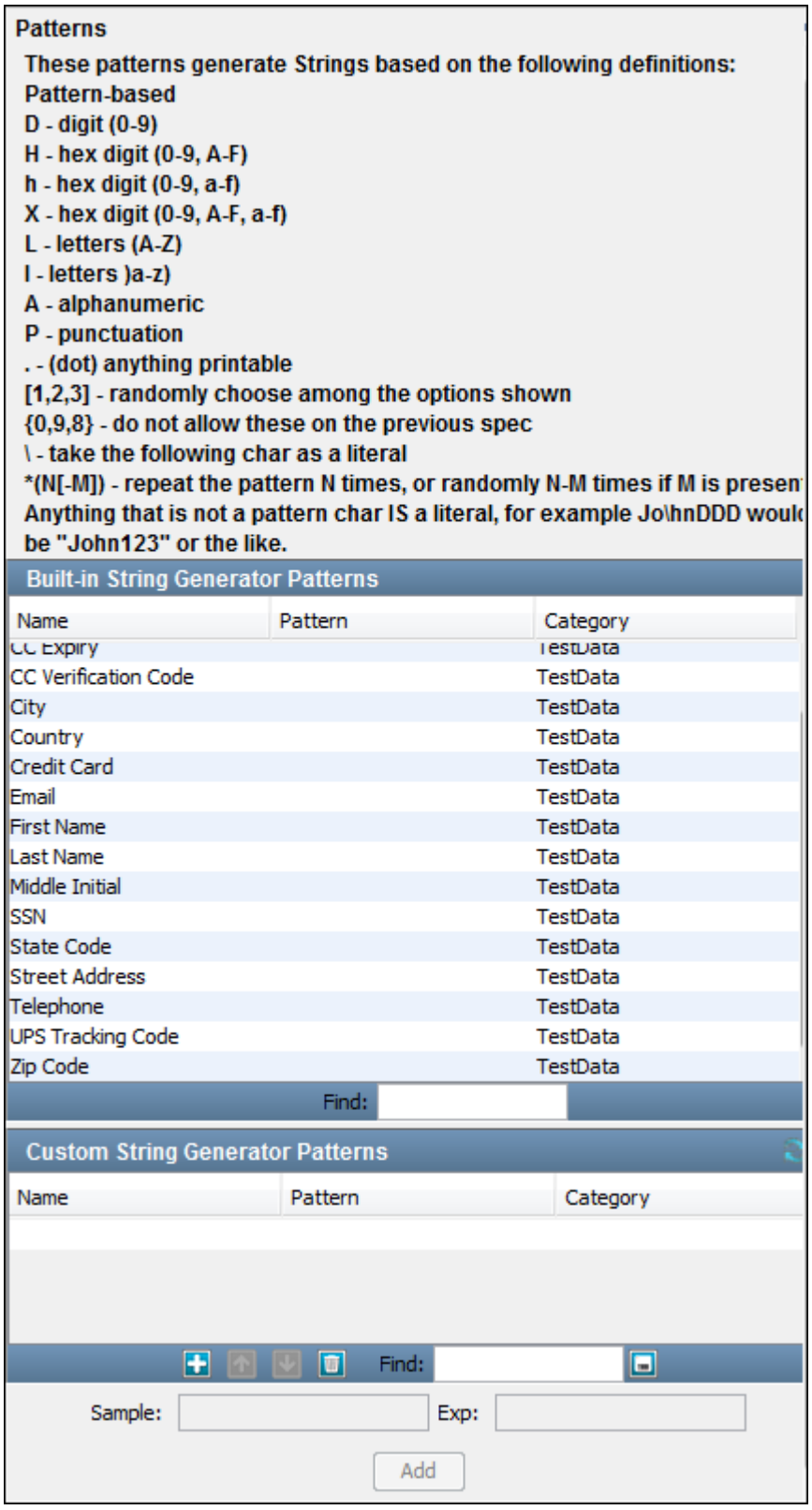
```
{{=[:First Name:]}} {{=[:Middle Initial:]}} {{=[:Last Name:]}}
{{=[:Street Address:]}}
{{=[:City:]}}, {{=[:State Code:]}}
{{=[:ZIP Code:]}} {{=[:Country:]}}
SSN: {{=[:SSN:]}}
Card: {{=[:Credit Card:]}} Expires {{=[:CC Expiry:]}}
Phone: {{=[:Telephone:]}}
Email: {{=[:Email:]}}
```

以下の応答が提供されます。

```
Marilyn M Mcguire
3071 Bailey Drive
Oelwein, IA
50662 US
SSN: 483-16-8190
Card: 4716-2361-6304-6128 Expires 3/2014
Phone: 319-283-0064
Email: Marilyn.C.Mcguire@spambob.com
```

ステップを再度実行すると、異なるデータが取得されます。DevTest では、テストデータ データベース内の行の数を追跡し、1 から N の間の行をランダムに選択します。

既存の文字列パターンがすべて表示され、ドキュメントが表示される以下のウィンドウを開くには、ページの右端にある垂直の［パターン］タブをクリックします。


Patterns
These patterns generate Strings based on the following definitions:
Pattern-based
D - digit (0-9)
H - hex digit (0-9, A-F)
h - hex digit (0-9, a-f)
X - hex digit (0-9, A-F, a-f)
L - letters (A-Z)
l - letters)a-z)
A - alphanumeric
P - punctuation
. - (dot) anything printable
[1,2,3] - randomly choose among the options shown
{0,9,8} - do not allow these on the previous spec
\ - take the following char as a literal
*(N[-M]) - repeat the pattern N times, or randomly N-M times if M is presen
Anything that is not a pattern char IS a literal, for example Jo\hnDDD woul
be "John123" or the like.
Built-in String Generator Patterns
Name
Pattern
Category
CC Verification Code
TestData
City
Country
Credit Card
Email
First Name
Last Name
Middle Initial
SSN
State Code
Street Address
Telephone
UPS Tracking Code
Zip Code
Find:
Custom String Generator Patterns
Name
Pattern
Category
Find:
Sample:
Exp:
Add

実装情報

デフォルトでは、このデータはレポート データベースの TESTDATA テーブルに保存されています。

DevTest ワークステーション は、起動時に、TESTDATA テーブルにデータがあるかどうかを確認します。 データがない場合は、lisa-core.jar 内の com/itko/lisa/test/data/TestData.csv が読み取られ、データベースがロードされます。何らかの理由で reports.db が削除された場合は、テスト データは再作成されます。データベースの読み取りは起動時にのみ行われ、約 15 秒かかります。

独自の文字列パターンの作成

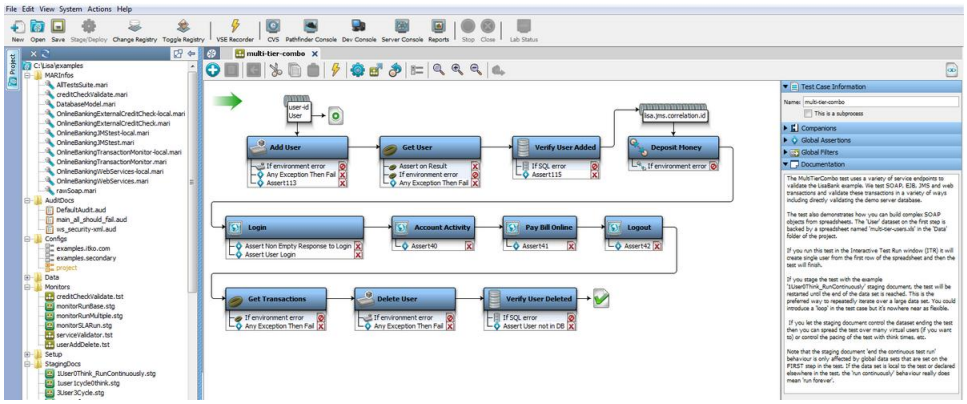
独自のデータを追加する場合に注意する唯一のことは、ID を正しく割り当てることです。 1 から開始し、行の数に到達するまで空番なしで増加させます。

文字列ジェネレータ コードは基本的に、ランダムなオブジェクトから「n」を取得した後、ID が「n」のテストデータからワイルドカード（*）で選択します。 したがって、効率的なルックアップを行うには、ID がテーブルのプライマリ キーである必要があります。

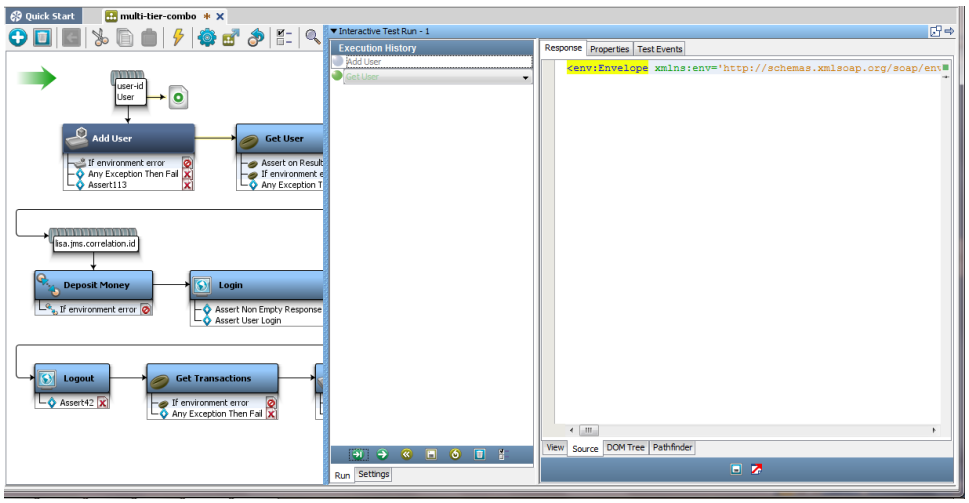
例

プロパティ式を使用する良い練習は、ステップが 1 つだけの簡単なテスト ケースであるログ メッセージの出力を作成することです。 このステップはログを 1 つ書き込むだけであり、対話型テスト ラン（ITR）の［応答］パネルにその応答が表示されます。 そのため、プロパティ式を使用して実験することができます。

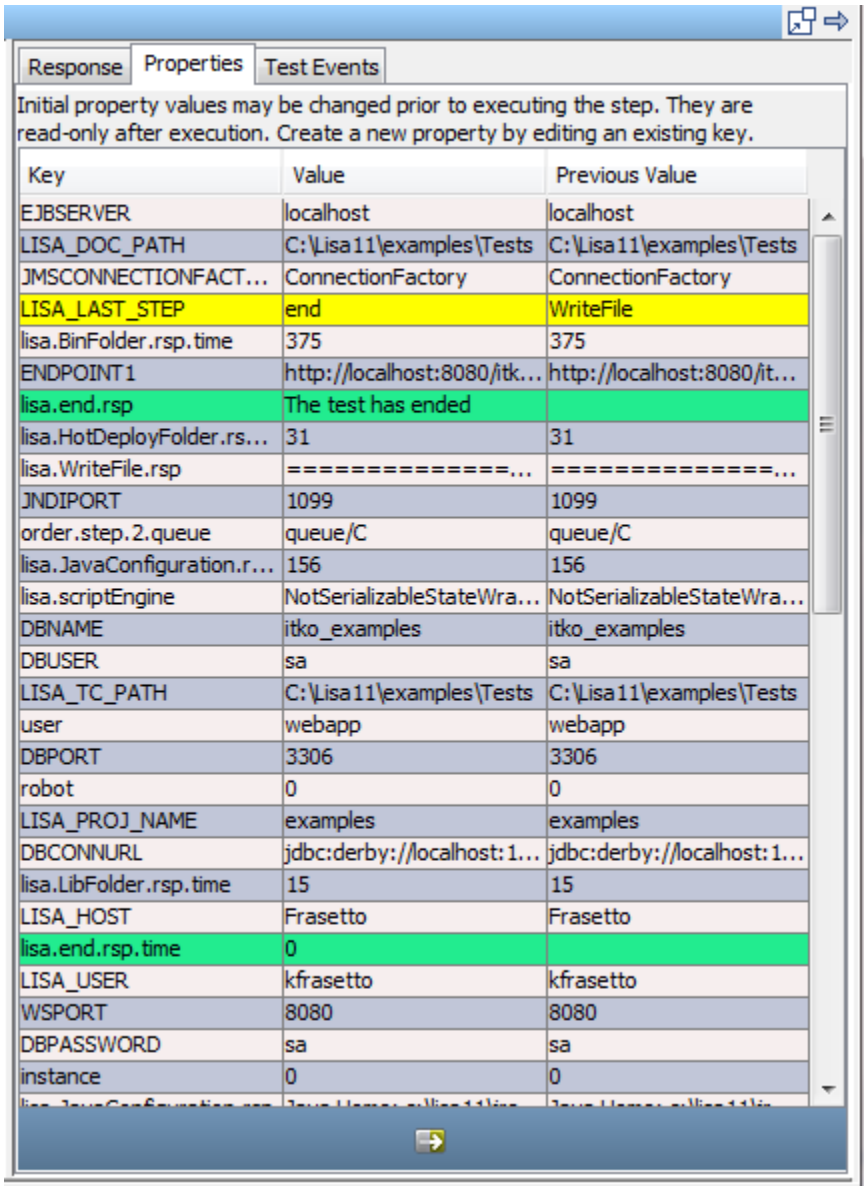
以下の例では、examples ディレクトリにある multi-tier-combo テスト ケース（multi-tier-combo.tst）を使用しています。



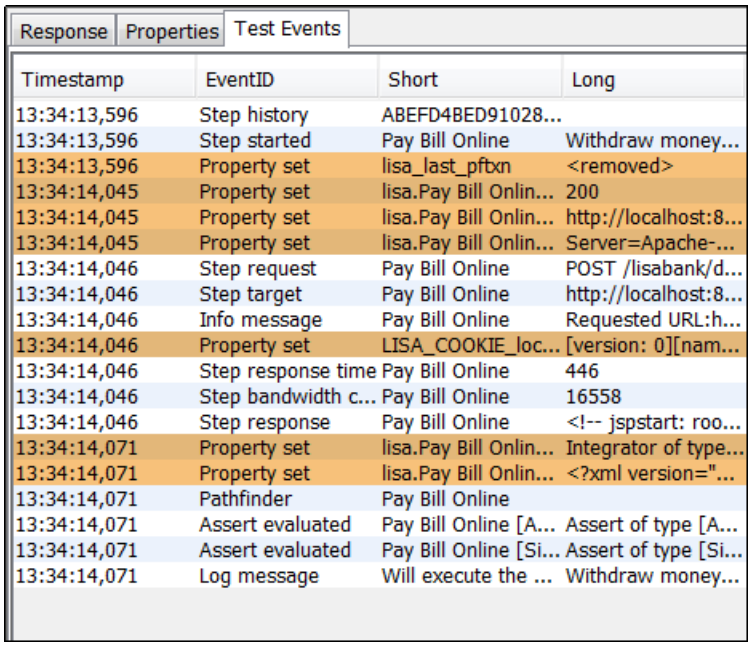
以下の図は ITR ユーティリティでこの例が表示された状態を示しています。



以下の図は、ITR の［プロパティ］タブを示しています。 表示されているタブは、Get User （ユーザの取得）ステップのものです。



以下の図は ITR の［テスト イベント］タブを示しています。



［テスト イベント］タブで［**プロパティの設定**］イベントを見つけてください。

Java 開発者は、JavaScript ステップのプロパティ式のテストに BeanShell 環境を利用することもできます。

プロパティのソース

プロパティは、以下のいくつかのソースを元にすることができます。

- DevTest
- 環境変数
- 起動時のコマンド ライン変数
- Configs
- コンパニオン
- テスト ステップ
- フィルタ
- データ セット
- 文字列パターン

プロパティは上書きすることができるため、プロパティの階層（プロパティがテスト ケースで読み取られる順番）を理解する必要があります。

以下の階層が使用されます。

1. テストのセットアップ中にロードされるプロパティ。
2. オペレーティング システムの環境変数（java.version または os.user など）。
3. DevTest プロパティ ファイル。
4. コマンド ライン属性。
5. デフォルト設定。
6. 任意の代替設定プロパティ（アクティブな設定ファイルまたはランタイム設定ファイルから）。
7. テストの実行時に設定されるプロパティ。
8. コンパニオン内のプロパティ。
9. （データ セット、フィルタ、およびステップ内で）テストの実行時に設定されるプロパティ。 テストの実行時に設定されるプロパティによって、それまでに設定された値が上書きされます。

共通のプロパティと環境変数

HOT_DEPLOY

プロジェクトに固有の hotDeploy ディレクトリを指します。

LASTRESPONSE

最後に実行されたステップに対する応答。

LISA_HOME

インストール ディレクトリを指し、自動的に設定されます。この値の最後にはスラッシュが含まれます。「examples」などのディレクトリを参照するには、以下のように指定します。

```
{{LISA_HOME}}examples
```

ディレクトリ名の前にスラッシュは必要ありません。

LISA_HOST

テスト環境が実行されているシステムの名前。

LISA_JAVA_HOME

使用する Java VM です。 このプロパティは、組み込みの VM を使用しない場合にのみ設定します。 Java がインストールされていない場合、DevTest はバンドルされている付属の JRE を使用します。また、インストール ディレクトリにある jre ディレクトリの名前を、jre_notinuse などに変更する必要があります。

LISA_POST_CLASSPATH

DevTest のクラスパスの後ろに情報を追加するために使用します。 DevTest は、OS 環境の CLASSPATH 変数を使用しません。 DevTest のクラスパスの後ろに独自の JAR を追加するには、LISA_POST_CLASSPATH を使用します。

LISA_PRE_CLASSPATH

DevTest のクラスパスの前に情報を追加するために使用します。 DevTest は、OS 環境の CLASSPATH 変数を使用しません。 DevTest のクラスパスの前に独自の JAR を追加するには、LISA_PRE_CLASSPATH を使用します。

LISA_PROJ_NAME

現在のドキュメントが属するプロジェクトの名前。 LISA_PROJ_NAME は、コールするメインのテスト プロジェクトを参照します。

LISA_RELATIVE_PROJ_NAME

現在のドキュメントが属するプロジェクトの名前。
LISA_RELATIVE_PROJ_NAME は、現在実行しているテストまたはサブプロセスのプロジェクトを参照します。テストがサブプロセスをコールしない場合、LISA_PROJ_NAME と LISA_RELATIVE_PROJ_NAME は同じです。

LISA_PROJ_PATH

プロジェクト ディレクトリの完全修飾パス。 値はオペレーティング システムによって異なります。 Windows では、円記号（¥）を区切り文字として使用します。その他のオペレーティング システムでは、スラッシュ（/）を区切り文字として使用します。 以下の例は、Windows インストールの場合です。

```
C:¥Program Files¥LISA¥examples
```

カスタム Java ステップで LISA_PROJ_PATH を使用するときには、構文 {{LISA_PROJ_PATH}} がサポートされないという制限があります。 カスタム Java ステップは、スクリプトをコンパイルするために Java コンパイラを呼び出し、Java は円記号を文字列内のエスケープ文字として扱います。 そのため、この文字列ではコンパイラ エラーが発生します。回避策は、変数として LISA_PROJ_PATH を使用することです。 以下に例を示します。

```
File f = new File ( LISA_PROJ_PATH );
```

LISA_RELATIVE_PROJ_PATH

現在実行しているテストまたはサブプロセスのプロジェクト ディレクトリの完全修飾パス。LISA_PROJ_PATH は、コールするメインのテストを参照します。詳細については、LISA_PROJ_PATH を参照してください。 テストがサブプロセスをコールしない場合、LISA_PROJ_PATH と LISA_RELATIVE_PROJ_PATH は同じです。

LISA_PROJ_ROOT

プロジェクト ディレクトリの完全修飾パス。 値はオペレーティング システムに依存しません。Windows を含むすべてのオペレーティング システムでは、スラッシュ（/）を区切り文字として使用します。 以下の例は、Windows インストールの場合です。

```
C:/Program Files/LISA/examples
```

LISA_RELATIVE_PROJ_ROOT

現在実行しているテストまたはサブプロセスのプロジェクト ディレクトリの完全修飾パス。LISA_PROJ_ROOT は、コールするメインのテストを参照します。詳細については、LISA_PROJ_ROOT を参照してください。 テストがサブプロセスをコールしない場合、LISA_PROJ_ROOT と LISA_RELATIVE_PROJ_ROOT は同じです。

LISA_PROJ_URL

プロジェクト ディレクトリの URL。 以下に例を示します。

```
file:/C:/Program%20Files/LISA/examples
```

LISA_RELATIVE_PROJ_URL

現在実行しているテストまたはサブプロセスのプロジェクト ディレクトリの URL。 LISA_PROJ_URL は、コールするメインのテストを参照します。 詳細については、LISA_PROJ_URL を参照してください。 テストがサブプロセスをコールしない場合、LISA_PROJ_URL と LISA_RELATIVE_PROJ_URL は同じです。

LISA_TC_PATH

テスト ケースを格納するディレクトリの完全修飾パス。

LISA_TC_URL

テスト ケースを格納するディレクトリの URL。

LISA_USER

テスト ケースをロードしたユーザ。

プロパティ ファイル

主なプロパティ ファイルを以下に示します。

- lisa.properties
- local.properties
- site.properties

プロパティの詳細については、以下の付録を参照してください。

- 付録 A (P. 523) - DevTest プロパティ ファイル（lisa.properties）
- 付録 B (P. 575) - カスタム プロパティ ファイル（local.properties、site.properties）

[プロパティ]ペインの使用

［プロパティ］ペインでは、テスト ケースに関連付けられているプロパティを表示できます。 また、このペインを使用すると、プロパティを特定のステップに追加するプロセスが簡略化されます。

次の手順に従ってください:

1. 変更するテスト ケースを開きます。
2. 以下のいずれかのアクションを実行して、［プロパティ］ペインを開きます。
   - ページの右端にある垂直の［プロパティ］タブをクリックします。
   - メイン メニューの［ヘルプ］-［プロパティを表示］をクリックします。
   - テスト ケース ツールバーにある  ［モデル プロパティの表示］をクリックします。

   ［プロパティ］ペインが開きます。
3. プロパティを追加するエレメント ツリーのテキスト フィールドに移動し、テキスト フィールド内をクリックします。
4. ［プロパティ］ペインで追加するプロパティを選択します。
   - 名前でプロパティを検索するには、ペインの下部にある［検索］フィールドを使用します。
   - 特定のカテゴリのプロパティにリストを制限するには、ペインの上部にあるリストからプロパティを選択します。
   - DevTest のこのインスタンスに対してグローバルなプロパティを表示するには、ページの右端にある［グローバル プロパティ］をクリックします。
5. ［プロパティ］ペインの下部にある［追加］をクリックします。

   プロパティが選択したフィールドに追加されます。

Configs

設定は、プロパティの名前付きのコレクションであり、通常はテスト中のシステムの環境に固有の値を指定します。

ハードコードされた環境データをテスト ケースからなくすことにより、別の設定を使用するだけで、異なる環境で同じテストを実行できます。設定は、DevTest 全体（テスト ケース ドキュメント、テスト スイート ドキュメント、ステージング ドキュメント、テスト ケースの実行、またはテスト スイートの実行など）で使用されます。

設定はプロジェクト レベルで定義する必要があります。 これらのプロパティの値は、テスト ケースの先頭で指定できます。

すべてのプロジェクトに対するデフォルト設定は **project.config** です。1 つのプロジェクトに複数の設定を作成して、特定のテスト ケースまたはスイートに対してそのうちの 1 つをアクティブにすることができます。

設定を作成する場合、新しいキーを追加することはできません。 新しい設定にキーを追加するには、project.config ファイルで追加します。その後、project.config ファイルに追加した新しいキーは、新しい設定ファイルのドロップダウンから選択できるようになります。

テストを作成するときに、プロパティが設定に自動的に追加されます。

たとえば、テストで WSDL 名を入力すると、定義されたサーバ名とポートが WSSERVER や WSPORT などのプロパティで置き換えられます。これらのプロパティの値は、デフォルトのプロジェクト設定に自動的に追加されます。 これで、いくつかのテスト ステップにハードコードされた値を探さなくても、設定を編集するだけで Web サービスの場所を変更できます。

別の例として、Enterprise JavaBeans （EJB）または Java オブジェクトを使用するときに、ホット デプロイ ディレクトリを切り替えたり、クラス パスに JAR ファイルを追加したりして、さまざまなバージョンの Java コードを使用することができます。 これらの場所には、HOT_DEPLOY および MORE_JARS という 2 つの標準プロパティがあります。 これらのプロパティは設定で設定できます。

その他のプロパティについては、「プロパティ (P. 101)」を参照してください。

設定は、テスト中のシステムに関連するプロパティを格納するためのものです。 これらを「テストのような」パラメータやグローバル パラメータの格納場所として使用しないでください。 これらのパラメータは、コンパニオン (P. 184)に格納できます。

注: 円記号（¥）は設定ファイルでは保持されません。 設定ファイルを手動で編集し、円記号を使用して何かを追加すると、円記号なしで上書きされます。

設定ファイルは、キー/値のペアとしてプロパティが記述されたテキストファイルであり、拡張子は .config です。 設定ファイルは、すべてのテスト ランに不可欠です。

設定ファイルは任意のテキスト エディタで作成および編集し、拡張子 .config を付けて保存できます。

テスト ランを開始するときに、設定ファイルを選択できます。 詳細については、「*CA Application Test の使用*」の「テスト ケース実行中の設定の適用 (P. 123)」を参照してください。

設定ファイルには命名規則を設定して、代替設定を見付けやすくすることをお勧めします。

これらの設定ファイルは、DevTest にインポートする必要があります。

以下のトピックが含まれます。

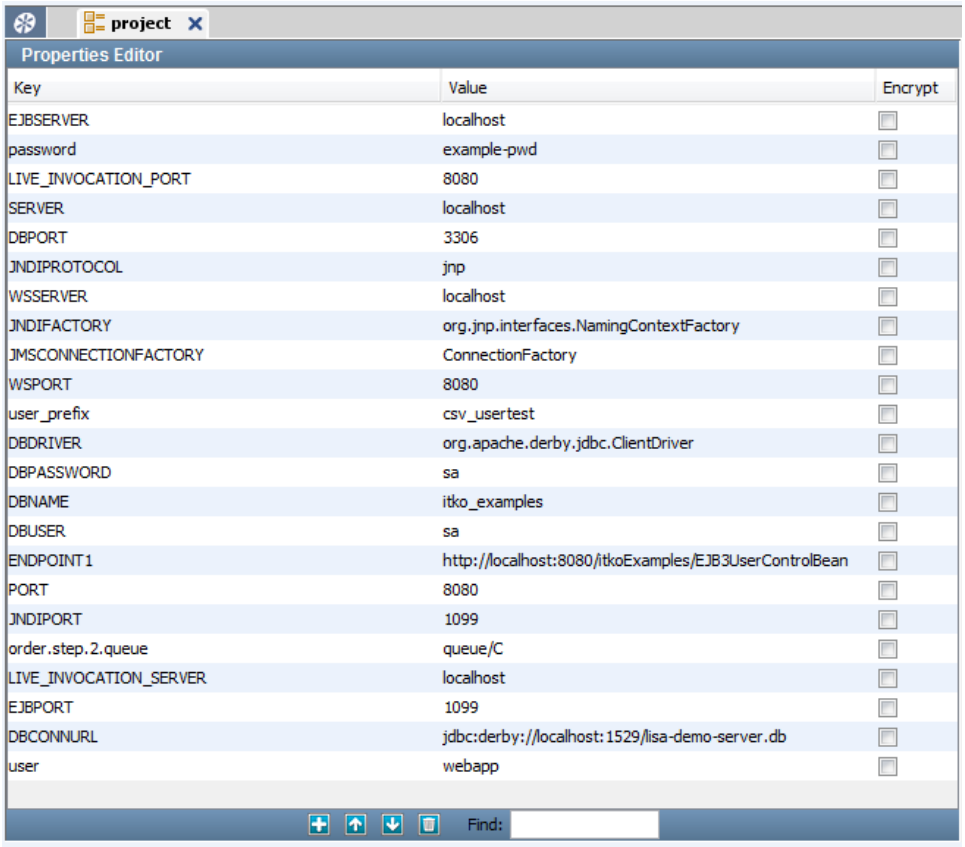
プロジェクト設定

すべてのプロジェクトには、**project.config** という名前の設定ファイルがあります。 このファイルは、*プロジェクト設定*とも呼ばれます。

プロジェクト パネルの Configs フォルダでは、プロジェクト設定がオレンジ色で表示されます。 ファイルの拡張子は表示されません。

プロジェクト設定の名前を変更したり、削除したりすることはできません。

プロジェクト設定をダブルクリックすると、［プロパティ エディタ］ウィンドウが表示されます。 以下の図は、［プロパティ エディタ］ウィンドウを示しています。 このエディタには、［キー］、［値］、［暗号化］という 3 つの列があります。



これらのパラメータは、すべての設定で使用できる標準のパラメータです。

プロジェクト設定には、その他のすべての設定で定義されているキーのスーパーセットが含まれます。 その他の設定にパラメータを追加するには、プロジェクト設定にパラメータを含める必要があります。 こうすることで、その他の設定のドロップダウン リストから選択できるようになります。

デフォルトでは、プロジェクト設定はプロジェクトのアクティブな設定です。 プロジェクトでアクティブにする設定を変更 (P. 120)できます。

設定の追加

1 つのプロジェクトには多数の代替設定を用意することができますが、アクティブな設定となるのは 1 つだけです。 どのような代替設定またはデフォルト設定もアクティブにできます。 代替設定は、デフォルト プロパティを上書きすることのみが可能です。

新しい設定にキーを追加するには、プロジェクト設定にそれらを追加します。 新しく定義したキーは、新しい設定ファイルのドロップダウン リストから選択できます。 新しい設定ファイルでは、新しいキーを追加することはできません。

設定ファイルを作成するときに追加できるキーは、すでにプロジェクト設定で定義されているものだけです。 いくつかの標準のキーがインストール時に用意されています。

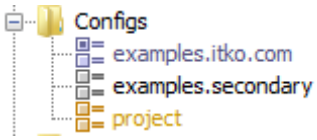
次の手順に従ってください:

1. プロジェクト パネルで Configs フォルダを右クリックし、［新規設定の作成］を選択します。
2. 新しい設定の名前を入力します。
3. ［OK］をクリックします。

設定をアクティブとしてマークする

プロジェクトに別の設定を適用する場合は、その設定をアクティブにします。 プロジェクト パネルの Configs フォルダでは、任意の設定をアクティブとしてマークできます。 アクティブな設定は、単一のテスト ケースではなく、プロジェクト全体に適用されます。

プロジェクト設定がアクティブな場合はオレンジでマークされます。その他の設定は黒で表示されます。 セカンダリ設定がアクティブな場合は紫でマークされ、プロジェクト設定はオレンジのままになります。



プロジェクト内の各テスト ケースでは、そのプロジェクトに適用されているアクティブな設定を共有します。 プロジェクト内の各テスト ケースに個別の設定を割り当てることはできません。

アクティブな設定が選択されている場合、クイック テストのステージングではそれが使用されます。 選択されていない場合は、デフォルト設定（project.config）が使用されます。

設定をアクティブとしてマークする方法

1. Configs フォルダ内の設定を右クリックし、［アクティブ化］を選択します。

設定の編集

プロジェクト設定以外の設定でプロパティを追加できるのは、そのプロパティがプロジェクト設定に存在している場合のみです。

ベスト プラクティスは、設定に格納されているパス名でプロパティを使用することです。**LISA_HOME** または **LISA_PROJ_ROOT** などのプロパティを使用すると、テスト ケースに移植性を持たせることができます。

プロパティ値には複数の行を含めることができます。

拡張ビューは、プロパティ値を編集するためのダイアログ ボックスで構成されています。 このビューは、値が長い場合や、値に複数の行が含まれる場合に便利です。 拡張ビューを表示するには、プロパティ値のセルを右クリックして［拡張ビューの起動］を選択します。

次の手順に従ってください:

1. Configs フォルダ内の設定をダブルクリックします。

   プロパティ エディタが開きます。

2. プロパティ エディタの下部にある  ［追加］をクリックしてプロパティを追加します。

   新しい行がプロパティ リストに追加されます。

3. ドロップダウン リストから、選択したキーを選択します。

   共通プロパティ名がドロップダウン リストに表示されます。

   値

   HOT_DEPLOY

   ホット デプロイ ディレクトリの場所を示します。

   MORE_JARS

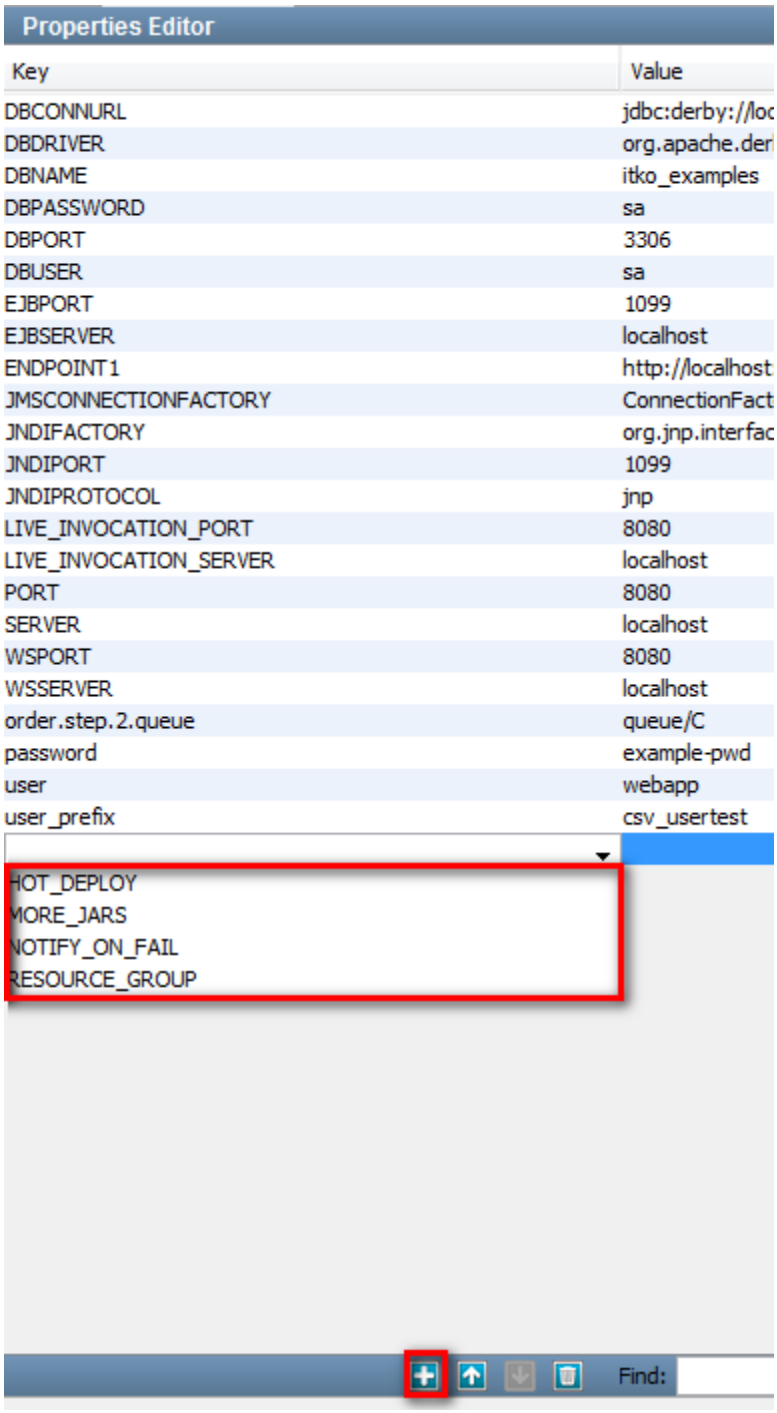
   JAR ファイルをクラス パスに追加します。

   NOTIFY_ON_FAIL

   テスト ケースが失敗したときに通知する電子メール アドレスを定義します。

   RESOURCE_GROUP

   リソース グループで定義されているリソースに基づいてコーディネータおよび VSE の選択内容をフィルタします。


Properties Editor
Key
Value
DBCONNURL
jdbc:derby://loc
DBDRIVER
org.apache.der
DBNAME
itko_examples
DBPASSWORD
sa
DBPORT
3306
DBUSER
sa
EJBPORT
1099
EJBSERVER
localhost
ENDPOINT1
http://localhost
JMSCONNECTIONFACTORY
ConnectionFact
JNDIFACTORY
org.jnp.interfac
JNDIPORT
1099
JNDIPROTOCOL
jnp
LIVE_INVOCATION_PORT
8080
LIVE_INVOCATION_SERVER
localhost
PORT
8080
SERVER
localhost
WSPORT
8080
WSSERVER
localhost
order.step.2.queue
queue/C
password
example-pwd
user
webapp
user_prefix
csv_usertest
HOT_DEPLOY
MORE_JARS
NOTIFY_ON_FAIL
RESOURCE_GROUP
Find:

テスト ケース実行中の設定の適用

テスト ケースの実行中に設定を適用する方法については、「*CA Application Test の使用*」の［テスト ケースのステージング (P. 340)］を参照してください。

アセット

*アセット*は、1 つの論理的な単位にグループ化される設定プロパティのセットです。

通常、アセットは、外部アプリケーション、またはアプリケーションとの通信に必要な中間のクライアント/サーバ コンポーネントとの通信ポイントを表します。アセットのタイプには、JMS 接続ファクトリ、JMS 送信先、JNDI コンテキスト、および SAP JCo 送信先が含まれます。

アセットの主な利点は再利用です。 同じプロパティを複数回入力する必要はありません。代わりに、プロパティをアセットの一部として 1 回定義します。

以下のパラメータはすべてのアセットに共通です。

- 名前
- 説明
- ランタイム スコープ

アセットは設定 (P. 116)に関連付けられます。 設定を開くと、アセット ブラウザがプロパティ エディタの右側に表示されます。 プロジェクト設定には、その他の設定のすべてのアセットのスーパーセットが含まれます。その他の設定の 1 つにアセットを設定して、プロジェクト設定の対応するアセットより優先させることができます。

アセットは、新規に作成またはテスト ステップから作成できます。

アセット ブラウザ

設定を開くと、アセット ブラウザがプロパティ エディタの右側に表示されます。

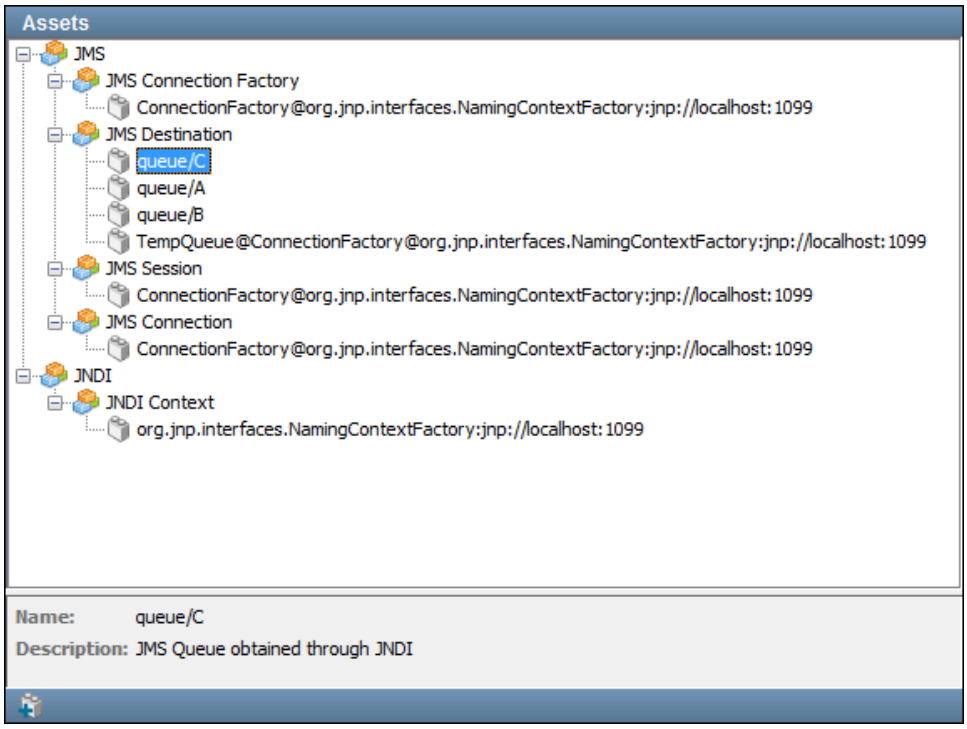
注: アセット ブラウザが空の場合、アセットは作成されていません。

メイン領域には、ツリー構造が表示されます。最上位ノードは*アセット カテゴリ*です。 2 番目のレベルのノードは*アセット タイプ*です。

名前および説明のパラメータがメイン領域の下に表示されます。

［追加］ボタンが左下隅に表示されます。

以下の図は、アセット ブラウザを示しています。 この例のアセットは、JMS と JNDI の 2 つのカテゴリに分類されます。 JMS のカテゴリには、JMS 接続ファクトリ、JMS 送信先、JMS セッション、および JMS 接続の 4 つのアセット タイプがあります。 JNDI のカテゴリには、JNDI コンテキストというアセット タイプが 1 つあります。



アセットがプロジェクト設定に関連付けられている場合、アセットを右クリックして［重複］を選択すると、アセットのコピーを作成できます。

アセットが追加設定に関連付けられている場合、アセットを右クリックして［プロジェクト設定で重複］を選択すると、アセットのコピーを作成できます。

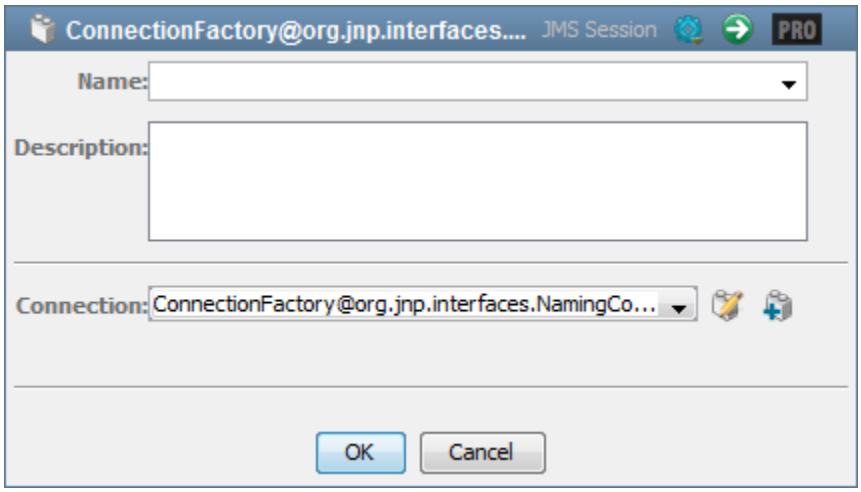
アセット エディタ

アセット エディタを使用して以下のタスクを実行します。

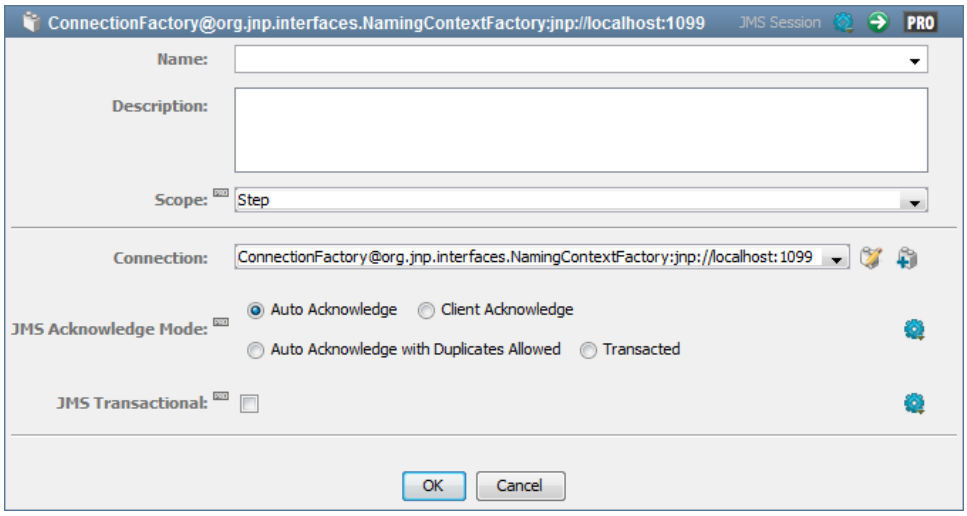
- アセットの新規作成
- アセットの変更

アセット エディタには、基本モードと詳細モードの 2 つのモードがあります。

以下の図は、基本モードを示しています。 名前および説明のパラメータはすべてのアセットの標準です。 最初の水平の分割線より下のパラメータはアセット タイプによって異なります。



以下の図は、詳細モードを示しています。 詳細モードを表示するには、右上隅の［PRO］をクリックします。



アセットには、その値が別のタイプのアセットであるパラメータを設定できます。たとえば、JMS セッション アセットには、JMS 接続アセットを指定するパラメータがあります。JMS 接続アセットには、JMS 接続ファクトリ アセットを指定するパラメータがあります。

一部のパラメータでは、値としてプロパティを入力できるようにエディタを変更できます。

値の個別のセットを提供するパラメータを使用すると、値を直接入力できるようにエディタを変更できます。

タイトル バーの緑色のボタンを使用して、アセットを確認 (P. 131)できます。

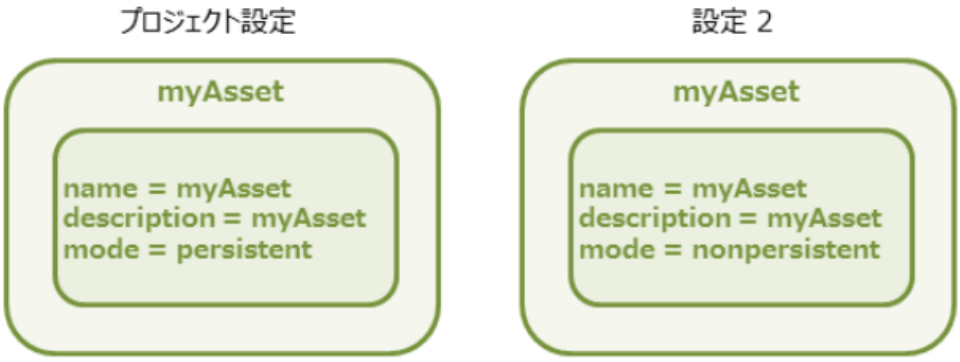
アセットの継承

すべてのプロジェクトには、プロジェクト設定があります。 プロジェクトには、1 つ以上の追加設定を設定することもできます。

プロジェクト設定には、その他の設定のすべてのアセットのスーパーセットが含まれます。

追加設定にアセットを設定して、プロジェクト設定の対応するアセットより優先させることができます。

以下の図に例を示します。プロジェクト設定には、**myAsset** という名前のアセットがあります。 このアセットには、名前、説明、およびモードのパラメータが含まれます。 2 番目の設定には、同じアセットがあります。ただし、2 番目の設定のモード パラメータの値は、プロジェクト設定のモード パラメータの値とは異なります。 したがって、2 番目の設定のアセットは、プロジェクト設定のアセットよりも優先されます。



アセットはタイプが同じである限り、別のクラスにすることもできます。

たとえば、プロジェクト設定に IBM MQ 接続ファクトリ アセットがあると仮定します。 追加設定で、アセットを変更して TIBCO 接続ファクトリ アセットにすることができます。 この変更が可能なのは、IBM MQ 接続ファクトリ アセットおよび TIBCO 接続ファクトリ アセットのタイプが同じ JMS 接続ファクトリであるためです。

アセット ブラウザ (P. 124)は、各追加設定のアセットの継承設定を示すために色分けを使用します。

- 灰色のアセットは、プロジェクト設定のアセットよりも優先されません。
- 黒のアセットは、プロジェクト設定のアセットよりも優先されます。

ランタイム スコープ

ランタイム スコープは、オープン アセット インスタンスがキャッシュされランタイム時に再利用される最小レベルを指定します。

アセットにはそれぞれランタイム スコープがあります。 また、テスト ステップの操作にランタイム スコープを設定できます。

有効な値は以下のとおりです。

- **ステップ：** オープン アセット インスタンスがキャッシュされ、それがアクセスされているステップで再利用されます。 ステップが完了すると、アセット インスタンスはクローズされます。
- **モデル：** オープン アセット インスタンスがキャッシュされ、それがアクセスされているモデルの実行時に再利用されます。 モデルの実行が完了すると、アセット インスタンスはクローズされます。
- **ステージング：** オープン アセット インスタンスは、それがアクセスされているモデルのすべてのインスタンスによって再利用されるようにキャッシュされます。 ステージングされたモデルのすべてのインスタンスが完了すると、アセット インスタンスはクローズされます。
- **グローバル：** オープン アセット インスタンスが、現在の JVM 内のすべてのクライアントによってグローバルに再利用されるようにキャッシュされます。アセット インスタンスがアイドルであった時間があると、アセット インスタンスはクローズされます。 このスコープは最大のスコープです。
- **デフォルト：** オープン アセット インスタンスが使用可能な最小スコープに割り当てられます。

アセットが操作で使用される場合、アセットと操作のランタイム スコープは異なる場合があります。 DevTest はランタイム スコープを評価し、使用する適切なランタイム スコープを判断します。

アセットの作成

アセット ブラウザ (P. 124)またはアセット エディタ (P. 126)でアセットを作成できます。

次の手順に従ってください:

1. 以下のアクションのいずれかを完了します。
   - アセット ブラウザで、左下隅の［追加］ボタンをクリックし、アセット タイプを選択します。
   - アセット エディタで、［新規アセットの追加］アイコンをクリックし、アセット タイプを選択します。使用可能なアセット タイプは、アイコンが表示されるパラメータによって異なります。
2. 必要な情報を入力します。［名前］フィールドを空白のままにすると、名前が自動的に生成されます。
3. ［OK］をクリックします。

テスト ステップからのアセットの作成

JMS 関連のテスト ステップからアセットを作成できます。

単一のテスト ステップから複数のアセットを作成できます。 たとえば、JMS メッセージング（JNDI）ステップを使用すると、以下のアセット タイプが作成できます。

- JMS 接続ファクトリ
- JMS 接続
- JMS セッション
- JMS 送信先
- JNDI コンテキスト

テスト ステップから再エクスポートすると、前のエクスポートからアセットに加えられた変更は無効になります。

次の手順に従ってください:

1. テスト ケースを開きます。
2. 1 つ以上のテスト ステップを選択します。
3. モデル エディタ ツールバーで、［選択されたステップからアセットを生成］アイコンをクリックします。

アセットの変更

このトピックでは、既存アセットに変更を加える方法について説明します。

加えることができる変更には、アセット クラスが含まれます。たとえば、JMS セッション アセットを JMS キュー セッション アセットまたは JMS トピック セッション アセットに変更できます。アセット クラスを変更するためのアイコンは、アセット エディタのタイトル バーにあります。

次の手順に従ってください:

1. 以下のいずれかの操作を実行します。
   a. アセット ブラウザで、アセットをダブルクリックします。
   b. アセット ブラウザで、アセットを右クリックし、［編集］または［上書き］を選択します。
   c. ステップ エディタで、［選択されたアセットの編集］をクリックします。
   d. パラメータとしてアセットが含まれる別のアセットのエディタで、［選択されたアセットの編集］をクリックします。
   e. 変更を加えます。
2. ［OK］をクリックします。

アセットの検証

以下の手順中にアセット エディタ (P. 126)からアセットを検証できます。

- アセットの作成
- アセットの変更

検証プロセスでは、アセットが表すオブジェクトにアクセスしようとします。 検証プロセスは機能の操作を実行しません。

次の手順に従ってください:

1. アプリケーションまたは少なくともアプリケーションが使用するエンドポイントが実行されており、DevTest から使用可能であることを確認します。
2. アセット エディタで、緑色の［検証］ボタンをクリックします。
3. ログ ウィンドウを表示して、成功か失敗かを示すメッセージを確認します。

フィルタ

フィルタはテスト ステップの前後に実行されるエレメントです。フィルタを使用すると、結果のデータを変更したり、プロパティに値を格納することができます。フィルタは、Web ページ、XML および DOM 応答、Java オブジェクト、テキスト ドキュメント、およびその他の多くのテスト ステップの応答から値を抽出するために使用できます。

ほとんどのフィルタは、ステップが実行された後に実行されます。

データをフィルタした後、そのデータはアサーションや後続のすべてのテスト ステップで使用できます。 フィルタは、通常、テスト中のシステムの応答に対して機能します。 たとえば、HTML ページの値を解析したり、応答に対して変換を実行するために使用します。 また、フィルタはその他の用途にも役立ちます。 フィルタは、プロパティ値をファイルに保存したり、プロパティを「最後の応答」になるよう変換したりするために使用できます。フィルタは主として、プロパティ セッタとして使用します。

フィルタは、グローバル フィルタまたはステップ フィルタとして適用できます。 使用可能なフィルタ タイプは同じですが、フィルタがどのように適用されるかが異なります。

グローバル フィルタ

テスト ケース レベルで定義されるフィルタがグローバル フィルタであり、グローバル フィルタを無視するよう設定されていない各テスト ステップの前または後に実行されます。 ステップの［ステップ情報］エレメントでは、グローバル フィルタを無視するようにステップを設定できます。

ステップ フィルタ

テスト ステップ レベルで定義されるフィルタがステップ フィルタであり、各テスト ステップの実行の前または後に実行されます。

グローバル フィルタおよびステップ フィルタは、必要な数だけ任意に追加できます。 フィルタは、テスト ケースに現れる順に実行されます。

以下のトピックが含まれます。

フィルタの追加 (P. 133)
フィルタのドラッグ アンド ドロップ (P. 150)

フィルタの追加

フィルタを追加する方法は以下のとおりです。

手動によるフィルタの追加 (P. 134)
HTTP 応答からのフィルタの追加 (P. 136)
JDBC 結果セットからのフィルタの追加 (P. 140)
返された Java オブジェクトからのフィルタの追加 (P. 147)

手動によるフィルタの追加

フィルタを手動で追加するには、リストからフィルタ タイプを選択して、そのフィルタのパラメータを入力します。

2 つのタイプのフィルタ（グローバル フィルタおよびステップ フィルタ）を手動で追加できます。

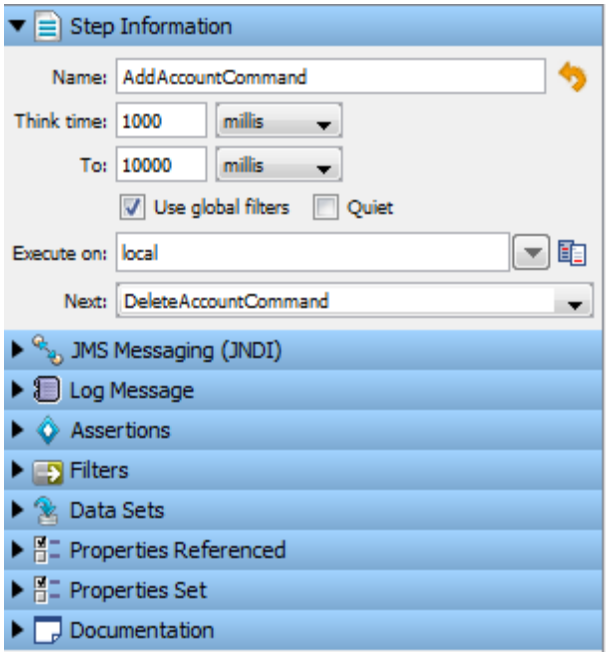
グローバル フィルタは、ステップに特に指定がない限り、テスト ケースの各ステップに適用され、自動的に実行されます。

ステップ フィルタは 1 つのステップのみに適用され、そのステップに対してのみ実行されます。

グローバル フィルタを手動で追加する方法

1. テスト ケースのエレメント パネルを開くには、テスト ケースを開き、エディタ内の任意の場所をクリックします。
2. ［グローバル フィルタ］エレメントで、［追加］ をクリックしてグローバル フィルタを追加します。
3. フィルタ タイプとして［CAI の DevTest 統合サポート］または［webMethods Integration Server の DevTest 統合サポート］を選択するよう求められます。

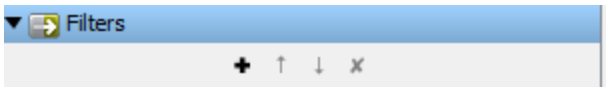
   これらのタイプのフィルタを追加する方法の詳細については、「*CA Application Test の使用*」の「CAI の統合サポート」または「webMethods Integration Server の統合サポート」を参照してください。
4. テスト ケースで各ステップに対して少なくとも 1 つのグローバル フィルタを設定している場合、デフォルトでは［グローバル フィルタの使用］チェック ボックスがオンになっています。ステップに対してグローバル フィルタを適用しない場合は、このチェック ボックスをオフにします。



ステップ フィルタを手動で追加する方法

次の手順に従ってください:

1. フィルタを適用するステップを選択し、右側のパネルで［フィルタ］エレメントをクリックします。



2. 使用可能なフィルタのリストを表示するには、フィルタ エレメントにある［追加］ をクリックするか、ステップを右クリックして［フィルタの追加］を選択し、このステップに適したフィルタを選択します。

   ［フィルタ］メニューが表示され、使用可能なフィルタのリストが表示されます。各フィルタにはそれぞれのエディタおよび適用可能なパラメータがあります。各フィルタ タイプの詳細については、「*リファレンス*」の「フィルタのタイプ」を参照してください。

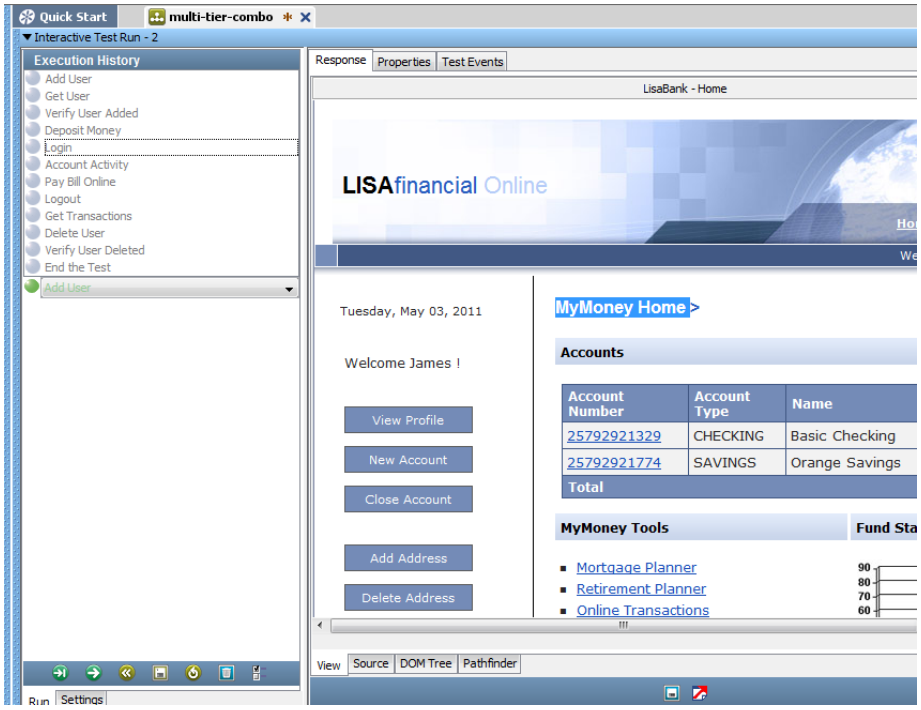
HTTP 応答からのフィルタの追加

HTTP ベースのステップからの応答にアクセスできる場合は、応答を使用してフィルタを直接追加できます。

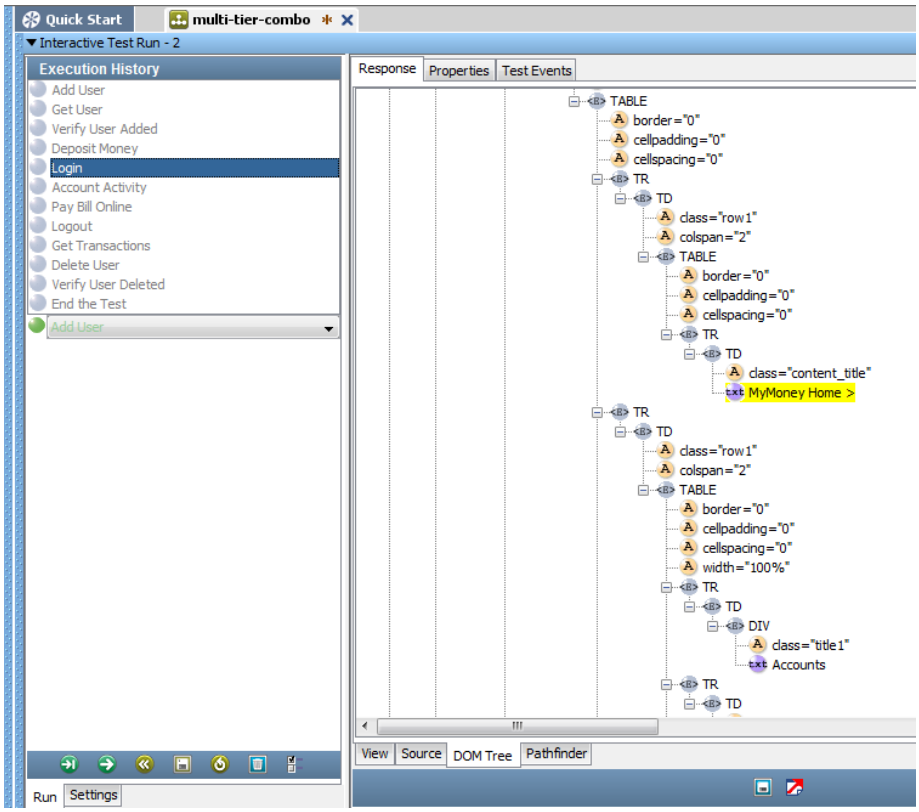
この例では、examples ディレクトリにある multi-tier-combo テスト ケースのログイン ステップからの応答を使用します。この例のポイントは、ウィンドウに現在「**MyMoney ホーム**」と表示されているテキストを取得することです （これは常に同じテキストとは限りません）。

次の手順に従ってください:

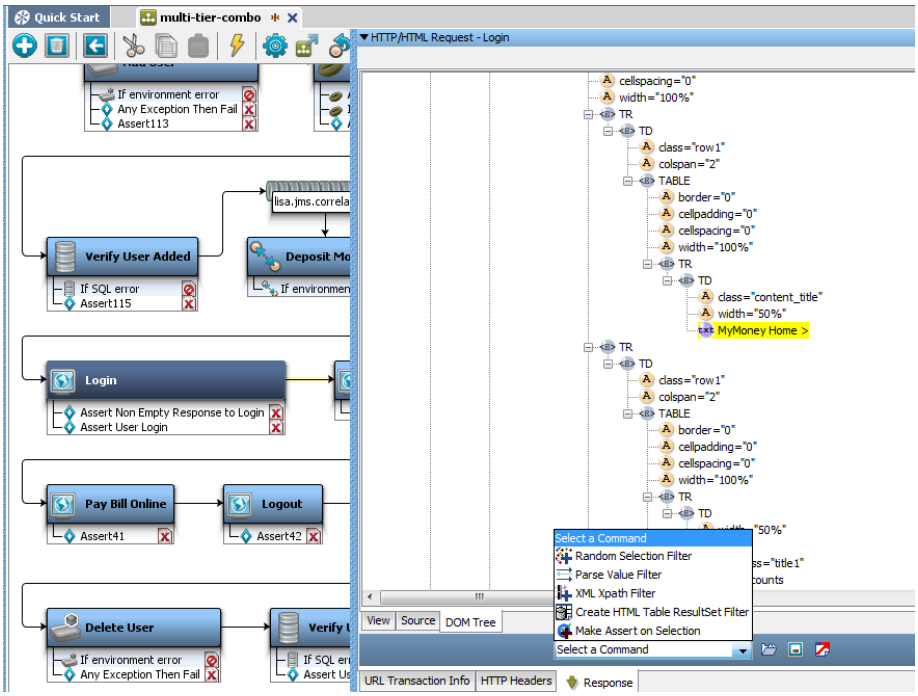
1. ITR で multi-tier-combo テスト ケースを実行し、ログイン テスト ステップを選択します。



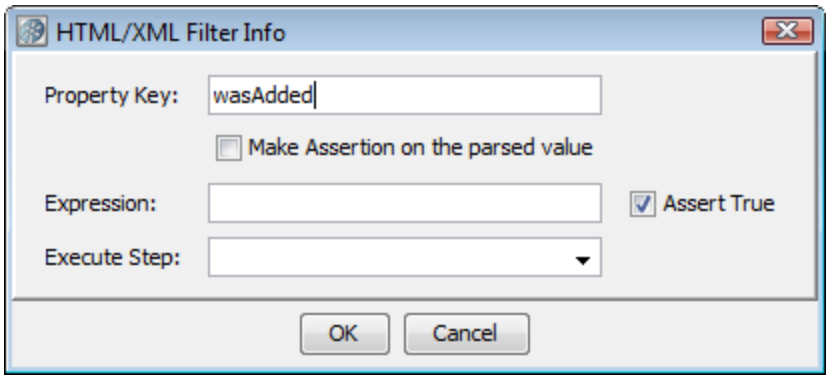
2. ［ビュー］タブ内の「MyMoney ホーム」というテキストを選択します。
3. このテキストがツリー ビューで選択されていることを確認するために、［DOM ツリー］タブをクリックします。



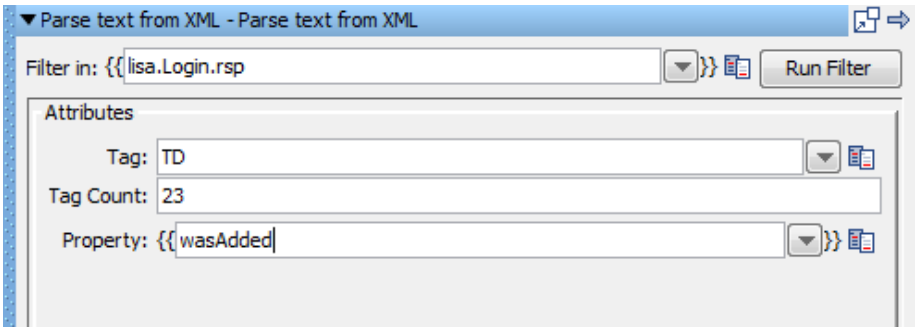
4. インライン フィルタを適用するには、モデル エディタでログイン ステップをダブルクリックして、HTTP/HTML ステップ エディタを開きます。



5. ［DOM ツリー］タブに移動して、DOM ツリー ビューで「MyMoney ホーム」を検索し、それを選択します。
6. ウィンドウの下部にある［コマンドの選択］ボックスで、ドロップダウン リストから［解析値フィルタ］を選択します。
7. 表示されるダイアログ ボックスに、プロパティ キーの名前「wasAdded」を入力します。



8. ［OK］をクリックします。



ここでアサーションも追加できます。 たとえば、**wasAdded** プロパティの値をテストして、実際に追加されたユーザと等しいかどうかを確認したい場合などがあります。 これについては「アサーションの追加」でより詳しく説明します。

生成されたフィルタは、ログイン テスト ステップでフィルタと見なすことができます。

HTML 応答がステップ エディタに表示される場合は、同じフィルタリング機能を使用できます。

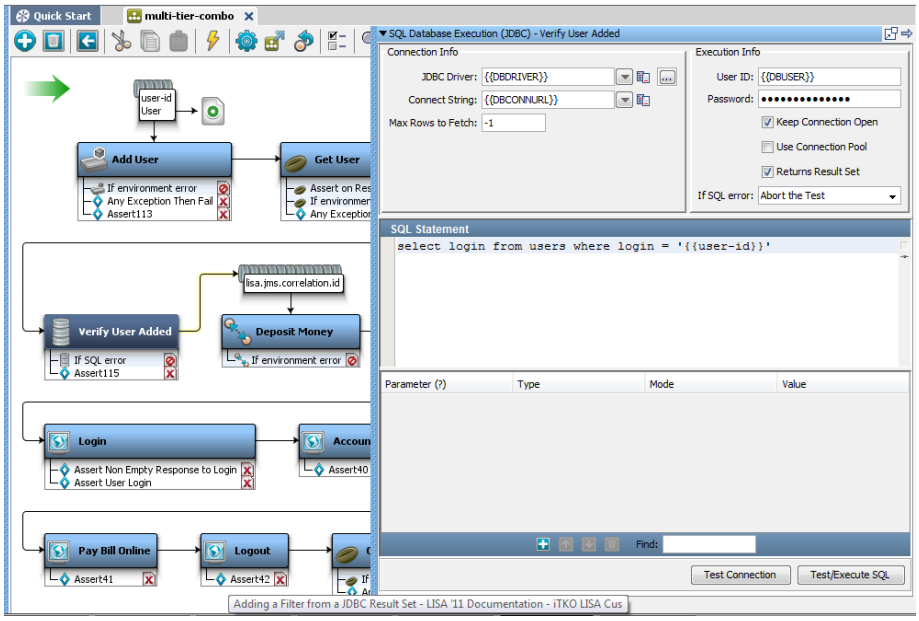
JDBC 結果セットからのフィルタの追加

JDBC ステップからの結果セット応答にアクセスできる場合は、応答を使用してフィルタを直接追加できます。

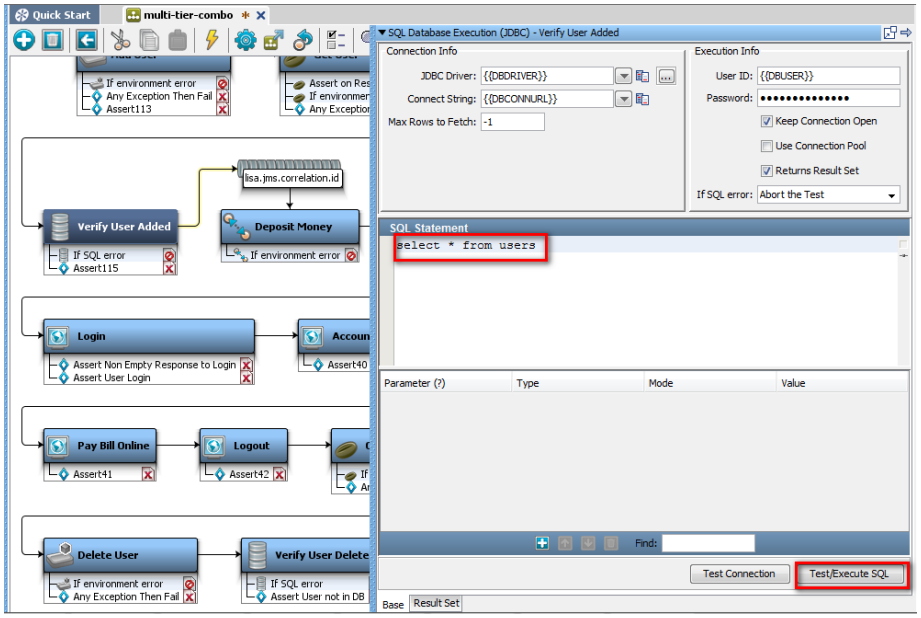
この例では、multi-tier-combo テスト ケースの Verify User Added （追加されたユーザの検証）ステップからの応答を使用して、JDBC 結果セット応答からフィルタを追加する方法について説明します。

次の手順に従ってください:

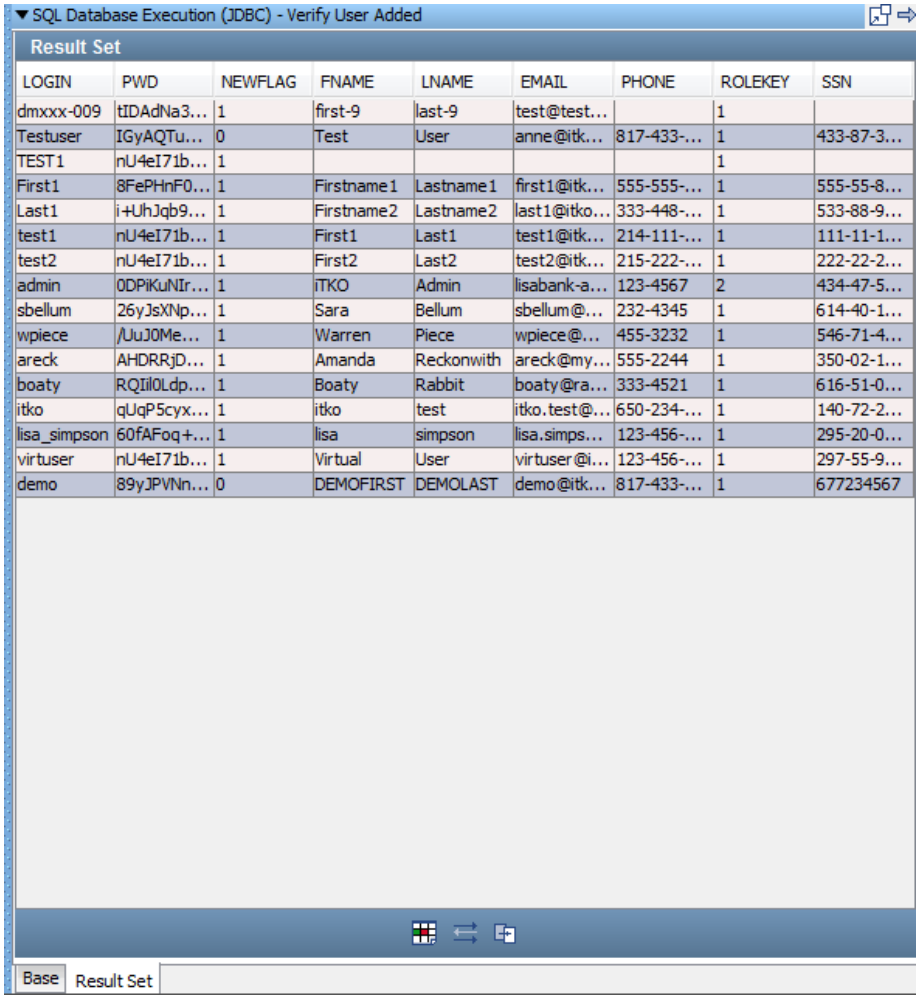
1. ステップ エディタを開くには、Verify User Added （追加されたユーザの検証）ステップをダブルクリックします。



2. 結果セット内の値を取得するには、SQL ステートメントを「**select * from users**」に編集して、［SQL をテスト/実行］をクリックします。



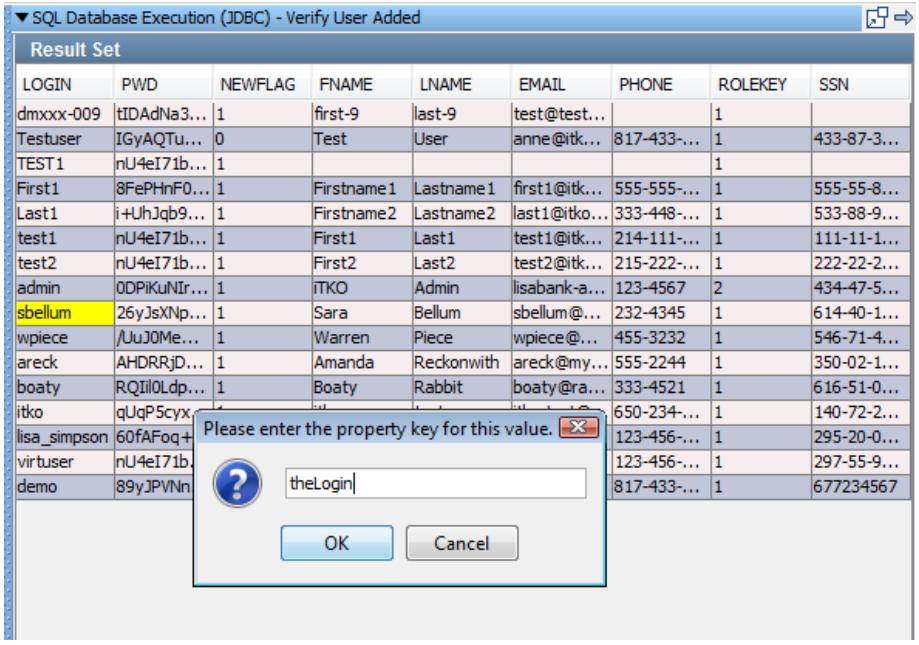
3. 結果セット内の値を取得するには、[結果セット]タブをクリックし、[SQL をテスト/実行] をクリックします。



LOGIN	PWD	NEWFLAG	FNAME	LNAME	EMAIL	PHONE	ROLEKEY	SSN
dmxxx-009	tIDAdNa3...	1	first-9	last-9	test@test...		1	
Testuser	IGyAQTu...	0	Test	User	anne@itk...	817-433-...	1	433-87-3...
TEST1	nU4eI71b...	1					1	
First1	8FePHnF0...	1	Firstname1	Lastname1	first1@itk...	555-555-...	1	555-55-8...
Last1	i+UhJqb9...	1	Firstname2	Lastname2	last1@itko...	333-448-...	1	533-88-9...
test1	nU4eI71b...	1	First1	Last1	test1@itk...	214-111-...	1	111-11-1...
test2	nU4eI71b...	1	First2	Last2	test2@itk...	215-222-...	1	222-22-2...
admin	0DPiKuNIr...	1	iTKO	Admin	lisabank-a...	123-4567	2	434-47-5...
sbellum	26yJsXNp...	1	Sara	Bellum	sbellum@...	232-4345	1	614-40-1...
wpiece	/UuJ0Me...	1	Warren	Piece	wpiece@...	455-3232	1	546-71-4...
areck	AHDRRjD...	1	Amanda	Reckonwith	areck@my...	555-2244	1	350-02-1...
boaty	RQIil0Ldp...	1	Boaty	Rabbit	boaty@ra...	333-4521	1	616-51-0...
itko	qUqP5cyx...	1	itko	test	itko.test@...	650-234-...	1	140-72-2...
lisa_simpson	60fAFoq+...	1	lisa	simpson	lisa.simps...	123-456-...	1	295-20-0...
virtuser	nU4eI71b...	1	Virtual	User	virtuser@i...	123-456-...	1	297-55-9...
demo	89yJPVNn...	0	DEMOFIRST	DEMOLAST	demo@itk...	817-433-...	1	677234567

4. キャプチャする情報の場所を表す結果セット タブ内のセルをクリックします（sbellum）。
5. ［現在の列/行の値用にフィルタを生成］  をクリックします。
6. 表示されたダイアログ ボックスにプロパティ キー「*theLogin*」を入力します。

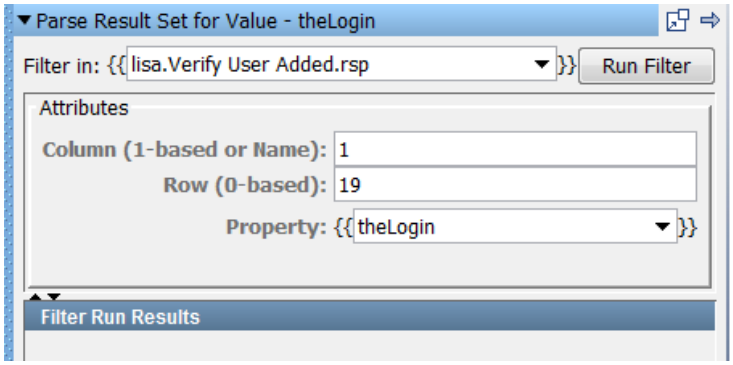
図 1: JDBC 結果セットからのフィルタの追加 - Verify User Added （追加されたユーザの検証） - 値のプロパティ キーの入力


SQL Database Execution (JDBC) - Verify User Added
Result Set
LOGIN PWD NEWFLAG FNAME LNAME EMAIL PHONE ROLEKEY SSN
dmxxx-009 tIDAdNa3... 1 first-9 last-9 test@test... 1
Testuser IGyAQTu... 0 Test User anne@itk... 817-433-... 1 433-87-3...
TEST1 nU4eI71b... 1 1
First1 8FePHnF0... 1 Firstname1 Lastname1 first1@itk... 555-555-... 1 555-55-8...
Last1 i+UhJqb9... 1 Firstname2 Lastname2 last1@itko... 333-448-... 1 533-88-9...
test1 nU4eI71b... 1 First1 Last1 test1@itk... 214-111-... 1 111-11-1...
test2 nU4eI71b... 1 First2 Last2 test2@itk... 215-222-... 1 222-22-2...
admin 0DPiKuNIr... 1 iTKO Admin lisabank-a... 123-4567 2 434-47-5...
sbellum 26yJsXNp... 1 Sara Bellum sbellum@... 232-4345 1 614-40-1...
wpiece /UuJ0Me... 1 Warren Piece wpiece@... 455-3232 1 546-71-4...
areck AHDRRjD... 1 Amanda Reckonwith areck@my... 555-2244 1 350-02-1...
boaty RQIil0Ldp... 1 Boaty Rabbit boaty@ra... 333-4521 1 616-51-0...
itko qUqP5cyx... 650-234-... 1 140-72-2...
lisa_simpson 60fAFoq+ 123-456-... 1 295-20-0...
virtuser nU4eI71b. 123-456-... 1 297-55-9...
demo 89yJPVNn 817-433-... 1 677234567
Please enter the property key for this value.
theLogin
OK
Cancel

7. ［OK］をクリックします。

   DevTest によって、リスト ユーザ ステップに「**結果セットの値を解析**」という名前のフィルタが追加されます。

8. フィルタを参照するには、フィルタ エディタをクリックします。



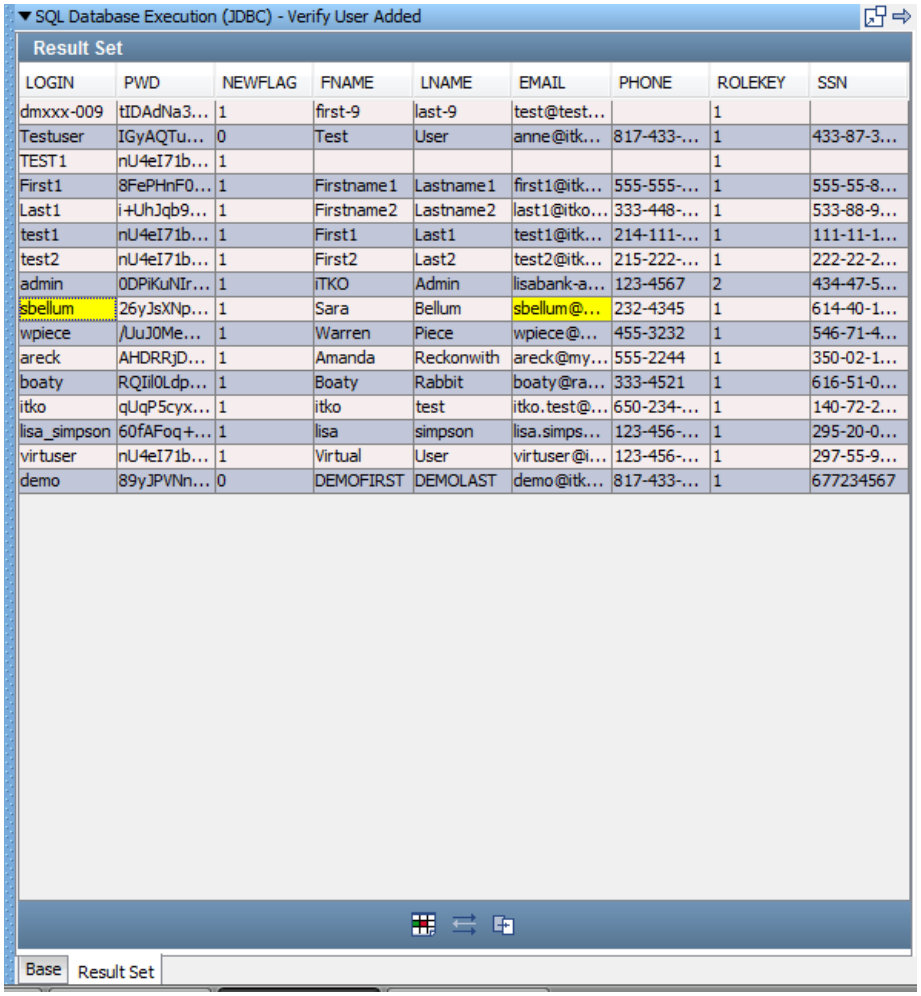
   この例では、8 行目の最初の列のセルの値（**sbellum**）が、theLogin プロパティに格納されます。

2 つ目のフィルタの適用

2 つ目のフィルタはここで適用できます。結果セットのある列で値を探した後、同じ行内の別の列から値をキャプチャできます。

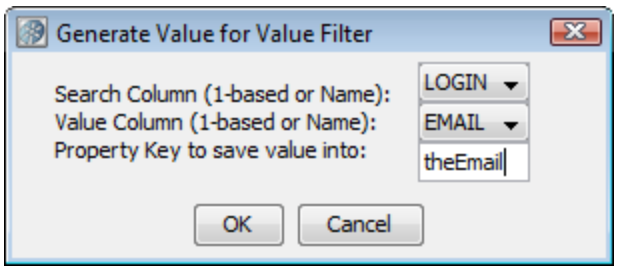
次の手順に従ってください:

1. 結果セットから、Ctrl キーを使用して、同じ行の別々の列から 2 つの値を選択します。



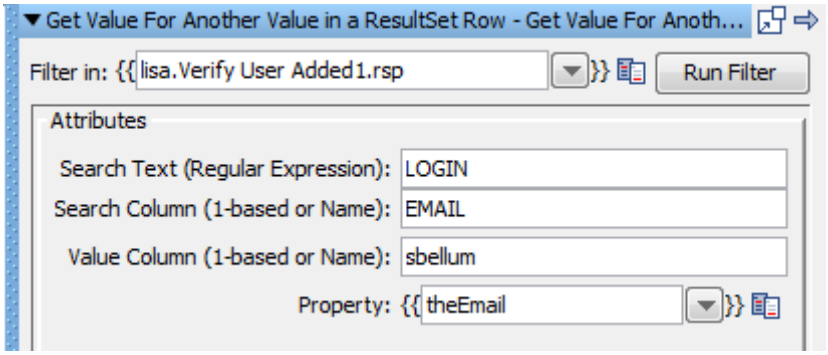
LOGIN	PWD	NEWFLAG	FNAME	LNAME	EMAIL	PHONE	ROLEKEY	SSN
dmxxx-009	tIDAdNa3...	1	first-9	last-9	test@test...		1	
Testuser	IGyAQTu...	0	Test	User	anne@itk...	817-433-...	1	433-87-3...
TEST1	nU4eI71b...	1					1	
First1	8FePHnF0...	1	Firstname1	Lastname1	first1@itk...	555-555-...	1	555-55-8...
Last1	i+UhJqb9...	1	Firstname2	Lastname2	last1@itko...	333-448-...	1	533-88-9...
test1	nU4eI71b...	1	First1	Last1	test1@itk...	214-111-...	1	111-11-1...
test2	nU4eI71b...	1	First2	Last2	test2@itk...	215-222-...	1	222-22-2...
admin	0DPiKuNIr...	1	iTKO	Admin	lisabank-a...	123-4567	2	434-47-5...
sbellum	26yJsXNp...	1	Sara	Bellum	sbellum@...	232-4345	1	614-40-1...
wpiece	/UuJ0Me...	1	Warren	Piece	wpiece@...	455-3232	1	546-71-4...
areck	AHDRRjD...	1	Amanda	Reckonwith	areck@my...	555-2244	1	350-02-1...
boaty	RQIil0Ldp...	1	Boaty	Rabbit	boaty@ra...	333-4521	1	616-51-0...
itko	qUqP5cyx...	1	itko	test	itko.test@...	650-234-...	1	140-72-2...
lisa_simpson	60fAFoq+...	1	lisa	simpson	lisa.simps...	123-456-...	1	295-20-0...
virtuser	nU4eI71b...	1	Virtual	User	virtuser@i...	123-456-...	1	297-55-9...
demo	89yJPVNn...	0	DEMOFIRST	DEMOLAST	demo@itk...	817-433-...	1	677234567

2.  アイコンを使用して、「指定の列を、指定の値で検索し、別の列の値を取得します」フィルタを選択します。 このフィルタを作成するには、同じ行内の 2 つのセルを選択します。1 つは検索列で、もう 1 つは値を抽出する列です。
3. 表示されるダイアログ ボックスで、検索と値の列を選択または再割り当てします。
4. プロパティ キー「*theEmail*」を入力します。



5. ［OK］をクリックします。

   DevTest によって、Verify User Added（追加されたユーザの検証）ステップに「**結果セット行の別の値に対する値の取得**」という名前のフィルタが追加されます。



   「結果セット行の別の値に対する値の取得」フィルタは、LOGIN 列で **sbellum** を検索します。 **sbellum** が見つかった場合は、**theEmail** という名前のプロパティ内の同じ行にある EMAIL 列に値を格納します。

注: JDBC 結果セットがステップ エディタに表示される場合は、同じフィルタリング機能を使用できます。

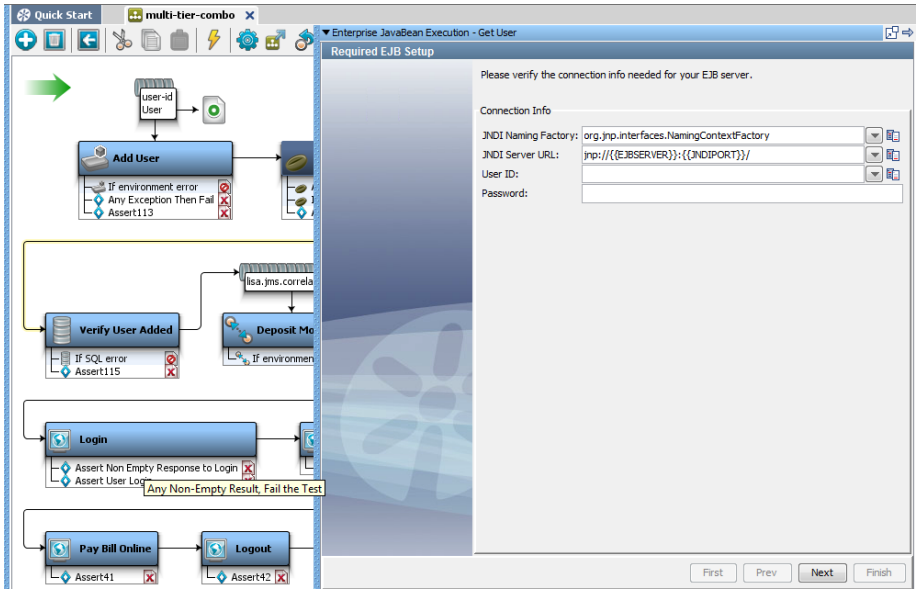
返された Java オブジェクトからのフィルタの追加

テスト ステップの結果が Java オブジェクトである場合、複合オブジェクト エディタ (P. 187)でインライン フィルタ パネルを使用して、メソッド コールからの戻り値を直接フィルタできます。

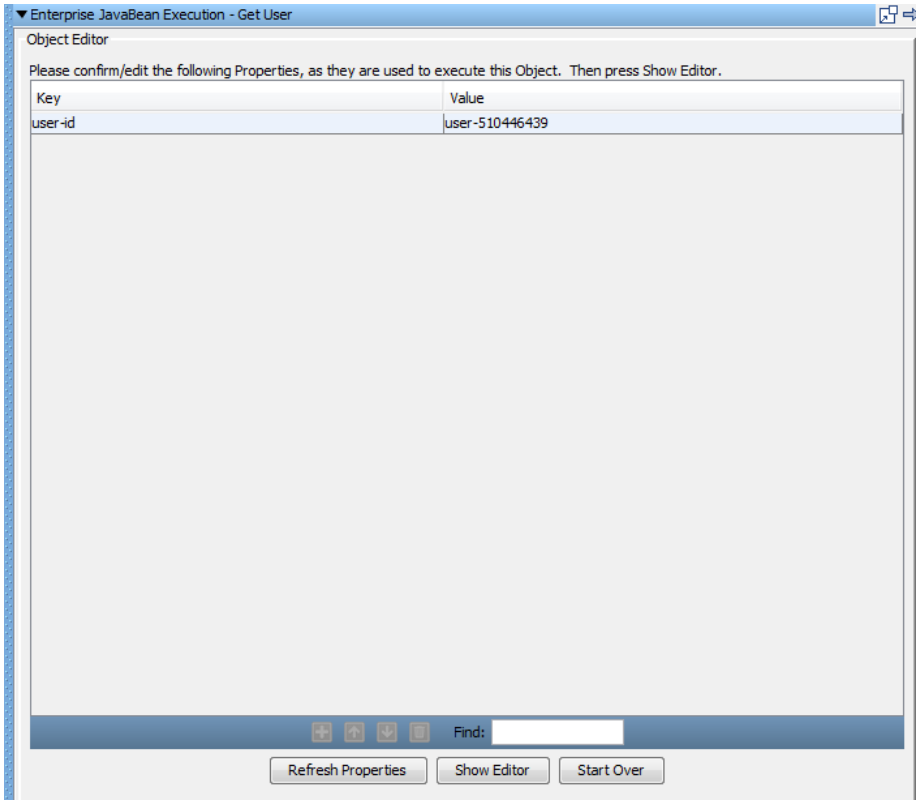
この例では、examples ディレクトリの multi-tier-combo テスト ケースの Get User （ユーザの取得）ステップ（EJB ステップ）を使用します。

次の手順に従ってください:

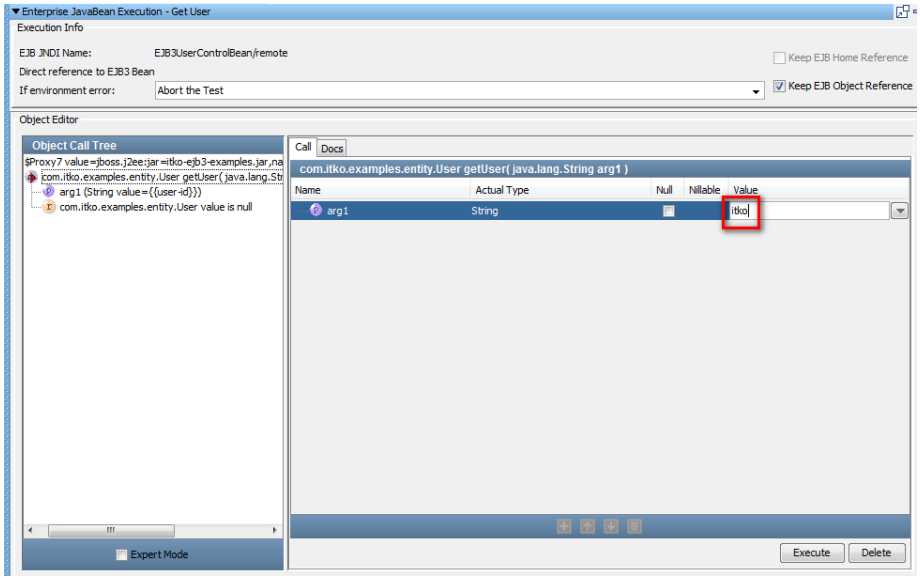
1. Get User （ユーザの取得）ステップのステップ エディタを開くには、ステップをダブルクリックします。



2. ［次へ］をクリックします。次のウィンドウで、［終了］をクリックします。



3. オブジェクト コール ツリーを開くには、［エディタを表示］をクリックします。

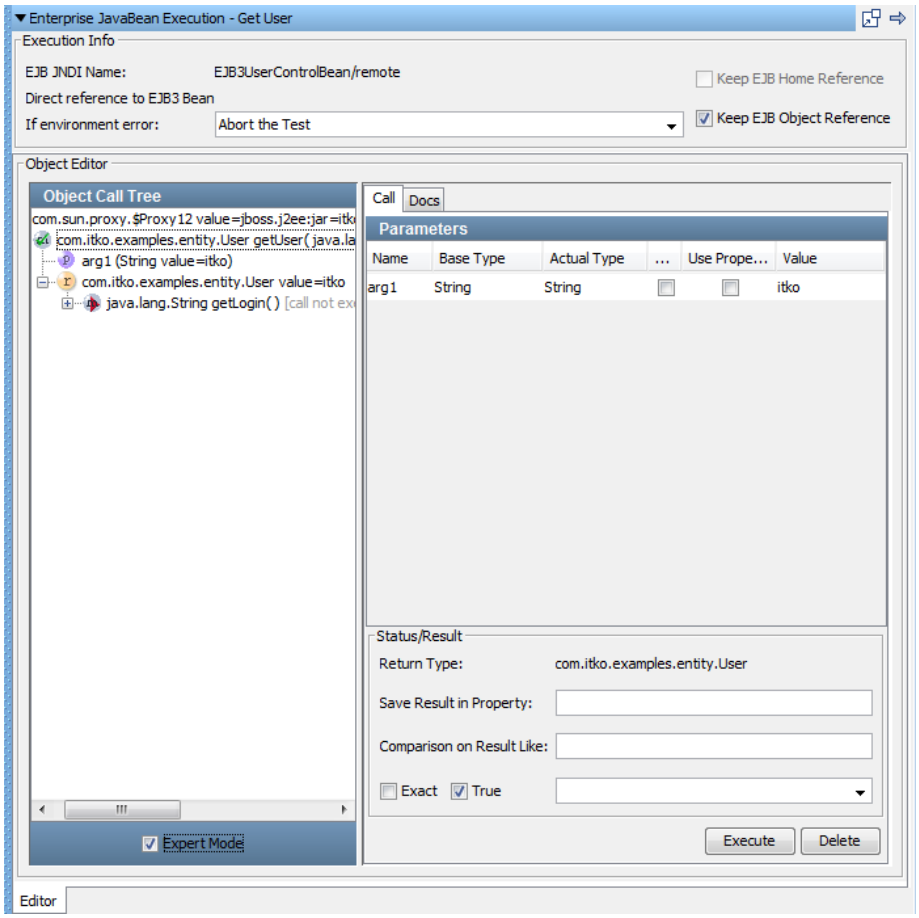


4. 値フィールドに入力パラメータ「itko」を入力します。
5. ［実行］をクリックしてこのメソッドを実行します。

getLogin メソッドの実行時の戻り値は、getUserObject プロパティに格納されます。 この場合、戻り値が（UserState タイプの）オブジェクトであることに注意してください。 ここでアサーションも追加できます。

6. ［エキスパート モード］チェック ボックスをオンにします。

   ［コール］タブにフィルタ パラメータが表示されます。



返されたオブジェクトに対してメソッドを呼び出して、このユーザのログイン値を取得し、別のプロパティにログインを保存することもできます。

インライン フィルタ（およびアサーション）は、エレメント ツリーのテスト ステップに追加されるフィルタにはなりません。 インライン フィルタの管理は、常に複合オブジェクト エディタで行います。

複合オブジェクト エディタの詳細については、「*CA Application Test の使用*」の「複合オブジェクト エディタ (P. 187)」を参照してください。

フィルタのドラッグ アンド ドロップ

モデル エディタでは、あるテスト ステップから別のテスト ステップにフィルタをドラッグ アンド ドロップできます。

次の手順に従ってください:

1. テスト ステップに関連付けられているフィルタ（たとえば Step1）をクリックします。
2. そのフィルタをモデル エディタ内の別のテスト ステップ（たとえば Step2）にドラッグ アンド ドロップします。

   ドラッグされたフィルタが Step2 に適用されます。

アサーション

アサーションは、あるステップとそのすべてのフィルタが実行された後に実行される DevTest コード エレメントです。アサーションにより、ステップの結果が予測と一致することが検証されます。

アサーションの結果はブール値（true または false）です。

結果によってテスト ステップが成功したか失敗したか、また、テスト ケース内の次に実行するステップを判断できます。 アサーションは、ワークフローに条件ロジック（分岐）を導入することにより、テスト ケースのワークフローを動的に変更するために使用します。 アサーションは、プログラムの条件ブロックの「if」と非常によく似た動作をします。

たとえば、特定の名前が結果セット内の 1 行だけに含まれることを確認するアサーションを JDBC ステップに作成できます。JDBC ステップの結果に複数の行が含まれていた場合は、アサーションは実行する次のステップを変更します。 このように、アサーションでは条件分岐機能を実行できます。

テスト ケース フローは、多くの場合、以下の 2 つの可能性のいずれかによってモデル化されます。

- 各ステップに対して定義される次のステップは、テスト ケースにおける次の論理的なステップである。この場合アサーションは失敗を示している。
- 次のステップは失敗として設定されており、アサーションはすべて次の論理的なステップを示している。

その選択は、通常は使用されている実際のロジックに依存します。

注: アサーションが未解決のプロパティを参照した場合は、モデル定義エラーが発生します。 モデル定義エラーによってテストは終了しませんが、これはテストの作成者に未解決のプロパティを参照したことを警告するものです。 未解決のプロパティが作成されるという問題は、アサーションに十分な情報がないため、アサーションで適切な評価を下すことができないことを意味します。 十分な情報がないことは、擬陽性または擬陰性の結果が生じることにつながります。 （未解決のプロパティを参照した場合、ほとんどのアサーションは「false」を返しますが、これは強制されたルールではありません。）テストを ITR で実行し、テスト イベント パネルでモデル定義エラーを確認することによって、未解決のプロパティが存在するかどうかを判断できます。

アサーションは必要なだけ追加して、複雑なワークフローを構築することができます。 アサーションは、DevTest のワークフローを変更できる唯一のオブジェクトです。

注: アサーションはそれらが出現する順に実行されます。また、ワークフロー ロジックは、通常はアサーションが適用される順番によって変化します。

アサーションが起動された後、そのアサーションの決定に従って次のステップを設定できます。また、残りのアサーションは無視されます。 アサーションが評価および起動されるたびに、イベントが生成されます。

グローバル アサーションとステップ アサーション

フィルタと同様に、アサーションをグローバルに適用できます。つまり、アサーションは、テスト ケース サイクル全体に適用するか、ステップ アサーションとして特定のステップにのみ適用することができます。

このセクションには、以下のトピックが含まれます。

アサーションの追加

テスト ケースにアサーションを追加するには、複数の方法があります。

アサーションの手動による追加

アサーションを手動で追加するには、リストからアサーション タイプを選択して、そのアサーションのパラメータを入力します。 手動によるアサーションには以下の 2 つのタイプがあります。

- グローバル アサーションはテスト ケース レベルで定義されます。 グローバル アサーションはテスト ケースのすべてのステップに適用され、ノードに特に指定がない限り、各ステップに対して自動的に実行されます。
- ステップ アサーションはテスト ステップ レベルで定義されます。 このタイプのアサーションはそのステップにのみ適用され、そのステップに対してのみ実行されます。

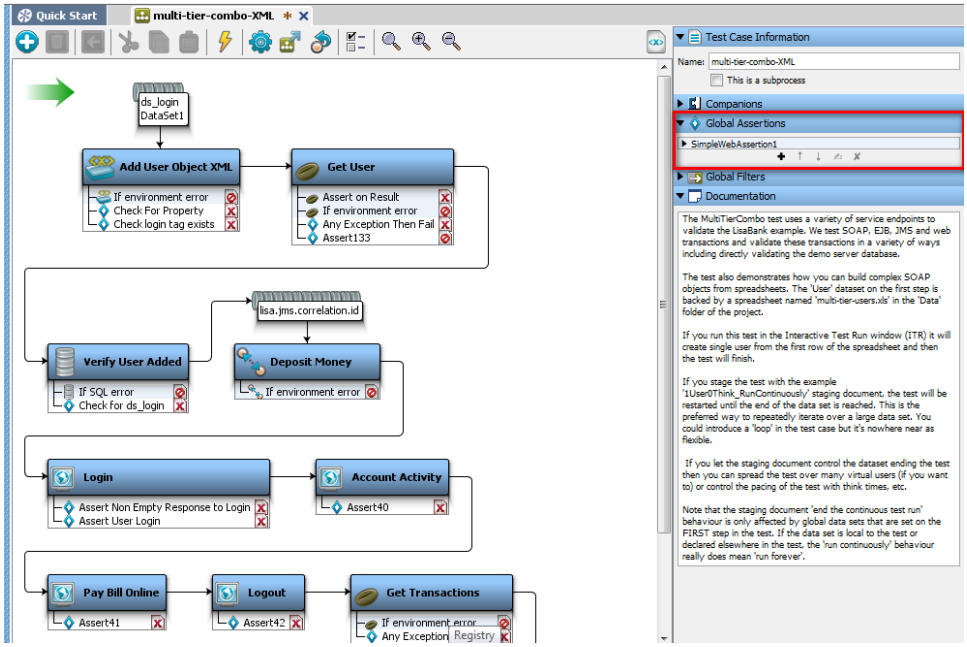
グローバル アサーションを追加する方法

1. テスト ケースを開き、右側のパネルで［グローバル アサーション］エレメントをクリックします。

以下のタイプのグローバル アサーションを適用できます。

- **HTTP**
  - シンプル Web アサーション
  - Web 応答のリンクを確認
- **XML**
  - ステップ応答時間を確認
- **その他**
  - 結果が式を含むことを確認
  - ステップ応答時間を確認
  - ファイルのコンテンツをスキャン

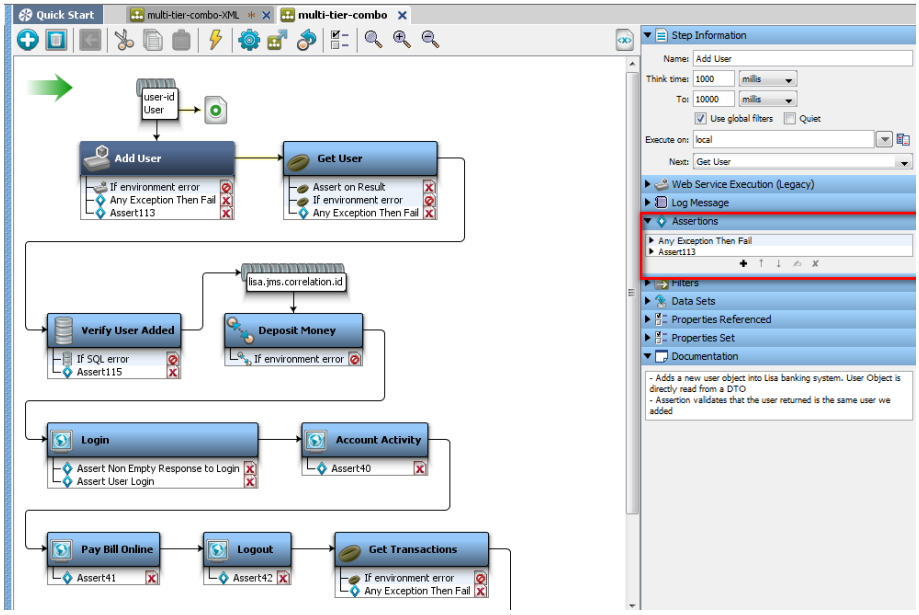
以下の図は、multi-tier-combo テスト ケースに適用されているグローバル アサーションを示しています。



ステップ アサーションを追加する方法

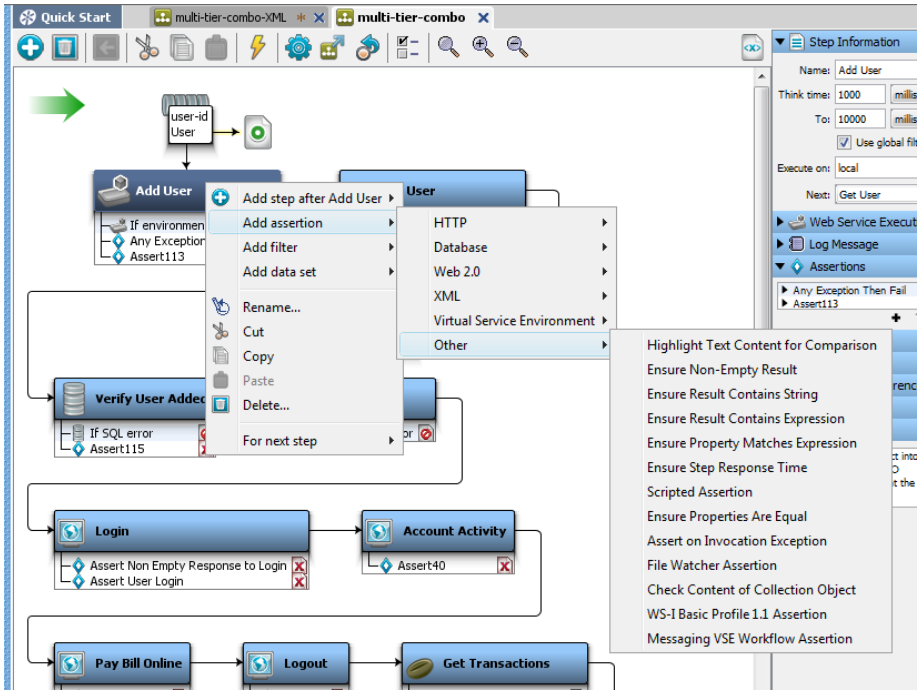
1. 以下のアクションのいずれかを完了します。
   - アサーションを適用するステップを選択し、右側のパネルで［アサーション］エレメントをクリックします。
   - ステップを右クリックし、［アサーションの追加］を選択して、ステップの適切なアサーションを選択します。

   以下の図は、multi-tier-combo テスト ケースのユーザの追加ステップに適用されているステップ アサーションを示しています。



2. アサーションを追加するには、アサーション ツールバーの [追加]  をクリックします。

   または、モデル エディタでステップを右クリックしてアサーションを追加することもできます。アサーション パネルが表示され、ステップに適用できるアサーションのメニューが表示されます。



アサーションを選択および編集する方法

1. 以下のアクションのいずれかを完了します。
   - アサーションが適用されるステップをクリックします。
   - ［アサーション］タブでそのステップに関連するアサーションをクリックします。
2. アサーション エディタを開くには、アサーションをダブルクリックします。 エディタはアサーションのタイプごとに異なります。

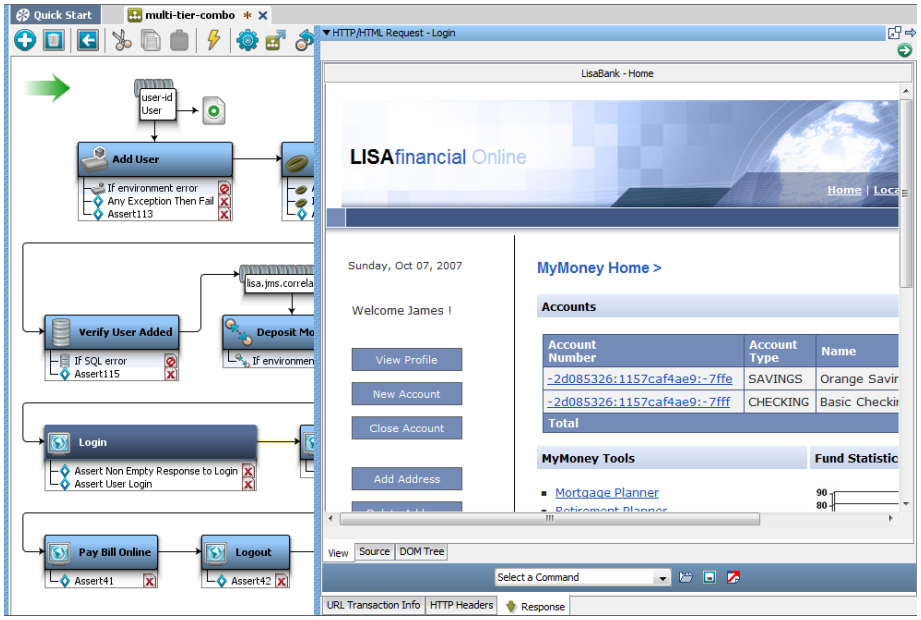
HTTP 応答からのアサーションの追加

HTTP ベースのステップからの応答にアクセスできる場合は、応答を使用してアサーションを直接追加できます。

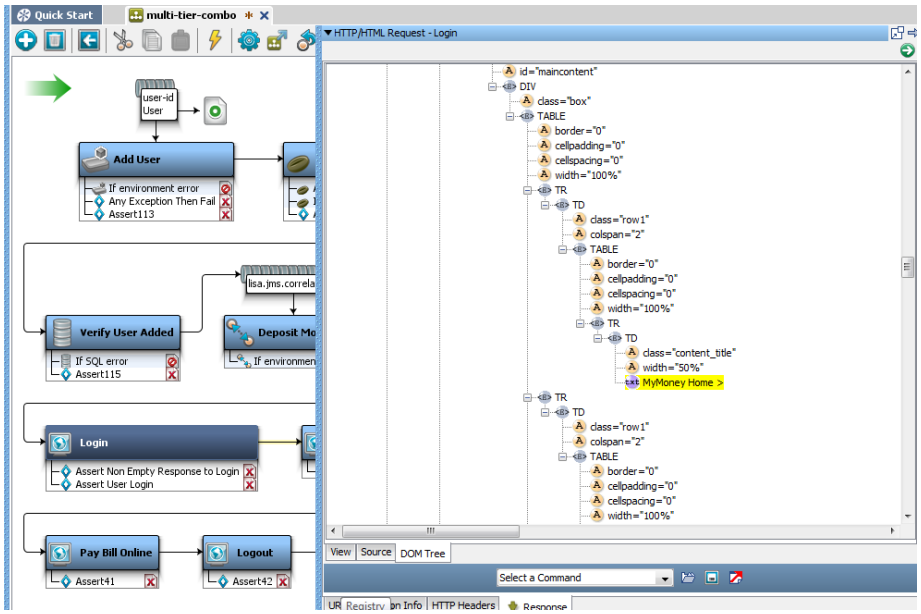
この HTTP/HTML 応答の例では、multi-tier-combo テスト ケースのログイン ステップを使用します。この例のポイントは、応答に「MyMoney ホーム」というテキストが表示されるかどうかをテストすることです。

次の手順に従ってください:

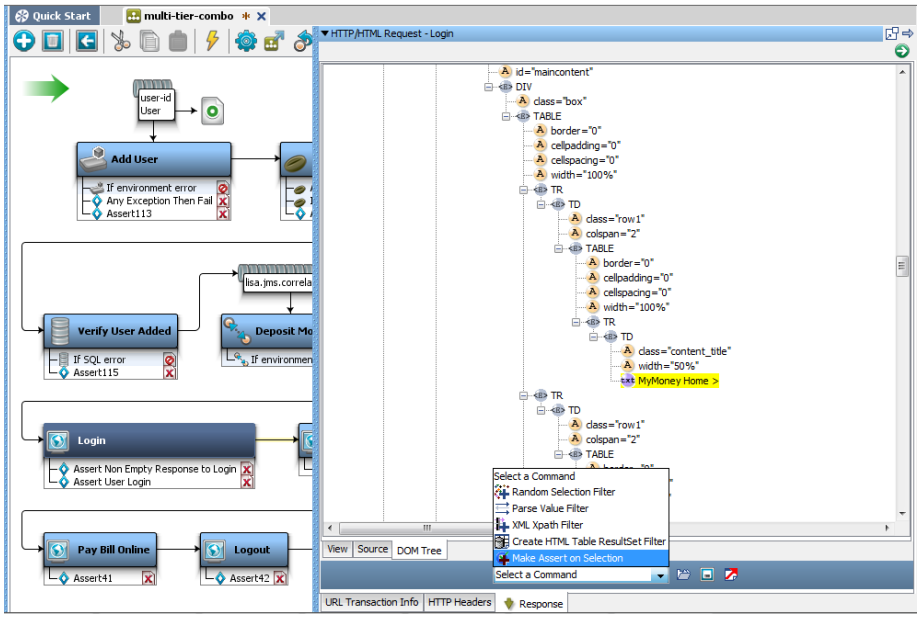
1. ITR で multi-tier-combo テスト ケースを実行します。
2. モデル エディタでログイン ステップをダブルクリックします。



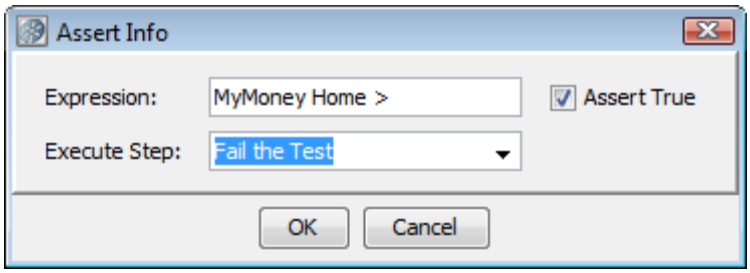
3. ［ビュー］タブ内の「MyMoney ホーム」というテキストを選択します。
4. このテキストがツリー で選択されていることを確認するために、［DOM ツリー］タブをクリックします。



5. パネルの下部の［コマンドの選択］プルダウン メニューから、［選択範囲にアサートを作成］を選択します。



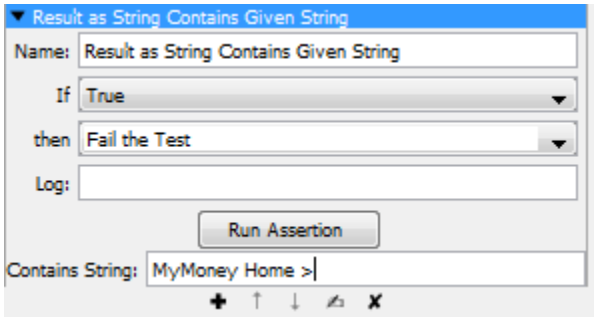
6. 表示されるウィンドウで、選択したテキストに一致させる式を入力し、適切なアサーションの動作を選択します。



この例では、「MyMoney ホーム」というテキストが存在しない場合、アサーションが起動され、失敗ステップにリダイレクトされます。

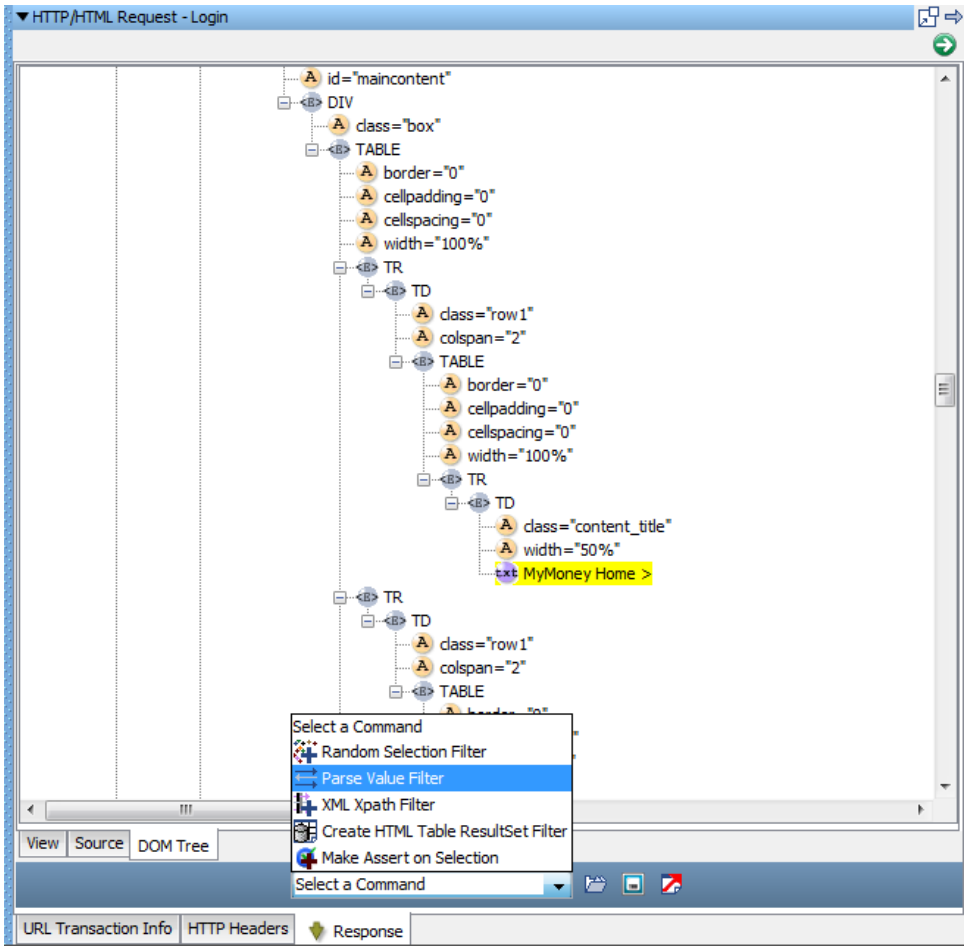
7. ［OK］をクリックして、アサーションを保存します。

生成されたアサーションは、ログイン ステップのアサーションとして表示されます。

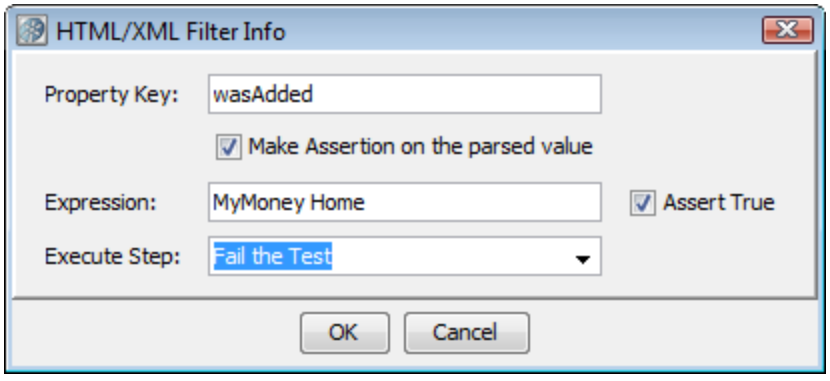


1 つのフィルタと 1 つのアサーションの実行

あるいは、フィルタで「MyMoney ホーム」という値をキャプチャし、アサーションとしてそれを実行する場合は、その両方が可能である解析値フィルタを使用します。



解析値フィルタによって表示されるウィンドウには、［プロパティ キー］の値に適用されるフィルタ、［式］に起動されるアサーションが表示されます。



結果として、1 つのフィルタおよび 1 つのアサーションがログイン ステップに追加され、モデル エディタに表示されます。

注: HTML 応答がステップ エディタに表示される場合は、同じアサーション機能を使用できます。

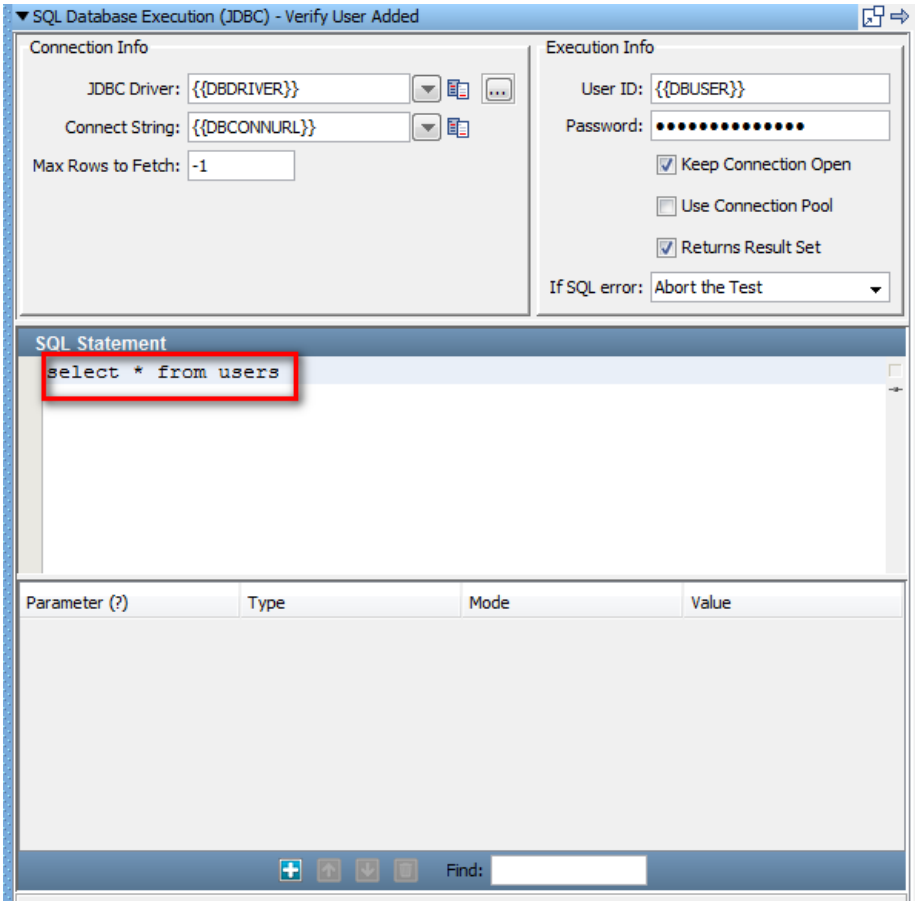
JDBC 結果セットからのアサーションの追加

JDBC ステップからの結果セット応答にアクセスできる場合は、応答を使用してアサーションを直接追加できます。 以下の例では、このようにアサーションを追加する方法を示します。

この例では、examples ディレクトリの multi-tier-combo テスト ケース（multi-tier-combo.tst）の Verify User Added （追加されたユーザの検証）ステップの応答を使用する結果セット応答の例を示します。

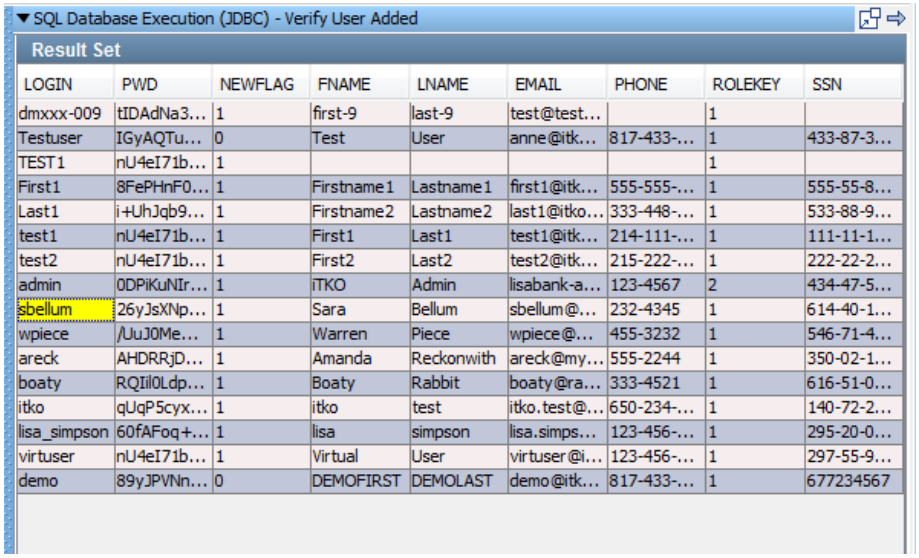
次の手順に従ってください:

1. Verify User Added （追加されたユーザの検証）ステップを選択し、ダブルクリックしてそのエディタ ウィンドウを開きます。 SQL ステートメントを「**select * from users**」に編集します。



2. クエリを実行するには、［SQL をテスト/実行］ボタンをクリックします。

3. ［結果セット］タブを選択し、テストする情報を表す結果セット内のセル（たとえば、**sbellum**）をクリックします。

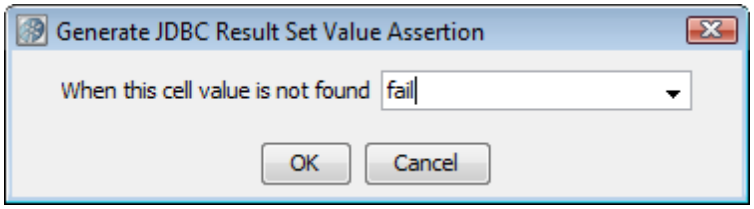


LOGIN	PWD	NEWFLAG	FNAME	LNAME	EMAIL	PHONE	ROLEKEY	SSN
dmxxx-009	tIDAdNa3...	1	first-9	last-9	test@test...		1	
Testuser	IGyAQTu...	0	Test	User	anne@itk...	817-433-...	1	433-87-3...
TEST1	nU4eI71b...	1					1	
First1	8FePHnF0...	1	Firstname1	Lastname1	first1@itk...	555-555-...	1	555-55-8...
Last1	i+UhJqb9...	1	Firstname2	Lastname2	last1@itko...	333-448-...	1	533-88-9...
test1	nU4eI71b...	1	First1	Last1	test1@itk...	214-111-...	1	111-11-1...
test2	nU4eI71b...	1	First2	Last2	test2@itk...	215-222-...	1	222-22-2...
admin	0DPiKuNIr...	1	iTKO	Admin	lisabank-a...	123-4567	2	434-47-5...
sbellum	26yJsXNp...	1	Sara	Bellum	sbellum@...	232-4345	1	614-40-1...
wpiece	/UuJ0Me...	1	Warren	Piece	wpiece@...	455-3232	1	546-71-4...
areck	AHDRRjD...	1	Amanda	Reckonwith	areck@my...	555-2244	1	350-02-1...
boaty	RQIil0Ldp...	1	Boaty	Rabbit	boaty@ra...	333-4521	1	616-51-0...
itko	qUqP5cyx...	1	itko	test	itko.test@...	650-234-...	1	140-72-2...
lisa_simpson	60fAFoq+...	1	lisa	simpson	lisa.simps...	123-456-...	1	295-20-0...
virtuser	nU4eI71b...	1	Virtual	User	virtuser@i...	123-456-...	1	297-55-9...
demo	89yJPVNn...	0	DEMOFIRST	DEMOLAST	demo@itk...	817-433-...	1	677234567

4. ［結果セット］ウィンドウの下部にあるツールバーの  ［セルの値に対するアサーションの生成］をクリックします。

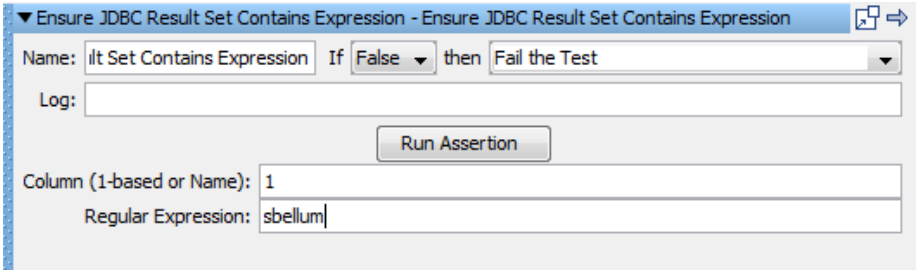
   LOGIN 列 のセルに表示される「**sbellum**」をテストします。

5. ［JDBC 結果セット値アサーションの生成］ダイアログ ボックスで、値が見つからない場合にリダイレクトするテスト ステップ（失敗）を入力します。



   DevTest は、Verify User Added （追加されたユーザの検証）ステップに「JDBC 結果セットに式が含まれることを確認」という名前のアサーションを作成します。

数式 1: アサーション：JDBC 結果セットからのアサーションの追加 - 追加されたアサーション



注: JDBC 結果セットがステップ エディタに表示される場合は、同じアサーション機能を使用できます。

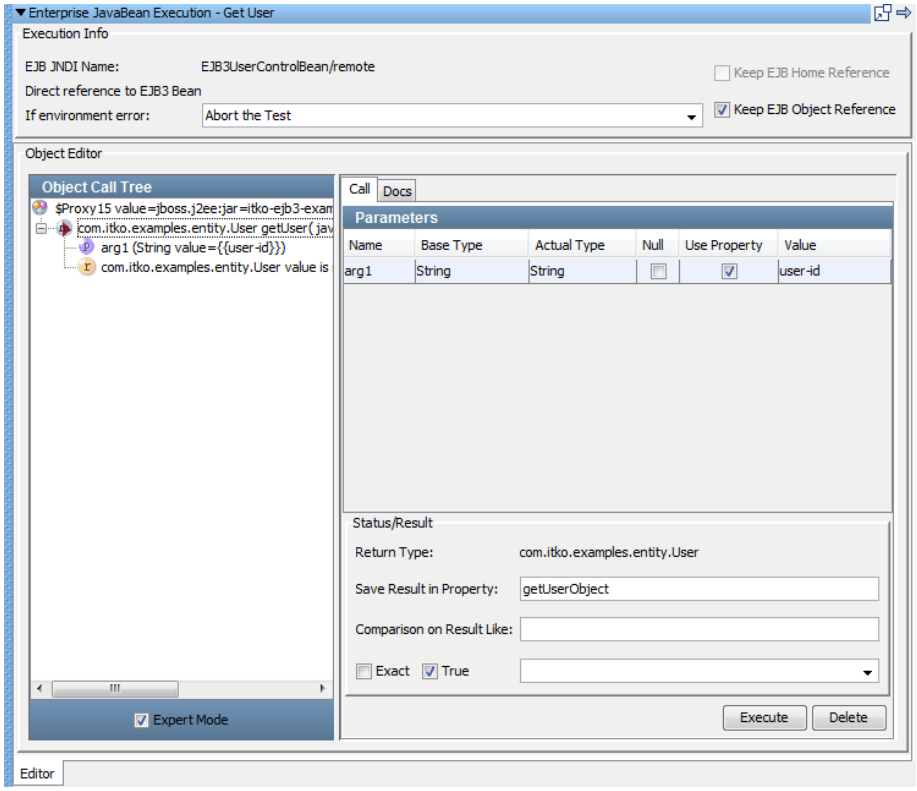
返された Java オブジェクトへのアサーションの追加

テスト ステップが Java オブジェクトを返す場合、複合オブジェクト エディタのインライン アサーション パネルを使用して、メソッド コールからの戻り値にアサーションを直接追加できます。 以下の例では、このようにアサーションを追加する方法を示します。

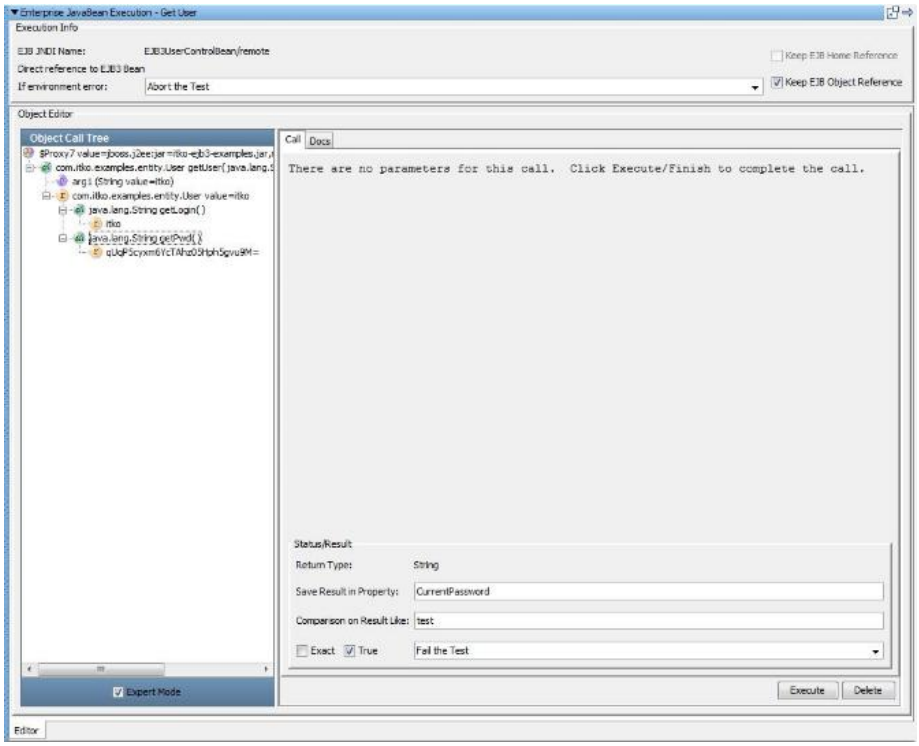
この例では、examples ディレクトリの multi-tier-combo テスト ケースの Get User （ユーザの取得）ステップ（EJB ステップ）を使用します。

次の手順に従ってください:

1. 入力パラメータ **itko** を入力し、メソッド コール getUser を実行します。次に、そのコールから返された UserState オブジェクトに getPwd コールを実行します。



2. アサーションを追加できる［ステータス/結果］ペインを開くには、左ペインの［エキスパート モード］チェック ボックスをオンにします。



getPwd メソッドの実行時の戻り値は、CurrentPassword プロパティに格納されます。

3. 戻り値が「test」という文字列に等しいかどうかをテストするアサーションを追加します。 等しくない場合は、失敗ステップにリダイレクトされます。
4. ［実行］をクリックしてこのステップを実行します。

注: インライン アサーション（およびフィルタ）では、アサーションはテスト ステップに追加されません。 インライン アサーションの管理は、常に複合オブジェクト エディタで行います。

複合オブジェクト エディタの詳細については、「*CA Application Test の使用*」の「複合オブジェクト エディタ（COE） (P. 187)」を参照してください。

アサーションのモバイル テスト ステップへの追加

以下のものにアサーションを適用できます。

- モバイル テスト ステップ。
- テスト ステップに含まれる各アクション。

  注: 1 つのテスト ステップには、ステップを完了するために必要な数だけジェスチャ（タップ、スワイプ）が含まれます。 テスト ステップをダブルクリックすると、これらのサブステップ（アクション）が［アクション］テーブルに表示されます。

次の手順に従ってください:

1. 更新するモバイル テスト ケースを開きます。
2. 各ステップの詳細を表示するには、以下を実行します。
   a. 確認するテスト ステップをクリックします。
   b. 右側のエレメント ツリーの［モバイル テスト ステップ］をクリックして、ステップの詳細を展開します。 テスト ステップをダブルクリックすることもできます。

      ［モバイル テスト ステップ］タブが開き、テスト アプリケーションのスクリーンショットが表示されます。タブの上部の［アクション］セクションには、テスト ステップで実行される個別のアクションが表示されます。
   c. 特定のアクションと関連付けられたスクリーンショットを表示するには、［アクション］セクション内のアクションをクリックします。
3. タブの下部にあるスクリーンショットまたは特定の画面エレメントを右クリックします。
4. ［アサーションの追加］をクリックし、追加するアサーションを選択します。

   選択したアサーションがテスト ステップに追加されます。
5. ［保存］をクリックします。

詳細情報:

アサーションの追加 (P. 152)

アサーションの選択および編集

次の手順に従ってください:

1. モバイル テスト ステップ エディタを閉じます。
2. 右側のエレメント ツリーの［アサーション］をクリックして、選択したテスト ステップのアサーションを表示します。

   ［アサーション］タブが展開されます。
3. 変更するアサーションをダブルクリックします。

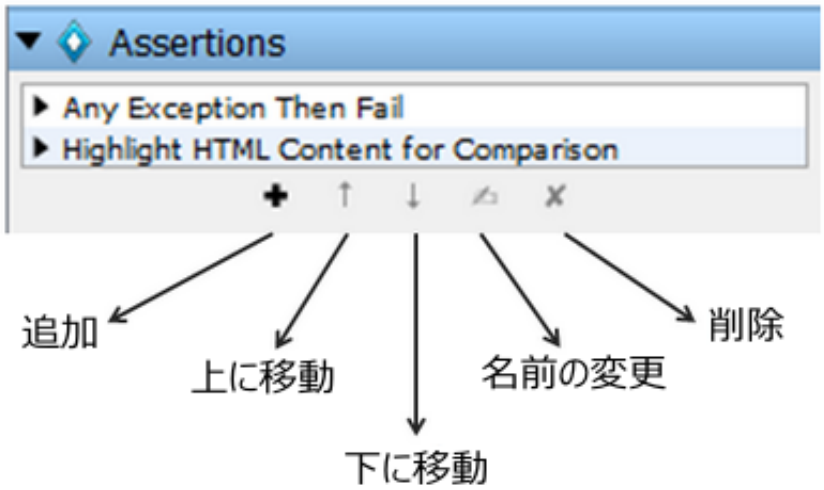
   アサーション エディタが開きます。
4. 選択したアサーションのフィールドに入力します。

   エディタはアサーションのタイプごとに異なります。特定のタイプのアサーションの詳細については、以下のトピックを参照してください。

   - Ensure Same Mobile Screen （モバイル画面が同一であることを確認）
   - Ensure Screen Element Matches Expression (式に一致する画面エレメントを確認)
   - Ensure Screen Element is Removed （画面エレメントの削除を確認）
5. ［保存］をクリックします。

アサーション ツールバー

すべてのエレメントには、追加/削除/並べ替えのための固有のツールバーがエレメントの下部にあります。



アサーションの並べ替え

アサーションは出現する順に評価されるため、アサーションを並べ替える必要がある場合があります。アサーションの順番の変更は、ワークフローに影響する可能性があります。

アサーションを並べ替える方法

- ［エレメント］タブでアサーションを選択し、ツールバーの ［上に移動］または ［下に移動］をクリックします。
- モデル エディタで、アサーションを目的の位置へドラッグ アンド ドロップします。

アサーションの名前の変更

次の手順に従ってください:

- アサーションを選択し、右クリックしてメニューを開きます。［名前の変更］をクリックし、アサーションの名前を変更します。
- アサーションを選択し、ツールバーの［名前の変更］アイコンをクリックします。
- デフォルトのアサーション名にアサーション名を戻すには、アサーションを選択して 名前をデフォルトに戻すボタンをクリックします。

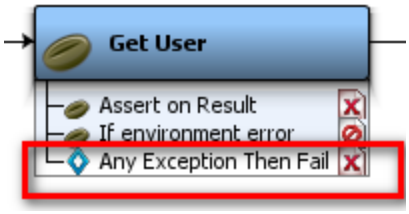
アサーションのドラッグ アンド ドロップ

モデル エディタ内のアサーションをあるテスト ステップから別のテスト ステップにドラッグ アンド ドロップする方法

1. あるテスト ステップ（たとえば Step1）のアサーションをクリックします。
2. アサーションを選択し、モデル エディタ内のその他のテスト ステップ（たとえば Step2）へドラッグします。
3. ドラッグされたアサーションが Step2 に適用されます。

アサーションの次のステップの設定

ステップに追加されたアサーションは、モデル エディタで表示できます。



アサーションがステップに追加された後、ワークフローを変更する場合は、実行される次のステップを選択できます。

アサーションの次のステップを設定する方法

1. メニューを開くには、モデル エディタ内のアサーションを右クリックします。
2. ［トリガされた場合］を選択し、以下のいずれかを実行します。
   - 警告またはエラーを生成することを選択します。
   - ステップの終了、失敗、または中止を選択します。
   - 実行される次のステップを選択します。

データ セット

データ セット (P. 616)は、テストの実行時にテスト ケースにプロパティを設定するために使用できる値のコレクションです。 この機能によって、テスト ケースに外部のテスト データを使用することができます。

データ セットは、多くの場合、名前/値ペアとしてプロパティに挿入できるデータの行です。 ただし、データ セットが単一のプロパティ値を返す場合もあります。

データ セットは、DevTest の内部または外部（たとえば、ファイルやデータベース テーブル）に作成できます。

テストの実行時に、DevTest はデータ セット エディタで指定されたステップにプロパティを割り当てます。 最後のデータ値がデータ セットから読み取られると、以下のいずれかのアクションが発生する場合があります。

- データが再利用可能となり、データ セットの先頭から開始される。
- テスト ケースの任意のステップにテストをリダイレクトできる。

データ セットは、グローバルまたはローカルです。

以下のトピックが含まれます。

グローバル データ セットおよびローカル データ セット

データ セットは、グローバルまたはローカルです。

グローバル データ セット

デフォルトでは、データ セットはグローバルです。

コーディネータ サーバは、すべてのテスト ステップにデータを提供する役割を担います。

ここでは、すべてのモデル インスタンスは、データ セットの単一のインスタンスを共有しています。

異なるシミュレータで実行される場合でも、グローバル データ セットは共有され、モデルのすべてのインスタンスに適用されます。

ローカル データ セット

データ セットの作成時に［ローカル］チェック ボックスをオンにすることで、データ セットをローカルにできます。

各インスタンスは、データ セットの（実質的に）固有のコピーを取得します。

ローカル データ セットは、実行中の各インスタンスにデータ セットのコピーを 1 つ提供します。

例

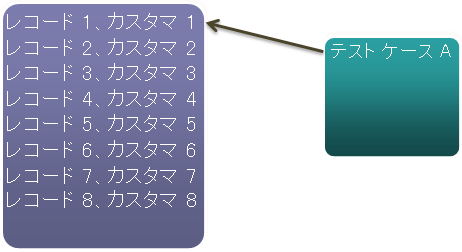
この例には、以下のコンポーネントが含まれます。

- 3 人の同時仮想ユーザ
- 100 行のデータのあるローカル データ セット
- 100 行のデータをループし、停止するテスト ケース

仮想ユーザはそれぞれ、データ セットの 100 行のデータをすべて参照します。

ローカル データ セットの場合

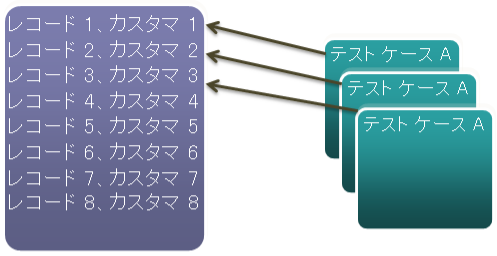
単一の実行（1 仮想ユーザ）：テスト ケース A、最初の行のデータ（レコード 1、カスタマ 1）を取得する


レコード 1、カスタマ 1
レコード 2、カスタマ 2
レコード 3、カスタマ 3
レコード 4、カスタマ 4
レコード 5、カスタマ 5
レコード 6、カスタマ 6
レコード 7、カスタマ 7
レコード 8、カスタマ 8
テスト ケース A

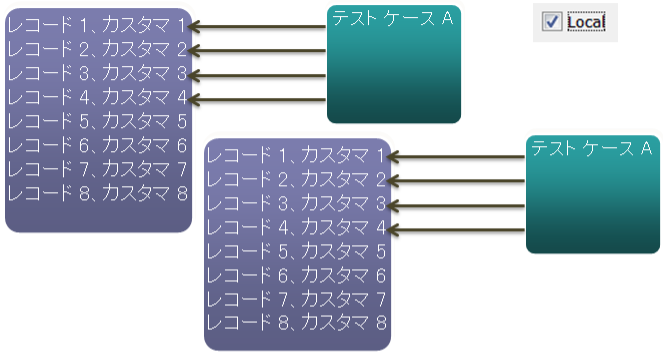
仮想ユーザ全体で共有されるデータ セットはグローバルです。

グローバル データ セットの場合

テスト ケース A が 3 人の仮想ユーザによってステージングされる場合、または継続的に実行される場合、テスト ケースはそれぞれデータの次の行を受け取ります。 DevTest は、ステージング ドキュメントで指定されている手順を使用して、複数の実行にわたってデータ セットを共有します。



各仮想ユーザ（インスタンス）は、ローカルであるデータ セットの固有のコピーを取得します。



ローカルがマークされ、テスト ケース A にループがある場合、テストはデータ セット内のデータのすべての行を読み取ります。 各実行は、データ セットの固有のコピーを取得します。

テスト ケースのループ

多くの場合、データ セット内のデータはテスト ケースが実行される回数を制御します。

ループを実装する方法は複数あります。

- テストは、データ セット内のデータがすべて使い果たされたときに完了するように設定されています。
- テストは、データが使い果たされたときにデータ セットを再利用するように設定されています。
- テスト ステップはそれ自体をコールできます。または、データ セット内のデータが使い果たされるまでループする一連のステップを設定できます。

特定のステップを特定の回数実行させるために数値カウンタ データ セットを使用できます。ステップ レベルまたはテスト ケース レベルで頻度を設定できます。

たとえば、100 のユーザ ID/パスワード ペアを使用して、アプリケーションのログイン機能をテストする場合を考えます。 このテストを実現するには、単一のステップを、データ セット（100 行のデータ）が使い果たされるまでそのステップ自体をコールするように設定します。 データ セットが使い果たされる場合、テストは、テスト ケースの次の通常のステップにリダイレクトするか、終了ステップにリダイレクトすることができます。 または、ユーザ ID/パスワード データ セットと一緒にカウンタ データ セットを使用することもできます。

テスト ケースの最初のステップのグローバル データ セットを、データ セットが使い果たされたときにテストを終了するように設定する場合、ステージングされた実行のすべてのテスト インスタンスが終了します。

これにより、安定状態時間などのその他のステージング パラメータが上書きされます。 ローカル データ セットでは、ステージングされた実行をこのように終了したり、最初のステップ以外のステップにデータを設定したりしません。

ランダム データ セット

ランダム データ セットは、特殊なタイプのデータ セットです。これは、別のデータ セットのラッパーと考えることができます。

特定のステップでデータ セットをランダムに選択できます。また、ランダムなレコードの最大数も選択できます。

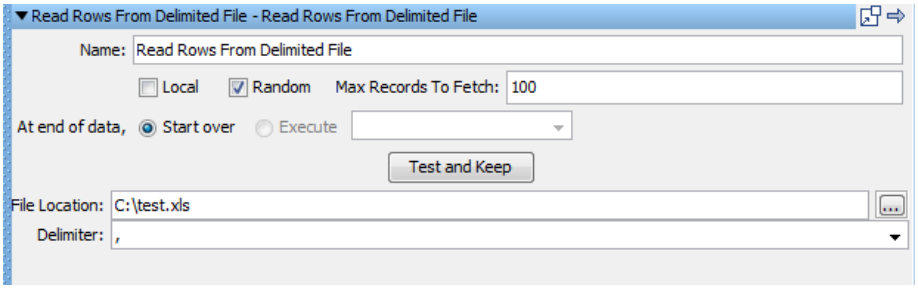
データ セットをランダムにラップする場合、データ セットの *n* 個のコピーがランダムにリストに追加されます（*n* = 最大行数）。 したがって、*n*=10 の場合、DevTest は、データ セットの行の 10 のコピーを作成します。

ステップがランダム データ セットを参照する場合、データ セットから行がランダムに選択されます。

「**RAND1**」という名前のランダム データ セットがあり、それが「**Fruit**」という名前の列を読み取り、［取得する最大レコード数］が「10」に設定されている場合、ランダムな数字の **RAND1** が行を選択するために生成されます。 **RAND1** の範囲は、0 ～ *n*-1 になります。 Fruit は、その行の「Fruit」列の値です。

データ セットをランダムにする方法

1. ランダムにするデータ セットを選択します。 この例では、「区切りデータ ファイルからの読み取り」を使用します。



2. ［ランダム］チェック ボックスをオンにします。
3. ［取得する最大レコード数］に数値を入力します。

   この数値は、データ セットから使用するレコードの最大数です。この数値がデータ セット内のレコード数より大きい場合、小さい数値を使用してランダムなセットが作成されます。

シナリオ例

データ セットを使用してテストの動作を確認するには、以下のシナリオを検討します。

15 人の仮想ユーザ、および 2 行のデータがあるデータ セットを使用するテスト。

シナリオ 1: データが終了した場合 -> やり直す

1 番目のユーザはデータの 1 番目の行を読み取り、2 番目のユーザは 2 番目の行を読み取ります。 3 番目のユーザはやり直し、1 番目の行を読み取り、これが繰り返されます。15 人のユーザ全員がテストを実行するまで、これが継続されます。 1 番目の行は 8 回読み取られ、2 番目の行は 7 回読み取られます。

シナリオ 2: データが終了した場合 -> 実行:テストを終了するステップ

1 番目のユーザはデータの 1 番目の行を読み取り、2 番目のユーザは 2 番目の行を読み取ります。 3 番目のユーザから 15 番目のユーザはテストを開始し、すぐに終了ステップにジャンプします。

100 人の仮想ユーザ、および 1500 行のデータがあるデータ セットを使用するテスト。テストは同時に実行されます。

シナリオ 1: データが終了した場合 -> やり直す

100 人のユーザが開始すると、これらのユーザはデータの最初の 100 行を読み取ります。 ステージング ドキュメントに応じて、サイクル終了時にユーザはテスト ケースの新しい実行を開始し、データ セットのより多くの行を処理します。 行がすべて処理されてもテスト ランが終了していない場合は、テスト ランが終了するまで、もう一度 1 番目の行からやり直します。

シナリオ 2: データが終了した場合 -> 実行:テストを終了するステップ

テスト ランは 1500 サイクルの後に終了します。

10 人の仮想ユーザ、および 10,000 行のデータがあるデータ セットを使用するテスト。ステージング ドキュメントでは、テストを 2 分間実行するように指定されています。

10 人のユーザが開始し、データの最初の 10 行を読み取ります。 ユーザは 10,000 の行からデータ行を処理し続けます。実行速度に応じて、どの時点でデータ セットが終了するかが決まります。2 分が経過すると、テストは終了します。

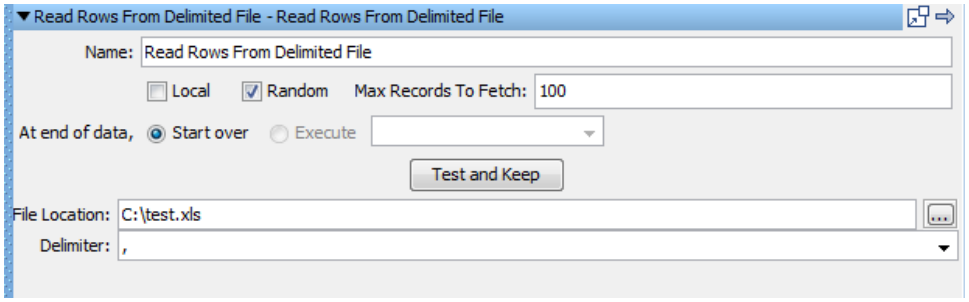
データ セットの追加

次の手順に従ってください:

1. モデル エディタでテスト ケースを選択します。
2. 右側のパネルの［データ セット］タブを展開します。
3. ［追加］をクリックして、共通データ セットが表示されるデータ セット パネルを開きます。
4. 必要なデータ セットをクリックして、適切なデータ セット エディタを開きます。 エディタは各データ セットに固有です。

データ セット エディタの例

以下の図は、区切りデータ ファイルからの読み取りデータ セットのエディタを示しています。



データ セット エディタの上部パネルは、すべてのデータ セット タイプに共通です。 オプションは、以下のとおりです。

名前

データ セットの名前。

ローカル

ローカル データ セットを使用する場合は、このチェック ボックスをオンにします。 デフォルトはグローバルです（オフ）。

ランダム

データ セットをランダム データ セットにする場合は、［ランダム］チェック ボックスをオンにし、取得する最大レコード数を入力します。

データが終了した場合

すべてのデータを読み取った後に続行する方法の指示。 以下の操作を行うことができます。

やり直す

データ セットの先頭からデータの読み取りを続行します。

実行

すべてのデータを読み取った後に実行するステップを選択します。プルダウン メニューには、テスト ケースで使用可能なすべてのステップがあらかじめ入力されています。

テスト アンド キープ

すべてのパラメータを入力した後、このボタンをクリックしてデータ セットをテストし、テスト ケースのステップにロードします。

データ セット エディタの下部パネルは、作成するデータ セットに固有です。 区切りデータ ファイルからの読み取りデータ セットの場合、以下を入力します。

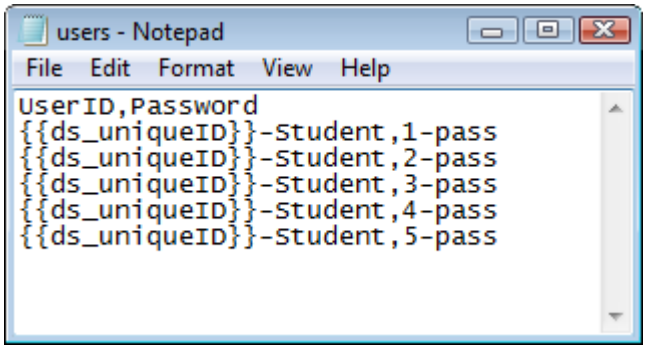
ファイルの場所

テキスト ファイルのフル パス名を入力するか、または［参照］ボタンを使用してファイルを参照します。パス名にはプロパティを使用できます（例： LISA_HOME）。

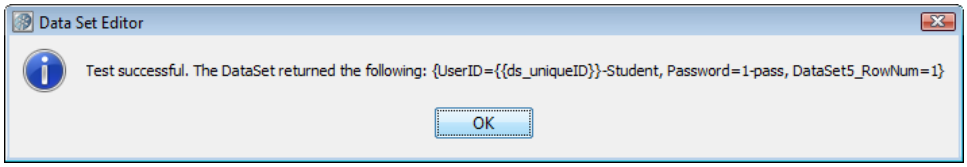
区切り文字

使用する区切り文字を入力します。任意の値を区切り文字として使用できます。ドロップダウン リストには、いくつかの一般的な区切り文字が含まれています。

このデータ セットは、区切られたテキスト ファイルを必要とします。 以下の例では、最初の行でプロパティ名（userid および password）を指定しています。 後続の行では、これらのプロパティで使用されるデータ値がリストされています。



［テスト アンド キープ］をクリックすると、データの最初の行がロードされます。 データ セットを読み取れたことを示す確認メッセージが表示され、データの最初の行が表示されます。



データ セットによるテスト ケースの終了

データ セットがテスト ランを終了するには、以下の条件を満たす必要があります。

- データ セットはグローバルである必要があります。
- データセットの［データが終了した場合：］が［実行：テストを終了する］に設定されている。
- テスト ケースの最初のステップで、データ セットを増加させる必要があります。

サイクルを完了する前にステージングされた実行全体を低下させるため、テスト ケースの最初のステップにグローバル データ セットを適用する場合は注意してください。

データ セットの名前の変更

データ セットの名前を変更する方法

- エレメント パネルでデータ セットを選択し、ツールバーの［名前の変更］アイコンをクリックします。
- モデル エディタでデータ セットを選択し、右クリックでメニューを開いて名前を変更します。
- エレメント パネルのデータ セットを選択し、  名前をデフォルトに戻すボタンをクリックして、デフォルトのデータ セット名にデータ セット名を変更します。

データ セットの移動

モデル エディタ内でデータ セットをドラッグ アンド ドロップすることによって、テスト ステップから別のテスト ステップに移動させることができます。

次の手順に従ってください:

1. テスト ステップに添付されたデータ セットをクリックします（たとえば Step 1）。
2. データ セットを選択し、モデル エディタで目的のテスト ステップ（たとえば、Step 2）にドラッグし、そこにドロップします。
3. ドラッグされたデータ セットが Step 2 に適用されます。

データ セットの次のステップの選択

データ セットが起動された後に実行する次のステップまたは終了ステップを選択できます。

次の手順に従ってください:

1. メニューを開くには、モデル エディタ内のデータ セットを右クリックします。
2. ［終了時の処理］メニューをクリックし、実行する次のステップを選択します。

データ セットおよびプロパティ

データ セットではプロパティを使用できます。

プロパティの重要な用途には、以下のようなものがあります。

- 外部データ セットの場所を指定する場合、パス名のハードコードされた値ではなく（LISA_HOME など）、プロパティを使用できます。 例：**LISA_HOME¥myTests¥myDataset.csv**

  ハードコードされた値の代わりにプロパティを使用すると、データ セットの移植性が向上します。 このように使用されるプロパティは、通常、テスト ラン内の早い段階で使用できる必要があるため、以下のいずれかの方法でそれらを定義する必要があります。

  - システム プロパティとして
  - 設定で

  設定ではダミーの値を定義でき、後でそれを変更できます。 この値によって、設計時にファイルを見つけることができます。 DevTest サーバ でテストを実行する場合、プロパティのこの使用方法は重要です。
- データ セット値にはプロパティを含めることができます。これらの値は、読み取られるときに評価されます。 たとえば、データ セットに student _1 としてログイン値を設定するとします。 その後、student の現在の値が **Bart** である場合、結果的なデータ値は **Bart_1** になります。
- ローカル データ セットでは、テスト ケースからプロパティを使用できます。ただし、グローバル データ セットは仮想ユーザが共有しているため、プロパティを使用できません。

コンパニオン

コンパニオンは、すべてのテスト ケース実行の前/後（またはその両方）に実行される LISA エレメントです。 コンパニオンはテスト ケースに対してグローバルな動作を設定するために使用されます。 これらの動作には、ブラウザ帯域幅およびブラウザの種類のシミュレート、負荷テストでの同期ポイントの設定、およびサンドボックス クラス ローダの作成などがあります。 ある意味では、コンパニオンは、テスト ケースの実行に対して一種のコンテキストを設定すると言えます。

ステップが実行される前に、コンパニオンはテスト ケースのヘルパーとして動作します。

以下のトピックが含まれます。

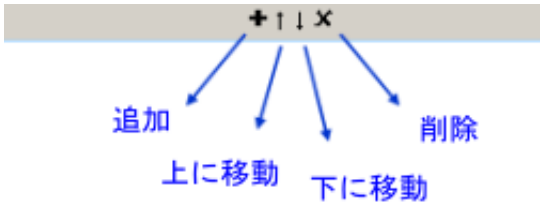
コンパニオンの追加

次の手順に従ってください:

1. テスト ケースを開き、右側のパネルのコンパニオン エレメントをクリックします。
2. ✚ ［追加］をクリックします。 ［コンパニオン］メニューが開きます。
3. コンパニオンにはそれぞれ別のエディタがあります。適切なエディタを開くには、必要なコンパニオンを選択します。 コンパニオンのタイプはそれぞれのパラメータと共に、「*リファレンス*」の「コンパニオンの説明」で詳細に説明されています。

コンパニオン ツールバー

各エレメントの下部には、追加/削除/並べ替えのためのアイコンのあるツールバーがあります。



DevTest フック

DevTest フックは、テスト ケースの開始/終了時にロジックを自動的に実行するグローバルな方法です。 フックが展開された環境のすべてのテスト ケースに、フックはそのロジックを適用します。

フックはコンパニオンと同様に動作します。フックは、テストが開始される前およびテストが完了した後に実行されます。

違いは、フックはアプリケーション レベルで定義され、DevTest のすべてのテスト ケースがフックを実行することです。 フックが存在する場合、それはすべてのテスト ケースに使用されます。

フックは、DevTest で実行されるすべてのテストに対して、テストのセットアップ ロジックまたは後処理ロジック、あるいはその両方を自動的に含めることができるメカニズムです。 フックはシステム全体にわたるコンパニオンと言えます。 フックが実行できるものはすべて、コンパニオンとしてモデル化できます。

フックは、テスト環境を設定するためや、正しく設定されていないテストまたは定義されたベスト プラクティスに従っていないテストの実行を防止するために使用されます。また、共通の操作を実施するために使用されます。

フックは DevTest クラス パスにある Java クラスです。 フックは .lisaextensions ファイルには定義されていませんが、local.properties または、lisa.properties では以下のように定義されています。

```
lisa.hooks=com.mypackage1.MyHook1, com.mypackage2.MyHook2
```

フックは、テスト ケース自体だけでなく、テスト ケースのすべてのサブプロセスの開始および終了に実行または呼び出されます。 テスト ケースに 3 つのサブプロセスがある場合、フック ロジックは 4 回実行されます。このロジックは、メインのテストに対して 1 回、3 つの各サブプロセスに対して 1 回ずつ実行されます。

サブプロセスで **startHook または endHook**（あるいはその両方）の実行を防ぐには、以下のアクションを実行します。

```
String marker =
(String)testExec.getStateValue(TestExec.FROM_PARENT_TEST_MARKER
_KEY)
```

サブプロセスの場合にのみ、「marker」は「true」に設定されます。

```
if (!"true".equals(marker))
```

サブプロセスでは実行しない開始/終了フック ロジックは、すべてここに記述します。

フックとコンパニオンの相違点

- フックはスコープ内でグローバルです。 コンパニオンには必要ですが、テスターは特にそのテスト ケースにフックを含めません。フックはシステム レベルで登録されます。すべてのテストにロジックを含める必要があり、それをユーザが誤って省略することを望まない場合、フックは良いメカニズムです。
- フックはテスト ケース レベルではなく、インストール レベルで展開されます。テストが 2 台のコンピュータで実行され、1 台のコンピュータでのみフックを登録する場合、フックが展開されているコンピュータにテストがステージングされる場合にのみフックが実行されます。テスト ケースで定義されているコンパニオンは、インストール レベルの設定にかかわらず実行されます。
- フックはユーザから実質的に見えないため、ユーザにカスタム パラメータを要求できません。 フックは、設定のプロパティまたはシステムからそれ自体のパラメータを取得します。コンパニオンはモデル エディタで表示されるため、コンパニオンにはカスタム パラメータがあります。

複合オブジェクト エディタ（COE）

複合オブジェクト エディタ（COE）では、Java コードを記述することなく、Java オブジェクトと対話できます。

入力パラメータの現在値を変更し、メソッド コールを実行して、戻り値を確認することができます。 また、簡単なインライン フィルタリングを実行し、戻り値に対する簡単なアサーションを追加できます。

多くのテスト ステップに、Java オブジェクトの操作が含まれています。複合オブジェクト エディタを使用して、Java オブジェクトを直接操作するか（動的 Java 実行ステップまたは Enterprise JavaBean（EJB）ステップでのように）、または間接的に操作します（Web サービスの入力パラメータや Enterprise Service Bus （ESB）から返されるメッセージのように）。

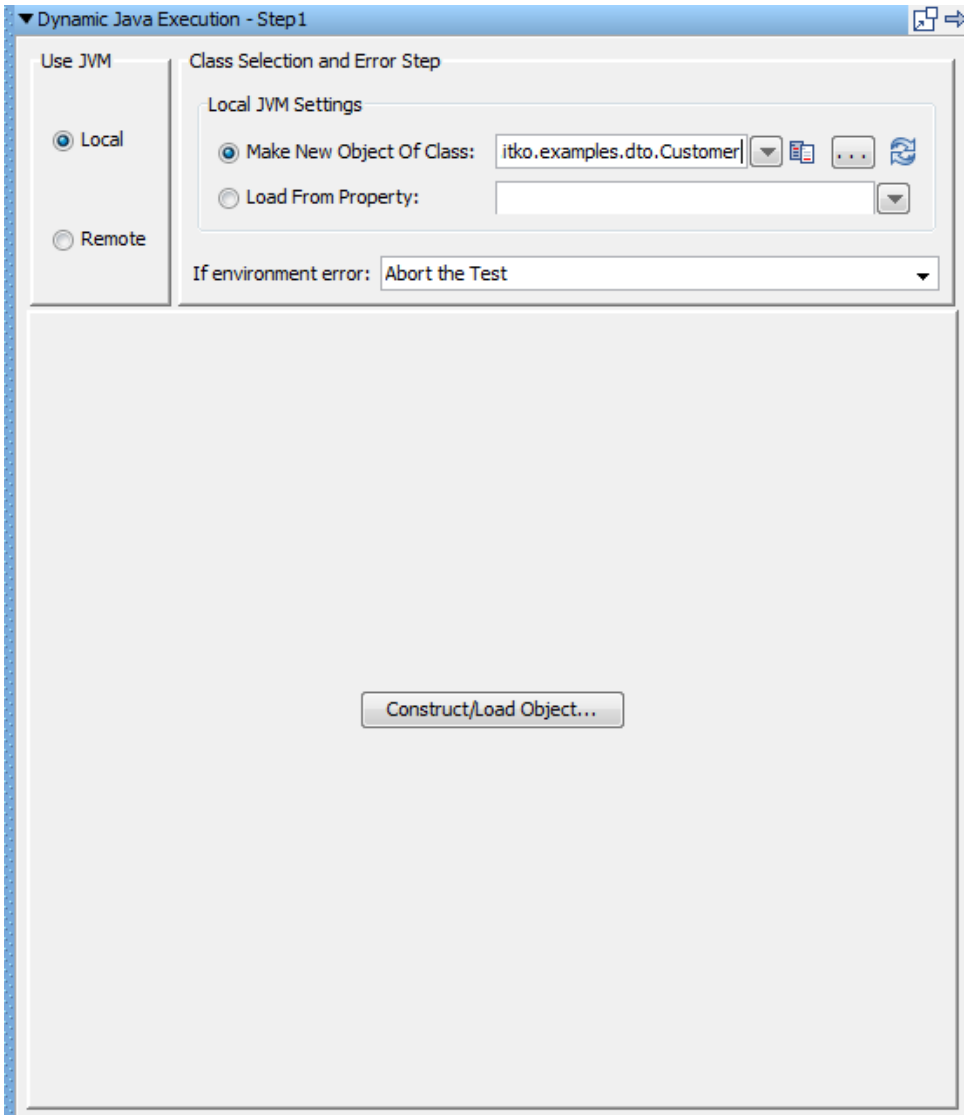
このセクションでは、まずユーザ インターフェースについて説明します。その後で、複数の使用シナリオを簡単なものから順番に説明します。

以下のトピックが含まれます。

COE の起動

表示される場所にかかわらず、複合オブジェクト エディタ（COE）の外観は同じです。

たとえば、Customer クラスを使用して動的 Java 実行ステップを呼び出した場合、以下のウィンドウが表示されます。

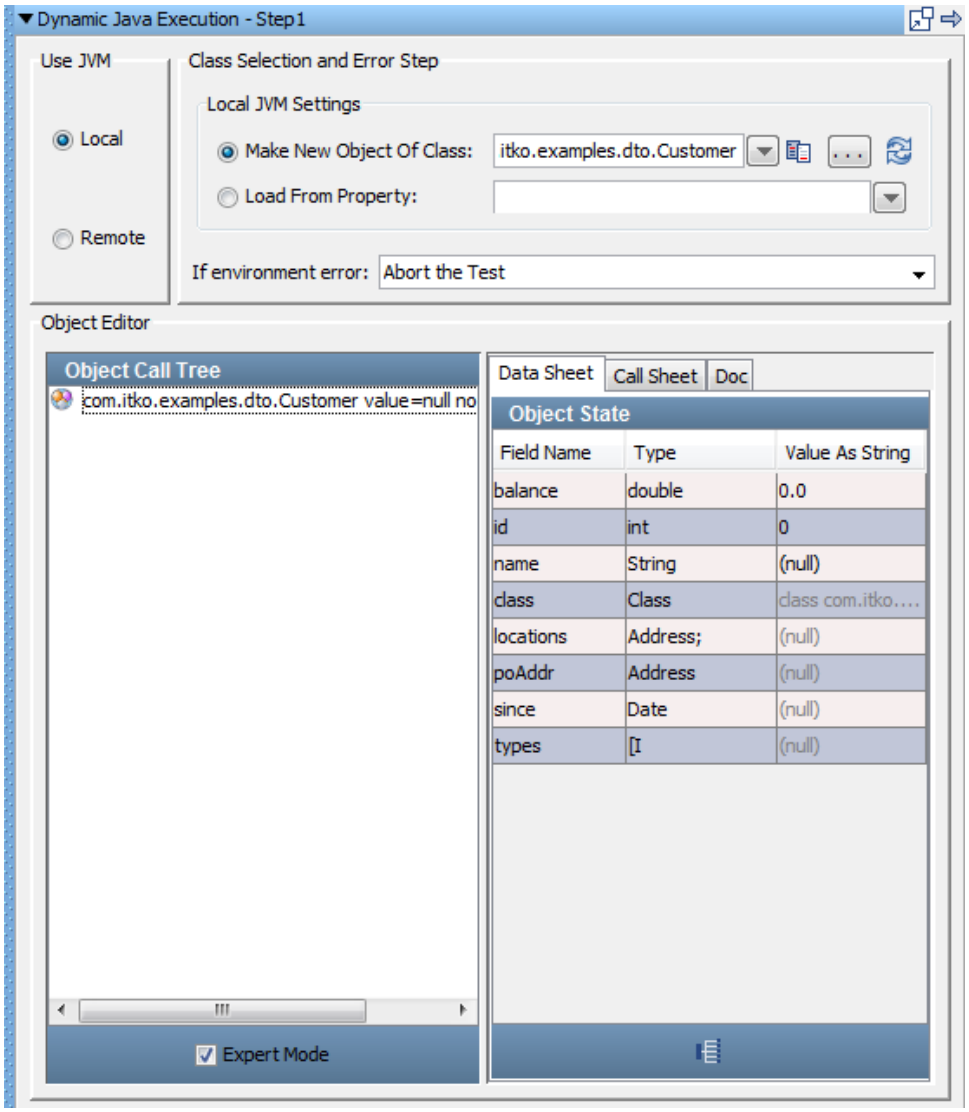


ローカル JVM の使用を選択し、［クラスの新規オブジェクトを作成］フィールドに「*com.itko.examples.dto.Customer*」と入力します。

オブジェクト エディタにオブジェクトをロードするには、［オブジェクトの構築/ロード］をクリックします。

複合オブジェクト エディタ

オブジェクトがロードされると、複合オブジェクト エディタが起動されます。



オブジェクト エディタは 2 つのパネルで構成されています。 左側の［オブジェクト コール ツリー］パネルは、メソッド呼び出し、およびそれらの入力パラメータと戻り値を追跡します。 ［オブジェクト コール ツリー (P. 192)］パネルについては、次のトピックで詳細に説明しています。

右側の［オブジェクト状態］パネルには、使用可能なオプションを示す動的なタブ（［データ シート］、［コール シート］、［ドキュメント］ (P. 194)）のセットがあります。

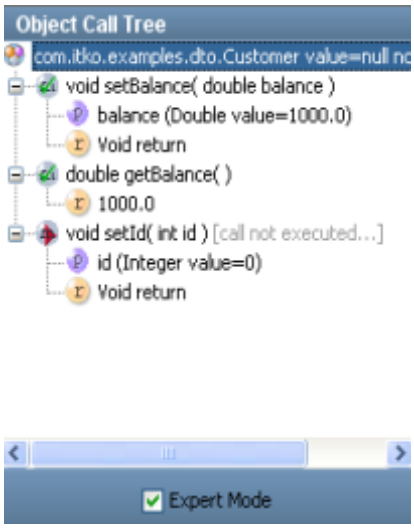
注: 応答が XML で 5 MB より大きい場合は、DOM ビューのないプレーン テキストで表示されます。 この 5 MB のデフォルトの制限は **gui.viewxml.maxResponseSize** プロパティで調節できます。

上記のウィンドウは、エディタでの**ロードされた Customer** タイプの Java オブジェクトを示しています。 このオブジェクトは、動的 Java 実行テスト ステップを使用してロードされていますが、多くの操作でこのオブジェクトをロードできます。 オブジェクトに対してコールは呼び出されていません。

[オブジェクト コール ツリー]パネル

Java オブジェクト（Customer）を操作する場合、呼び出されたコールおよび関連するパラメータ値を追跡するには、オブジェクト コール ツリーを展開します。

以下に、複数のメソッドが呼び出された後のオブジェクト コール ツリーの例を示します。



オブジェクト コール ツリー アイコン

オブジェクト コール ツリーのブランチの識別には、以下のアイコンを使用します。

アイコン	説明
	現在ロードされているオブジェクトのタイプ（クラス）。その後にオブジェクトの toString メソッドのコールの応答が表示されます。
	コールされたコンストラクタ。 これは、コンストラクタが複数存在する場合に表示されます。
	まだ実行されていないメソッド コール。
	すでに実行されたメソッド コール。
	エンクロージング メソッドの入力パラメータ（タイプと現在の値）。

エンクロージング メソッドの戻り値（コールが実行された場合は現在の値）。

オブジェクト コール ツリーの項目をクリックすると、右側のパネルの［データ シート］および［コール シート］に適切なタブのセットが表示されます。

オブジェクト コール ツリーの項目を右クリックすると、メニューが表示されます。

オブジェクト コール ツリーでは、コールをすべて実行できます。また、未実行としてコールをすべてマークできます。

[データ シート]および[コール シート]パネル

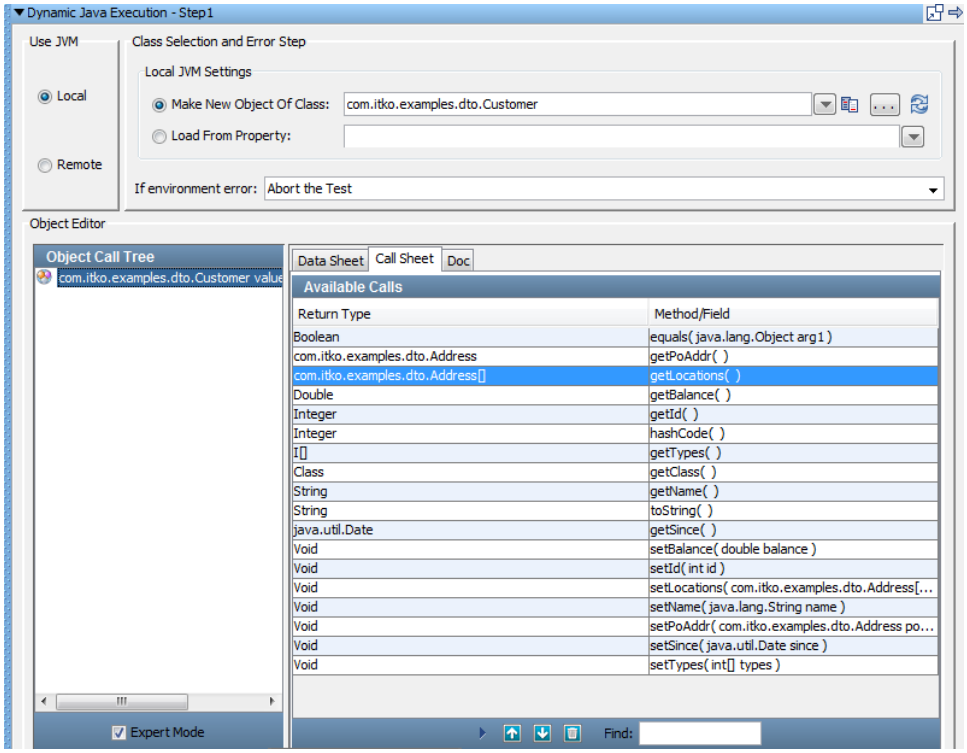
［データ シート］パネル

右側のパネルには 3 つのタブが表示され、デフォルトでは[データ シート]タブがアクティブになります。

黒で表示されたデータは、このタブで編集できます。データ値は［String 型の値］列で編集します。 これらの値は常にプリミティブまたは文字列です。 淡色表示された値は、［データ シート］パネルでは編集できませんが、別のウィンドウで編集できます。たとえば、アドレス フィールドは、このタブで編集できない Address 型のオブジェクトです。

［コール シート］パネル

COE の［コール シート］タブには、使用可能なメソッド/フィールド コールおよびそれらの戻り値型が表示されます。

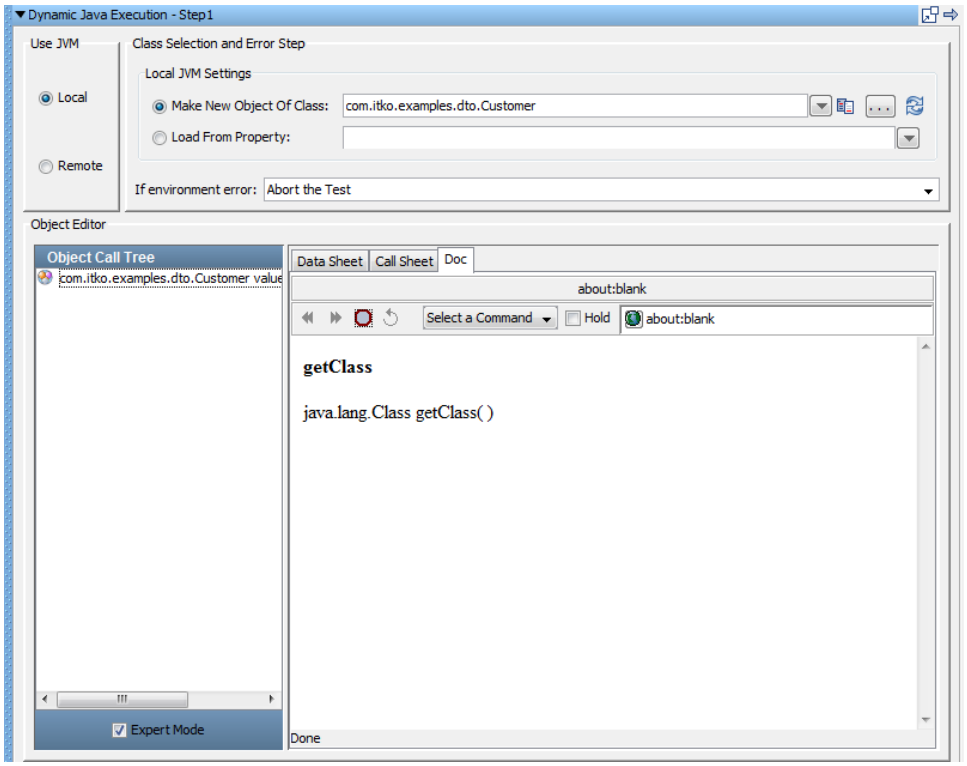


メソッドを追加するには、メソッド/フィールドのリストでメソッドを選択し、タブの下部にあるメソッドの追加アイコン  をクリックします。選択したメソッドが、オブジェクト コール ツリー（左側のパネル）に表示されます。

オブジェクト コール ツリーでは、入力パラメータを指定したり、メソッドを呼び出したりすることができます。

［ドキュメント］パネル

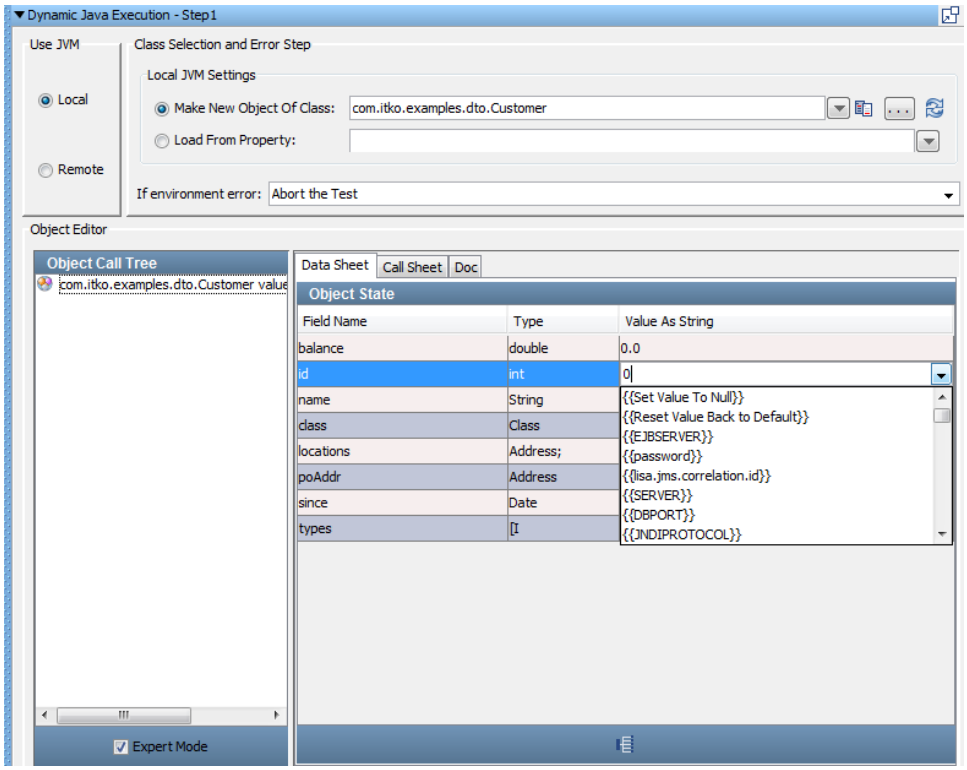
［ドキュメント］タブには、このクラスで使用可能な Java API ドキュメントが表示されます。



オブジェクト操作パネル

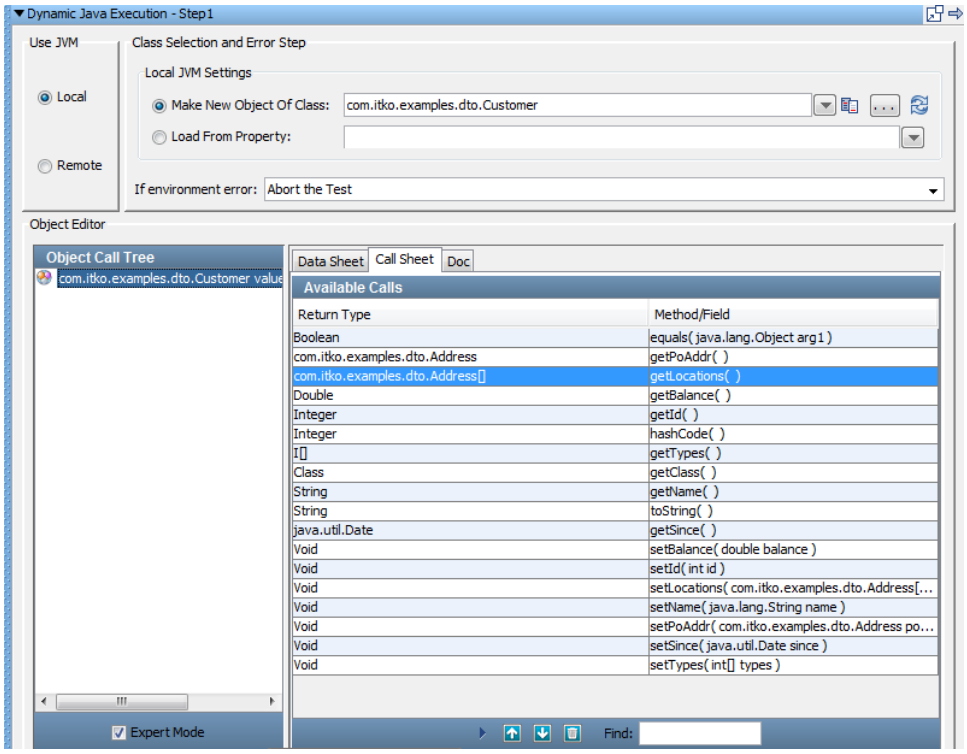
［データ シート］および［コール シート］に表示されるアクションには個別のインターフェースがあります。

タブおよびそのインターフェースは、左側の［オブジェクト コール ツリー］パネルで何を選択したかによって変わります。



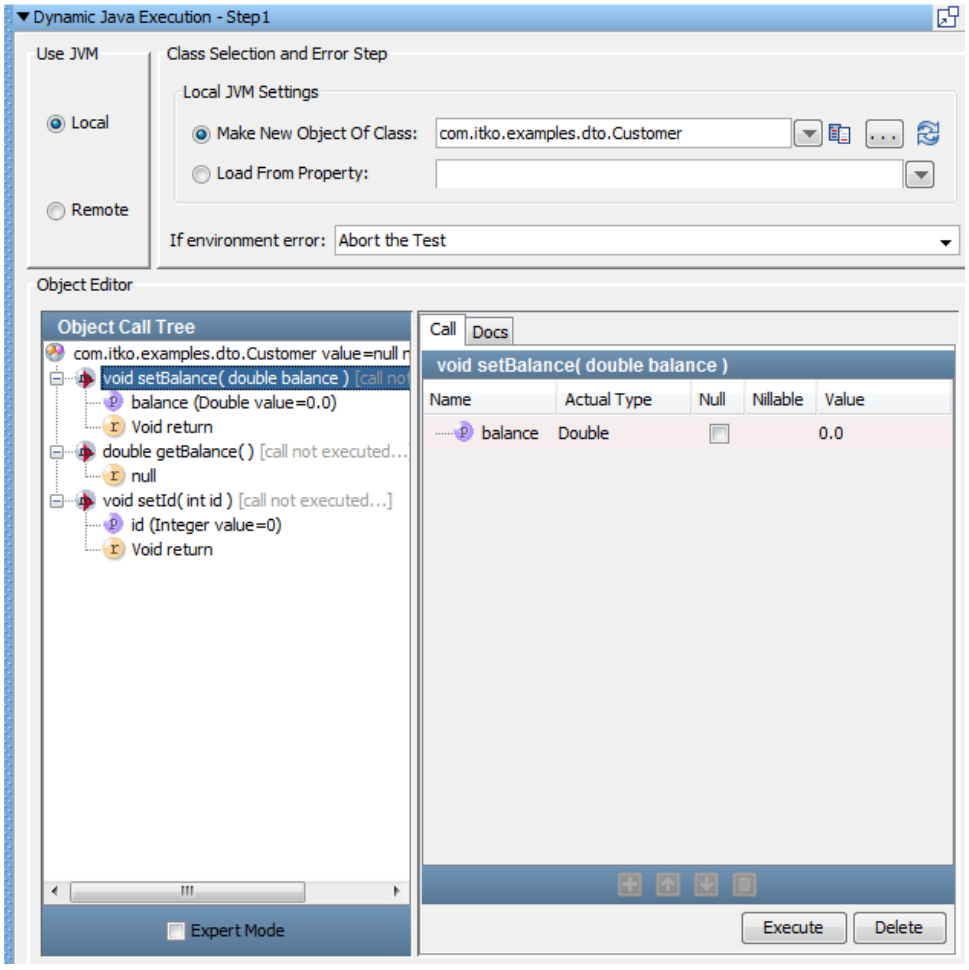
オブジェクト パネル

オブジェクトがオブジェクト コール ツリーで選択されている場合、［データ シート］、［コール シート］および［ドキュメント］タブが使用可能です。



メソッド コール パネル

オブジェクト コール ツリーでメソッド コールを選択すると、右側のパネルで［コール］および［ドキュメント］タブが使用可能になります。



ここでは、入力パラメータの値を指定します。

各パラメータに関する情報は提供されているため、値のみを入力します。この例は単純です。 単一のパラメータは「double」型です。したがって、［値］列には値またはプロパティ名を入力できます。

プルダウン メニューには、現在のプロパティのリストが表示されます。入力パラメータとしてオブジェクトを入力する必要がある場合は、より多くの作業が必要となります。 以下のセクションでは、オブジェクトを指定する複数の方法について説明します。

左側のパネルの下部にある［エキスパート モード］がオンになっていないことに注目してください。 これは、シンプル モードであることを示します。

シンプル モードおよびエキスパート モード

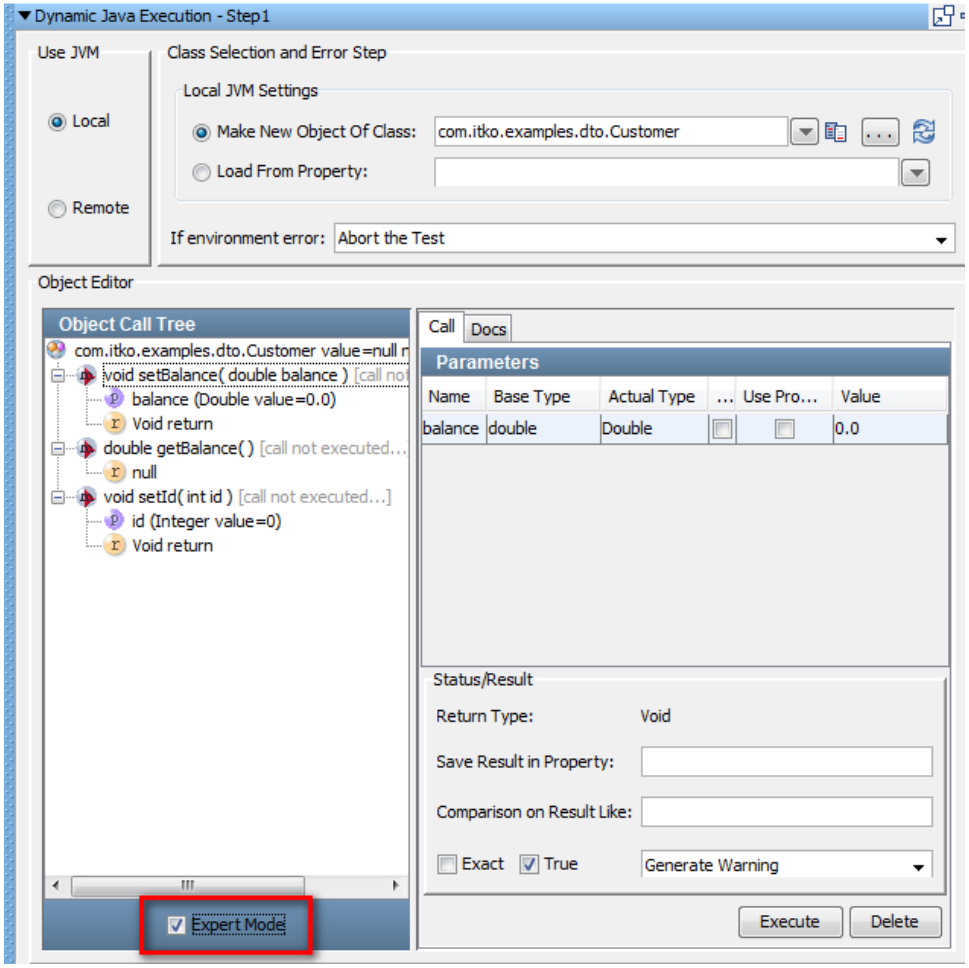
2 つの編集モード（シンプルおよびエキスパート）を使用できます。

オブジェクトがデフォルトのコンストラクタと複数の setter/getter メソッドだけの Java Bean などの単純なオブジェクトである場合、シンプル モードが有用です。 これは従来のデータ転送オブジェクト（DTO）です。これらのオブジェクトは、Web サービス コールへの入力として一般的です。このタイプのオブジェクトでは、シンプル モードとエキスパート モードを切り替えることができます。 プロパティとして DTO が含まれる DTO はシンプル モードで操作できます。 このアクティビティの例については後で説明します。

エキスパート モードは、複数のコンストラクタがあるオブジェクトなどの、より複雑なオブジェクトに使用します。 その他の複雑なオブジェクトが含まれる一部の複合オブジェクトでは、シンプル モードを使用できません。また、実際には、現在のオブジェクトがエキスパート モードを必要とする場合、シンプル モード オプションは無効になります。

上記の図では、すべてシンプル モードを使用しています。

以下の図は、上記の図の例を示していますが、エキスパート モードが選択されています。



上記の図のように、新しく［ステータス/結果］パネルが表示されます。

オブジェクト エディタで、インライン フィルタ（結果を保存するプロパティ）およびアサーション（結果の比較 - Like）を追加できます。

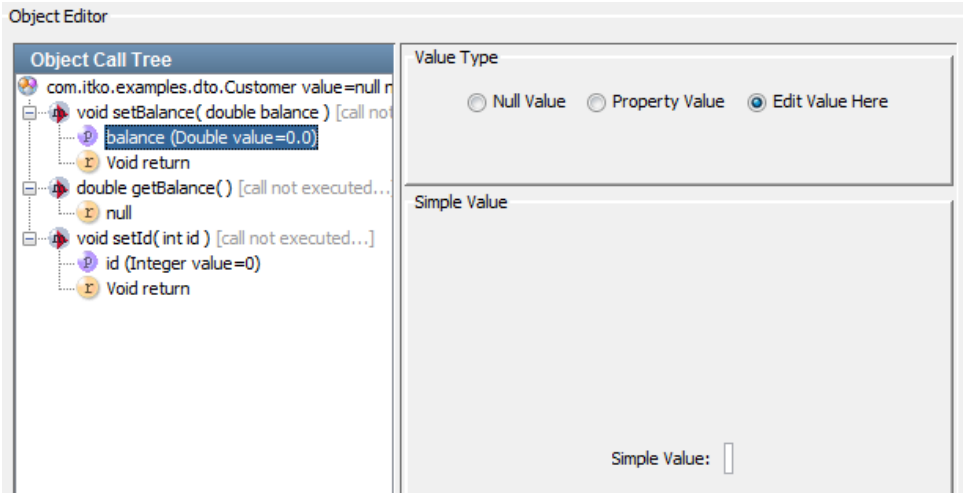
注: 適用されるインライン フィルタおよびアサーションは、フィルタまたはアサーション リストに表示されません。

その他のいくつかの相違点については、後の例で説明します。

入力パラメータ パネル

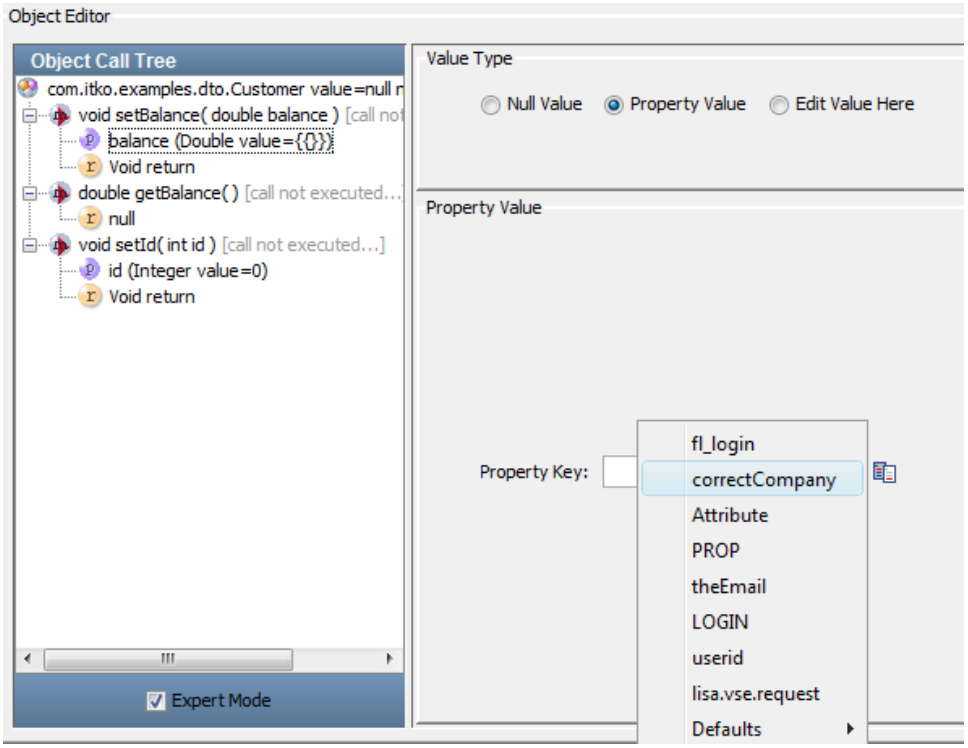
オブジェクト コール ツリーで入力パラメータを選択した場合、右側のパネルで使用可能なタブは入力パラメータのタイプ（プリミティブ/文字列、またはオブジェクト）によって変わります。

入力パラメータがプリミティブまたは文字列である場合、以下のパネルが表示されます。



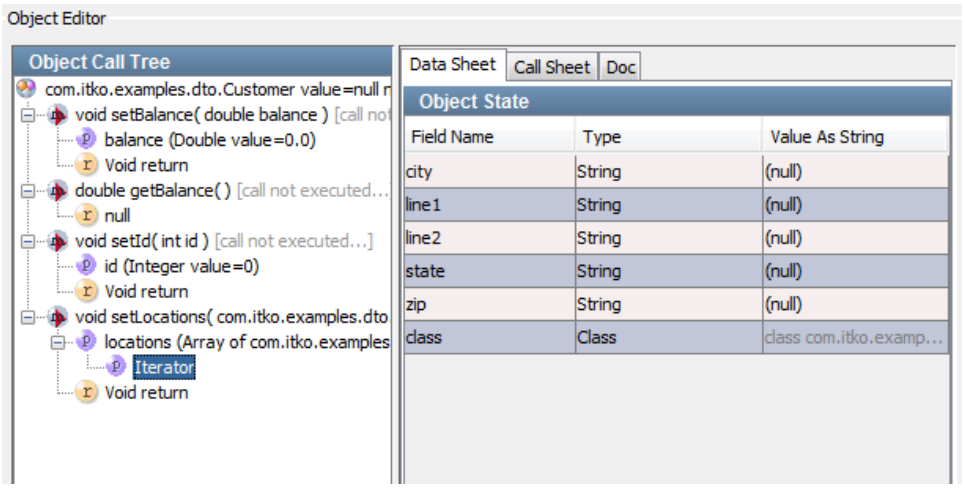
このパネル内の値は、［シンプル値］フィールドで編集できます。

値としてプロパティを使用するには、［プロパティの値］オプション ボタンをクリックして、以下のパネルを表示します。



使用可能なプロパティ キーを開くには、プロパティ名を入力するか、プルダウン メニューを使用するか、または  [リスト]をクリックします。

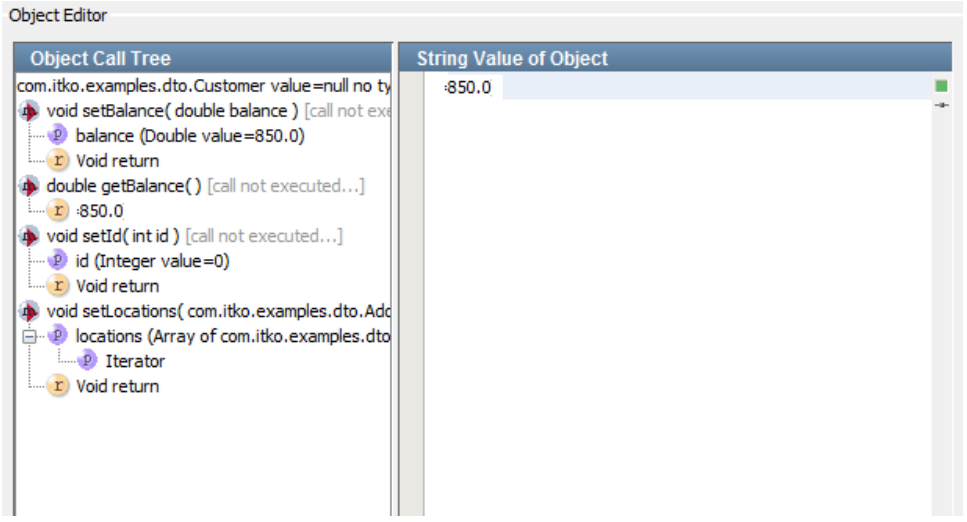
入力パラメータがオブジェクトである場合、以下のパネルが表示されます。



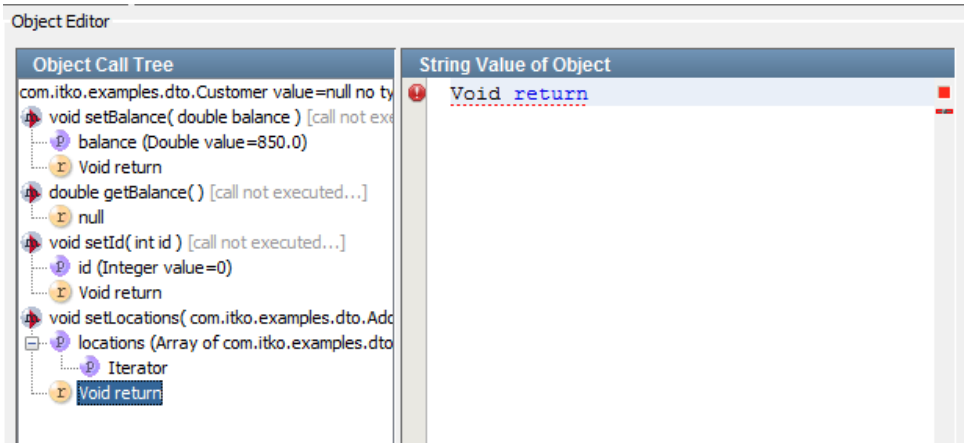
戻り値パネル

オブジェクト コール ツリーで戻り値を選択した場合、右側のパネルで使用可能なタブは入力パラメータのタイプによって変わります。

入力パラメータがプリミティブまたは文字列である場合、以下のパネルが表示されます。



入力パラメータがオブジェクトである場合、以下のパネルが表示されます。

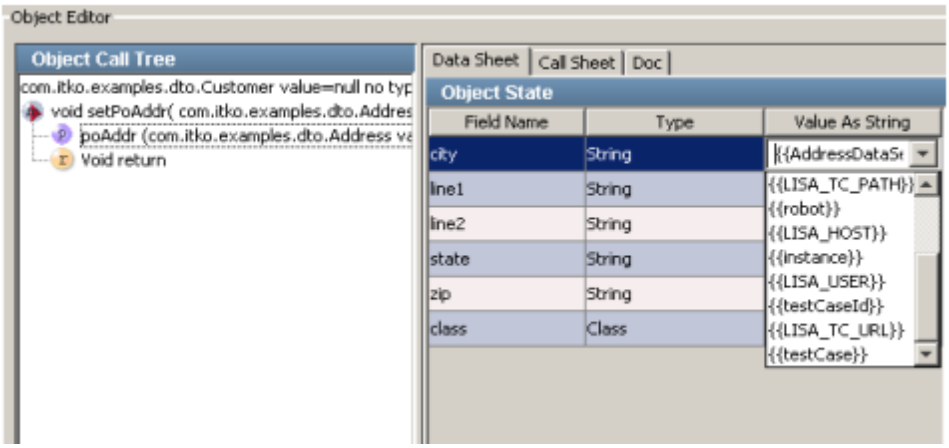

Object Editor
Object Call Tree
com.itko.examples.dto.Customer value=null no ty
void setBalance(double balance) [call not exe
balance (Double value=850.0)
Void return
double getBalance( ) [call not executed...]
null
void setId(int id) [call not executed...]
id (Integer value=0)
Void return
void setLocations(com.itko.examples.dto.Adc
locations (Array of com.itko.examples.dto
Iterator
Void return
String Value of Object
Void return

COE でデータ セットを使用する方法

Java DTO オブジェクトにデータを提供する一般的な方法は、データ セットです。

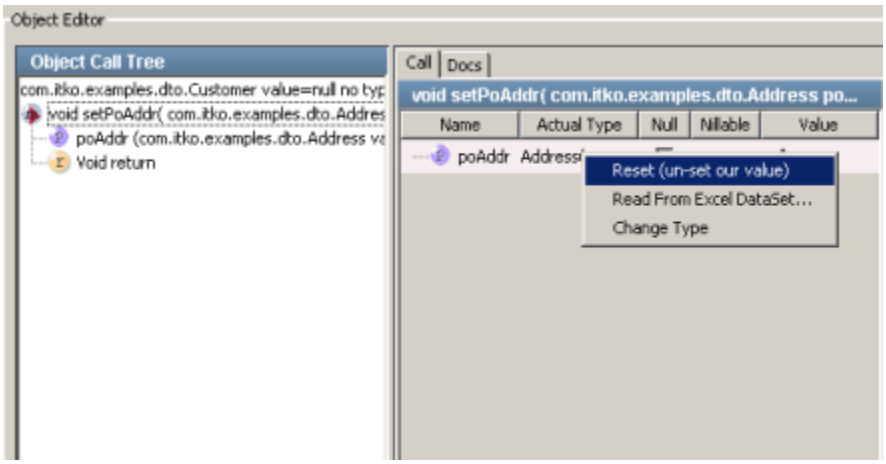
データ セットと Excel ファイルからの DTO の読み取りは、この目的のために提供されています。

Excel データ セットが存在する場合は、データ セットが含まれるプロパティを DTO オブジェクトの値として入力できます。

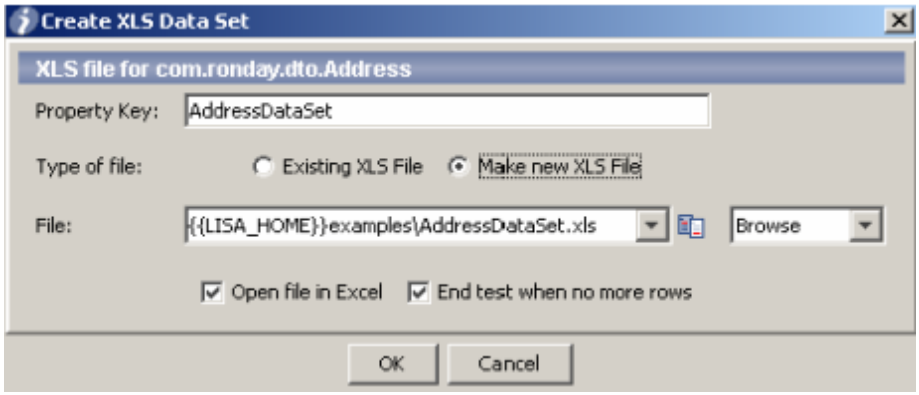


オブジェクト エディタで新しい DTO データ セットの作成を開始する方法

1. DTO オブジェクトがコール リストに表示されている場合に、［名前］または［実際のタイプ］を右クリックしてメニューを開きます。



2. ［Excel データセットから読み取り］を選択して、以下のウィンドウを表示します。



3. このウィンドウのパラメータは、このデータ セットの作成を開始するために必要です。

以下のパラメータを入力します。

プロパティ キー

データ セットからの現在値を格納するプロパティ。

ファイル タイプ

既存の XLS ファイルまたは新しい XLS ファイルの作成を選択します。

ファイル

Excel ファイル（DTO データのテンプレート）の名前。

ファイルを Excel で開く

Excel でスプレッドシートを開きます。

これ以上の行がない場合にテストを終了する

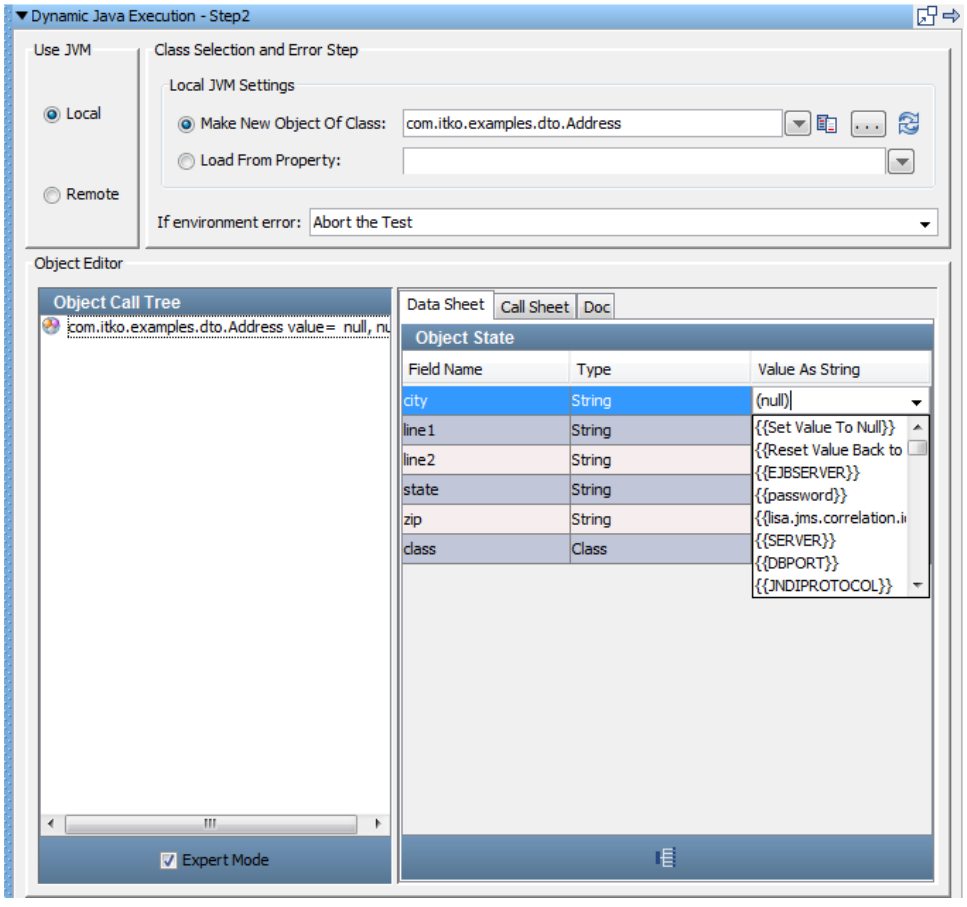
行がすべて読み取られた後、テストを終了します。

単純なオブジェクトの使用シナリオ

以下の例は、単純なオブジェクトの標準の Java クラスに基づいています。環境で再現が簡単なように、DevTest に付属しているデモ サーバのクラスを使用しています。

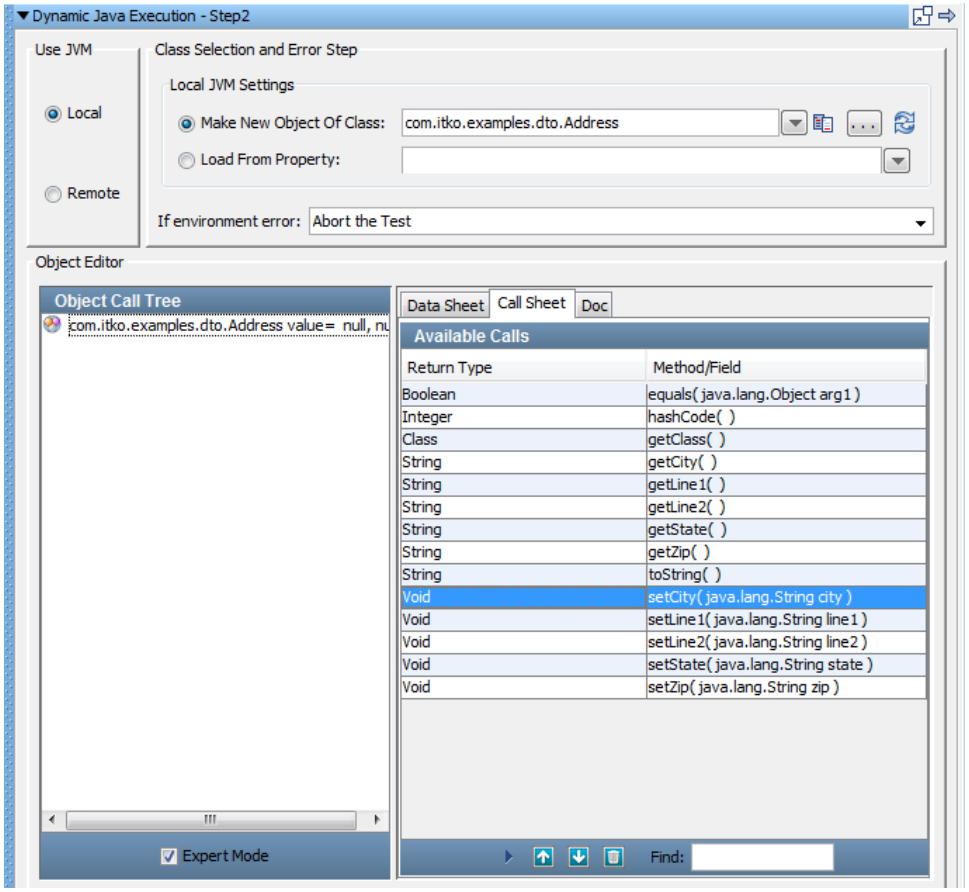
単純な DTO オブジェクト(シナリオ 1)

単純な DTO の com.itko.examples.dto.Address は、動的 Java 実行ステップを使用して COE にロードされています。 Address クラスには単純なプロパティのみがあります。

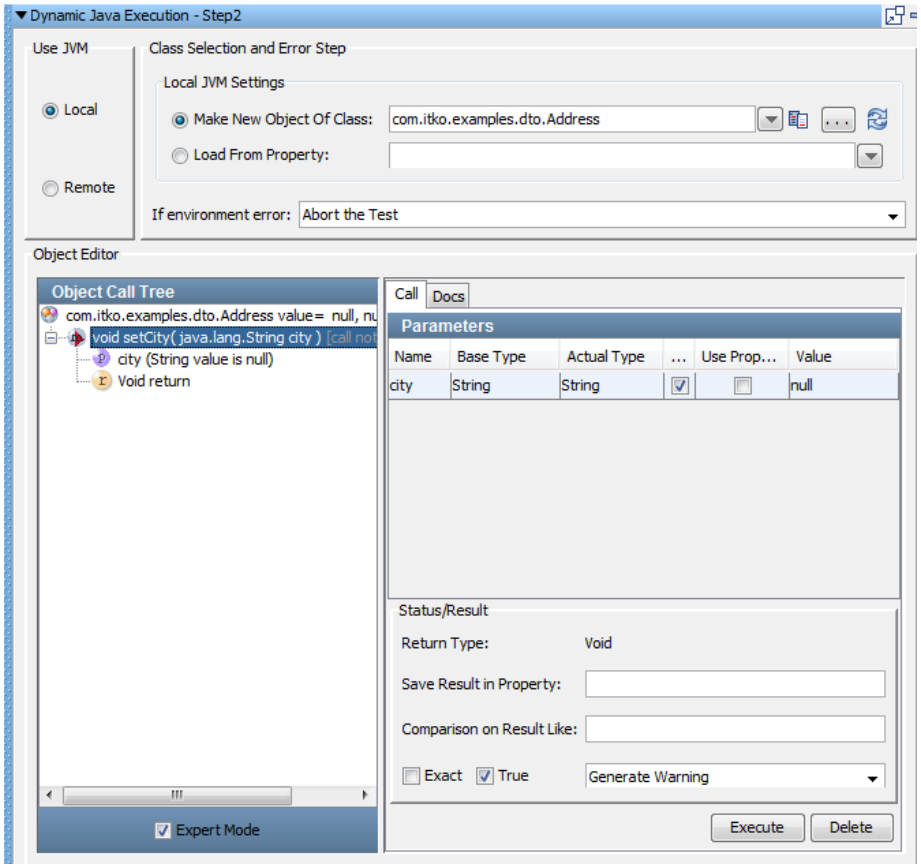


単純な DTO の場合、パラメータ値は固定値またはプロパティとして [データ シート] パネルで入力できます。 上記の例では、city が currentCity プロパティと同等に設定されます。

[コール シート] パネルで DTO セッターを使用してパラメータを入力する方法



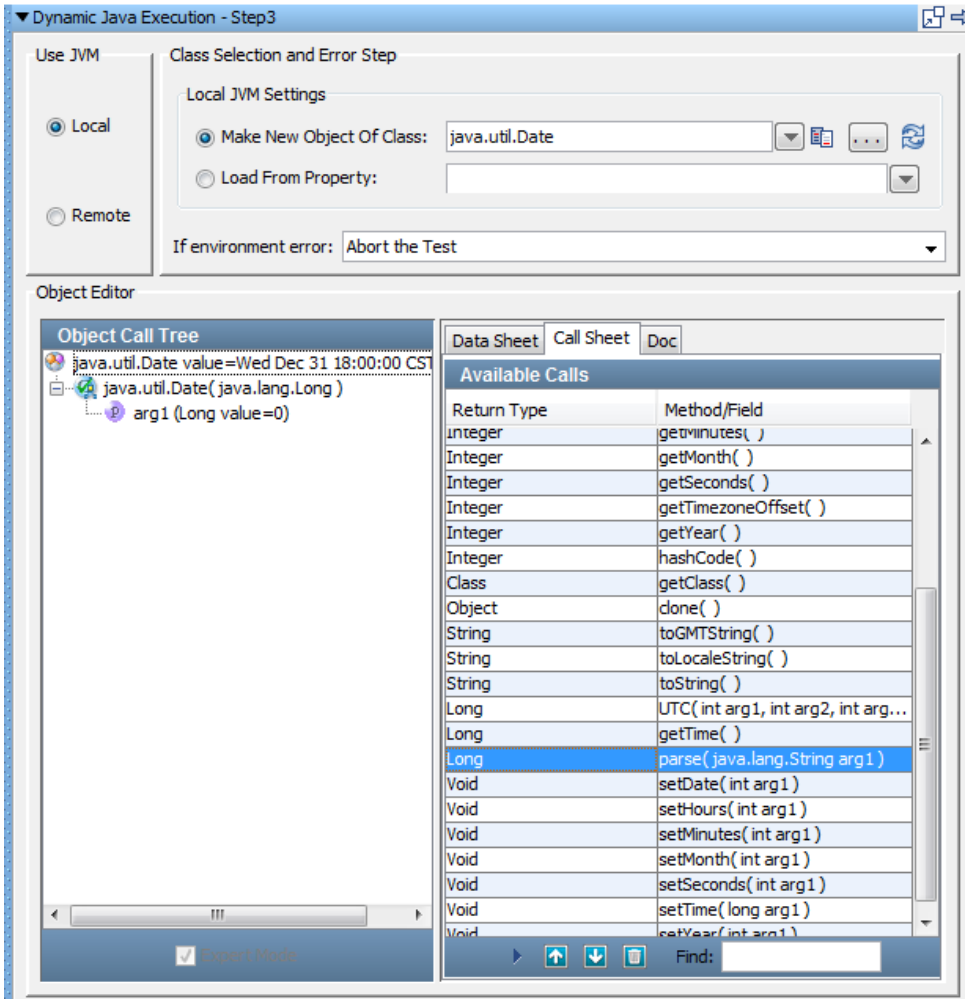
1. ［コール シート］タブで、setCity(java.lang.String city) などのゲッターを選択し、メソッドの追加アイコン  をクリックします。
   メソッドが実行され、COE の［コール］タブが開きます。



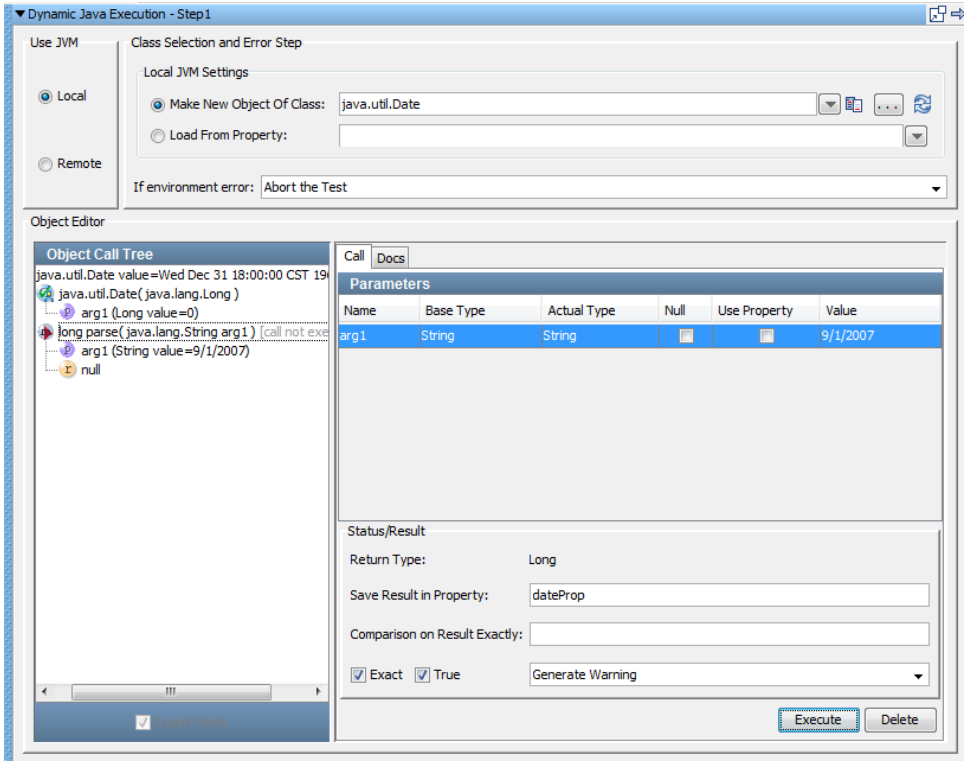
2. 固定値またはプロパティとしてパラメータ値を入力します。 DevTest プロパティ構文の propname を使用します。
3. ［実行］ボタンをクリックして、メソッドを呼び出します。
4. この手順を繰り返して、その他の DTO プロパティを設定します。

単純な Java オブジェクト（シナリオ 2）

単純な Java オブジェクト（**java.util.Date**）は、動的 Java 実行ステップを使用して COE にロードされています。



この場合、Date 型は DTO ではないため、エキスパート モードを使用する必要があります。 実際には、COE がエキスパート モードを強制します。



ただし、今までどおりパラメータ値を入力し、メソッドを呼び出すことができます。上記の例では、1 つの入力パラメータを必要とする parse メソッドを実行します。 文字列値として「9/1/2007」を入力しています。

注: エキスパート モードでは、［NULL］または［プロパティの使用］チェック ボックスを使用してパラメータ タイプを指定します。 どちらもオンになっていない場合、固定値が入力されます。 ここでは {{}} プロパティ構文を使用しません。 文字列値ではなくプロパティを入力する場合でも、プロパティ名のみを入力します。

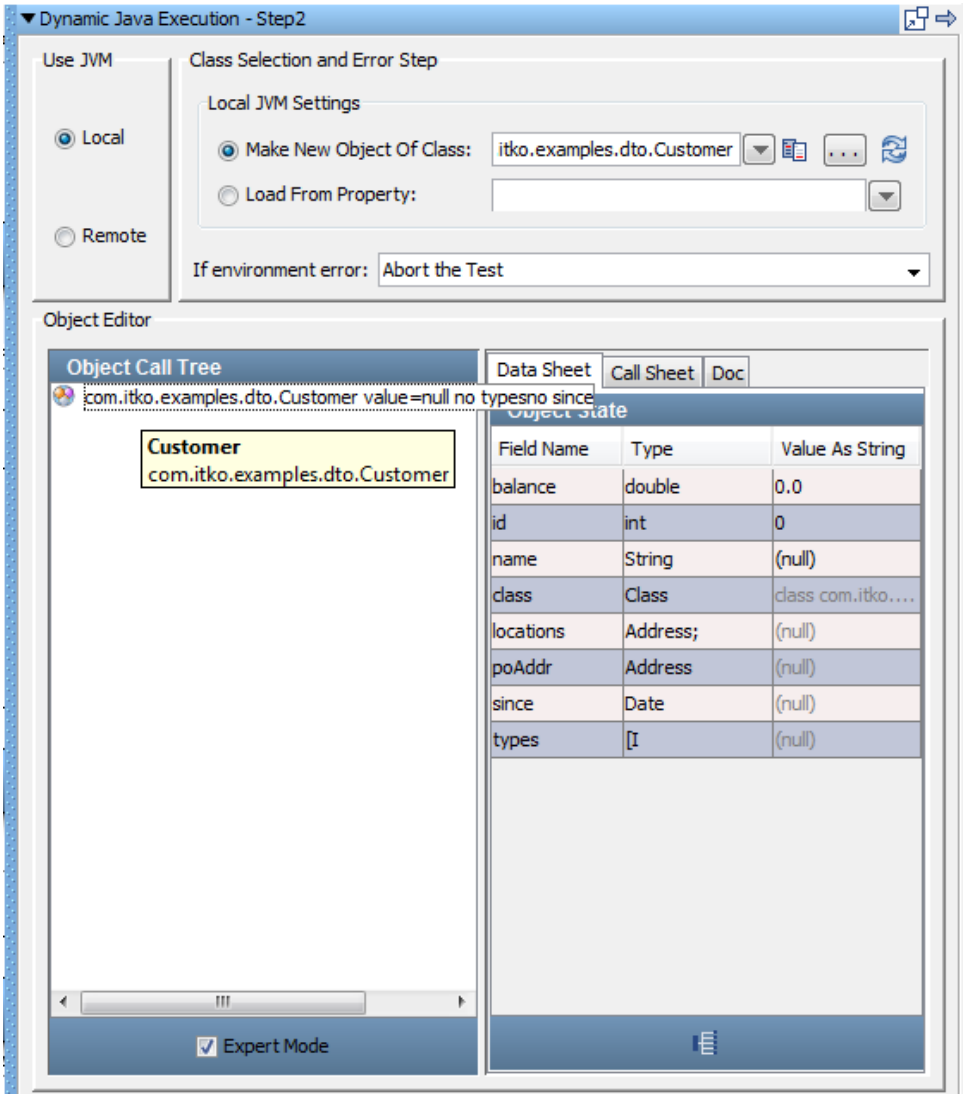
エキスパート モードであるため、インライン フィルタおよびアサーションを追加できます。

複雑なオブジェクトの使用シナリオ

以下の例は、複雑なオブジェクトの標準の Java クラスに基づいています。環境で再現が簡単なように、DevTest に付属しているデモ サーバのクラスを使用しています。

複雑な DTO オブジェクト(シナリオ 1)

複雑な DTO オブジェクト (com.itko.examples.dto.Customer) は、動的 Java 実行ステップを使用して複合オブジェクト エディタにロードされます。



一部のプロパティはプリミティブまたは文字列などの単純な値ではないため、この DTO は複雑です。 ただし、DTO 構造を備えているため、シンプル モードを引き続き使用できます。

Customer オブジェクトを使用する前に、各プロパティの値を指定する必要があります。

locations

Address オブジェクトの配列

poAddr

Address オブジェクト

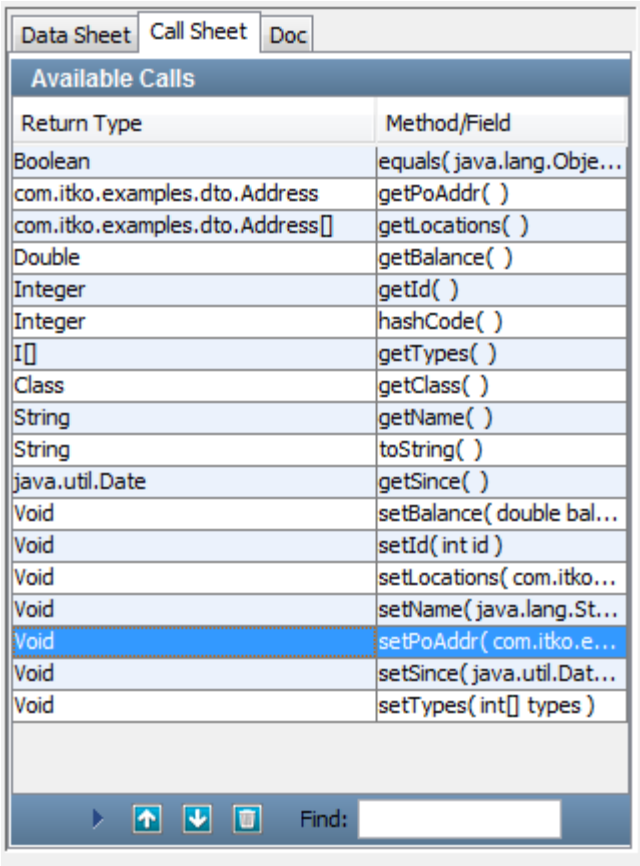
since

Java Date オブジェクト

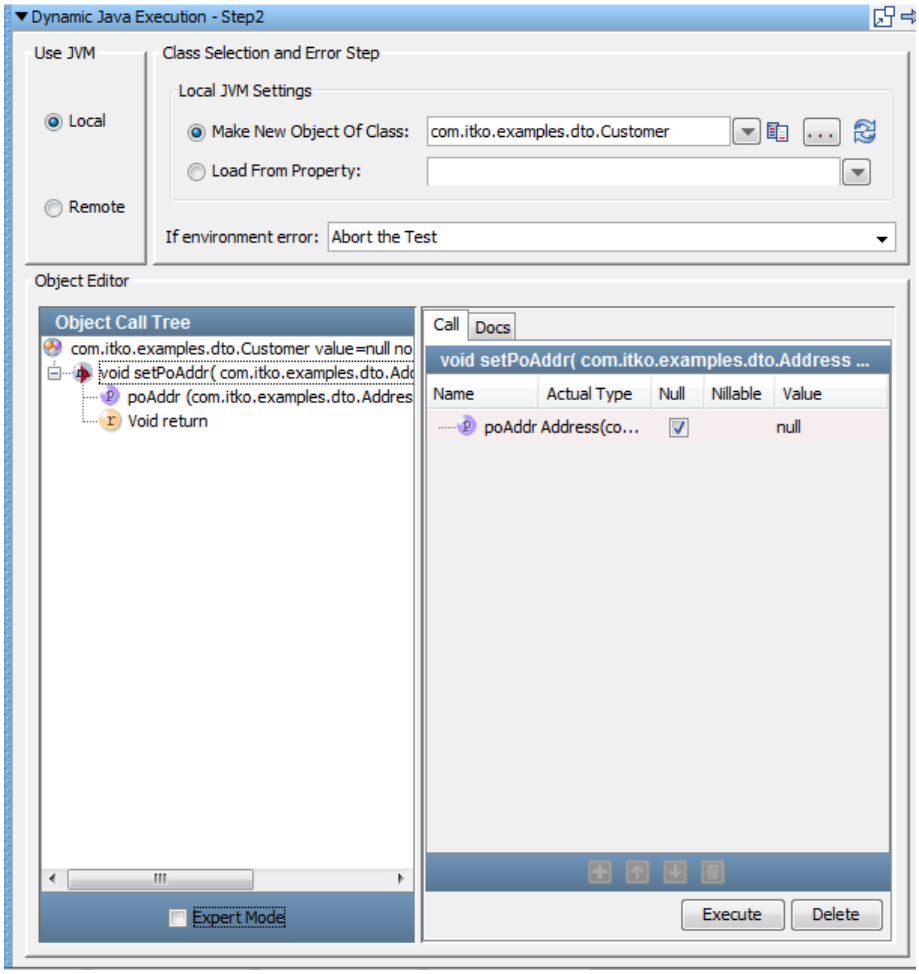
types

整数の配列

1. poAddr オブジェクトから始めて、［コール シート］の setPoAddr メソッドを指定します。

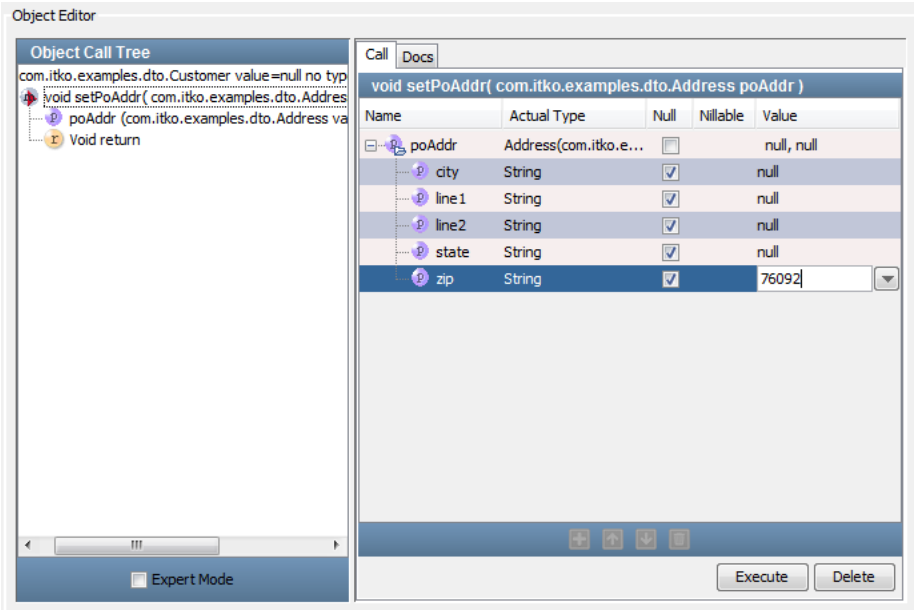


2. setPoAddr メソッドを選択して、ダブルクリックするか、または  メソッドの追加 をクリックして、このメソッドを実行します。



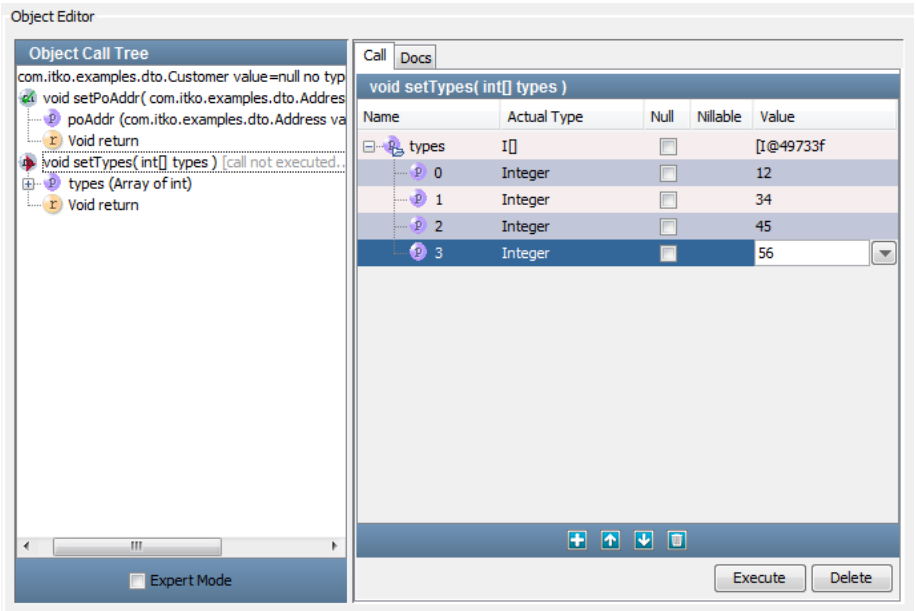
3. この DTO では、シンプル モードを使用できます。［エキスパート モード］チェック ボックスをオンにしないでください。 poAddress プロパティは Address 型として識別されます。

4. シンプル モードでは、［NULL］パラメータをオフにすると、Address オブジェクトはそのプロパティを表示するために展開されます。前述の単純なデータ オブジェクトのシナリオの図から、Address には単純なプロパティがあることがわかります。値またはプロパティとして［値］列にそれらを入力できます。



5. ［実行］ボタンをクリックして、setPoAddr メソッドを呼び出します。

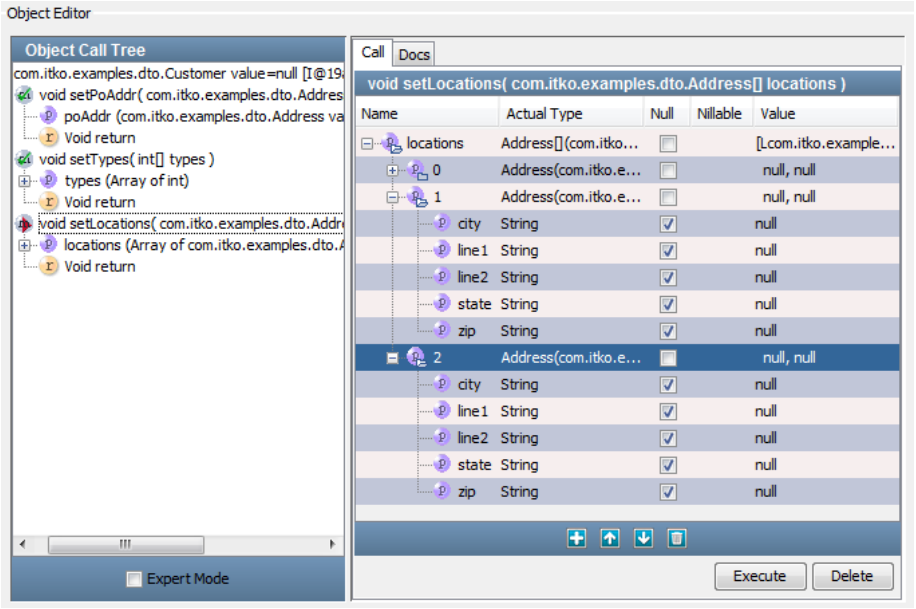
6. ［コール シート］で setTypes メソッドを選択し、  アイコンをクリックしてメソッドを呼び出します。



7. **Types** は整数の配列です。 配列に要求な数の整数を追加するには、下部にある  ［追加］をクリックします。 上記の例では、4 つのエレメントを追加し、それぞれに値を入力しています。

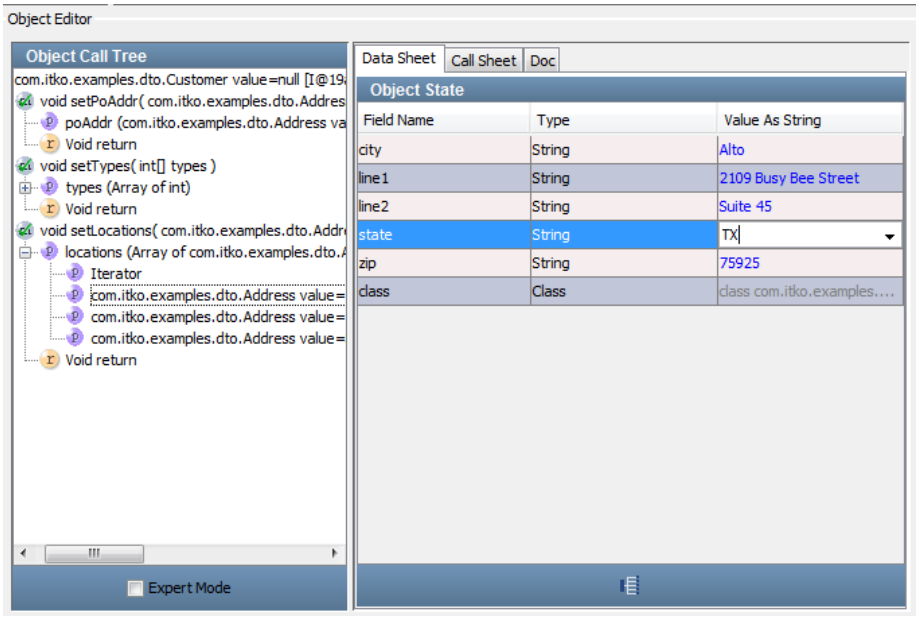
8. ［実行］ボタンをクリックして、setTypes メソッドを呼び出します。

9. ［コール シート］で setLocations メソッドを選択し、  アイコンをクリックしてメソッドを呼び出します。



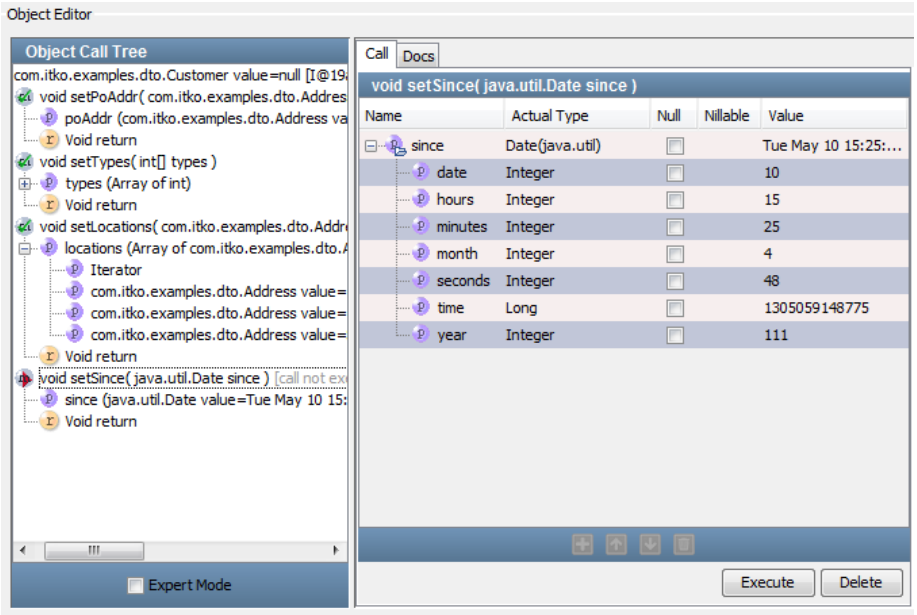
10. **Locations** は Address オブジェクトの配列です。  ［追加］をクリックして、必要な数のエレメント（Address 型）を追加します。

11. 上記の例では、3 つの Address オブジェクトを追加しています。2 つは完了しており、プロパティ値を入力するために、3 つ目の Address オブジェクトを展開します。入力が完了したら、［実行］をクリックして、setLocations メソッドを呼び出します。 オブジェクト コール ツリーの Location エレメントの 1 つをクリックすると、［データ シート］タブでプロパティを表示および編集できることに注目してください。



これは、オブジェクト コール ツリーに表示されるすべてのプロパティに当てはまります。

12. ［コール シート］で getSince メソッドを選択し、  アイコンをクリックしてメソッドを呼び出します。



Data オブジェクトの入力パラメータが表示され、値が示されます。

13. ［実行］ボタンをクリックします。

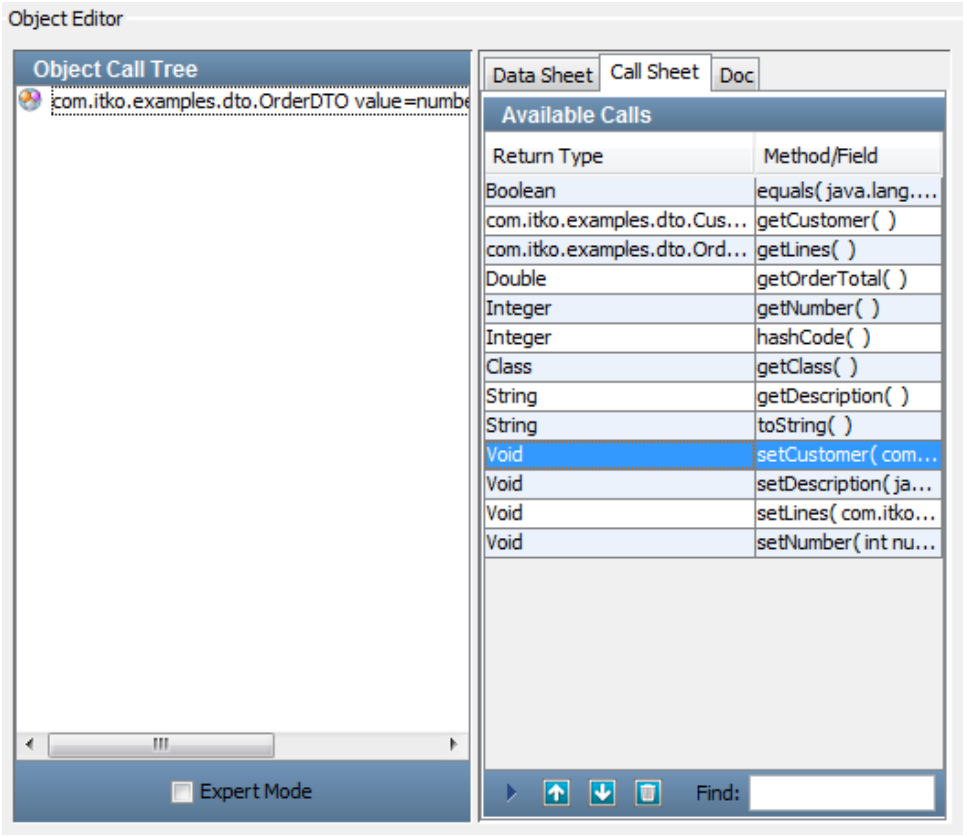
これで Customer オブジェクトが完全に指定され、テスト ケースで使用できます。

複雑な DTO オブジェクト(シナリオ 2)

最後のシナリオでは、前述の 3 つのシナリオに基づいた例を示します。

この DTO （com.itko.examples.dto.OrderDTO）には、そのプロパティの 1 つとして Customer オブジェクトがあります。 このシナリオでは、setter メソッドをコールせずにシンプル モードで Customer オブジェクトを構築することがどれくらい容易かを示します。

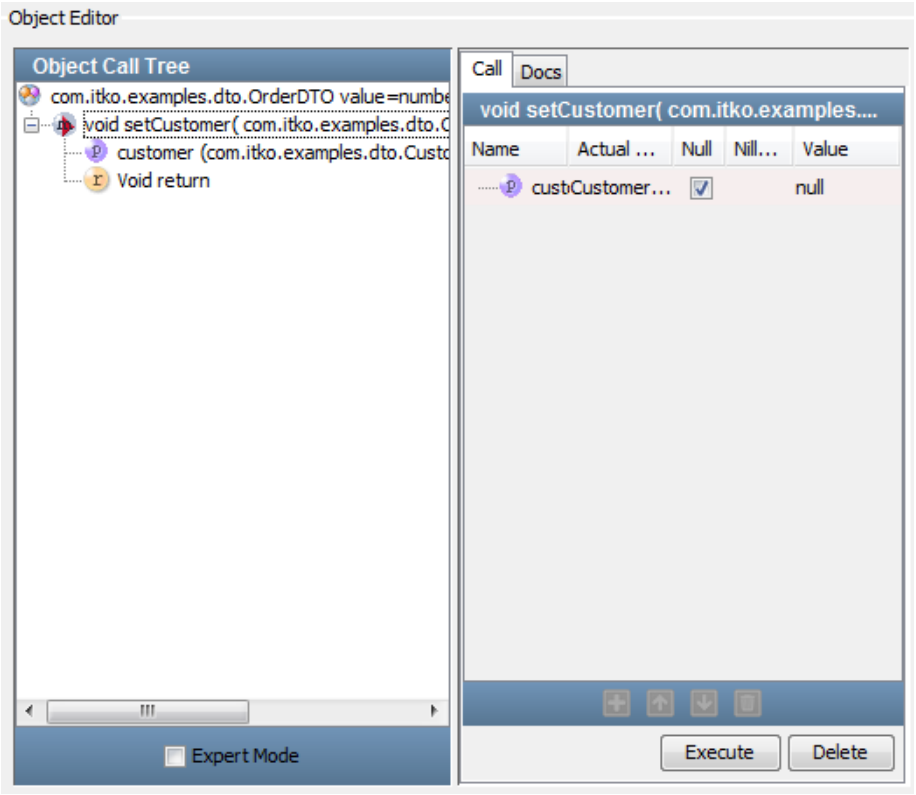
OrderDTO オブジェクトは、動的 Java 実行ステップを使用して COE にロードされています。



ここでも、OrderDTO オブジェクトにシンプル モードを使用できます。

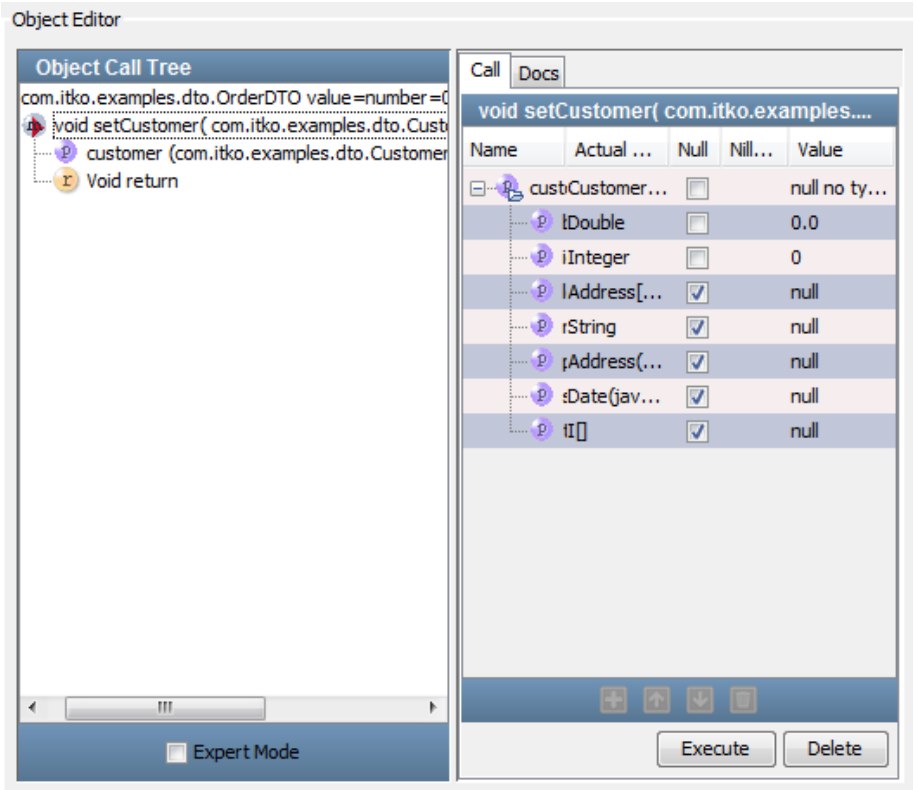
次の手順に従ってください:

1. ［コール シート］で setCustomer メソッドを選択し、  アイコンをクリックしてメソッドを呼び出します。 予期したとおり、入力パラメータは Customer オブジェクトです。

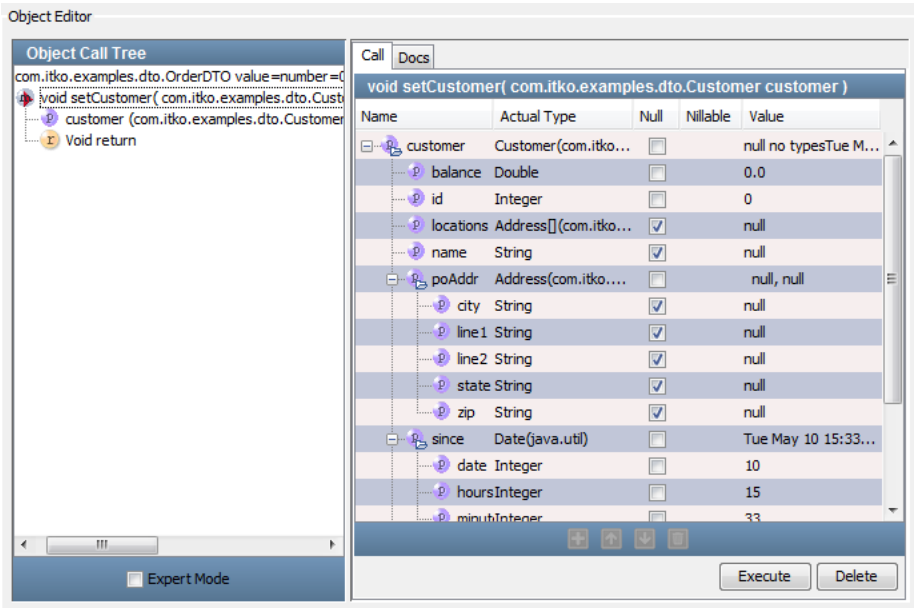


2. ［NULL］列のチェック ボックスをオフにします。

Customer の行が展開され、プロパティが表示されます。



プロパティはすべてこのウィンドウで編集できます。 Integer および String プロパティは［値］列で追加できます。 残りのプロパティは、プロパティの［NULL］ボックスをオフにすると展開され、そのプロパティが表示されます。プロパティが単一のオブジェクトである場合は、展開され、そのプロパティが表示されます。 プロパティが配列またはコレクションである場合は、適切な数のエレメントを追加できます。この図は、編集プロセスのスナップショットを示しています。



locations プロパティには 2 つのエレメントがあり、types プロパティには 3 つのエレメントがあります。 poAddr オブジェクトは Address 型であり、展開されると単純な文字列プロパティが表示されます。

テスト ステップの作成

テスト ステップは、実行される単一のテスト アクションを表すテスト ケース ワークフローのエレメントです。 テスト ステップには、2 つの主なカテゴリがあります。

- ほとんどのテスト ステップはテスト中のシステムに対して動作し、応答を評価します。 一般的な例としては、メッセージ サービス プロバイダによる Enterprise JavaBean （EJB）メソッド、Web サービス、またはメッセージのテストなどがあります。
- もう 1 つのカテゴリのテスト ステップは、データ変換、データ操作 (エンコーディングなど) 、ログ、ファイルへの情報の書き込みなどのユーティリティ機能を実行します。

注: DevTest は、テスト ステップ、プロパティ、またはその両方からの I/O ストリームの送信をサポートしていません。

どちらのカテゴリのステップもテスト ケースの作成に使用されます。

このセクションには、以下のトピックが含まれます。

テスト ステップの追加

テスト ステップを追加する方法

以下のいずれかの操作を実行します。

- ツールバーの［ステップの追加］をクリックします。
- メイン メニューから［コマンド］-［新規ステップ作成］を選択します。

また、メニューを開くワークフローのステップを右クリックし、ワークフローの特定の場所にステップを追加できます。 ［～の後にステップを追加］をクリックし、適切なテスト ステップを選択します。

詳細:

テスト ステップの追加 - 例

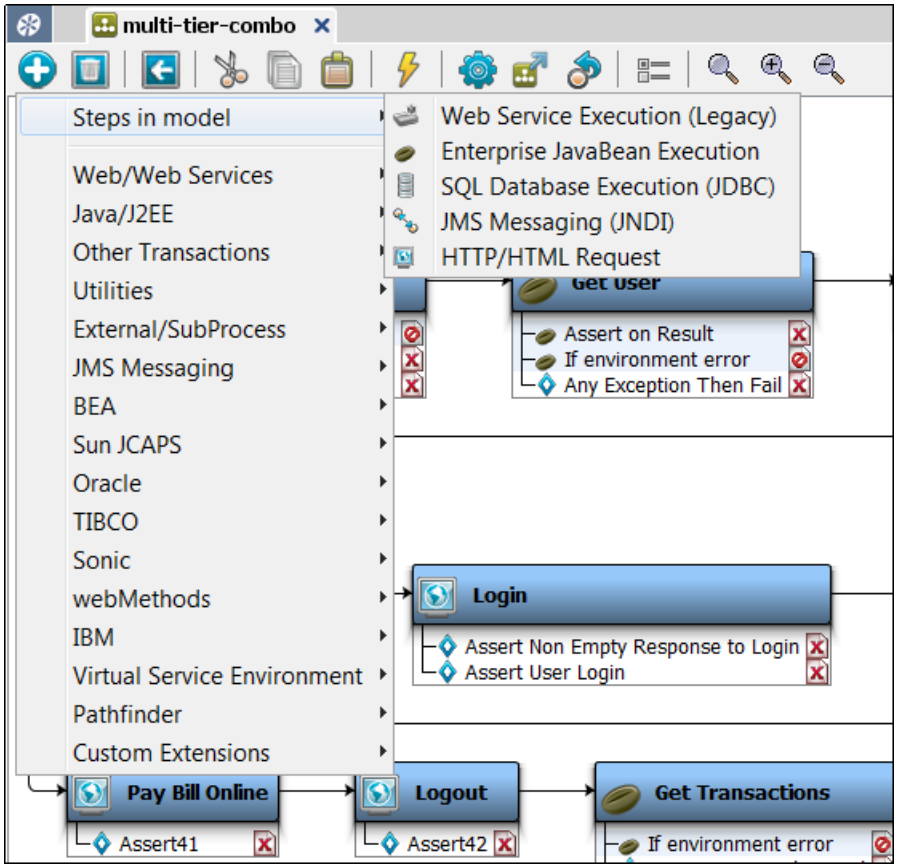
この例では、examples ディレクトリの multi-tier-combo テスト ケースにステップを追加します。新しい動的 Java 実行ステップが、multi-tier-combo テスト ケースの Get User （ユーザの取得）ステップの後に追加されます。

次の手順に従ってください:

1. ［ステップの追加］ボタンをクリックします。

   一般的なテスト ステップをリスト表示するパネルが表示されます。

   - このテスト ケースのようにいくつかのステップがテスト ケースにすでに作成されている場合は、パネルの一番上に［モデルのステップ］が表示されます。 ［モデルのステップ］は、テスト ケースに存在するすべてのステップをリスト表示します。
   - 新しいテスト ケースの場合、このリストは空です。multi-tier-combo テスト ケースを開くと、このテスト ケースのステップが［モデルのステップ］メニューに表示されます。



2. 設定するステップの主なカテゴリを選択します（Web/Web サービス、Java/J2EE、ユーティリティなど）。

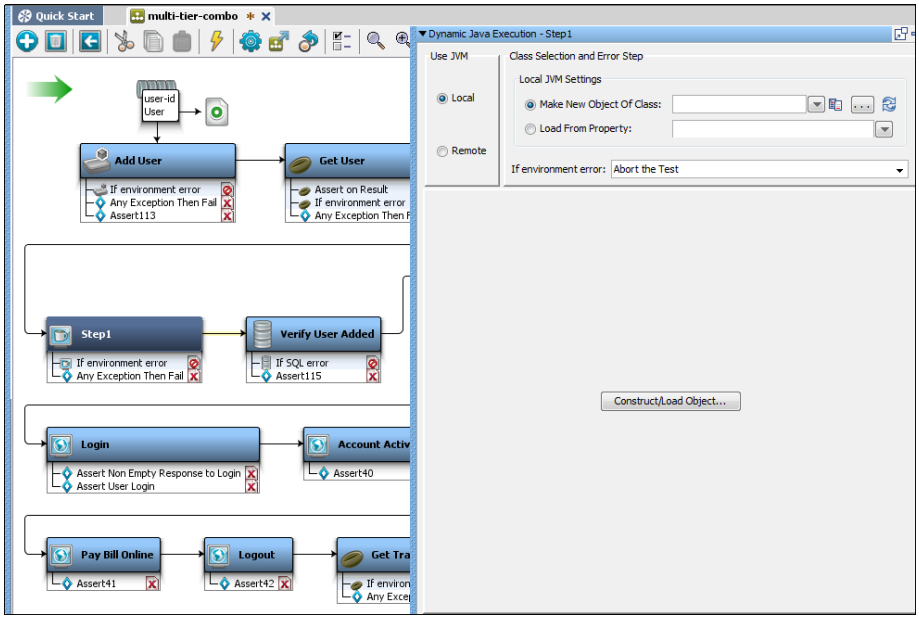
   サブカテゴリが表示されます。

3. テスト ケースにステップを追加するには、ステップをクリックします。

   選択したテスト ケースのステップ エディタが開きます。 各ステップ タイプには個別のステップ エディタがあります。

4. Get User （ユーザの取得）ステップの後に動的 Java 実行ステップを追加するには、Get User ステップを右クリックしてメニューを開き、動的 Java 実行ステップを選択します。

   ステップが追加され、動的 Java 実行ステップのステップ エディタが開きます。

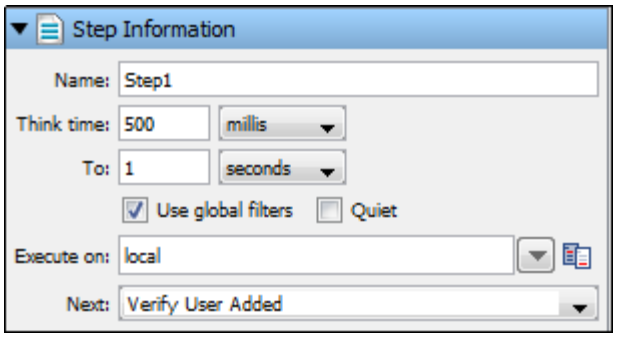


ステップ情報の追加

次の手順に従ってください:

1. 基本的なステップ情報を追加するには、右側のパネルのエレメント ツリーの［ステップ情報］タブを開いて展開します。

   ステップ情報エディタが開きます。



2. 以下のパラメータを入力します。

   名前

   ステップの名前。 このテキスト ボックスでステップの名前を変更できます。

   反応時間

   このステップを実行する前に、テスト ケースが待機する時間の長さ。 反応時間は、アクションを実行する前に何を実行するかユーザが決定するためにかかる時間の長さを DevTest にシミュレートさせます。 時間の計算方法を指定する方法

   a. 該当するドロップダウンをクリックして時間単位（ミリ秒、秒、分）を選択します。

   b. ［反応時間］フィールドに開始値を入力します。

   c. ［終了］フィールドには、終了値および時間インジケータを入力します。

      DevTest は、指定された範囲内のランダムな反応時間を選択します。たとえば、500 ミリ秒から 1 秒の間のランダムな間隔のユーザ反応時間をシミュレートするには、「500 ミリ秒」 および 「1 秒」 を入力します。

   グローバル フィルタの使用

ステップにグローバル フィルタの使用を指定するには、このボックスを選択します。 フィルタの詳細については、「*CA Application Test の使用*」の「フィルタの追加 (P. 133)」を参照してください。

クワイエット

DevTest にこのステップの応答時間イベントおよびパフォーマンスの計算を無視させる場合は、このチェック ボックスをオンにします。

実行

ステップが実行されるシミュレータを指定します。 特定のコンピュータで実行される必要があるステップには、特定のシミュレータを指定します。 たとえば、ログ ファイルを読み取る場合、ステップはログが存在するコンピュータで実行する必要があります。

次へ

テストで実行する次のステップ。 このステップがテスト ケースの実行を終了する場合は、次のステップを指定しません。このステップで起動されるアサーションはこの値を上書きします。

テスト ステップの設定

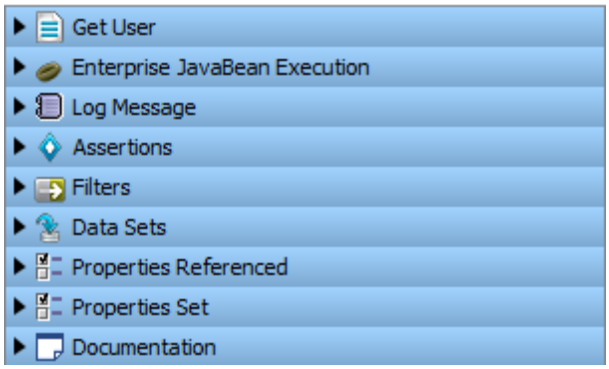
次の手順に従ってください:

1. モデル エディタでステップを追加します。

   ステップが追加されると、別のテスト ステップ パネルがエレメント ツリーに表示されます。

2. 設定エレメント（アサーション、フィルタ、データ セットなど）のパラメータを設定することにより、テスト ステップを設定します。

各テスト ケース/ステップ エレメントの詳細は、設定エレメントの隣のエレメント矢印  をクリックして展開することによって表示できます。



ステップ情報

［ステップ情報］エレメントはエレメント パネル内の最初のエレメントで、ステップの名前をそのラベルとして保持します。 上記の例では、［ステップ情報］エレメントは **Get User （ユーザの取得）** です。 テスト ステップを展開した後にその名前を変更できます。その後、現在のステップが完了した後に実行される次のステップを設定します。 その他のオプションについては、「テスト ステップの追加 (P. 222)」で説明しています。

ステップ タイプ情報

このフィールドには、選択したステップ タイプ（例では Enterprise JavaBean 実行］ ）のタイトルが表示されます。 ステップはそれぞれ異なり、それ自体の特定の設定要件を持っています。そのため、ステップを実行してテスト（および応答を取得）するために必要な情報を入力するカスタム エディタが用意されています。 このカスタム エディタは、エレメントが展開されたときに開きます。

ログ メッセージ

DevTest でステップが追加されると、このメッセージが表示され、ステップが実行された後に表示されるログ メッセージを設定できます。

アサーション

1 つ以上のアサーションの追加および設定。 アサーション設定では、アサーションが起動されたときに実行される次のステップも設定する必要があります。

フィルタ

フィルタの追加および設定。 フィルタは各テスト ステップのフィルタ エレメントの下に追加されます。

データ セット

テスト ステップに適用される 1 つ以上のデータ セットの追加および設定。データ セットはテスト ステップを実行する前に起動されます。 データ セットが設定された任意のプロパティをテスト ステップで使用できます。

参照プロパティ

ステップが参照する（読み取る）プロパティの読み取り専用リスト。

設定プロパティ

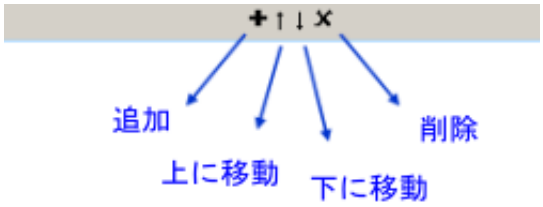
ステップが設定する（値を割り当てる）プロパティの読み取り専用リスト。

ドキュメント

テスト ステップに添付されるメモ。

ステップ エレメント ツールバー

すべてのエレメント（アサーション、フィルタ、およびデータ セット）には、［エレメント］タブの下部に固有のツールバーがあります。

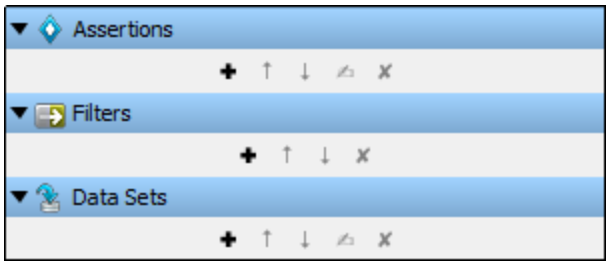


個別のエレメントの下部のアイコンをクリックすることにより、ステップに対してエレメントを追加/削除できます。

- フィルタの追加の詳細については、「フィルタの追加 (P. 133)」を参照してください。
- アサーションの追加の詳細については、「アサーションの追加 (P. 152)」を参照してください。

ステップへのフィルタ/アサーション/データ セットの追加

フィルタ、アサーション、およびデータ セットは、右側のパネルの各ステップに対応するエレメントの下に追加されます。



フィルタ、アサーション、またはデータ セットを追加するには、ツールバーの［追加］ をクリックします。

- フィルタの追加および設定の詳細については、「フィルタ (P. 132)」を参照してください。
- アサーションの追加および設定の詳細については、「アサーション (P. 151)」を参照してください。
- データ セットの追加および設定の詳細については、「データ セット (P. 171)」を参照してください。

次のステップの設定

次のステップの割り当て

テスト ケースのワークフローでは、選択したテスト ステップに「次のステップ」を割り当てることができます。

ワークフローの選択したステップを実行すると、実行用に定義した次のステップに移動します。

「次のステップ」を設定して、ワークフローのその他のステップに移動するか、以下のアクションを指示できます。

- テストを終了する
- テストを失敗させる
- テストを中止する

次の手順に従ってください:

1. 次のステップを決定するステップをクリックします。
2. 右クリックして［次のステップ］を選択し、対象の次のステップをクリックします。

モデル エディタでワークフローが変更されます。ステップ エディタの［次のステップ］フィールドの情報も変更されます。

また、テストを終了、失敗、中止させることもできます。

終了ステップ

終了ステップはワークフローの終了を意味し、ワークフローが正常に完了する場合に実行されます。実行がこのステップに到達すると、テスト ケース全体が成功と見なされます。

失敗ステップ

失敗ステップはワークフローの終了を意味し、ワークフローがエラー イベントにより失敗する場合に実行されます。 実行がこのステップに到達すると、テスト ケース全体が失敗と見なされます。 失敗ステップは、多くの内部 DevTest 例外（EB 例外など）のデフォルトです。 ただし、アサーションで失敗ステップを次のステップとして設定してテスト ケースを失敗させることができます。

中止ステップ

中止ステップもワークフローの終了を意味し、ワークフローが中止される場合に実行されます。 このステップに到達すると、テスト ケースは（完了せずに）中止されたと見なされます。

任意のステップまでの再生

テスト ケースをケース内の任意のステップまで迅速に再生できます。

ステップまで再生する方法

ステップをクリックし、右クリックして［ここから再生］を選択します。

選択したステップまでケースが再生されます。

スタータ ステップの設定

任意のテスト ステップをワークフローのスタータ ステップに設定できます。

その後、スタータ テスト ステップはテスト ケース ワークフローを開始します。

スタータ ステップを設定する方法

ステップをクリックし、右クリックして［スタータとして設定］を選択します。

選択したステップは、ワークフローの最初のステップとして設定されます。このオプションはテスト ケースの最初のステップには使用できません。

警告およびエラーの生成

次のステップとして、その他の 2 つのタイプのテストを設定できます。

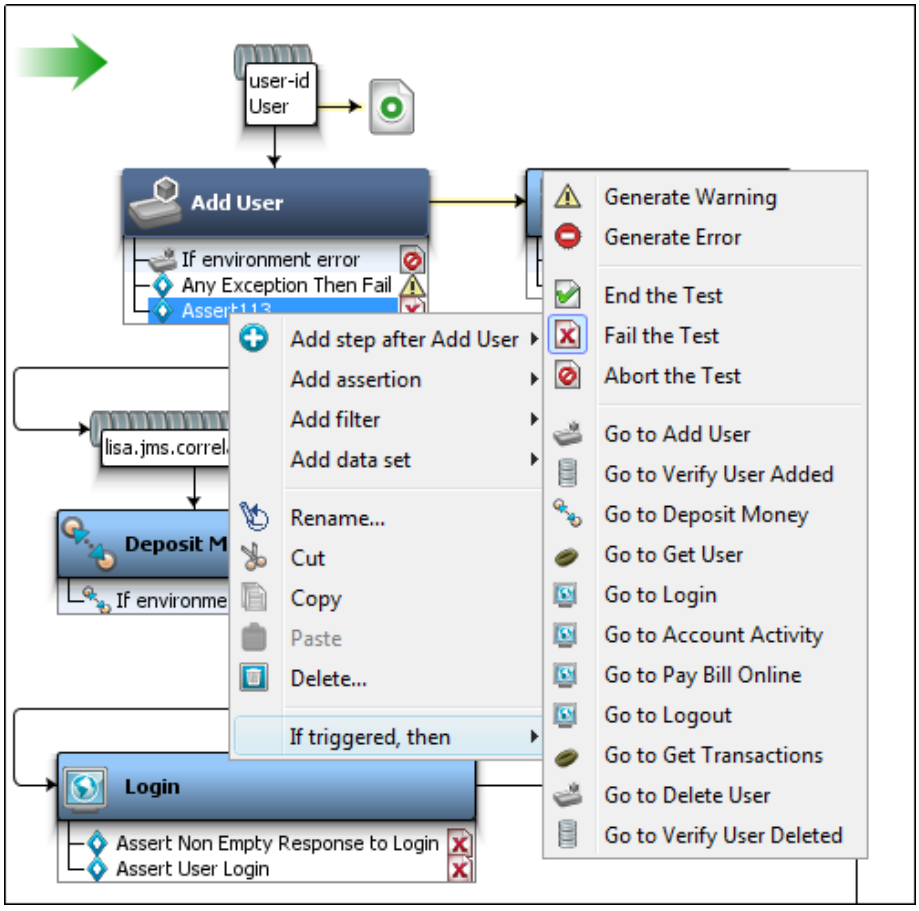
- 警告を生成する
- エラーを生成する

例として、Get User （ユーザの取得）ステップのアサーションを使用します。

次の手順に従ってください:

1. メニューを開くには、ステップを選択してアサーションを右クリックします。
2. ［警告を生成する］を選択します。

   アサーションがトリガされた場合のみ、テスト ケースはこの次のステップ（警告を生成する）に移動します。



警告を生成するステップ

テスト ステップが失敗した場合、DevTest は、アラームやイベントを生成しない「無視」タイプのステップ ロジック（警告を生成するステップ）を使用します。警告を生成するステップは、テスト ケースのワークフローを変更しません。

エラーを生成するステップ

テスト ステップは成功または失敗します。 失敗する場合、テスト ステップは実際にはテストを失敗させません。

テストを失敗させるために、テスト ステップはテスト ケースのワークフローで「失敗」ステップを実行するように設定します。 このアクションにより、ステップは暗黙的に失敗と見なされます。

明示的に失敗させる場合は、エラーを生成して NODEFAILED イベントを生成し、テスト ステップを続行します。

テスト ケースの作成

テスト ケースは、「テスト中のシステム」（または場合によっては完全な「テスト中のシステム」）のビジネス コンポーネントをテストする方法の完全な仕様です。

テスト ケースは XML ドキュメントとして保持され、特定のコンポーネントまたはシステムをテストするために必要な情報がすべて含まれています。テスト ケースは、DevTest ワークステーション で作成および管理されます。

テスト ケースの作成の最初のステップは、プロジェクトを作成 (P. 54)することです。 このプロジェクトには、単一または複数のテスト ケースを作成できます。

このセクションには、以下のトピックが含まれます。

テスト ケースの作成

プロジェクトを開くまたはプロジェクトを作成する (P. 54)ことにより、テスト ケースを作成できます。

たとえば、デフォルト プロジェクト「examples」では、Configs、Data、Staging Doc、Suites、Tests、およびその他のフォルダを持ったツリー構造が表示されます。

テスト ケースを作成する方法

1. プロジェクト パネルで Tests フォルダを右クリックし、［新規テスト ケースの作成］をクリックします。
2. ダイアログ ボックスで、テスト ケースを保存するディレクトリを参照し、新しいテスト ケースの名前を入力します。

   詳細については、「モデル エディタのテスト ケース (P. 236)」を参照してください。

テスト ケースを開く

既存のテスト ケースを開くまたは表示する方法

1. メイン メニューから［ファイル］-［開く］-［テスト ケース］を選択します。

   最近開かれたテスト ケースが表示されます。 テスト ケースは使用された順にリスト表示されます。 最新のものは、リストの先頭に表示されます。

2. 目的のテスト ケースがリストに存在する場合は、それを選択します。

   目的のテスト ケースがリストに存在しない場合は、［ファイル システム］、［クラス パス］、または［URL］をクリックして参照します。

3. 開くファイルを選択し、［開く］をクリックします。

   選択したテスト ケースがモデル エディタで開かれて表示されます。 また、プロジェクト パネルでダブルクリックすることにより、既存のテスト ケースを開くこともできます。

   これで、テスト ケースに新しいエレメント（フィルタ、アサーション）を追加したり、既存のエレメントを変更したりすることができます。

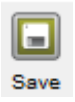
テスト ケースの保存

テスト ケースを保存する場合、DevTest は、テスト ケースの各ステップの結果も同じディレクトリの応答ドキュメント（「.rsp」というサフィックスの付いたファイル）に保存します。 詳細については、「応答ドキュメント (P. 241)」を参照してください。

テスト ステップ エレメント（フィルタ、アサーション、またはデータ セット）の必須フィールドが空白の場合、テスト ケースを保存できません。

テスト ケースを保存する方法

以下のいずれかを実行します。

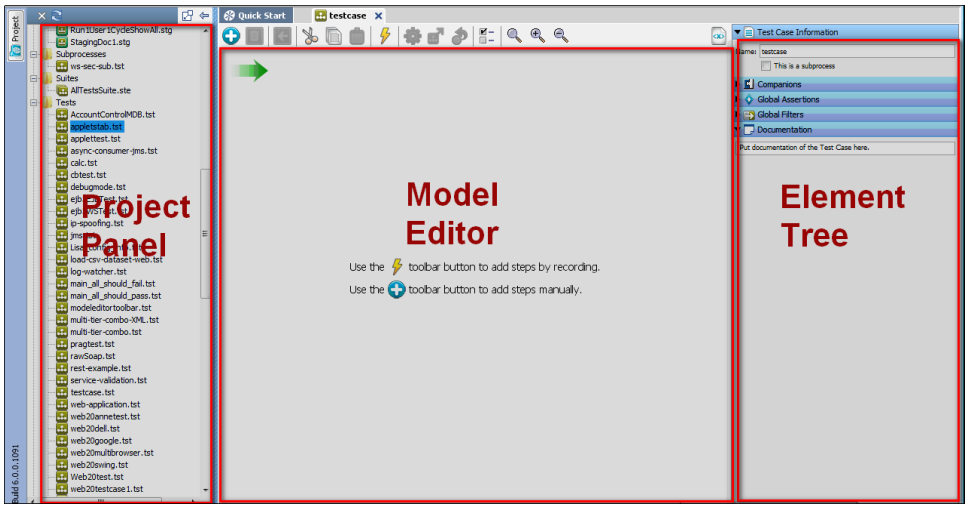
- ツールバーの［保存］ 

をクリックします。
- メイン メニューから［ファイル］-［保存］を選択します。 元のテスト ケース名が表示されます。

モデル エディタのテスト ケース

テスト ケースを作成または開く場合、モデル エディタが開きます。

モデル エディタのビューは、テスト ケースとそれに関連するすべてのテスト ステップおよびエレメントをグラフィカルに表示します。例として、**LISA_HOME/examples** ディレクトリのテスト ケースを開くことができます。



モデル エディタには、以下の 3 つのセクションがあります。

- プロジェクト パネル：プロジェクトを作成し、そのプロジェクトのテスト ケースまたはその他のドキュメントを作成、表示、または編集します。
- モデル エディタ：テスト ケースに対して作成されるテスト ステップを追加、編集、削除します。
- エレメント パネル：テスト ケースまたはテスト ステップに対して、フィルタ、アサーション、データ セット、またはコンパニオンを適用します。

テスト ステップの追加

ステップはいくつかの方法で追加できます。

テスト ステップを追加する方法

以下の手順のいずれかを実行します。

- メイン メニューから［コマンド］-［新規ステップ作成］を選択します。
- テスト ケース ツールバーの［ステップの追加］アイコンをクリックします。
- ワークフローのステップを右クリックし、［～の後にステップを追加］をクリックします。 ステップは、そのステップの直後/直下に追加されます。

ステップが既存のワークフローに挿入されると、挿入されたステップの両側のステップが自動的に更新され、ワークフローの線形が維持されます。

ワークフローに分岐またはループが含まれる場合、次のステップは自動的に設定されません。

次のステップの設定

モデル エディタで次のステップを設定する方法

1. 目的のテスト ステップを選択して右クリックし、メニューを開きます。
2. ［次のステップ］をクリックし、目的の次のステップを選択します。

テスト ケースのテスト ステップの並べ替え

モデル エディタでステップを並べ替える方法

1. ステップを選択して右クリックし、ワークフローで並べ替えます。
2. 次のステップとして設定するステップを選択します。

また、モデル エディタ内のステップにドラッグ アンド ドロップすることもできます。 ワークフローが線形の場合、ステップはすべて自動的に更新されます。 非線形のワークフローには、分岐およびループが含まれます。 ワークフローが非線形の場合、次のステップは更新されません。 非線形のワークフローの場合、次のステップは、そのステップの［ステップ情報］タブで更新できます。

テスト ケース内の分岐およびループ

テスト ケース内の分岐

テスト ケース内の分岐は、アサーションを使用して実行されます。

任意の数のアサーションをテスト ステップに適用できます。 すべてのアサーションには、true または false を評価する条件があります。アサーションの条件が満たされると最初のアサーションが起動され、ワークフローのパスが変更されます。 多くのステップでは、デフォルト エラー条件が失敗に対するアサーションとして自動的に作成されます。 アサーションを作成する場合、整合性を保ち、常に true または常に false で分岐することをお勧めします。

アサーションの詳細については、「アサーションの追加 (P. 152)」で説明しています。

テスト ケース内のループ

ループは、テスト ステップから後続のテスト ステップへとフローが流れ、特定のステップがフローの先頭のテスト ステップに制御を返す場合に作成されます。 重要なのはループから抜ける機能です。 条件で抜けるループ（プログラミングでの「while」ループ）は、ループを構成しているテスト ステップに対してアサーションを設定することにより実現されます。

指定された回数実行するか、特定のデータが使い果たされるまで実行するタイプのループ（プログラミングでの「for」または「foreach」ループ）は、データ セットで実現されます。 データ セットは、1 つ以上のプロパティに有限の回数の値を割り当てるために使用されます。 データ セットが使い果たされた後に実行される次のステップは、データ セット定義で指定されます。この定義は、ループから抜けるために使用できます。

たとえば、データ セットにシステムにログ記録する必要のある 20 行のユーザが含まれている場合、データ セットの各行に対してログイン ステップを実行するループを作成できます。 あるいは、特定のステップを特定の回数実行させるために数値カウント データ セットを使用できます。

単一のステップがそれ自体をコールし、データ セットに対してループすることができます。 複数のステップをループで実行して、データ セットのデータを使用することができます。 ステップのグループの最終ステップが、グループの最初のステップを指している場合に、これが実行されます。

データ セットの詳細については、「データ セット (P. 171)」で説明しています。

テスト ケースのインポート

現在の作業プロジェクト ディレクトリへ、古いテスト ケースをインポートできます。

テスト ケースをインポートする方法

1. プロジェクト パネルで Tests フォルダを右クリックし、［ファイルのインポート］を選択します。
2. そのフォルダにインポートするファイルを選択します。
3. ［OK］をクリックします。

   インポート プロセスが開始され、選択したフォルダにファイル（サブディレクトリ内のファイルを含む）がすべてコピーされます。テスト ケースおよび仮想サービス モデルの設定はすべて、プロジェクト設定にマージされます。

古いバージョンのファイル（LISA 5.0 より古いバージョン）をインポートする方法

1. メイン メニューから［ファイル］-［新規］-［プロジェクト］を選択します。

   ［新規プロジェクトの作成］ダイアログ ボックスが表示されます。
2. 古いバージョンのテスト ケースがあるディレクトリを選択します。
3. ［以下のいずれかに基づいて新しいプロジェクトを作成します］チェック ボックスをオンにします。
4. ［既存の DevTest ドキュメント ディレクトリからプロジェクトを作成］オプションを選択します。
5. ［作成］をクリックします。

   このアクションにより、新しいバージョンに変換された古いバージョンのすべてのファイルを使用してプロジェクトが作成されます。同じ名前のプロジェクトが存在する場合、新しいプロジェクトに別の名前を付けるよう求められます。

プロジェクト作成メッセージには、新しいバージョンの要件を満たすために変換されたすべてのテスト ケースおよび仮想サービス モデルの自動変換メッセージも含まれています。

完了後、以下のメッセージを確認できます。

- プロジェクトのマイグレーション
- 設定の変更の詳細
- テスト ケースの変更の詳細

インポートされたすべてのファイルが選択されています。

応答(.rsp)ドキュメント

Web サイトから記録するか、サーバと通信する場合、応答は応答ドキュメントに保存されるため、情報を後で使用できます。

応答ドキュメントは自動的に作成および管理され、テスト ケース ファイルと同じ名前と .rsp 拡張子を持つファイルとして保存されます。 テスト ケース ファイルと同様、応答ドキュメントは XML ファイルです。

応答ドキュメントには、テスト ケースの各 HTTP ベースのステップに対する HTTP 応答、Web サービス コールからの応答情報、およびデータベース クエリからの JDBC 結果が保持されます。

たとえば、対話型テスト ラン（ITR）ユーティリティを使用して HTTP ベースのステップを実行した場合、保存された応答ドキュメントにはテストの各 HTTP ベースのステップの結果の DOM ツリー全体が含まれます。 この応答情報は、データの検証、簡単なフィルタの作成、または簡単なアサーションの作成に使用できます。

HTTP 応答を使用したフィルタの作成の詳細については、「フィルタ (P. 132)」を参照してください。

HTTP 応答を使用したアサーションの作成の詳細については、「アサーション (P. 151)」を参照してください。

一部のステップには、応答ドキュメントに保存できない結果があります。

テスト ケース ファイルを別の場所にコピーする場合は、関連する応答ドキュメントをコピーすることを検討してください。そうすれば、保存された応答が失われません

テストの実行には必要ないため、応答ドキュメントはオプションです。応答ドキュメントは、結果の表示、DOM ツリーの表示、JDBC テーブルの表示などの機能を提供するために存在します。

再生機能を使用する場合、情報は応答ドキュメントから読み取られます。

別のプロジェクトのファイルへのアクセス

LISA_PROJ_ROOT プロパティを使用すると、現在のプロジェクトと同じディレクトリ レベルに位置する別のプロジェクトのファイルにアクセスすることができます。

たとえば、以下のプロジェクトがあると仮定します。

- C:¥Lisa¥Projects¥Project1
- C:¥Lisa¥Projects¥Project2

Project2 で作業している場合、以下のタイプのパスを指定することにより、Project1 のファイルにアクセスできます。

```
{{LISA_PROJ_ROOT}}/../Project1/Data/AccountNumbers.xls
```

環境全体で有効なパスを指定してください。

テスト ケース ツールバー

このツールバーは、モデル エディタでテスト ケースを開くと表示されます。 このツールバーのタスクはすべてテスト ケースに固有です。

アイコン	説明
	新しいステップを作成します。
	テスト ステップを削除します。
	現在選択しているステップをワークフローの開始ステップとして設定します。
	選択したテキストを切り取ります。
	選択したテキストをコピーします。
	選択したテキストを貼り付けます。

	クリックすると、メニューが開き、DevTest ブラウザでテスト ケースを記録することにより、ステップを作成します。 以下のものを使用してテスト ケースを記録できます。 ■ Web レコーダ（HTTP プロキシ） ■ モバイル レコーダ
	クリックして表示されるメニューで、以下から選択します。 ■ Selenium テスト ステップを作成するために JSON スクリプトをインポートする ■ Selenium テスト ステップを JSON スクリプトとしてエクスポートする
	新しい対話型テスト ラン（ITR）を開始します。
	クイック テストをステージングします。
	指定した時点までテストを再生します。
	モデル プロパティを表示します。
	ズーム スケールを 1 ： 1 にリセットします。
	ズーム イン。
	ズーム アウト。
	XML ソースを表示します。

サブプロセスの作成

サブプロセスは、スタンドアロンのテスト ケースとして実行されるのではなく、別のテスト ケースから呼び出されるテスト ケースです。

サブプロセスはその他のテスト ケースでモジュールとして使用できるため、再利用性が向上します。 多くのテスト ケースで共有可能なサブプロセスのライブラリを作成できます。

プログラミング言語では、サブプロセスは関数またはサブルーチンと呼ばれます。

テスト ケースは自己完結型である必要があります。 テスト ケースで使用されるすべてのプロパティの値は、テスト ケースの内部で取得する必要があります。 サブプロセスは、自身を実行していくつかのプロパティ値を提供するテスト ステップを予期します（入力プロパティ）。 サブプロセスが完了すると、コール元のステップでプロパティ値を使用できるようになります（戻り値プロパティ）。

サブプロセスはネストすることができます。 サブプロセスは別のサブプロセスをコールすることができます。

注: 別のサブプロセスをコールするサブプロセスが Quiet とマークされた場合、コールされたサブプロセスはすべて Quiet になります。

以下の相違点を考慮して、通常のテスト ケースに対して行うのと同じ方法でサブプロセスのステップを作成します。

- （「*CA Application Test の使用*」の「サブプロセス テスト ケースの作成 (P. 246)」で説明されているように）テスト ケースの［テスト ケース情報］タブで、テスト ケースをサブプロセスとしてマークします。
- テスト ケースのように、データ セットを追加しないでください。 代わりに、データ セットはケースのコールの一部である必要があります。データ セットが呼び出されると、現在の値がサブプロセスに渡されます。 例外は、データ セットがサブプロセス ロジック自体の一部である場合です。 このシナリオでは、グローバル データ セットを使用しません。
- コール元のステップから渡されると予期される任意のパラメータを初期化するために、サブプロセス内で設定ファイルまたはコンパニオンを使用しないでください。 テスト目的のために、これらの値は別の場所に追加します。 サブプロセスがコールされると、コール元のステップはこれらの値を渡します。

注: 親テスト ケース（サブプロセスをコールするテスト ケース）の反応時間パラメータは、サブプロセスに伝達されます。 サブプロセスの反応時間をコール元プロセスとは無関係に実行するには、**lisa.subprocess.setThinkScaleFromParent** という名前の testExec プロパティを「false」に設定して、各サブプロセスについて決定できるようにします。グローバルな上書きについては、local.properties で **lisa.subprocess.setThinkScaleFromParent=false** を設定します。

ゼロからサブプロセスを作成できます。または、既存のテスト ケースをサブプロセスに変換できます。

サブプロセスの実行テスト ステップは、サブプロセス テスト ケースのコールを容易にします。

このセクションには、以下のトピックが含まれます。

サブプロセス テスト ケースの作成

次の手順に従ってください:

1. テスト ケースを作成するか、または既存のテスト ケースを開きます。
2. テスト ケースの［テスト ケース情報］タブを開きます。

   ［テスト ケース情報］タブを開くには、モデル エディタ内の空白の領域のどこかをクリックします（ステップを選択しないでください）。［テスト ケース情報］タブが右側のパネルに表示されます。
3. このテスト ケースをサブプロセスにするには、［サブプロセス］チェック ボックスをオンにします。

   デフォルトでは、テスト ケースは、サブプロセスとして指定されません。
4. ［サブプロセス入力パラメータ］および［サブプロセス出力プロパティ］の新しい 2 つのタブが追加されます。
5. ［ドキュメント］タブは、サブプロセスのいくつかの詳細なドキュメントを提供します。

   このテキストは、サブプロセスをコールするすべてのテスト ステップで表示されます。
6. サブプロセス ステップの追加を完了したら、サブプロセスの入力および出力プロパティを設定します。

サブプロセス入力プロパティの定義

次の手順に従ってください:

1. ［サブプロセス入力パラメータ］タブをクリックします。
2. 入力パラメータおよび予期される入力パラメータを定義します。

   サブプロセスが必要とするパラメータのリストが［サブプロセス入力パラメータ］フィールドに表示されます。 いくつかのパラメータは自動的に追加されます。 その他のパラメータは、必要に応じて追加できます。

3. プロパティを追加するには、このパネルの下部の［追加］ をクリックします。
4. ［キー］（プロパティ名）、［説明］、および［デフォルト値］の値を入力します。

   対話型テスト ラン （ITR）機能でサブプロセスを実行する場合は、デフォルト値が使用されます。 この値では、それが通常のテスト ケースであるかのように、サブプロセスをテストできます。 別のテスト ステップがサブプロセスを呼び出す場合、これらのデフォルト値は無視されます。

5. 不要なプロパティを削除するには、［削除］ボタン を使用します。

   予期される入力パラメータ（必要なパラメータである場合があります）

   DevTest が、サブプロセスで、入力プロパティである可能性があるプロパティを検出すると、そのプロパティがこのフィールドにリスト表示されます。 それが有効な入力プロパティである場合は、［追加］ボタンを使用して、［サブプロセス入力パラメータ］リストにこのプロパティをプロモートします。

サブプロセス出力パラメータの定義

サブプロセス出力（結果）プロパティ：サブプロセスが設定するすべてのプロパティのリスト。 サブプロセスが必要とするパラメータのリストが［サブプロセス出力プロパティ］フィールドに表示されます。 いくつかのパラメータは自動的に追加されますが、一部のパラメータは手動で追加する必要がある場合があります。

プロパティを追加するには、このパネルの下部の［追加］ボタン  を使用します。

コール元のステップに対して使用可能にしないプロパティがここにある場合は、［削除］  ボタンを使用してそれらを削除します。

入力および出力プロパティが確認され、デフォルト値がすべての入力パラメータに設定された後、ITR でサブプロセスを実行できます。

既存のテスト ケースのサブプロセスへの変換

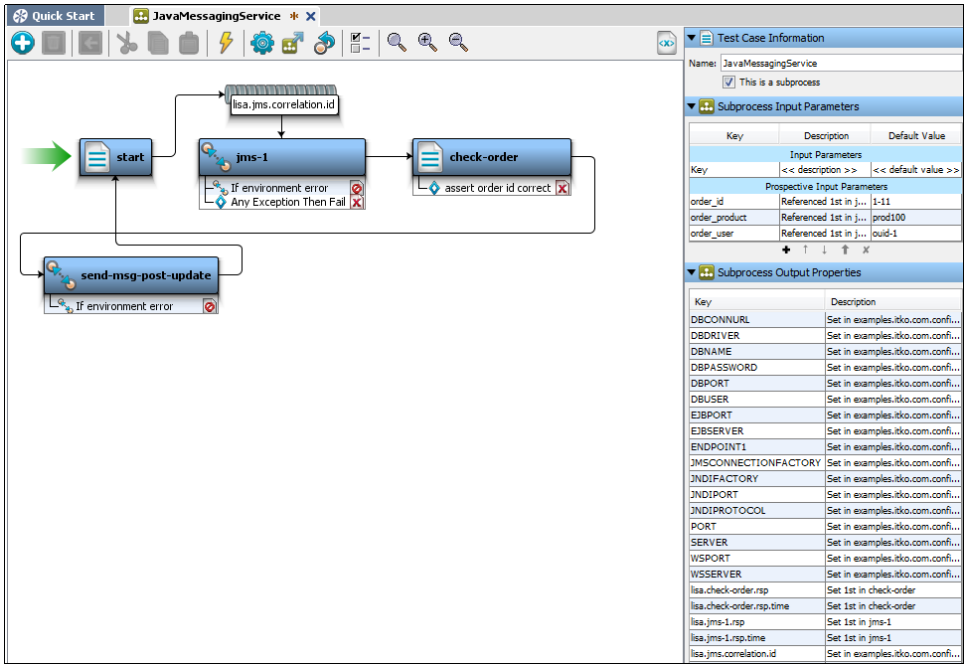
次の手順に従ってください:

1. テスト ケースを開き、適切に名前を変更します。
2. テスト ケースの［テスト ケース情報］タブで、テスト ケースをサブプロセスとしてマークします。
3. 新しいサブプロセスの入力プロパティであるプロパティの値を提供するデータ セットを削除します。

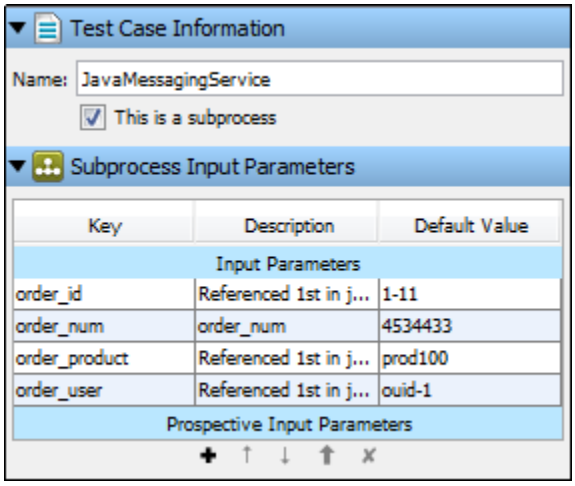
   例外は、データ セットがサブプロセス ロジック自体の不可欠な部分である場合です。
4. 設定ファイルから、任意のプロパティまたは新しいサブプロセスの入力プロパティであるパラメータを初期化するコンパニオンを削除します。
5. 入力および出力プロパティが確認され、デフォルト値がすべての入力パラメータに設定された後、ITR でサブプロセスを実行できます。

サブプロセスの例

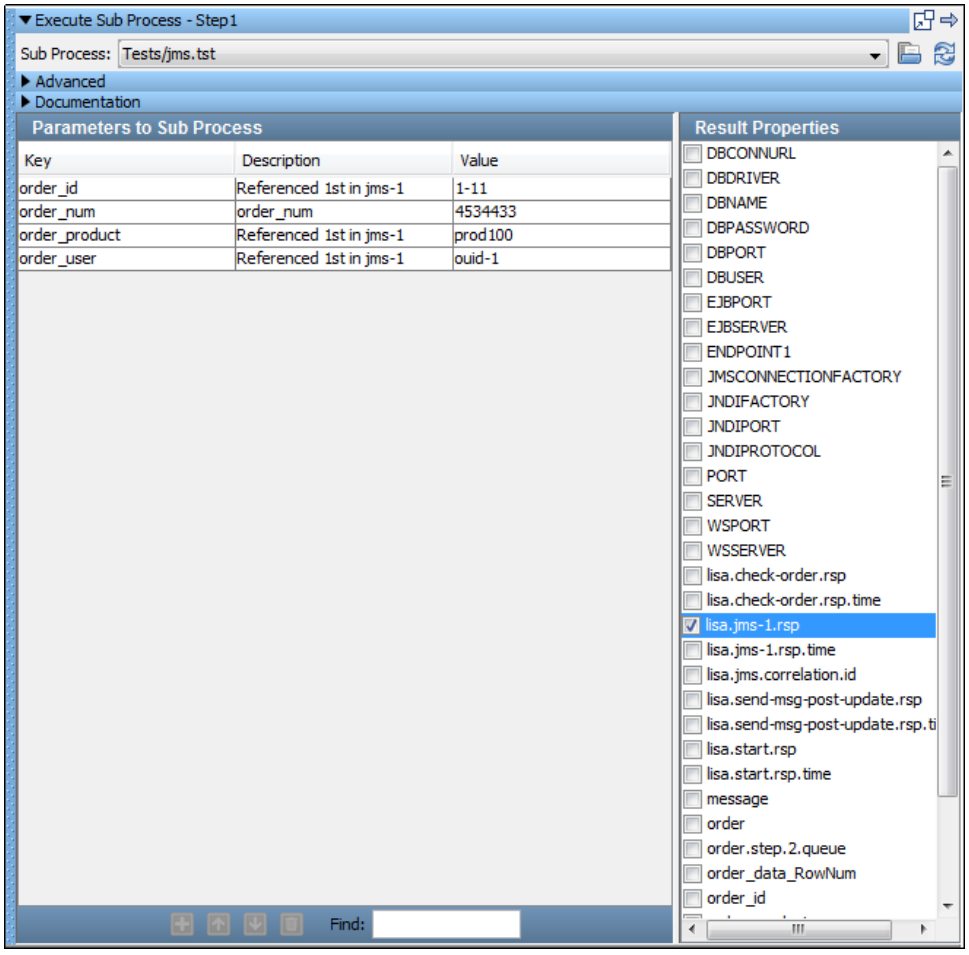
以下の例では、DevTest の examples ディレクトリの jms.tst テスト ケースから派生したサブプロセスとそれをコールするテスト ケースを示します。



1 つのデータ セット（order_data）が元のテスト ケースから削除されています。 order_data データ セットは、元のテスト ケースの **order_num** の値を提供していました。ここで、**order_num** プロパティは入力プロパティです。



以下の図は、1 つのテスト ステップ（サブプロセスの実行）を持ったテスト ケースを示しています。



ステップ 1 は［サブ プロセスの実行］タイプです。これは、サブプロセス **jms** （上記に示されています）を実行します。

入力プロパティは一致しています。また、コール元のステップは、サブプロセスの実行が完了した後に **lisa.jms-1.rsp** プロパティを利用可能にするように指定しています。

第 11 章: ドキュメントの作成

ステージング ドキュメントには、テスト ケースを実行する方法に関する情報が含まれます。

監査ドキュメントには、スイート内のテスト ケースに対する成功条件を設定できます。

テストが実行された後にテスト ケースをモニタするために使用するメトリックを適用したり、イベントを追加したりすることができます。

スイート ドキュメントはテスト ケースのコレクションを実行するために使用できます。

このセクションには、以下のトピックが含まれています。

ステージング ドキュメントの作成

ステージング ドキュメントには、テスト ケースを実行する方法に関する情報が含まれます。

このセクションには、以下のトピックが含まれます。

ステージング ドキュメントの作成

ステージング ドキュメントは DevTest ワークステーション で作成します。

テスト ケースの最初のステップにグローバル データ セットが含まれており、データ セットが使い果たされたときにテストを終了するようにデータ セットが設定されている場合、ステージングされた実行のすべてのテスト インスタンスが終了します。 安定状態時間などのその他のステージング パラメータは、上書きされます。

ローカル データ セットでは、ステージングされた実行をこのように終了したり、最初のステップ以外のステップにデータを設定したりしません。

次の手順に従ってください:

1. メイン メニューから［ファイル］-［新規］-［ステージング ドキュメント］を選択します。

   ［新規ステージング ドキュメント］ダイアログ ボックスが表示されます。

2. 新しいステージング ドキュメントの名前を入力します。
3. ［OK］をクリックします。

   ステージング ドキュメント エディタが表示されます。

4. ［ベース］タブ (P. 254) で、ステージング ドキュメントに関する基本情報を指定します。

   この情報には、ステージング ドキュメント名、ロード パターン、および配分パターンが含まれます。

5. ［レポート］タブ (P. 265) で、実行時に作成するレポートのタイプを指定します。
6. ［メトリック］タブ (P. 268) で、実行時に記録するメトリックを指定します。
7. ［ドキュメント］タブ (P. 271) で、ステージング ドキュメントに関する説明を入力します。
8. （オプション） ［IP スプーフィング］タブ (P. 272) で、IP スプーフィングを有効化および設定します。
9. （オプション）［ソース ビュー］タブ (P. 276) で、ステージング ドキュメントの XML バージョンを確認します。
10. メイン メニューから［ファイル］-［保存］を選択します。

ステージング ドキュメント エディタ

ステージング ドキュメント エディタでは、テスト ケースの実行条件を指定します。

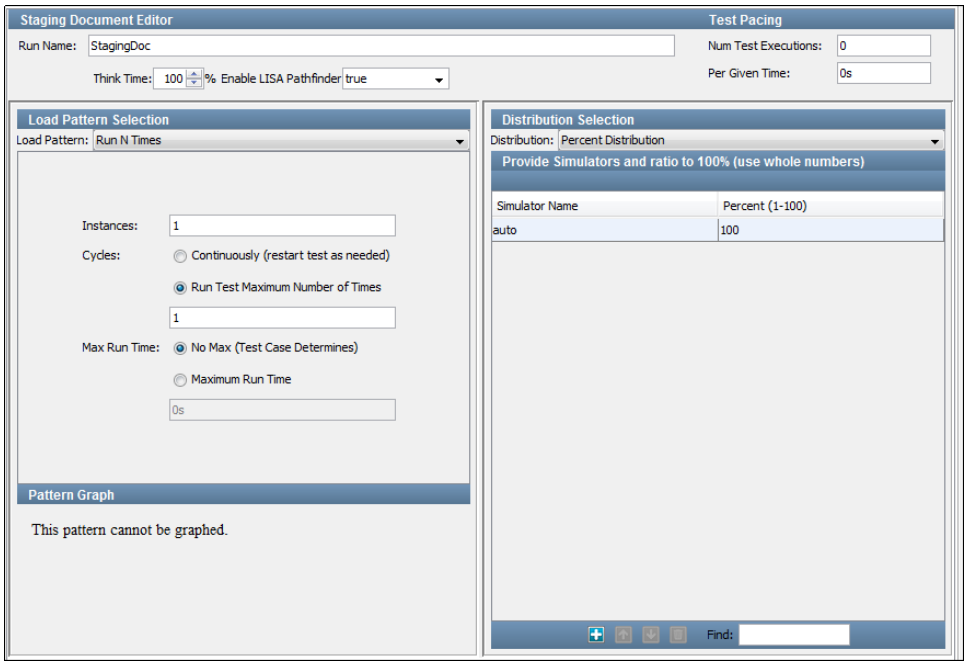
ステージング ドキュメント エディタには以下のタブが含まれます。

- ベース (P. 254)： 基本パラメータを指定します。
- レポート (P. 265)： レポートを選択および追加します。
- メトリック (P. 268)： メトリックを選択し、サンプリング間隔を指定します。
- ドキュメント (P. 271)： ステージング ドキュメントに関する説明を入力します。
- IP スプーフィング (P. 272)：IP スプーフィングの詳細を入力します。IP スプーフィングを使用すると、ネットワーク要求を作成する場合に、ネットワーク インターフェースで複数の IP アドレスを使用できます。
- ソース ビュー (P. 276)： ステージング ドキュメントの XML ソースを表示します。

ステージング ドキュメント エディタ - [ベース]タブ

ステージング ドキュメント エディタの［ベース］タブでは、テスト ケースの基本パラメータを設定します。

- 反応時間のグローバルな調整
- テスト期間（経過時間または実行数）
- テストのペースを調整する情報（特定の数のテストを指定した期間で完了するなど）
- 仮想ユーザ数
- 仮想ユーザのロード パターン
- 仮想ユーザの分散パターン



［ベース］タブは、以下のパネルで構成されています。

- 上部パネル
- ［ロード パターンの選択］パネル
- ［配分の選択］パネル

上部**パネル**

上部パネルでは、以下のパラメータを設定できます。

ラン名

ステージング ドキュメントの名前。

反応時間

テスト ケース内のすべてのテスト ステップの反応時間（パーセンテージ）。 各テスト ステップでは、ステップ情報セクションで反応時間を宣言できます。 ここでは、値のパーセンテージとして、それらの反応時間にグローバルな変更を適用できます。 以下に例を示します。

- 反応時間を削除するには、パーセンテージを 0% に設定します。
- 反応時間を半減させるには、パーセンテージを 50% に設定します。
- 反応時間を倍増させるには、パーセンテージを 200% に設定します。

CA CAI の有効化

CA Continuous Application Insight を有効にするか無効にするかを選択できます。

テスト実行回数

特定の時間で完了するテスト実行の数。

指定の実行時間

テストを実行する期間（実際の時間）。

値： h で時、**m** で分、**s** で秒

2,500 のテストが 8 分で完了するように指定できます。

反応時間は必要なペースを達成するためには変更されません。

速すぎてテスト ペースを達成できない場合、DevTest はテスト中に一時停止せずに実行します。 DevTest は、テストが要求より遅いペースで実行されていることをログに記録します。

指定された仮想ユーザが少なく、テスト ペースを達成できない場合、DevTest はユーザを追加しません。 必要な仮想ユーザ数を予測するには、オプティマイザ (P. 357) ユーティリティについての説明を参照してください。

［ロード パターンの選択］パネル

［ロード パターンの選択］パネルでは、以下のアクションを実行できます。

- テストの期間を設定する。
- 仮想ユーザ（インスタンス）の数を設定する。
- 仮想ユーザのロード パターン（複数の仮想ユーザが存在する場合）を設定する。

詳細については、「ロード パターンの選択 (P. 257)」を参照してください。

［配分の選択］パネル

［配分の選択］パネルでは、シミュレータを実行する仮想ユーザ（インスタンス）を配分できます。 詳細については、「配分の選択 (P. 262)」を参照してください。

ロード パターンの選択

［ステージング ドキュメント エディタ］の［ベース］タブには、［ロード パターンの選択］パネルが含まれます。

ロード パターン オプションは、以下のとおりです。

- すぐに起動
- 手動ロード パターン
- 実行回数を指定
- 段階的ステップ
- 加重平均パターン

注: ペースは、以下のロード パターンでのみサポートされています。

- 実行回数を指定
- すぐに起動
- 段階的ステップ

ペースは、安定状態のインスタンス数を予測するロード パターンの機能によって異なります。 動作している安定状態のインスタンスの数は、ステップが実行されるペースに影響します。 そのペースを予測できるのは、これらのロード パターンのみです。

すぐに起動

［すぐに起動］パターンは、少数の仮想ユーザ（インスタンス）を実行する場合に適用できますが、テストの期間を指定します。どのロード パターンにも影響を受けません。 このパターンは、すべての仮想ユーザを同時に開始します。

このパターンを設定するには、以下のパラメータを入力します。

インスタンス

仮想ユーザの数。

最大実行時間

「最大値なし」（テスト ケースによって時間が決定されます）か「最大実行時間」のいずれかを選択します。 後者の場合、［最大実行時間］を指定します。 この設定は、［サイクル］に入力されている値を上書きします。 **h** で時、**m** で分、**s** （または文字なし）で秒（デフォルト）を指定できます。

下部のグラフは、パターンのグラフィカル ビューを提供します。

手動ロード パターン

［手動ロード パターン］は、ユーザが仮想ユーザ（インスタンス）のロードおよびアンロードの大部分を制御できます。 このパターンは［段階的ステップ］パターンに似ていますが、追加する仮想ユーザ数およびパターンの各ステップの時間間隔の両方を個別に指定できます。

このパターンを設定するには、テーブルの行として各ステップを定義します。各行で、時間間隔および仮想ユーザ数を定義して、そのステップ（［インスタンスの変化］列）に対して追加または削除を行います。 経過時間（［合計時間］列）および仮想ユーザの総数（［実行中インスタンス］列）は、自動的に計算されます。

［間隔］列で、数値の後に、時の **h**、分の **m**、秒（デフォルト）の **s** （または文字なし）を使用して時間を入力します。

ステップの現在の順番を追加、削除、変更するには、テーブルの下部のツールバーにある標準アイコンを使用します。

これらのアイコンを表示するには、スクロールする必要がある場合があります。 または、行を選択して以下のショートカットを使用できます。

- 行の追加： Ctrl + Shift + A
- 行の削除： Ctrl + Shift + D
- 行を上に移動： Ctrl + Shift + 上方向キー
- 行を下に移動： Ctrl + Shift + 下方向キー
- 拡張ビュー： Ctrl + Shift + L

下部のグラフは、パターンのグラフィカル ビューを提供します。

実行回数を指定

［実行回数を指定］パターンは、1 人または少数の仮想ユーザ（インスタンス）のみを実行する場合に適用できますが、テストを実行する回数を指定します。どのロード パターンにも影響を受けません。このパターンは、すべての仮想ユーザを同時に開始します。

このパターンを設定するには、以下のパラメータを入力します。

インスタンス

仮想ユーザの数。

サイクル

［最大実行時間］に設定されている時間に達するまで継続的に実行するか、または最大回数を実行するかのいずれかを選択します。

最大実行時間

「最大値なし」（テスト ケースによって時間が決定されます）か「最大実行時間」のいずれかを選択します。 後者の場合、［最大実行時間］を指定します。 この設定は、［サイクル］に入力されている値を上書きします。 **h** で時、**m** で分、**s** （または文字なし）で秒（デフォルト）を指定できます。

段階的ステップ

［段階的ステップ］パターンは、一回ではなく、詳細に定義された段階を使用してシステムへ仮想ユーザ（インスタンス）を導入します。 安定状態のユーザ、ランプアップおよびランプダウン時間、ランプのステップ数を指定します。

このパターンを設定するには、以下のパラメータを入力します。

安定状態のインスタンス

安定状態で実行する仮想ユーザの最大数。

ステップ数

仮想ユーザの最大数に達するために使用するステップ数。 各ステップで導入する仮想ユーザ数は、安定状態のインスタンスをステップ数で割った数になります。

ランプアップ時間

仮想ユーザが追加され、仮想ユーザの最大数に到達する期間。 ステップ間の時間間隔は、ランプアップ時間をステップ数で割った時間になります。

安定状態時間

有意なテスト ランの期間。

ランプダウン時間

仮想ユーザが削除される期間。 テストは「停止」要求の後に完了するように動作するため、ランプダウン時間はおおよその時間です。

数値の後に、時の **h**、分の **m**、秒（デフォルト）の **s** （または文字なし）を使用して時間を入力します。

下部のグラフは、パターンのグラフィカル ビューを提供します。

加重平均パターン

［加重平均パターン］は統計の計算に基づいて仮想ユーザ（インスタンス）を追加および削除します。 DevTest は、加重移動平均を使用して、ステップ数およびステップ間の期間を毎秒ごとに計算します。 この分布は正規分布曲線に近似します。 大部分の仮想ユーザは、2 つの標準偏差のロードまたはアンロード ランプ時間の中間に追加されます。

このパターンを設定するには、以下のパラメータを入力します。

安定状態のインスタンス

安定状態で実行する仮想ユーザの最大数。

ランプアップ時間

仮想ユーザが追加され、仮想ユーザの最大数に到達する期間。

安定状態時間

有意なテスト ランの期間。

ランプダウン時間

仮想ユーザが削除される期間。 テストは「停止」要求の後に完了するように動作するため、ランプダウン時間はおおよその時間です。

数値の後に、時の **h**、分の **m**、秒（デフォルト）の **s**（または文字なし）を使用して時間を入力します。

下部のグラフは、パターンのグラフィカル ビューを提供します。

配分の選択

ステージング ドキュメント エディタの［ベース］タブには、［配分の選択］パネルが含まれます。

DevTest ワークステーション を使用する場合、仮想ユーザはローカルで（DevTest ワークステーション の一部であるシミュレータを使用して）実行されるため、［配分の選択］パネルを必要としません。

いくつのシミュレータ サーバがアクティブのまま DevTest サーバ を使用する場合、［配分の選択］パネルで、シミュレータ間での仮想ユーザの配分方法を指定できます。

配分のオプションは以下のとおりです。

- インスタンスの能力に基づいてバランス
- DCM による動的シミュレータ スケーリング
- パーセント配分
- ラウンド ロビン配分

インスタンスの能力に基づいてバランス

［インスタンスの能力に基づいてバランス］配分は、現在のロード（各シミュレータのロード率）の評価に基づいて、DevTest が仮想ユーザ（インスタンス）の割り当てを決定することを要求します。

各シミュレータは、定義された割り当て可能な仮想ユーザ数で開始されます。デフォルトは 255 です。DevTest は動的にロード率を追跡し、仮想ユーザを特定のシミュレータへ追加して、ロード率をできるだけ均等に維持しようとします。 そのため、最低のロード率のシミュレータは、システムに導入される次の仮想ユーザに対する候補となります。

この配分は、システムがすでにその他のいくつかのテスターからのテストを実行しており、ロード配分を最適化したい場合に役立ちます。

パラメータは不要です。

DCM による動的シミュレータ スケーリング

［DCM による動的シミュレータ スケーリング］配分は、テストの実行要件を満たすためにより多くの容量が必要になる場合（自動的にラボを展開することが必要になる場合）を DevTest が自動的に決定することを要求します。

この配分パターンには、以下のパラメータが含まれます。

チェックポイント時間

より多くのシミュレータが必要かどうかを DevTest が評価する間隔。 値は秒（例： 300s）または分（例： 5m）で入力します。

DCM 動的ラボ名

既存のコーディネータにステージングしてテストを実行するためにより多くの容量が必要と判定された場合に、このパラメータは、追加のシミュレータを実行するためにどのラボをアクティブ化するかを示します。 VLM プレフィックスおよび完全修飾ラボ名を含めます。

展開の最大数

展開時に作成されるシミュレータの最大数。 ユーザごとのシミュレータ数を制限するポリシーがある場合、このパラメータを使用してポリシーを強制できます。 デフォルト値の 0 は無制限を意味します。

チェックポイントの評価時に、DevTest は、現在実行されているシミュレータのパフォーマンスおよびオンラインにする必要がある仮想ユーザ数を確認します。 追加のシミュレータが必要な場合、DevTest はラボを展開するプロセスを開始します。 シミュレータがオンラインになると、DevTest はそれらのシミュレータへのトラフィックの送信を開始します。

パーセント配分

［パーセント配分］は、指定したパーセンテージに基づいてシミュレータに仮想ユーザ（インスタンス）を配分します。

実行されているシミュレータ（ローカル）の名前は［シミュレータ名］列のドロップダウン リストで使用可能です。

シミュレータを選択し、そのシミュレータ用の仮想ユーザのパーセンテージを指定します。100 パーセントになるまで、含めるすべてのシミュレータに対してこの操作を繰り返します。 パーセンテージには整数値を使用します。

注: ドロップダウン リストの選択肢として［自動］が表示されますが、推奨されていません。 「自動」は、明示的な配分の選択と競合するため、仮想ユーザの割り当てを混乱させる結果になります。

ラウンド ロビン配分

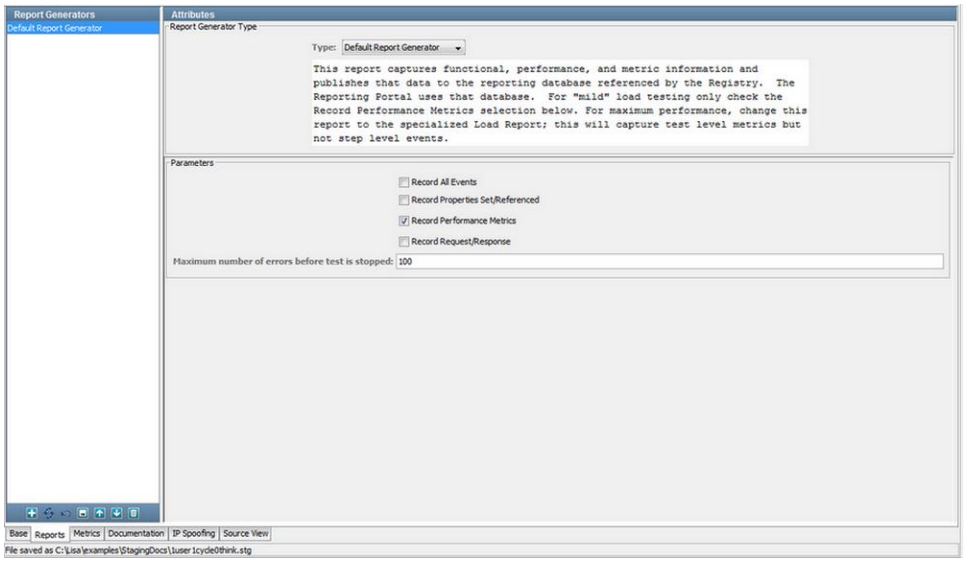
［ラウンド ロビン配分］は、単純なラウンド ロビン配分パターンに基づいて、DevTest が仮想ユーザ（インスタンス）の割り当てを制御することを要求します。

DevTest は、シミュレータをランダムに選択して仮想ユーザを追加し、それから別のシミュレータに移動してユーザを追加し、必要に応じてこの方法で追加を続けます。 すべてのシミュレータが使用されると、プロセスは最初のシミュレータで続行されます。 この配分は、システムで単一の大規模な負荷テストを行う場合に役立ちます。

パラメータは不要です。

ステージング ドキュメント エディタ - [レポート]タブ

ステージング ドキュメント エディタの［レポート］タブでは、すべてのテスト ケースまたはテスト スイートに対して呼び出すレポート ジェネレータを指定できます。



以下のレポート ジェネレータが利用可能です。

デフォルトのレポート ジェネレータ

機能、パフォーマンス、およびメトリックの情報をキャプチャし、そのデータをレジストリによって参照されるレポート データベースにパブリッシュします。レポート ポータルはこのデータベースを利用します。

ロード テスト レポート ジェネレータ

何千もの仮想ユーザを使用した負荷テスト用に設計されています。 このレポートでは負荷メトリックがキャプチャされますが、ステップ レベルのメトリックはキャプチャされません。 ステップ レベルのメトリックをキャプチャすると、データ量が多くなり過ぎ、レポート データベースによってテストのスピードが遅くなるおそれがあります。

XML レポート ジェネレータ

キャプチャ可能なすべてのデータを使用して XML ファイルを作成します。キャプチャされるデータは、レポート オプションを使用して制限することができます。このレポートを表示するには、レポート ポータルにファイルをインポートします。

各レポート ジェネレータで使用可能なデータ、レポートの表示、レポートの内容、および出力オプションの詳細については、「レポート (P. 435)」で説明しています。

レポート用にキャプチャするメトリックについては、「メトリックの生成 (P. 286)」でより詳細に説明しています。

レポートは以下のモジュールで選択でき、レポート ビューアで参照できます。

- クイック テストのステージング (P. 330)
- ステージング ドキュメントの作成 (P. 251)
- テスト スイートの実行 (P. 349)

要求された後、それらを後で表示および管理できます。

ドロップダウンからレポートを選択する場合、レポートのサマリがその下の領域に表示されます。

選択したレポートによって、パラメータは異なります。 必要な場合および場所に応じて、パラメータを設定します。

［レポート］タブは、以下の領域で構成されています。

右側のパネル

右側のパネルでは、追加するレポートを選択します。

属性

［属性］領域には、レポート ジェネレータ タイプ、およびレポートに必要なパラメータが含まれます。

- レポート ジェネレータ タイプ: プルダウン メニューに使用可能なレポートのタイプが表示されます。
- パラメータ : 選択したレポート ジェネレータを設定するために必要なパラメータ。

  ［テストが停止するまでの最大エラー数および失敗数］フィールドは、テストが中止されるまでに許可されるエラー数の制限を設定します。「0」のエントリは、すべてのエラーを許可することを意味します。

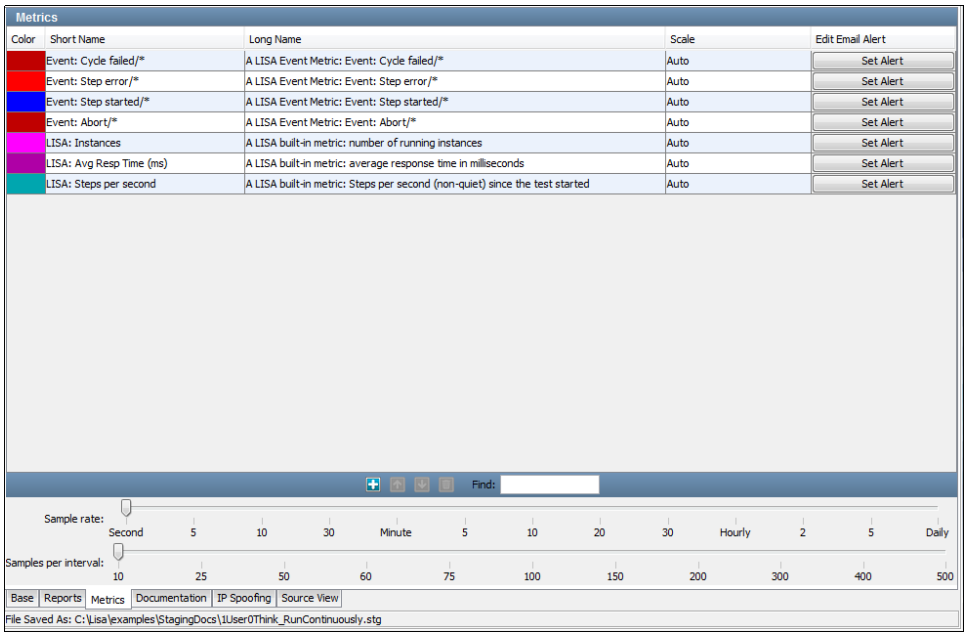
左側のパネル

左側のパネルには、追加されたレポートのリストが表示されます。 パネルの下部のツールバーを使用して、レポートを追加、保存、移動、および削除できます。

ステージング ドキュメント エディタ - [メトリック]タブ

ステージング ドキュメント エディタの［メトリック］タブでは、記録するテスト メトリックを選択できます。

また、メトリック収集のサンプリング仕様を設定でき、選択したメトリックに対して電子メール アラートを設定できます。



［メトリック］タブは、2 つのセクションで構成されています。 上部パネルには、デフォルトのメトリックが色分けされて表示されます。 下部パネルには、サンプリング パラメータが表示されます。

ステージング ドキュメントでメトリックを表すために使用される色を変更できます。色をクリックし、新しい色を選択して、ステージング ドキュメントを保存します。 テストをステージングする前に色を変更します。

上部パネルでは、メトリックの追加や削除、および電子メール アラートの設定を行えます。

以下のタイプのメトリックが使用できます。

- DevTest 包括テスト メトリック
- DevTest テスト イベント メトリック
- SNMP メトリック
- JMX 属性リーダ（JMX メトリック）

- TIBCO Hawk
- Windows Perfmon メトリック
- SSH 経由の UNIX メトリック

各メトリックの詳細については、「メトリック」を参照してください。

メトリックの追加

次の手順に従ってください:

1. ツールバーの［追加］アイコンをクリックします。

   メトリックを追加するためのダイアログ ボックスが、追加できるメトリックのリストと共に表示されます。

2. 目的のメトリック タイプを選択し、［OK］をクリックします。

   メトリック選択のダイアログ ボックスが表示されます。

3. メトリックの目的のサブカテゴリを選択し、［OK］をクリックします。

   サブカテゴリはメトリックごとに異なります。新しく追加されたメトリックが、既存のメトリックのリストに別の色で表示されます。

すべてのカテゴリのすべてのメトリックの説明、およびレポートに含めるようそれらを設定する方法の詳細については、「メトリックの生成 (P. 286)」を参照してください。

電子メール アラートの設定

個別のメトリックに対して電子メール アラートを設定する方法

1. メトリックに対応する［アラートの設定］ボタンをクリックします。

   ［アラートの編集］ダイアログ ボックスが表示されます。

2. メトリックの許容範囲（上限値および下限値）および電子メールで送信される詳細を設定できます。 電子メール サーバの名前および電子メール アドレスのリストを入力します。

   ステージング ドキュメントに追加される電子メール アラートは、SMTP パスワードを暗号化された値として保存します。 また、既存のステージング ドキュメントを開いて再保存すると、SMTP パスワードは暗号化された値として保存されます。

   電子メール アドレスを追加および削除するには、ウィンドウの下部の［追加］および［削除］ボタンを使用します。

3. 完了したら、［閉じる］をクリックします。

   ［メトリック］タブの［アラートの設定］ボタンが［アラートの編集］ボタンに変わっていることに注目してください。

注: 電子メール アラートが動作するには、lisa.properties ファイルで SMTP サーバ関連のパスを設定する必要もあります。

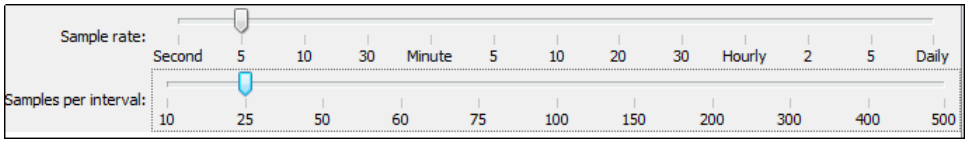
サンプリング パラメータ

下部パネルには、サンプリング パラメータを設定できる 2 つのスライダ バーがあります。

- サンプル レート：このパラメータは、サンプルを収集する（メトリックの値を記録する）頻度を指定します。［サンプル レート］は期間として指定します。これは、サンプル レートの逆数です。
- サンプル数（間隔あたり）：このパラメータは、メトリックのサマリ値を計算する間隔を設定するために使用されるサンプル数を指定します。

サンプル値を毎分取得（サンプル レート = 1 分）して、60 サンプルごとに平均（サンプル数（間隔あたり） = 60）すると、メトリック値が毎分生成され、サマリ値（平均）が毎時間計算されます。

たとえば、デフォルトでは、サンプル レートは 1 秒でサンプル数（間隔あたり）は 10 （間隔は 10 秒）です。



上記の例では、1 つのサンプルあたり 5 秒、および 1 つの間隔あたり 25 のサンプルを使用します。 これらの値は、5 秒ごとにメトリック値、および 125 秒ごとにサマリ値を作成します。

メトリック値は、レポートに含めるために XML ファイルまたはデータベース テーブルに格納されます（「レポート (P. 265)」を参照）。

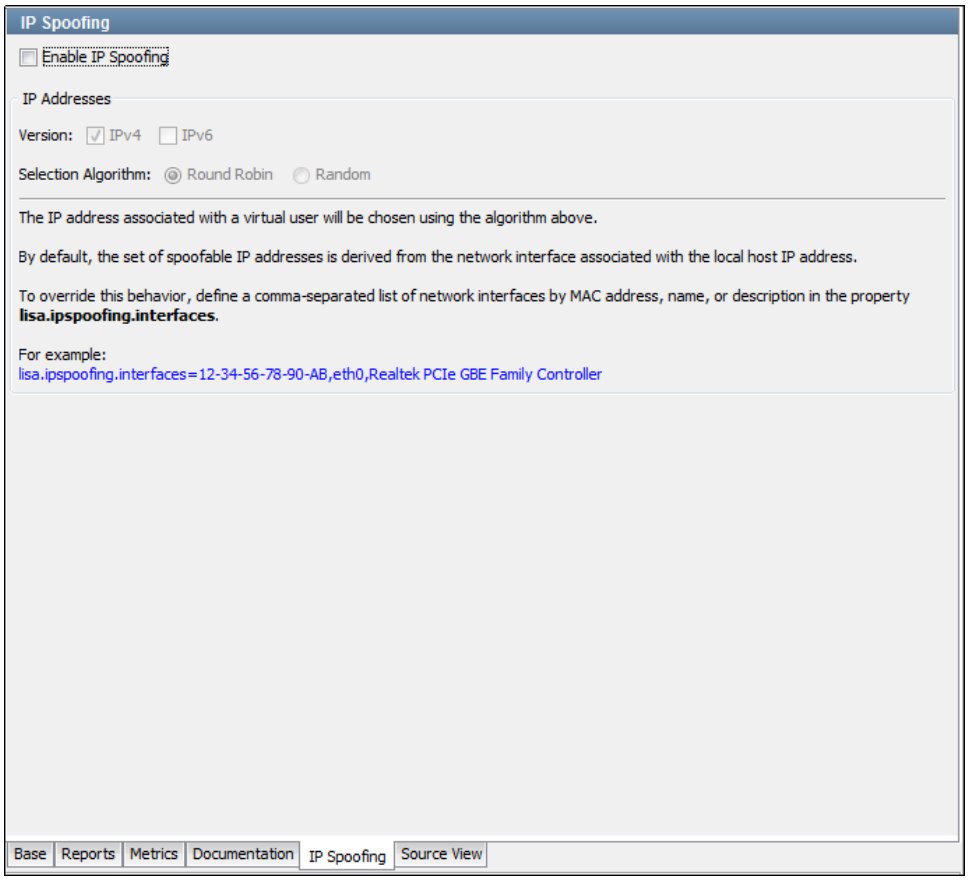
ステージング ドキュメント エディタ - [ドキュメント]タブ

ステージング ドキュメント エディタの［ドキュメント］タブでは、ステージング ドキュメントに関する説明を入力できます。

ステージング ドキュメント エディタ - [IP スプーフィング]タブ

ステージング ドキュメント エディタの［IP スプーフィング］タブでは、ネットワーク 要求を作成する場合に、ネットワーク インターフェースで複数の IP アドレスを使用することができます。

パフォーマンス テスト シナリオで、テスト中のシステムに対して IP スプーフィングを有効にすることにより、要求がさまざまな仮想ユーザから送信されているようにできます。 Web アプリケーションなどの一部のシステムでは、この結果は実際の動作と非常に似ています。



IP スプーフィングの有効化

選択すると、IP アドレスはテスト ケース内のすべてのサポートされたテスト ステップに対してスプーフィングされます。

バージョン

- IPv4： 選択すると、IPv4 アドレスがスプーフィングされます。
- IPv6： 選択すると、IPv6 アドレスがスプーフィングされます。

IPv4 または IPv6 のいずれかを選択する必要があります。

選択アルゴリズム

- ラウンド ロビン： スプーフィングされる IP アドレスがラウンド ロビン形式で選択されます。
- ランダム： スプーフィングされる IP アドレスがランダムな形式で選択されます。

IP スプーフィングのサポート

IP スプーフィングは HTTP に対してのみサポートされています。

以下のステップに対して、IP スプーフィングの使用を設定できます。

- HTTP/HTML 要求
- REST ステップ
- Web サービス実行（XML）
- RAW SOAP 要求

IP スプーフィング シナリオ

IP スプーフィングがステージング ドキュメントで設定される場合、以下の状況が予想されます。

ループのある単一ユーザの単一のテスト ケース

テストの 1 つのサイクルのループはそれぞれ、IP アドレス リストのシーケンシャルまたはランダムな順番に基づいて、異なる IP アドレスを取得します。

ループのない単一ユーザの単一のテスト ケース（テストはすべてのサイクルで開始および終了）

IP アドレスが異なるように、ランダムなアドレスを使用します。［**ランダム**］を選択しない場合、各ユーザは常に最初の IP アドレスから開始します。

ループのある複数のユーザの単一のテスト ケース（単体として動作）

テストの実行またはサイクルでは、使用する IP アドレスのリストが保持されます。

ループのない複数のユーザの単一のテスト ケースでは[ランダム]が選択されている必要があります。

［**ランダム**］を選択することにより、サイクルの開始時に、リストの 1 番目の IP アドレスではなく、ランダムな IP アドレスが使用されます。

Windows での IP アドレスの設定

このセクションでは、IP スプーフィングのためのネットワーク インターフェースに IP アドレスを追加する方法について説明します。

注: IP アドレスを追加する前に、管理者であることを確認してください。

Windows では、IP アドレス情報はコマンドライン ユーティリティの **ipconfig** を使用して取得できます。

単一の IP アドレス 192.168.0.201 を追加するには、**netsh** コマンドライン ユーティリティを使用します。

```
netsh in ip add address "Local Area Connection" 192.168.0.201 255.255.255.0
```

複数の IP アドレスを追加するには、**netsh** をループで使用します。

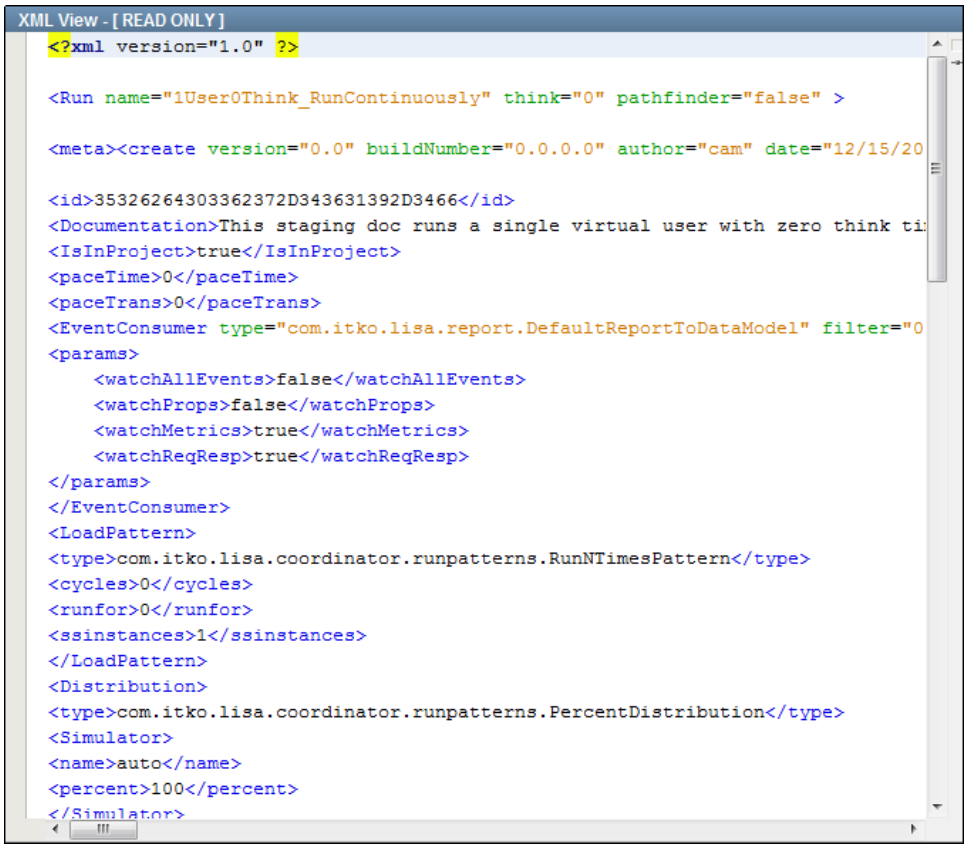
たとえば、以下のコマンドは、192.168.0.202 ～ 192.168.0.210 の 9 つの IP アドレスを追加します。

```
for /L %i in (202, 1, 210) do netsh in ip add address "Local Area Connection"
192.168.0.%i 255.255.255.0
```

これらのコマンドが成功した場合、コマンドライン ユーティリティの **ipconfig /all** を使用して新しい IP アドレスを確認できます。

ステージング ドキュメント エディタ - [ソース ビュー]タブ

ステージング ドキュメント エディタの [ソース ビュー] タブには、ステージング ドキュメントの XML ソースが表示されます。



ステージング ドキュメントの例

サンプルのステージング ドキュメントは、examples プロジェクト (P. 58) の StagingDocs フォルダで提供されています。 これらはすべて、［実行回数を指定］ロード パターンおよび［パーセント配分］パターンを使用します。

これらのドキュメントは、新しいステージング ドキュメントのベースとして使用できます。 次に、名前を変更して保存します。

- **1User0Think_RunContinuously.stg :** このステージング ドキュメントは、ゼロの反応時間で単一の仮想ユーザを実行します。 このステージング ドキュメントはテストを継続的に実行します。これは必ずしも永久という意味ではありません。
- **1user1cycle0think.stg :** このステージング ドキュメントは、ゼロの反応時間で単一の仮想ユーザを 1 回実行します。
- **ip-spoofing.stg :** このステージング ドキュメントでは、IP スプーフィングのサポートをテストできます。
- **jboss-jmx-metrics-localhost.stg :** このステージング ドキュメントは、440 秒間、3 人の同時仮想ユーザを 1 回実行します。
- **Run1User1Cycle.stg :** このステージング ドキュメントは、100 パーセントの反応時間で単一の仮想ユーザを 1 回実行します。
- **Run1User1CyclePF.stg :** このステージング ドキュメントは、100 パーセントの反応時間で単一の仮想ユーザを 1 回実行します。 CAI は有効です。
- **Run1User1CycleShowAll.stg :** このステージング ドキュメントは、100 パーセントの反応時間で単一の仮想ユーザを 1 回実行します。 CAI は有効です。

監査ドキュメントの作成

監査ドキュメントには、スイート内のテスト ケースに対する成功条件を設定できます。

監査ドキュメントでは以下のことを追跡できます。

- テスト中に発生する必要があるイベント、または発生してはならないイベント
- テスト の実行にかかる時間が短すぎないか、または長すぎないか

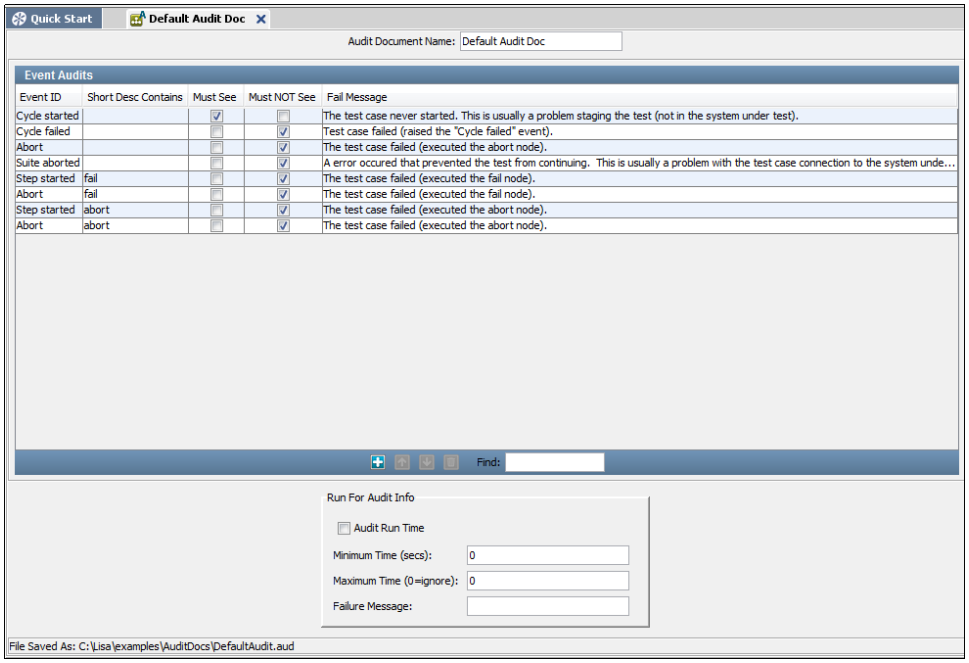
監査ドキュメントは、スイート ドキュメント エディタの［ベース］タブ (P. 289)で指定できます。 ［レポート］ タブ (P. 294)の［すべてのイベントを記録］チェック ボックスをオンにします。

監査ドキュメントは、プロジェクトの AuditDocs フォルダに配置されます。ファイル拡張子は **.aud** です。

プロジェクトを作成すると、AuditDocs フォルダに **DefaultAudit.aud** という名前のデフォルトの監査ドキュメントが含まれます。

注: 監査ドキュメントを使用する場合、テスト ステップ名にはテストで使用される監査ドキュメントの簡単な説明を含めることができません。 たとえば、**Abort** が監査ドキュメント内の簡単な説明である場合、**Abort** という単語をテスト ステップ名に使用することはできません。

以下の図は、監査ドキュメント エディタを示しています。 このエディタには、［イベント監査］パネルと［監査実行情報］パネルが含まれます。



監査ドキュメントを作成する方法

1. メイン メニューから［ファイル］-［新規］-［テスト監査］を選択します。
2. 監査ドキュメントの名前を入力し、［OK］をクリックします。

   監査ドキュメント エディタが開きます。
3. イベントを監査する場合は、［イベント監査 (P. 280)］パネルのパラメータを設定します。
4. 実行時間を監査する場合は、［監査実行情報 (P. 281)］パネルのパラメータを設定します。
5. メイン メニューから［ファイル］-［保存］を選択します。

[イベント監査]パネル

［イベント監査］パネルでは、テスト中に発生する必要があるか、発生してはならないイベントを指定できます。

イベントを追加する方法

1. ［追加］ をクリックします。
2. 新しい行で、［イベント ID］列のドロップダウン リストからイベント名を選択します。

   各行には以下のパラメータがあります。

   短い説明の内容

   イベントの短い説明の値に基づいてイベントをフィルタする場合は、この列で短い説明からキーワード（複数可）を入力します。

   発生する

   イベントがテスト中に発生する必要がある場合は、このチェックボックスをオンにします。

   発生しない

   イベントがテスト中に発生してはならない場合は、このチェックボックスをオンにします。

   失敗時のメッセージ

   この監査が失敗した場合にログに記録されるメッセージ。
3. 含める各イベントに応じて行を追加します。

   論理的に競合するイベント監査を追加しようとすると、エディタに警告メッセージが表示されます。

   ［上へ］ および［下へ］ アイコンを使用して、行を並べ替えることができます。

   ［削除］ アイコンを使用して、行を削除できます。

[監査情報実行]パネル

［監査情報実行］パネルでは、実行時間を監査できます。

このパネルには、以下のパラメータがあります。

実行時間の監査

実行時間の監査を有効にするには、このチェック ボックスをオンにします。

最小時間

テストを実行する必要のある最小時間（秒単位）。

最大時間

テストを実行でき、正常な監査と見なすことができる最大時間（秒単位）。 最大時間の制約がない場合は、「0」を入力します。

失敗時のメッセージ

この監査が失敗した場合にログに記録されるメッセージ。

イベントについて

イベントは、すべての関係者にアクションが発生したことを伝える DevTest からのメッセージ ブロードキャストです。

このセクションには、以下のトピックが含まれます。

- イベントの概要（P. 282）
- イベントの追加および表示（P. 284）

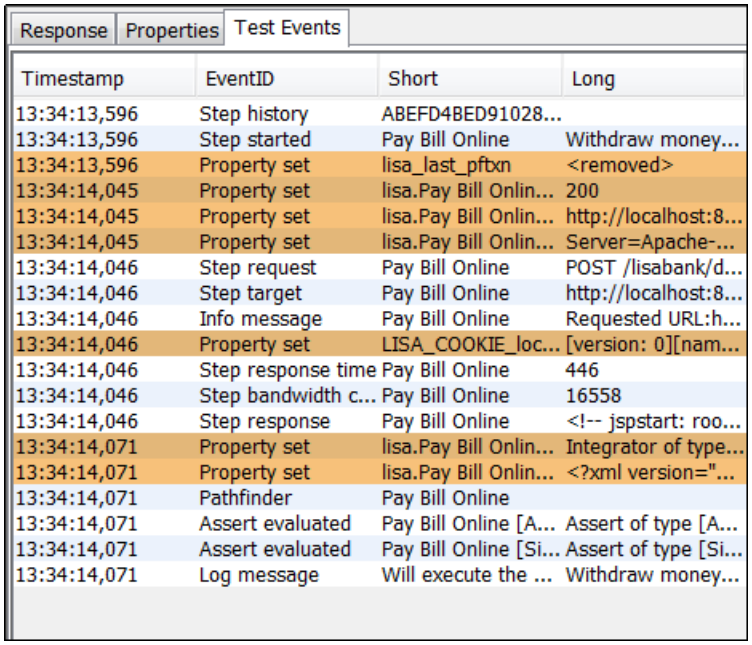
イベントの概要

イベントは、すべての関係者にアクションが発生したことを伝えるブロードキャスト メッセージです。 イベントは、テスト ケースで発生するすべての主要なアクションに対して作成されます。

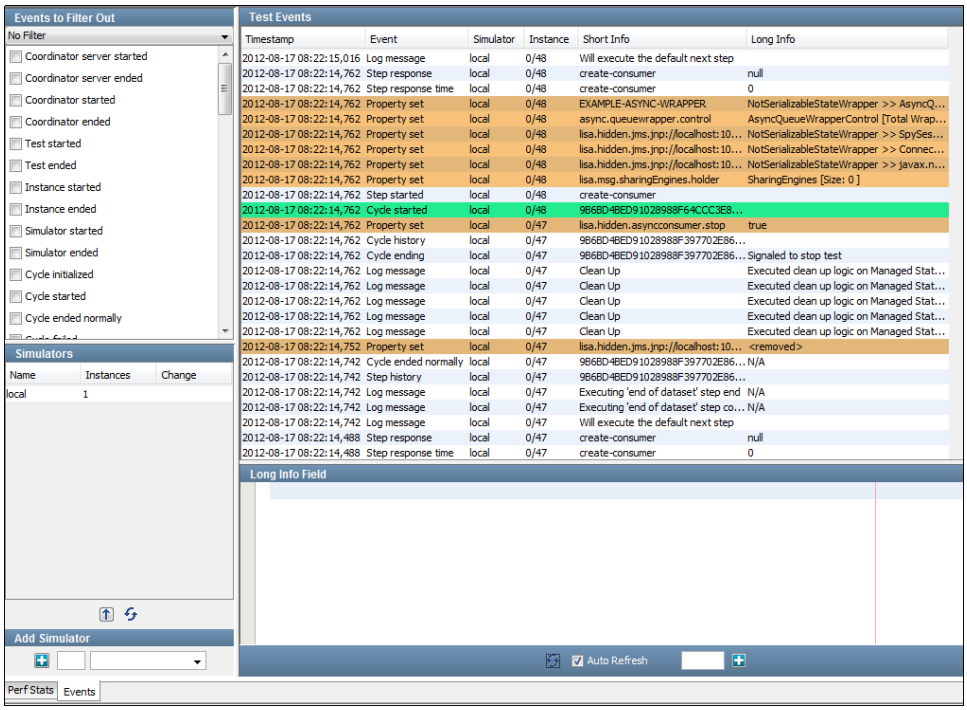
イベントは、ステップの実行、プロパティの設定、応答時間のレポート、結果の受信、テストの成功または失敗などで毎回作成されます。 すべてのイベントを参照し、関係のないイベントをフィルタできます。

テストのモニタやテスト結果の分析において、イベントは重要です。

ITR の [テスト イベント] タブをクリックすると、対話型テスト ラン（ITR）中にイベントを観察できます。 以下の図は、［テスト イベント］ タブを示しています。



クイック テストのイベントを観察およびフィルタできます。



同じことは、ステージングされたテストまたはテスト スイートのモニタ時にも実行できます。

レポートに含めるイベントを選択できます。また、モニタおよびレポートに含めることができるメトリックとして使用するイベントを選択できます。

レポートの詳細については、「レポート (P. 435)」を参照してください。メトリックの詳細については、「メトリックの生成 (P. 286)」を参照してください。

注: DevTest の内部では、ステップはノードとも呼ばれます。これが、一部のイベントがイベント ID 内に「node」を持つ理由です。

イベント ID、Short 値、および Long 値は、イベントの特徴を示します。イベント ID はイベントのタイプを示すキーワードです。Short 値および Long 値は、イベントについての情報を含みます。 それらのコンテンツはイベントのタイプによって異なります。

イベントの追加および表示

イベントを追加および表示できます。

- ITR を使用
- クイック テストまたはステージング ドキュメントを使用

ITR によるイベントの追加および表示

対話型テスト ラン（ITR）ユーティリティの［設定］タブでいくつかのイベントを有効にできます。

それらのイベントを表示するには、［コール結果を表示］および［ノード レスポンスを表示］チェック ボックスをオンにします。

［テスト イベント］タブをクリックして、ITR でテスト イベントを表示できます。

クイック テスト/ステージング ドキュメントによるイベントの追加および表示

［テストを開始］または［Start Suite］（スイートの開始）オプションを使用してテスト ケースを開始すると、テストがステージングされ、関連するグラフが［パフォーマンス統計］タブに表示されます。

テスト イベントを表示するには、［イベント］タブをクリックします。

左側のパネルの［フィルタで除外するイベント］を選択するか、または事前定義済みのフィルタ セットから選択できます。

テスト イベントのリストを更新するには、［イベント］タブの［自動リフレッシュ］チェック ボックスをオンにします。テストが実行されると、［イベント］タブに指定したイベントが表示されます。

テスト スイートのイベントの表示

テスト スイートをステージングした場合、イベントも観察できます。

レポートに含めるイベントを選択できます。また、モニタおよびレポートに含めることができるメトリックとして使用するイベントを選択できます。

レポートの詳細については、「レポート (P. 435)」を参照してください。メトリックの詳細については、「メトリックの生成 (P. 286)」を参照してください。

イベントには、イベント ID、Short 値、および Long 値があります。イベント ID はイベントのタイプを示すキーワードです。 Short 値および Long 値は、イベントについての情報を含みます。 それらのコンテンツはイベントのタイプによって異なります。

注: DevTest の内部では、ステップはノードとも呼ばれます。これが、一部のイベントがイベント ID 内に「node」を持つ理由です。

メトリックの生成

メトリック収集（CA 独自のメトリック計算方法）は、拡張可能なレポート メカニズムであり、さまざまな出力のレポートを生成できます。

メトリックにより、テストおよびテスト中のシステムのパフォーマンス/機能面に定量的手法および測定単位を適用できます。

ソフトウェア メトリックは、ソフトウェア、ハードウェア システム、またはそれらの仕様のプロパティの測定単位です。 メトリックを使用する定量的手法は、コンピューティングの複数の分野で非常に強力であると証明されており、テストもその例外ではありません。

メトリックは、以下の 2 つのグループに大別されます。

- **ゲージ：** ゲージは、応答時間や CPU 使用率などの値を瞬間的に読み取るものです。
- **カウンタ：** カウンタは、失敗したテストの数など、プロパティの継続的な数を示します。 メトリックがカウンタである場合、これは基本的にテストの実行時に継続的に増加する数 (エラーの数) です。メトリックにカウンタとしてフラグが立てられた場合、レポート コンソールで異なってグラフ化されます。 グラフ化される値は、現在のサンプルと前のサンプルの差異です。

ほとんどのメトリックは、メトリックのタイプ（ゲージまたはカウンタ）がすでにわかっています。 メトリックをどちらかのタイプでも使用できる場合は、目的のタイプを指定できます。

メトリックは、以下のステージング ドキュメントやテスト スイート ドキュメントに追加できます。 テスト モニタおよびクイック テストでは、メトリックを生成できます。 各ドキュメントまたはテストに対するメトリックの指定や生成については、各エディタのセクションで説明しています。

- クイック テスト (P. 330)： テストをモニタするために使用します。
- ステージング ドキュメント (P. 268)： レポートに含めるために使用します。
- テスト スイート ドキュメント (P. 296)： レポートに含めるために使用します。
- テスト モニタ (P. 331)： テストをモニタするために使用します。

テスト スイートの作成

DevTest では、テスト ケースを組み合わせて一緒に実行したり、ステージングしたりすることができます。 一緒に実行されるテスト ケースのコレクションは、テスト スイートという形態になります。

このセクションには、以下のトピックが含まれます。

テスト スイートの作成

テスト スイートは、DevTest ワークステーション から作成します。

次の手順に従ってください:

1. メイン メニューから［ファイル］-［新規］-［スイート］を選択します。

   スイート ドキュメント エディタが開きます。

2. ［ベース］タブ (P. 289) で、スイートの基本情報を指定します。

   この情報には、スイート名およびスイートのエントリが含まれます。

3. ［レポート］タブ (P. 294) で、実行時に作成するレポートのタイプを指定します。

4. ［メトリック］タブ (P. 296) で、実行時に記録するメトリックを指定します。

5. ［ドキュメント］タブ (P. 298) で、スイートに関する説明を入力します。

6. メイン メニューから［ファイル］-［保存］を選択します。

テスト スイート エディタ

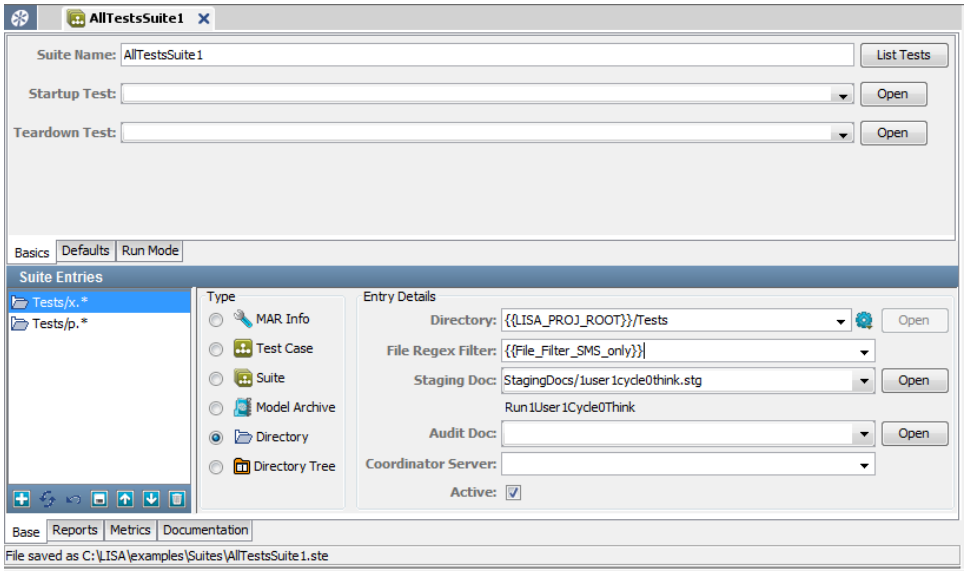
テスト スイートを作成する場合および既存のテスト スイートを開く場合には、テスト スイート エディタが開きます。

このページの特定のタブの詳細については、以下を参照してください。

- テスト スイート エディタ - ［ベース］ タブ (P. 289)
- テスト スイート エディタ - ［レポート］ タブ (P. 294)
- テスト スイート エディタ - ［メトリック］ タブ (P. 296)
- テスト スイート エディタ - ［ドキュメント］ タブ (P. 298)

テスト スイート エディタ - [ベース]タブ

テスト スイート エディタの［ベース］タブには、テスト スイートに関する基本情報が含まれます。



上部パネル

［ベース］タブの上部パネルには、テスト スイート全体に関する基本情報が含まれます。

また、スイートを設定するために、すべてのサブ タブでパラメータを入力する必要があります。

［基本］タブ

このタブには、以下のパラメータがあります。

スイート名

スイートの名前。

スタートアップ テスト/後処理テスト

スイートが開始される前に実行されるスタートアップ テスト、およびスイートの完了後に実行される後処理テストを指定できます。スタートアップ テストおよび後処理テストは、テストの統計には含まれません。 これらのテストのイベントは、レポートに表示されます。 スタートアップ テストが失敗した場合、スイートのテストは続行されません。 スタートアップまたは後処理テストとして MAR 情報ファイルを使用できます。MAR 情報ファイルのプライマリ アセットはテスト ケースである必要があります。

テストのリスト

スイートに現在含まれているテストのリストを示すダイアログ ボックス ウィンドウを表示します。 スイートを保存すると、このリストが更新されます。

［デフォルト］タブ

このタブには、以下のパラメータがあります。

ベース ディレクトリ

完全パスが含まれていない個々のテストのベース ディレクトリとされるディレクトリの名前。 テストがほかの場所で見つからない場合、DevTest はこのディレクトリを確認します。 ベース ディレクトリを指定しない場合、スイート ドキュメントが保存されるときにデフォルトが作成されます。

デフォルト ステージング ドキュメント

ステージング ドキュメントが個々のテストに対して指定されていない場合に使用されるステージング ドキュメント。

デフォルト監査ドキュメント

監査ドキュメントが個々のテストに対して指定されていない場合に使用される監査ドキュメント。

デフォルト コーディネータ サーバ

コーディネータ サーバが個々のテストに対して指定されていない場合に使用されるコーディネータ サーバ。プルダウン メニューには、使用可能なコーディネータ サーバが表示されます。 このパラメータは、DevTest サーバ を実行している場合にのみ必要です。

［実行モード］タブ

このタブには、以下のパラメータがあります。

シリアル

テストはスイート ドキュメントに出現する順に実行されます。 この設定は、機能テストおよびレグレッション テストに使用されます。

パラレル

テストは同時に実行されます。 この設定は、負荷テストおよびパフォーマンス テストに使用されます。 テストをすべて同時に実行できるように十分な仮想ユーザが存在する必要があります。

テストの重複実行を許可

同じテストを 2 回以上実行するかどうかを選択します。 含まれるスイート、ディレクトリ、またはディレクトリ ツリーに同じテストが表示される場合、テストを 1 つのスイートで重複実行できます。

親のスイートの実行モードは、子のスイートに自動的に適用されます。

下部パネル

［ベース］タブの下部パネルには、スイートおよび関連ドキュメントの詳細に追加できるドキュメントのタイプが表示されます。

このパネルでは、個々のテスト、既存のスイート、テストが含まれるディレクトリおよびディレクトリ ツリーを追加してスイートを作成します。

すべてのテストがスイート ドキュメントに追加および保存されると、スイートのエントリのリストが左側のパネルに表示されます。

スイートのエントリは、以下のタイプから選択することによって個別に追加されます。選択内容によって、［エントリの詳細］領域は異なります。

タイプ

以下のいずれかのスイート エントリ タイプを選択します。

- MAR 情報： MAR 情報ファイルの名前。
- テスト ケース： テスト ケースの名前。

- スイート： スイート ドキュメントの名前。 テストはすべてこのスイートから抽出され、個々のテストとして実行されます。
- モデル アーカイブ： MAR ファイルの名前。
- ディレクトリ：スイートに含められるテストが含まれるディレクトリの名前。 このディレクトリのすべてのテストには、同じステージング ドキュメント、監査ドキュメント、およびコーディネータ サーバが使用されます。 新しいテスト ケースがディレクトリに追加されると、そのテスト ケースはこのスイートに自動的に含められます。
- ディレクトリ ツリー： スイートに含められるテストが含まれるディレクトリの名前。 DevTest は、指定されたディレクトリとそのツリーのすべてのサブディレクトリを再帰的に検索します。 このディレクトリ ツリーのすべてのテストには、同じステージング ドキュメント、監査ドキュメント、およびコーディネータ サーバが使用されます。 新しいテスト ケースがディレクトリ ツリーに追加されると、そのテスト ケースはこのスイートに自動的に含められます。

ディレクトリおよびディレクトリ ツリーについては、名前でテスト ケースをフィルタできます。 結果をフィルタするには、［ファイル正規表現フィルタ］フィールドに正規表現を入力します。ツールバーの［追加］ をクリックし、新しいフィルタ情報を入力することにより、スイートに対して複数のフィルタを入力できます。 また、プロパティでフィルタを定義し、［ファイル正規表現フィルタ］フィールドでそのプロパティを使用できます。 スイートに含まれるテストを表示するには、［テストのリスト］ボタンを使用します。最新の結果を参照するために、忘れずにスイートを保存します。

［アクティブ］チェック ボックスでは、選択したエントリを実行するかどうかを制御できます。 たとえば、スイートのいずれかのテスト ケースの［アクティブ］チェック ボックスをオフにできます。

スイートへのドキュメント タイプの追加

選択されているドキュメント タイプによって、エントリの詳細フィールドは異なります。

例として、MAR 情報ドキュメントを追加します。 ディレクトリ ツリーに表示されているエントリの詳細が変更されます。

MAR 情報

MAR 情報の名前。 名前を入力するか、プルダウン メニューから選択するか、ファイルまたはディレクトリを参照します。選択した後、［開く］をクリックしてそれぞれのファイルを開きます。

ステージング ドキュメント

このエントリのステージング ドキュメントの名前。名前を入力するか、プルダウン メニューから選択するか、ドキュメントを参照します このエントリを空白のままにした場合は、デフォルトのステージング ドキュメントが使用されます。 選択した後、［開く］をクリックしてステージング ドキュメントを開きます。このスイートで使用するステージング ドキュメントを選択すると、ステージング ドキュメントの名前が［ステージング ドキュメント］フィールドの下に表示されます。 この名前は、ドキュメントの編集時にエディタ タブに表示される［ラン名］と同じです。

監査ドキュメント

このエントリの監査ドキュメントの名前。 名前を入力するか、プルダウン メニューから選択するか、ドキュメントを参照します このエントリを空白のままにした場合は、デフォルトの監査ドキュメントが使用されます。 選択した後、［開く］をクリックして監査ドキュメントを開きます。

コーディネータ サーバ

このエントリと一緒に使用するコーディネータ サーバの名前。プルダウン メニューに表示される使用可能なコーディネータ サーバのリストから選択します。 このパラメータは、DevTest サーバ を実行している場合にのみ必要です。

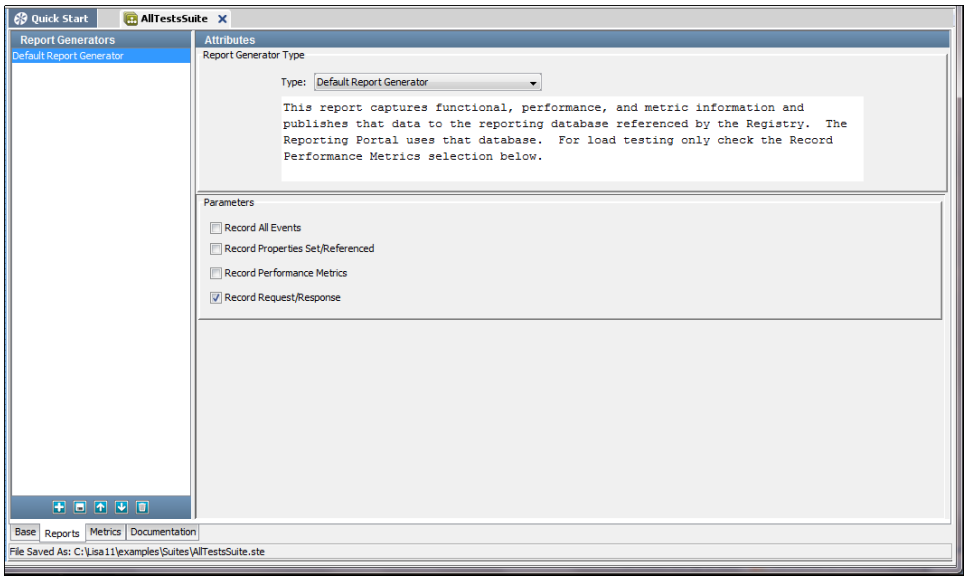
ツールバーの［追加］ をクリックして、左側のパネルに表示されるリストにこのエントリを追加します。 ［削除］ をクリックすると、エントリを削除できます。 ［上に移動］アイコン および［下に移動］アイコン を使用すると、エントリを並べ替えることができます。

テストのエントリをすべて入力したら、テスト スイート ドキュメントを保存します。

テスト スイート エディタ - [レポート]タブ

テスト スイート エディタの［レポート］タブでは、作成するテスト レポートおよびテスト スイート レベルでキャプチャするイベントを指定します。

各レポート ジェネレータで利用可能なレポート、レポートの表示、レポートの内容、および出力オプションの詳細については、「レポート (P. 435)」で説明しています。 レポート用にキャプチャするメトリックについては、「メトリック (P. 286)」でより詳細に説明しています。



［レポート］タブには以下のパネルが含まれます。

- レポート ジェネレータ
- 属性

レポート ジェネレータ

［レポート ジェネレータ］パネルには、選択されたレポートのリストが表示されます。

レポートを追加するには、リスト パネルの下部の［追加］ をクリックし、［レポート］パネルの中央の［タイプ］プルダウン メニューからレポート タイプを選択します。

レポートを削除するには、リストからレポートを選択して、［削除］  をクリックします。

属性

［属性］パネルには、利用可能なレポート タイプのリストが表示されます。

以下のレポート タイプが利用可能です。

- デフォルトのレポート ジェネレータ (P. 436)
- XML レポート ジェネレータ (P. 437)

パラメータ

すべてのイベントを記録

選択すると、すべてのイベントが記録されます。

設定/参照されたプロパティを記録

選択すると、設定または参照されたプロパティが記録されます。

パフォーマンス メトリックを記録

選択すると、パフォーマンス メトリックが記録されます。 この値を負荷テストに使用します。

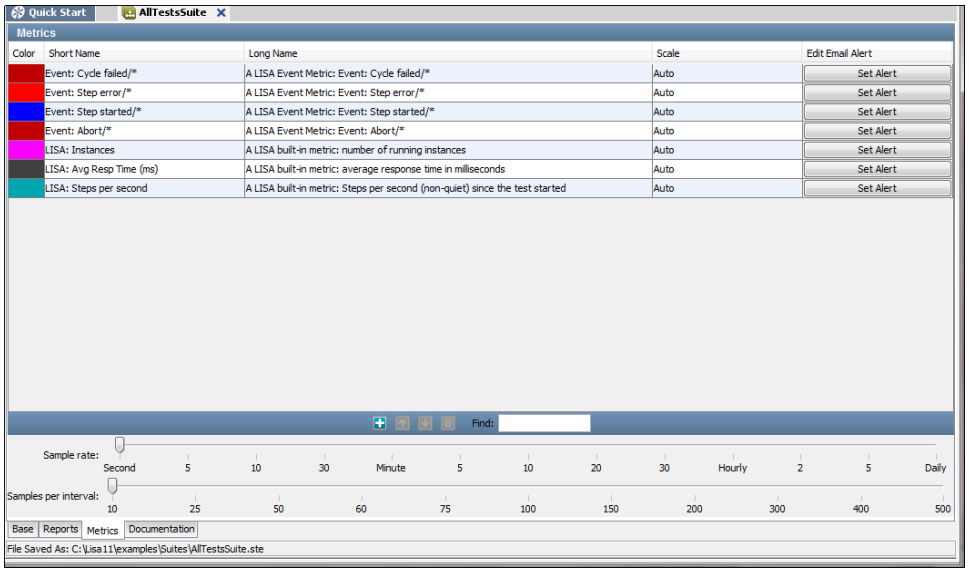
要求/応答を記録

選択すると、要求および応答が記録されます。

テスト スイート エディタ - [メトリック]タブ

テスト スイート エディタの［メトリック］タブでは、記録するテスト メトリックを選択します。 このタブでは、メトリック収集のサンプリング仕様も設定します。

また、選択したメトリックに対して電子メール アラートを設定できます。



［メトリック］タブは、2 つのセクションで構成されています。

上部パネルには、スイートに追加されるデフォルトのメトリックが表示されます。 ツールバーの［追加］ をクリックすると、メトリックを追加できます。

下部パネルでは、2 つのスライダ バーを使用して以下のサンプリング パラメータを設定できます。

- サンプル レート： この値は、サンプルを収集する（メトリックの値を記録する）頻度を指定します。 このパラメータは期間として指定します。 これは、サンプル レートの逆数です。
- サンプル数（間隔あたり）： この値は、メトリックのサマリ値を計算する間隔を設定するために使用されるサンプル数を指定します。

サンプル値を毎分取得（サンプル時間（間隔あたり） = 1 分）して、60 サンプルごとに平均（サンプル数（間隔あたり） = 60）すると、メトリック値が毎分生成され、サマリ値（平均）が毎時間計算されます。

たとえば、デフォルトでは、サンプル レートは 1 秒でサンプル数（間隔あたり）は 10 （間隔は 10 秒）です。

メトリック値は、レポートに含めるために XML ファイルまたはデータベース テーブルに格納されます（「レポート」を参照）。

メトリックの追加

以下のメトリック カテゴリを使用できます。

- DevTest 包括テスト メトリック
- DevTest テスト イベント メトリック
- JMX メトリック
- SNMP メトリック
- TIBCO Hawk メトリック
- Windows Perfmon メトリック
- SSH 経由の UNIX メトリック

メトリックを追加する方法

1. ツールバーの［追加］ をクリックします。

   ［メトリックの追加］ダイアログ ボックスが表示されます。
2. 目的のメトリックを選択し、［OK］をクリックします。
3. その他のカテゴリのメトリックを設定する場合は、この手順を繰り返します。

すべてのカテゴリのすべてのメトリックの説明、およびレポートに含めるようそれらを設定する方法の詳細については、「メトリックの生成 (P. 286)」を参照してください。

電子メール アラートの設定

次の手順に従ってください:

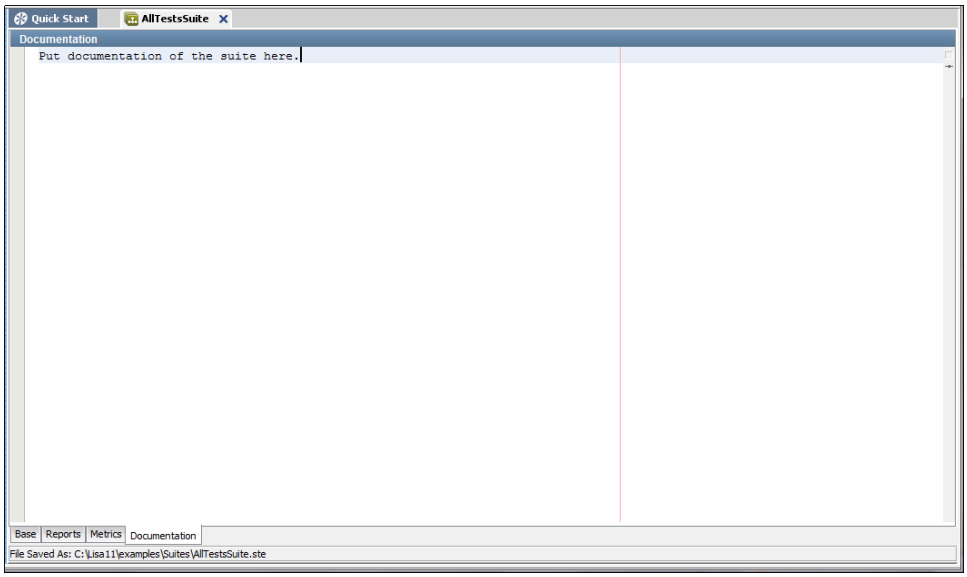
1. メトリックに対応する［アラートの設定］ボタンをクリックします。

   ［アラートの編集］ダイアログ ボックスが表示されます。

2. メトリックの許容範囲（上限値および下限値）および電子メールで送信される詳細を設定できます。 電子メール サーバの名前および電子メール アドレスのリストを入力する必要があります。
3. ウィンドウの下部の［追加］および［削除］アイコンを使用して、電子メール アドレスを追加および削除します。
4. 完了したら、［閉じる］をクリックします。

テスト スイート エディタ - [ドキュメント]タブ

テスト スイート エディタの［ドキュメント］タブでは、関連するドキュメントを入力します。



第 12 章: モデル アーカイブ（MAR）の操作

主要な展開アーティファクトは、モデル アーカイブ（MAR）という名前のファイル タイプです。

MAR は、DevTest ワークステーション から、または Make Mar コマンドライン ユーティリティを使用して設定できます。

このセクションには、以下のトピックが含まれています。

モデル アーカイブ(MAR)の概要

主要な展開アーティファクトは、モデル アーカイブ(MAR)と呼ばれるファイル タイプです。 ファイル拡張子は **.mar** です。

MAR ファイルには、以下のアイテムが含まれます。

- プライマリ アセット
- プライマリ アセットを実行するために必要なすべてのセカンダリ ファイル
- 情報ファイル
- 監査ファイル

モデル アーカイブ(MAR)

MAR ファイルはプロジェクト内のファイルから作成されます。 MAR ファイルが作成された後は、プロジェクトには依存しません。

MAR のコンテンツ

MAR ファイルには、以下のいずれかのプライマリ アセットが含まれます。

- テスト ケース
- スイート
- 仮想サービス
- テスト ケース モニタ
- スイート モニタ

また、MAR ファイルには、プライマリ アセットに必要なすべてのセカンダリ ファイルが含まれます。

たとえば、プライマリ アセットが仮想サービス モデルである場合、MAR ファイルにはサービス イメージが含まれます。

さらに、MAR ファイルには以下のファイルが含まれます。

- MAR 情報ファイル（P. 302）
- MAR 監査ファイル（P. 302）

MAR ファイルが最適化されるように指定することができます。 MAR ファイルが作成される際には、必要なプロジェクト ファイルのみが追加されます。 ただし、必要ではないプロジェクト ファイルがいくつか含まれる最適化された MAR ファイルを設定することもできます。

MAR ファイルを最適化しない場合は、すべてのプロジェクト ファイルが追加されます。

注: アーカイブは通常、メモリ内に保持されます。 そのため、最適化されたアーカイブを使用することを強くお勧めします。

MAR 情報ファイル

MAR 情報ファイルには、MAR を作成するために必要な情報が含まれます。ファイル拡張子は **.mari** です。

MAR 情報ファイルで指定される情報は、プライマリ アセットのタイプによって異なります。

プライマリ アセット	指定される情報
テスト ケース	テスト ケース、設定ファイル、およびステージング ドキュメント
スイート	スイートおよび設定ファイル
仮想サービス	仮想サービス モデル、設定ファイル、同時実行数、反応時間スケール、および自動再起動フラグ
テスト ケース モニタ	テスト ケースに指定されたすべての情報、サービス名、通知電子メール、優先度、および実行スケジュール
スイート モニタ	スイートに指定されたすべての情報、サービス名、通知電子メール、優先度、および実行スケジュール

MAR 情報ファイルはプロジェクトの Tests、Suites、または VirtualServices フォルダに配置されます。 ステージングや展開に関連するオプションを使用する場合は、MAR 情報ファイルは作成されますが保存されません。

examples プロジェクトには、MAR 情報ファイルが含まれており、確認できます。

MAR 監査ファイル

MAR 監査ファイルには、MAR の作成に関するメタデータが含まれています。 ファイル拡張子は **.maraudit** です。

MAR 監査ファイルには、以下のメタデータが含まれています。

- MAR が作成された日時。
- MAR が作成されたコンピュータの名前。
- MAR が作成されたプロジェクトのルート ディレクトリ。
- MAR を作成したユーザの名前。

明示的および暗黙的な MAR の作成

MAR を作成するには、以下の 2 つの方法があります。

- 明示的 (P. 303)
- 暗黙的 (P. 304)

アセットを展開するために Web ベースのダッシュボードまたはコマンド ライン ユーティリティを使用するには、MAR を明示的に作成する必要があります。 例外はテスト ランナー (P. 366) コマンドライン ユーティリティです。これは、MAR にパッケージ化されていないテスト ケースやスイートを実行するために使用できます。

DevTest ワークステーション からテスト ケースのステージング、スイートのステージング、仮想サービスの展開、テスト ケースのモニタとしての展開、またはスイートのモニタとしての展開を行う場合は、暗黙的な方法を使用します。

明示的な MAR の作成

明示的な方法では、MAR 情報ファイルを作成するステップを実行します。その後、MAR を作成するためにそれを使用します。

次の手順に従ってください:

1. 実行するプライマリ アセットを指定します。
2. MAR 情報ファイルを作成します。
3. MAR を作成します。

暗黙的な MAR の作成

暗黙的な方法では、MAR 情報ファイルと MAR の両方が自動的に作成されます。

次の手順に従ってください:

1. 実行するプライマリ アセットを指定します。
2. DevTest ワークステーション から、またはテスト ランナー (P. 366) を使用して、以下のいずれかのアクションを実行します。
   - テスト ケースのステージング
   - スイートのステージング
   - 仮想サービスの展開
   - モニタとしてのテスト ケースの展開
   - モニタとしてのスイートの展開

MAR 情報ファイルの作成

MAR 情報ファイルは、既存のテスト ケース、スイート、または仮想サービスから作成できます。

既存のテスト ケースから MAR 情報ファイルを作成する方法

1. DevTest ワークステーション のプロジェクト パネルで、テスト ケースを右クリックして［MAR 情報ファイルの作成］を選択します。

   ［MAR 情報ファイルの作成］ダイアログ ボックスが表示されます。
2. MAR 情報ファイルの名前を［名前］フィールドに入力します。
3. ［設定］フィールド リストから、このテスト ケースの設定ファイルを選択します。
4. ［ステージング ドキュメント］フィールド リストから、このテスト ケースのステージング ドキュメントを選択します。
5. ［コーディネータ サーバ］フィールド リストから、このテスト ケースのコーディネータ サーバ（該当する場合）を選択します。
6. ［OK］をクリックします。

   .mari ファイルはテスト ケースが配置されている同じフォルダに作成されます。

既存のスイートから MAR 情報ファイルを作成する方法

1. DevTest ワークステーション のプロジェクト パネルで、スイートを右クリックして［MAR 情報ファイルの作成］を選択します。

   ［MAR 情報ファイルの作成］ダイアログ ボックスが表示されます。
2. MAR 情報ファイルの名前を［名前］フィールドに入力します。
3. ［設定］フィールド リストから、このスイートの設定ファイルを選択します。
4. ［OK］をクリックします。

   .mari ファイルはスイートが配置されている同じフォルダに作成されます。

既存の仮想サービスから MAR 情報ファイルを作成する方法

1. DevTest ワークステーション のプロジェクト パネルで、仮想サービスを右クリックして［MAR 情報ファイルの作成］を選択します。

   ［MAR 情報ファイルの作成］ダイアログ ボックスが表示されます。

2. MAR 情報ファイルの名前を［名前］フィールドに入力します。

3. ［設定］フィールド リストから、この仮想サービスの設定ファイルを選択します。

4. この仮想サービスを仮想サービス グループ (P. 614)に含める場合は、［グループ タグ］フィールドにグループ名を入力します。グループ タグは英数字で始まる必要があり、英数字およびピリオド (.) 、ダッシュ（-）、アンダースコア（_）、ドル記号（$）の 4 つのその他の文字を含めることができます。

5. 負荷容量を指定するには、［同時実行数］フィールドに数値を入力します。

   デフォルト値は 1 ですが、より多くの容量を指定するために数値を増加させることもできます。 容量は、同時に何人の仮想ユーザ（インスタンス）が仮想サービス モデルで実行できるかを意味します。 ここでの容量は、このサービス モデルに対する要求を処理するスレッドの数を表します。

6. ［反応時間スケール］フィールドに反応時間のパーセンテージ（記録される反応時間に対応）を入力します。

   反応時間を 2 倍にするには、「200」と入力します。 反応時間を半分にするには、「50」と入力します。 デフォルト値は 100 です。

7. ［展開時にサービスを開始する］チェック ボックスをオンまたはオフにします。

   このチェック ボックスは、展開時にサービスがすぐに開始されるかどうかを指定します。 デフォルトではサービスが開始されます。

8. ［サービスが終了した場合、自動的に再起動する］チェック ボックスをオンまたはオフにします。

   このチェック ボックスをオンにすると、エミュレーション セッションがそのエンド ポイントに到達した後もサービスの実行が継続されます。

9. ［OK］をクリックします。

   .mari ファイルは仮想サービスが配置されている同じフォルダに作成されます。

モニタ MAR 情報ファイルの作成

モニタ MAR 情報ファイルは、既存のテスト ケースまたはスイートから作成できます。

モニタの詳細については、「継続的検証サービス（CVS） (P. 417)」を参照してください。

次の手順に従ってください:

1. DevTest ワークステーション のプロジェクト パネルで、テスト ケースまたはスイートを右クリックして［モニタ MAR 情報ファイルの作成］を選択します。

   ［モニタ MAR 情報ファイルの作成］ダイアログ ボックスが表示されます。

2. MAR 情報ファイルの名前を［名前］フィールドに入力します。
3. ［モニタ情報］タブ (P. 308)に関係するモニタ情報を指定します。
4. ［スケジュール］タブ (P. 309)に関係するタイム スケジュール情報を指定します。
5. （テスト ケース）［テスト ケース情報］ (P. 310)タブに関係するテスト ケース情報を指定します。
6. （スイート）［スイート情報］タブ (P. 310)に関係するスイート情報を指定します。
7. ［OK］をクリックします。

   .mari ファイルは、テスト ケースまたはスイートが配置されている同じフォルダに作成されます。

［モニタ情報］タブ

以下のパラメータを入力します。

サービス名

サービスの名前。この名前は CVS ダッシュボードに表示されます。

電子メール通知

通知する電子メール アドレス。

優先度

優先度を［高］、［中高］、［中］、［中低］、または［低］に設定します。

[スケジュール]タブ

以下のパラメータを入力します。

開始

モニタを開始する日時。時刻の値には、24 時間形式を使用します。

停止

モニタを停止する日時。時刻の値には、24 時間形式を使用します。停止日時は、開始日時より後にする必要があります。

実行

モニタを実行する頻度。 選択内容に応じて、追加のオプションが表示されます。

- 1 回： モニタが開始日時に 1 回実行されます。
- 間隔： 頻度を分単位で入力します。モニタは NN 分ごとに実行されます。 ただし、前の実行が完了していない場合、CVS は実行をスキップし、次にスケジュールされた刻に再試行します。CVS は開始されたモニタを中断しません。
- 日単位： モニタは、開始時刻に指定された時間に 1 日 1 回実行されます。
- 週単位： 曜日を 1 つ以上選択します。 モニタは、選択された各曜日の開始時刻に指定された時間に 1 回実行されます。
- 月単位： 月の日付を 1 つ以上入力します。 複数の日付を入力する場合は、それぞれをスペースで区切ります。たとえば、「1 15 30」と入力します。 特定の月に存在しない日付（31 日など）を指定すると、その月のその日は無視されます。 モニタは、各日付の開始時刻に指定された時間に 1 回実行されます。
- CRON： 標準の cron の指定を入力します。 cron は、時間間隔を指定するための標準の UNIX 構文です。 実行スケジュールを指定する場合には、これが最も柔軟性の高い方法になります。

[テスト ケース情報]タブ

以下のパラメータを入力します。

設定

設定の名前。 このフィールドを空白にすると、テスト ケース内の有効な設定が使用されます。

ステージング ドキュメント

ステージング ドキュメントの名前。

コーディネータ サーバ

コーディネータ サーバの名前。

[スイート情報]タブ

以下のパラメータを入力します。

設定

設定の名前。

MAR 情報ファイルの編集

MAR 情報エディタでは、MAR 情報ファイルの作成時に指定された情報を変更できます。

このエディタには、最適化用チェック ボックスがあります。 デフォルトでは、このチェックボックスはオンになっています。 MAR ファイルが作成される際には、プライマリ アセットで必要なプロジェクト ファイルだけが追加されます。 ただし、このエディタでは、必要ではないプロジェクト ファイルがいくつか含まれる最適化された MAR ファイルを指定できます。

最適化用チェック ボックスをオフにすると、すべてのプロジェクト ファイルが MAR ファイルに追加されます。

MAR 情報ファイルを編集する方法

1. DevTest ワークステーション のプロジェクト パネルで、MAR 情報ファイルをダブルクリックします。

   エディタが開きます。

   上部の領域で編集できる情報は、MAR 情報ファイルのタイプによって異なります。

2. ［メモ］領域に、MAR 情報ファイルに関する注意事項を入力します。
3. 最適化用チェック ボックスをオンまたはオフにします。

   MAR にすべてのプロジェクト ファイルを含める場合は、最適化用チェック ボックスをオフにします。［含めるファイル］および［除外するファイル］領域は無効になります。

   注: アーカイブは通常、メモリ内に保持されます。 そのため、最適化されたアーカイブを使用することを強くお勧めします。 このチェックボックスはデフォルトでオンになっています。

4. MAR に必要でないプロジェクト ファイルを含める場合は、そのファイルを［含めるファイル］領域に移動させます。

   ［含めるファイル］領域には、必要なプロジェクト ファイルのリストが表示されます。［除外するファイル］領域には、必要でないプロジェクト ファイルのリストが表示されます。

5. メイン メニューから［ファイル］-［保存］を選択します。

MAR の作成

MAR 情報ファイルを作成および編集した後、このファイルを使用してモデル アーカイブ（MAR） (P. 300)を作成することができます。

MAR はプロジェクト ディレクトリの内部または外部に保存できます。

次の手順に従ってください:

1. DevTest ワークステーション のプロジェクト パネルで、MAR 情報ファイルを右クリックして［モデル アーカイブの構築］を選択します。

   ［モデル アーカイブの構築］ダイアログ ボックスが表示されます。

2. MAR を保存するフォルダに移動します。
3. ［保存］をクリックします。

CVS に展開

モニタ MAR 情報ファイル (P. 307)は CVS に展開できます。

次の手順に従ってください:

1. DevTest ワークステーション のプロジェクト パネルで、.mari ファイルを右クリックして［CVS に展開］を選択します。

   確認メッセージが表示されます。

2. 新しく追加されたモニタを確認するには、CVS ダッシュボードに移動します。

Make Mar ユーティリティ

Make Mar コマンドライン ユーティリティを使用すると、MAR 情報ファイル（スタンドアロンまたはアーカイブ内）のコンテンツを表示したり、MAR 情報ファイルからモデル アーカイブ ファイルを作成したりすることができます。

このユーティリティは、**LISA_HOME/bin** ディレクトリにあります。

オプション

各オプションには、短いバージョンと長いバージョンがあります。 短いバージョンは、1 つのダッシュから始まります。 長いバージョンは、2 つのダッシュから始まります。

-h、--help

ヘルプ テキストを表示します。

-s ファイル名、--show=ファイル名

MAR 情報ファイルのコンテンツを表示します。

ファイル名が MAR 情報ファイルを指している場合は、そのファイルが表示されます。ファイル名がアーカイブを指している場合は、監査と情報の両方のエントリが表示されます。

-c、--create

MAR 情報ファイルから 1 つ以上のモデル アーカイブを作成します。

アーカイブを作成するための MAR 情報ファイルの名前を指定するには、--marinfo 引数を使用します。

作成するアーカイブの名前を指定するには、--archive 引数を使用します。

--source-dir 引数を使用すると、--marinfo 引数はディレクトリ ツリー全体で検索するための名前パターンとして使用されます。 この場合、作成したアーカイブを保存する場所を指定するために --target-dir 引数を使用する必要があります。このコマンドには、作成するアーカイブに自動的に名前を付ける機能もあります。

-m MAR 情報ファイル名、--marinfo=MAR 情報ファイル名

読み取る MAR 情報ファイルの名前（--source-dir 引数を使用しない場合）、または検索する MAR 情報ファイルの名前パターン（--source-dir 引数を使用する場合）を指定するパラメータ。 後者の場合、指定しなかった場合にはデフォルトで .mari になります。

-s ディレクトリ、--source-dir=ディレクトリ

アーカイブを作成する元となる MAR 情報ファイルをツールが検索するディレクトリを指定します。 --marinfo 引数で指定したパターンに一致するファイルが、ディレクトリ ツリー全体で検索されます。

-t 出力ディレクトリ、--target-dir=出力ディレクトリ

アーカイブが書き込まれるディレクトリを指定します。

この引数を使用する場合、作成されるアーカイブには、ファイルが上書きされないように MAR 情報ファイルの名前と数値のサフィックスに基づいて名前が自動的に付けられます。

-a アーカイブ ファイル、--archive=アーカイブ ファイル

作成するアーカイブ ファイルの名前を指定します。

この引数を使用する場合は、--marinfo 引数で既存の MAR 情報ファイルを 1 つ指定する必要があります。 --source-dir 引数および --target-dir 引数は使用できません。

--version

バージョン番号を出力します。

例

以下の例では、MAR 情報ファイルのコンテンツを表示します。

```
MakeMar --show=C:¥Lisa¥examples¥MARInfos¥AllTestsSuite.mari
```

以下の例では、**rawSoap.mar** という名前のモデル アーカイブを作成します。

```
MakeMar --create --marinfo=C:¥Lisa¥examples¥MARInfos¥rawSoap.mari
--archive=rawSoap.mar
```

以下の例では、**examples¥MARInfos** ディレクトリにある各 MAR 情報ファイル用のモデル アーカイブを作成します。 このコマンドを実行する前に、保存先ディレクトリ **examples¥MARs** が存在している必要があります。

```
MakeMar --create --source-dir=examples¥MARInfos --target-dir=examples¥MARs
```

第 13 章: テスト ケースおよびスイートの実行

テスト ケースを実行するための多数のオプションがあります。

- DevTest ワークステーション では、対話型テスト ラン（ITR） (P. 317) ユーティリティを使用して、テスト ケースのステップ全体を検証することができます。
- DevTest ワークステーション では、テスト ケースの特定のステップまで (P. 231) 最小限の設定でテスト ケースを迅速に実行できます。
- DevTest ワークステーション では、最小限の設定でテスト ケースを迅速に実行 (P. 330) できます。
- DevTest ワークステーション では、テスト ケースのステージング (P. 340) を行うことができます。 このオプションを使用するには、設定、ステージング ドキュメント、およびコーディネータ サーバを指定する必要があります。 同じテスト ケースと別のステージング ドキュメントを使用して、別のテストを実行することもよくあります。 機能テスト、レグレッション テスト、負荷テストごとに別々のテスト ケースを作成する必要はありません。その代わりに、同じテスト ケースを使用する別々のステージング ドキュメントを用意します。
- テスト ランナー (P. 366) は、テストをバッチ プロセスとして実行できるコマンドライン ユーティリティです。
- LISA Invoke (P. 372) は、テスト ケースおよびスイートを URL で実行できる、REST のような Web アプリケーションです。
- サーバ コンソールでは、ラボにテスト ケース用のモデル アーカイブ（MAR） を展開 (P. 413) できます。
- 継続的検証サービス（CVS） (P. 417) を使用すると、テストを一定の時間間隔で実行されるようスケジュールできます。
- Ant および JUnit を使用すると、テストを自動的なビルド プロセスの一部として実行できます。

このセクションには、以下のトピックが含まれています。

REST API (P. 378)
HP ALM - Quality Center プラグインの使用 (P. 380)
HTTP および SSL デバッグ ビューア (P. 389)

対話型テスト ラン（ITR）ユーティリティの使用

対話型テスト ラン（ITR）ユーティリティでは、テスト ケースのステップ全体を検証することができます。 このユーティリティを使用すると、テスト ケースをステージングする前に、テスト ケースが正しく実行されることを検証できます。ITR は、テスト ケース ドキュメントをロードするのではなく、メモリにあるテストを実行します。 テスト ケースにリアルタイムで変更を加えて、テストのプロパティやワークフローを変更することができます。

たとえば、自然なワークフロー シーケンスから特定のテスト ステップを実行することができます。 テストの実行中に行ったリアルタイムの変更は、テスト ケースへ組み込まれます。 テスト ケースの実行は続行され、再起動は必要ありません。 例外はアクティブな設定を変更するようなグローバルな変更です。 あるステップを実行し、予期しない結果が得られたときには（不正なパラメータが原因の場合など）、ITR を開いたままでステップを編集できます。 その後 ITR に戻り、再度同じステップを実行できます。

ITR では現在の ITR の状態を保存することができます。また、以前に保存した ITR の状態をロードすることもできます。 この機能により、状況に応じてチームの別のメンバにエラーを伝えることができます。 テスト ケースおよび保存された ITR の状態を送信して、正確なアクションや観察した結果を提供することができます。

このセクションには、以下のトピックが含まれます。

- ITR の実行の開始 (P. 318)
- ITR の実行結果の確認 (P. 322)
- グラフィカル テキスト差分ユーティリティ (P. 326)

ITR の実行の開始

対話型テスト ラン（ITR）ユーティリティは、DevTest ワークステーション内の開いているテスト ケースから実行します。

次の手順に従ってください:

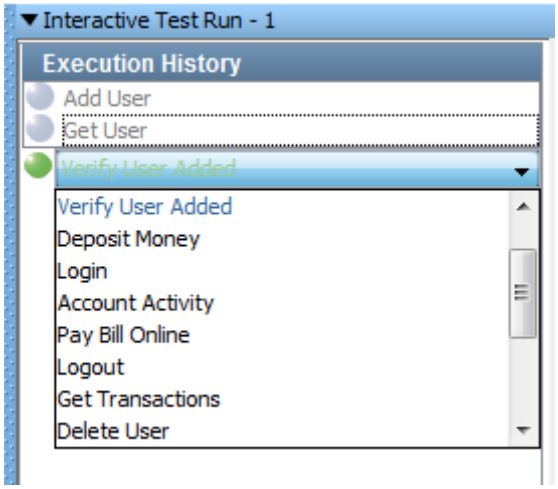
1. 以下のいずれかを実行します。
   - メイン メニューから［アクション］-［対話型テスト ラン（ITR）の開始］を選択します。
   - ツールバーの［ITR］アイコン  をクリックします。 このアイコンには、新しい ITR の実行を開始するか、以前の ITR の実行を開くかのオプションがあります。

   ［対話型テスト ラン（ITR）］ウィンドウが表示されます。
2. （オプション）設定を変更するには、［設定］タブ（P. 321）を使用します。
3. テスト ケースの全体または一部を実行するには、［実行］タブ（P. 319）を使用します。

ITR の[実行]タブ

ITR ウィンドウの［実行］タブには、すでに実行されたステップ（灰色）と次に実行されるステップ（緑）が表示されます。

以下の図では、最初の 2 つのステップ (ユーザの追加および Get User（ユーザの取得）) がすでに実行されています。 3 つ目のステップ（Verify User Added （追加されたユーザの検証））が次に実行されるステップです。



各ステップにはプルダウン メニューがあり、これにはテスト ケース内のすべてのステップが表示されます。 このメニューを使用して、次に実行されるステップを変更できます。 ステップを選択すると、現在の次のステップが選択したステップに置き換えられます。 ステップを変更した後、ワークフローは新しいステップから続行されます。

ITR の管理は、［実行］タブの下部にあるツールバーを使用して行います。

テスト ケースの次のステップを実行します。 ステップが実行された後、右側のパネルで結果を確認 (P. 322)できます。 再度アイコンをクリックしてリストされている次のステップを実行するか、先に説明したように実行する別のステップを選択します。

テスト ケースのステップをすべて実行します。 テストが実行されている間、アイコンは［停止］アイコンに変わります。［停止］アイコンをクリックすると、テストを停止させることができます。 最後のステップが実行されると、ダイアログ ボックスにテストが完了したことが表示されます。

テストを再度開始します。 この機能は、テスト ケースを変更して最初から実行する場合に便利です。

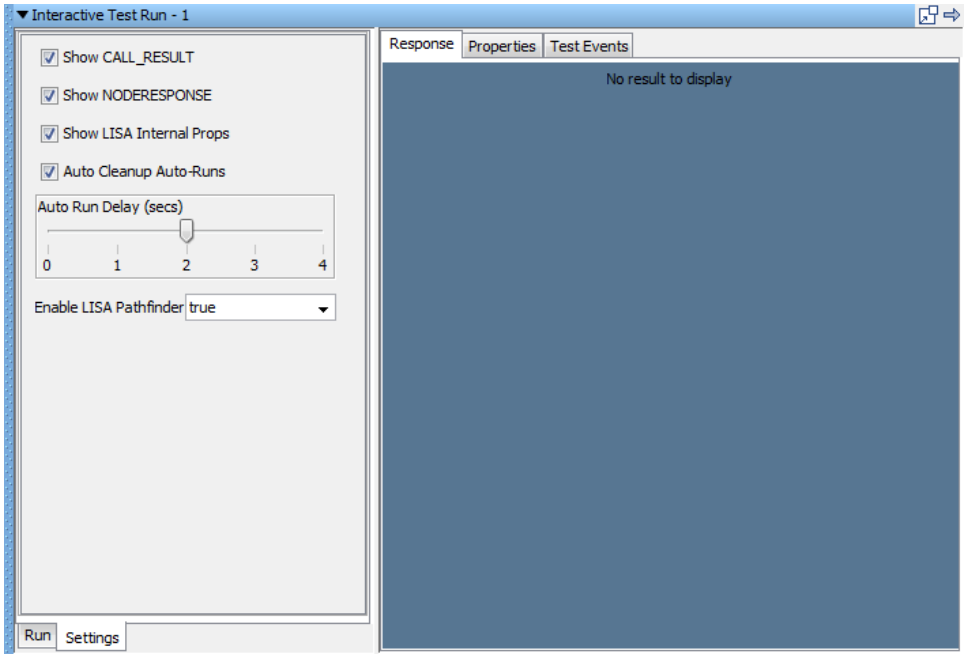
現在の ITR の状態を保存します。ダイアログ ボックスで保存ファイルの名前を入力します。 サフィックス .itr が名前に付加されます。 この後の保存では、このファイルに上書きします。

保存されている ITR をロードします。 ロードする .itr ファイルを参照します。 保存されている ITR の状態には、タブに表示される情報と、実行されたすべてのテストからキャプチャされる情報が含まれます。保存されている状態を使用するには、まず元のテストが実行されたときに使用したテスト ケースを開きます。

プロパティを追加または確認するには、プロパティ ウォッチ ウィンドウを開きます。

ITR の[設定]タブ

［設定］タブでは、ITR のさまざまな面を制御できます。



結果のタブからイベントおよびプロパティをフィルタできます。 いくつかの冗長なイベントを表示しないようにすることができます。 同様に、プロパティ リストに DevTest の内部プロパティが多数表示されないようにすることもできます。

コール結果を表示

［テスト イベント］リストに EVENT_CALL_RESULT イベントを表示します。

ノードレスポンスを表示

［テスト イベント］リストに EVENT_NODERESPONSE イベントを表示します。

内部プロパティを表示

［初期状態］タブおよび［実行後の状態］タブのプロパティ リストに内部イベントをすべて表示します。

自動実行の自動クリーンアップ

自動実行を自動的にクリーンアップします。

自動実行遅延（秒）

［自動的にテストを実行］モードでは、各ステップの実行の間で ITR が一時停止します。 この設定により、一時停止の間隔を変更できます。ITR では反応時間が考慮されないため、この設定によって各ステップの実行の間に一定の遅延を加えることができます。

CA CAI の有効化

CAI が有効かどうかを制御します。

デフォルト： false

ITR の実行結果の確認

テスト ステップの実行結果は、実行中、2 つのステップ間、または実行後に確認できます。

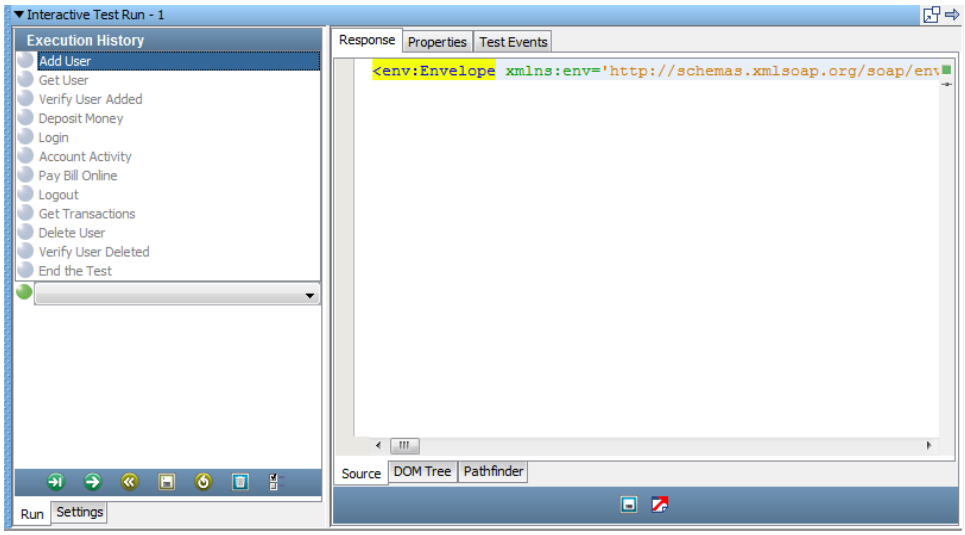
［実行履歴］リストでステップを選択します。 右側のパネルで情報を確認します。以下のタブがあります。

- ［応答］タブ (P. 323)
- ［プロパティ］タブ (P. 324)
- ［テスト イベント］タブ (P. 325)

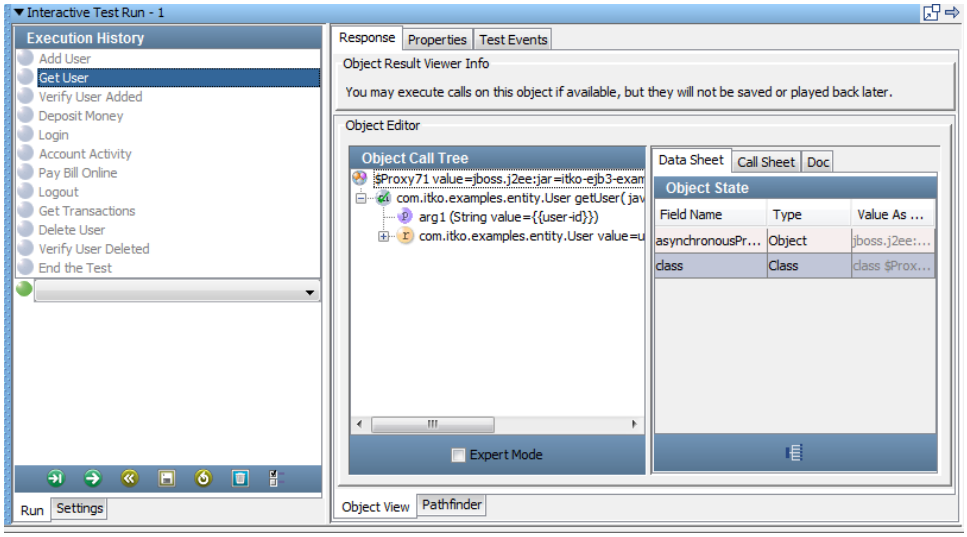
ITR の結果の[応答]タブ

[応答]タブには、ステップを実行した後のステップ応答が表示されます。

その表示には、応答のタイプに適したエディタが使用されます。 たとえば、multi-tier-combo テスト ケースのユーザの追加ステップでは、HTML/XML 応答が呼び出されます。



multi-tier-combo テスト ケースの Get User（ユーザの取得）ステップでは、複合オブジェクト エディタ (P. 187)に表示されるオブジェクトを返す EJB が呼び出されます。



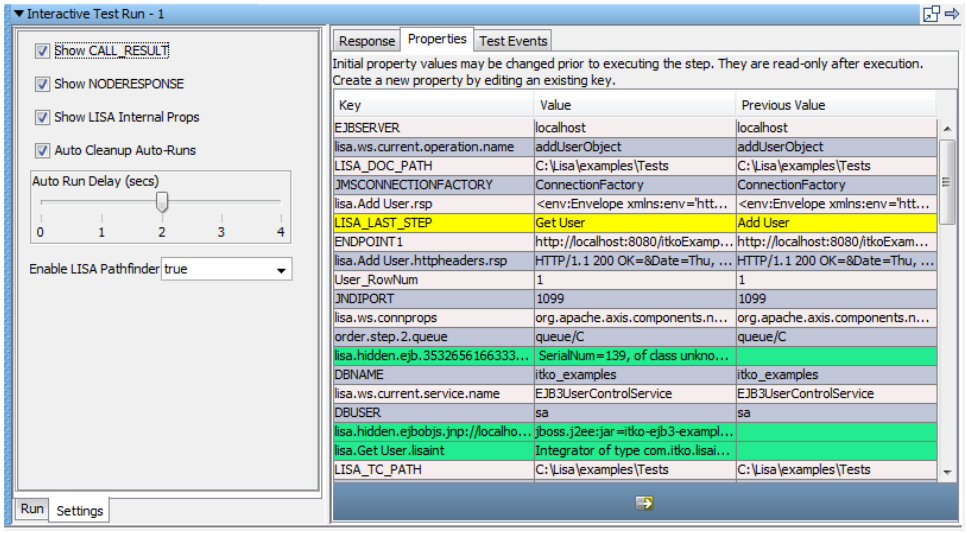
一部のエディタには、ITR の状態を保存するアイコン と、外部のブラウザを起動するアイコン があります。

注: 応答が XML で 5 MB より大きい場合は、DOM ビューのないプレーン テキストで表示されます。 この制限は **gui.viewxml.maxResponseSize** プロパティで調節できます。

ITR の結果の[プロパティ]タブ

［プロパティ］タブには、実行後（値）と、実行直前（前の値）のステップの状態が表示されます。

DevTest の内部プロパティを表示または非表示にするには、［設定］タブの［LISA 内部プロパティを表示］チェック ボックスをオンまたはオフにします。

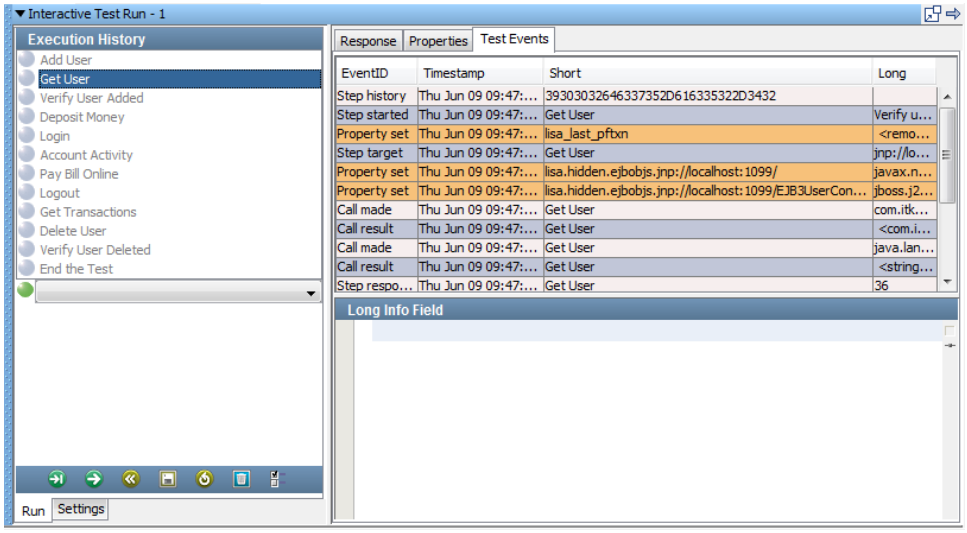


ステップで設定されたプロパティは緑で強調表示されます。 ステップで変更されたプロパティは黄色で強調表示されます。

ITR の結果の[テスト イベント]タブ

［テスト イベント］タブには、ステップが実行されたときに起動されたイベントが、起動された順に表示されます。

イベントを表示または非表示にするには、［設定］タブの［コール結果を表示］および［ノードレスポンスを表示］チェック ボックスをオンまたはオフにします。



［テスト イベント］タブには、以下の列があります。

イベント ID

イベントの名前。

タイムスタンプ

イベントの日時。

概略

イベントの簡単な説明。

詳細

イベントの詳細な説明。 詳細な説明は切り詰めて表示することができます。テキスト全体を表示するには、セルをクリックします。［詳細フィールド］パネルに全体の内容が表示されます。

［詳細フィールド］には、イベントの詳細または全体的な説明を入力できます。

イベントは、起動または評価されるすべてのアサーションに対して生成されます。

警告が発生した場合は、行は黄色で表示されます。

アサーションが起動された場合は、行は紫で表示されます。

プロパティが設定された場合は、行はオレンジで表示されます。

エラーが発生した場合は、テストは失敗となるか、または中止され、行は赤で表示されます。

注: ステップでアサーションが含まれるサブプロセスがコールされると、［テスト イベント］タブにはアサーションに対応するイベントが表示されます。 たとえば、［テスト イベント］タブには、サブプロセスで発生した［アサーションの起動］イベントを表示できます。

グラフィカル テキスト差分ユーティリティ

グラフィカル XML 差分エンジンおよびビジュアライザを使用すると、XML ファイルを比較し、その差分をグラフィカルに表示できます。

グラフィカル テキスト差分ユーティリティの起動

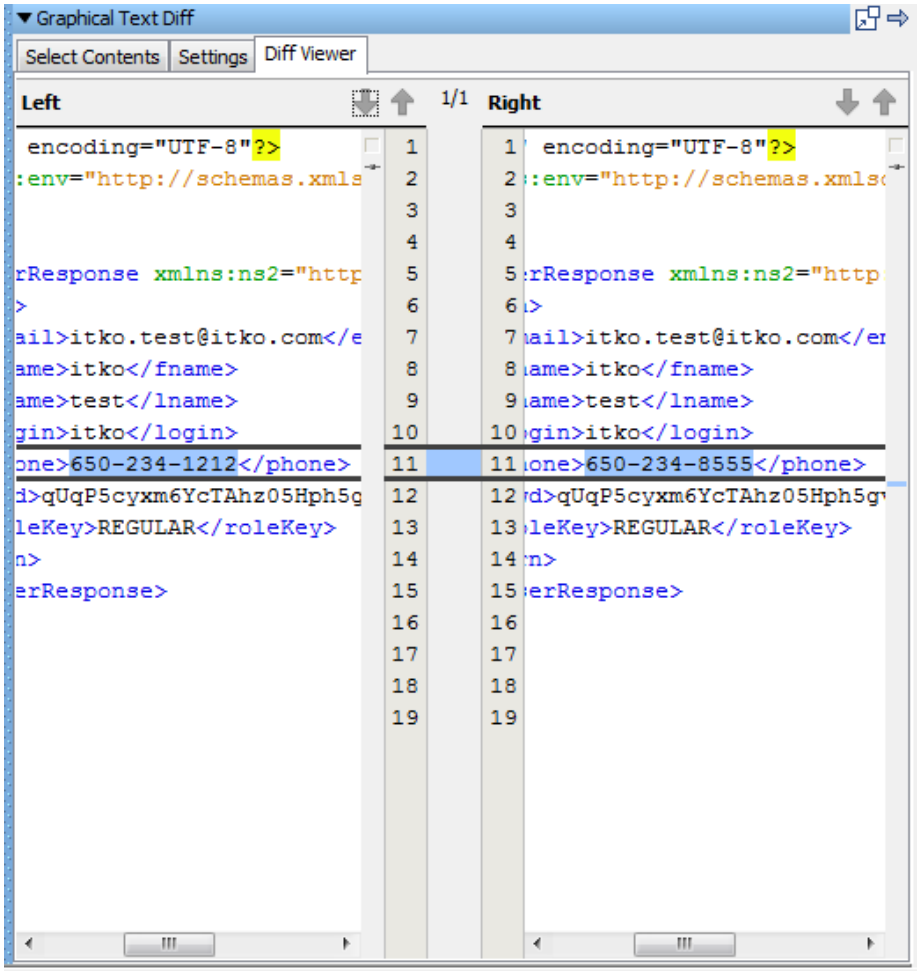
次の手順に従ってください:

1. メイン メニューから［アクション］-［グラフィカル テキスト差分］を選択します。

   ［グラフィカル テキスト差分］ウィンドウが表示されます。このパネルでは、2 つの XML ファイルの内容を比較できます。

2. ［比較］  をクリックして、比較を開始します。

   以下のウィンドウが表示され、実際の比較結果が表示されます。［差分ビューア］タブに比較が表示されます。



対話型テスト ラン(ITR)ユーティリティからグラフィカル テキスト差分ユーティリティを起動する方法

1. ITR で［テスト イベント］タブを選択します。
2. テーブルの行を右クリックし、［比較する左テキストの選択］を選択します。
3. 比較するテーブルの行を右クリックし、［～と比較］を選択します。

   比較のためにグラフィカル テキスト差分ユーティリティが表示されます。

グラフィカル テキスト差分ユーティリティでは、ファイルを選択してコンテンツをロードすることもできます。

次の手順に従ってください:

1. ［グラフィカル テキスト差分］ウィンドウの［コンテンツの選択］タブをクリックします。
2. ファイルをロードするには、［ファイルからロード］をクリックします。

   差分の結果がグローバル トレイ パネル コンポーネントに表示されます。

注:［設定］タブで比較オプションを設定することもできます。比較オプションの詳細については、「*CA Application Test の使用*」の「グラフィカル XML 比較」を参照してください。

グラフィカル テキスト差分ユーティリティのプロパティ

グラフィカル XML 差分ユーティリティでは、以下のプロパティを使用します。これらのプロパティは **local.properties** で上書き、または実行時に上書きすることができます。

lisa.graphical.xml.diff.engine.max.differences

このプロパティでは、検出される差分の最大数を設定します。この数に達すると、XML 差分アルゴリズムは停止します。 「-1」に設定すると、すべての差分が計算されます。差分の数が多いと、この設定によって XML 差分アルゴリズムが早く終了してしまうことがあります。

デフォルト： 100

lisa.graphical.xml.diff.report.max.linewidth

このプロパティでは、XML 差分結果レポートのテキスト行の最大幅を設定します。入力された XML に長いテキスト行がある場合は、このプロパティによって XML 差分結果レポートで消費するメモリを少なくすることができます。

デフォルト： 80

lisa.graphical.xml.diff.report.max.numberoflines

このプロパティでは、XML 差分結果レポートの差分のコンテキスト行の最大数を設定します。比較している XML に大きな差分エレメントがある場合は、このプロパティによって XML 差分結果レポートで消費するメモリを少なくすることができます。

デフォルト： 5

注: ほとんどの場合は、これらの値を変更するべきではありません。 これらのプロパティのデフォルト値を変更すると、グラフィカル XML 差分エンジンで CPU 使用量が増えたり、メモリが使い果たされたり、あるいはその両方が生じるおそれがあります。

クイック テストのステージング

DevTest ワークステーション では、最小限の設定でテスト ケースを迅速に実行できます。 クイック テストは、テスト ケース ドキュメントをロードするのではなく、メモリにあるテストを実行します。 クイック テストでは、シンプルな事前作成済みのステージング仕様と少数のオプションを使用します。 このテストのインスタンスは最小限であるため、ステージングや実行が容易です。ただし、正式なテスト ケース (P. 340)のようにステージング ドキュメント (P. 251)を使用してステージングするテストに比べると、多くの機能は備わっていません。 クイック テストでは、イベントやメトリックを選択およびモニタし、標準的なパフォーマンス レポートを確認できます。

次の手順に従ってください:

1. DevTest ワークステーション でテスト ケースを開きます。
2. メイン メニューから［アクション］-［クイック テストのステージング］を選択します。

   ［クイック テストのステージング］ダイアログ ボックスが表示されます。

   レジストリに接続しているコーディネータやシミュレータがあれば、接続しているコーディネータ サーバやシミュレータ サーバが表示されます。
3. 以下のパラメータを指定します。

   ラン名

   クイック テストの名前。

   インスタンス数

   使用される同時ユーザ（インスタンス）の数。 指定できるユーザの最大数は、ライセンスによって異なります。

   インスタンスのステージング先

   テストがステージングされるコーディネータ サーバの名前。

   テストが終了したら再起動する

   手動で停止するまでテストを継続して実行する場合は、このチェック ボックスをオンにします。
4. ［OK］をクリックします。

   テスト モニタ (P. 331) ウィンドウが表示されます。

テスト モニタ ウィンドウ - クイック テスト

クイック テストをステージング (P. 330)すると、DevTest ワークステーション にテスト モニタ ウィンドウが表示されます。クイック テスト用のテスト モニタ ウィンドウは、テスト ケース用のテスト モニタ ウィンドウと同様です。

最初にテストをステージングします。まだテストの実行は開始されていません。

テストの実行時にモニタするメトリックおよびイベントを選択できます。

テスト モニタは、以下の 2 つのタブで構成されています。

- ［パフォーマンス統計 (P. 332)］タブには、収集されたメトリックが時間の関数として表示されます。 テストを開始した後は、この表示を変更できません。
- ［イベント］タブ (P. 335)には、テストの実行時に記録されたイベントが表示されます。

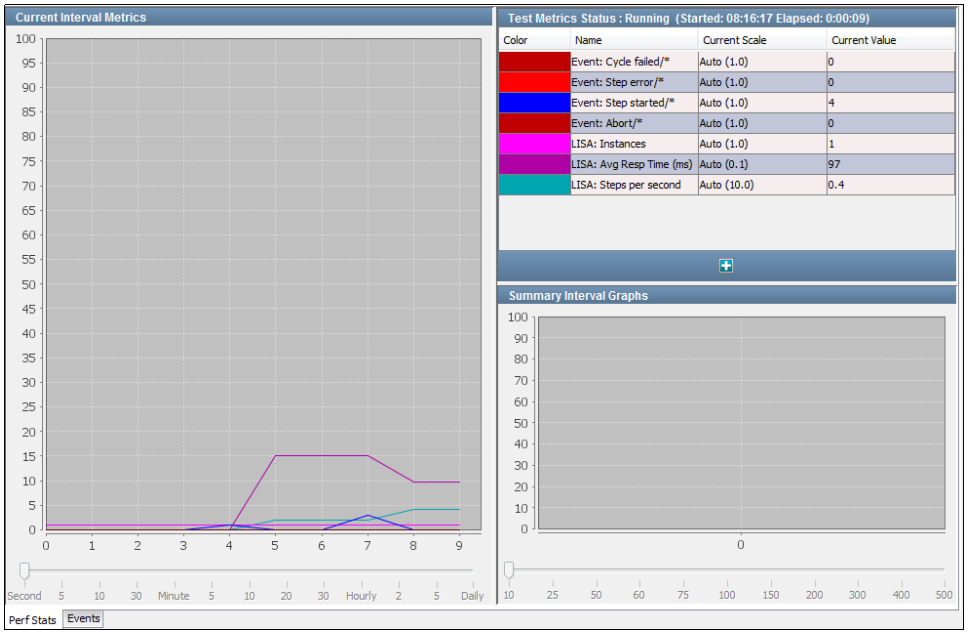
テスト ケースを実行し、テストが失敗した場合は、［失敗 (P. 337)］という 3 つ目のタブが表示されます。

ベースライン テスト ケースを実行する場合は、［統合ベースライン］タブも表示されます。詳細については、「*CA Continuous Application Insight の使用*」を参照してください。

local.properties ファイルに **lisa.coord.failure.list.size** プロパティを追加することで、失敗イベントのサイズを制限できます。

[パフォーマンス統計]タブ - クイック テスト

[パフォーマンス統計] タブでは、テストの実行時にモニタするメトリックおよびイベントを追加できます。



[パフォーマンス統計] タブは以下のパネルで構成されています。

- テスト メトリック ステータス
- 現在の間隔メトリック
- サマリ間隔グラフ

テスト メトリック ステータス

[テスト メトリック ステータス] パネルには、現在モニタされているメトリックのリストが表示されます。

いくつかのメトリックはデフォルトで表示されています。 メトリックは識別しやすくするために色分けされています。

[追加] をクリックすると、メトリックを追加できます。 プルダウン メニューには、追加できるメトリックのカテゴリがすべて表示されます。

[テスト メトリック ステータス] パネルには、以下の列があります。

色

グラフの色分けに使用される色。

名前

メトリックの名前。 短い名前を使用してメトリックをフィルタしている場合、［名前］フィールドには、スラッシュの後に短い名前が表示されます。 アスタリスクは、メトリックのイベント名のみを使用していることを意味します。

現在のスケール

グラフで使用される縦方向のスケール。 任意のメトリックの［現在の間隔メトリック］グラフのグラフ スケールを調節できます。［現在のスケール］セルをダブルクリックしてドロップダウン リストから新しい値を選択することにより、スケールを調節します。［現在の間隔メトリック］グラフで、メトリックのスケールが変更されます。

注: ［現在のスケール］のデフォルト設定は、自動スケールの 100 です。() 内の数値は、自動スケールが同じ 0 ～100 のグラフにデータをすべて対応させるために決定したスケールの値を示します。この値とスケールを乗算したものがグラフの Y 座標になります。スケールを変更しても現在の値は変わりません。 ただし、グラフで使用される Y 座標を決定する計算では、異なる Y 座標が算出される場合があります。 座標が 100 を超えている場合は、より大きな値に対応できるように Y 軸の制限も増加させる必要があります。

現在の値

メトリックの現在の値。

メトリックの詳細については、「*CA Application Test の使用*」の「メトリックの生成 (P. 286)」を参照してください。

現在の間隔メトリック

［テスト メトリック ステータス］パネルの各メトリックに対して、［現在の間隔メトリック］パネルに実行時の指定された時間間隔での値が表示されます。

時間間隔を指定するには、パネルの下部にあるスライダを使用します。実行が開始された後は、値は調節できません。

サマリ間隔グラフ

［サマリ間隔グラフ］パネルには、［テスト メトリック ステータス］パネルの各メトリックの値を指定された時間間隔で平均したものが表示されます。

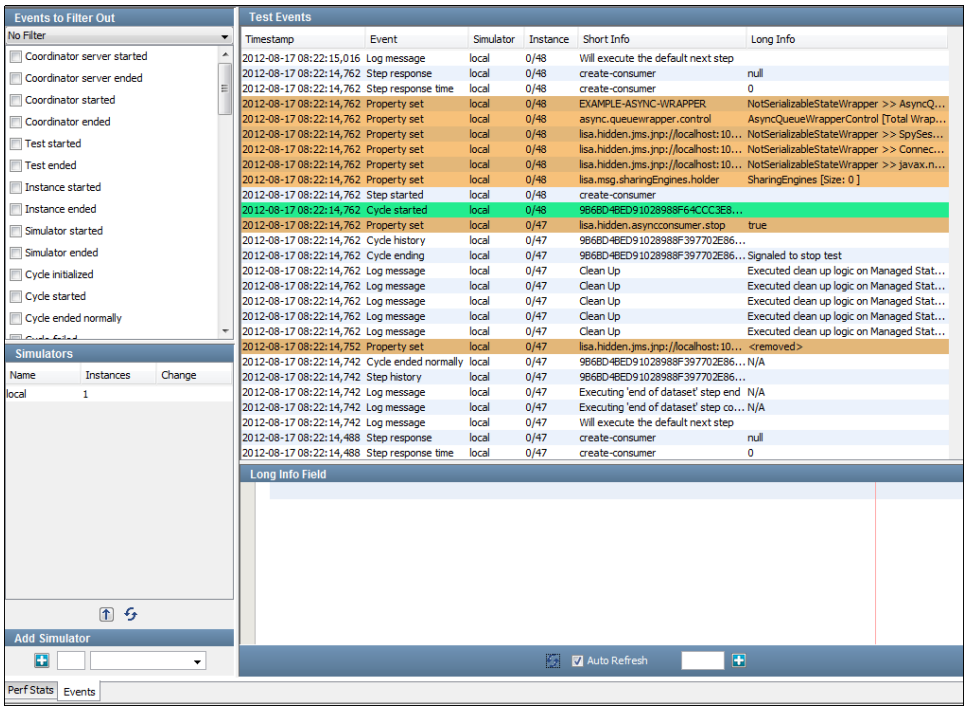
時間間隔あたりのサンプル数を指定するには、パネルの下部にあるスライダを使用します。 実行が開始された後は、値は調節できません。

[イベント]タブ - クイック テスト

［イベント］タブには、実行時に生成されるイベントが表示されます。

［イベント］タブは以下のパネルで構成されています。

- フィルタで除外するイベント
- シミュレータ
- テスト イベント



フィルタで除外するイベント

［フィルタで除外するイベント］パネルでは、実行時に表示されるイベントを制限することができます。 このパネルには、イベントのリストが表示されます。 イベントに対するチェック ボックスをオンにすると、そのイベントは実行時に表示されなくなります。

このパネルにはドロップダウン リストがあります。このオプションでは、事前定義済みのイベントのセットを含めてすべてのイベントをフィルタしないか、またはすべてのイベントをフィルタすることを指定できます。オプションは以下のとおりです。

- フィルタなし
- 簡易イベント セット
- 共通イベント セット
- 詳細イベント セット
- テスト セットのロード
- カスタム イベント セット
- すべてのイベントをフィルタ

注: 負荷テストでは、これらの UI エレメントは無効になります。 DevTest が決定するテスト ケースが負荷テストである場合は、個別のステップ応答などを送信するテストを変更できません。 基本的なイベント以外を送信すると、負荷テストの値が低下します。

シミュレータ

［シミュレータ］パネルには、使用中のシミュレータのステータスが表示されます。

テスト イベント

［テスト イベント］パネルには、イベントが表示されます。 最新のイベントが最上部に表示されます。 最も古いイベントが最下部に表示されます。

パネルの下部にある［自動リフレッシュ］チェック ボックスをオンにすると、イベントをリアルタイムでリスト表示できます。また［自動リフレッシュ］ボックスがオフの状態で［リフレッシュ］アイコンをクリックすると、リストを手動でリフレッシュできます。 フィルタされていないイベントのみが表示されます。

各イベントに対して、以下の情報が表示されます。

タイムスタンプ

イベントの時刻

イベント

イベントの名前

シミュレータ

イベントが生成されたシミュレータの名前。

インスタンス

インスタンス ID および実行番号（スラッシュ区切り）

概要

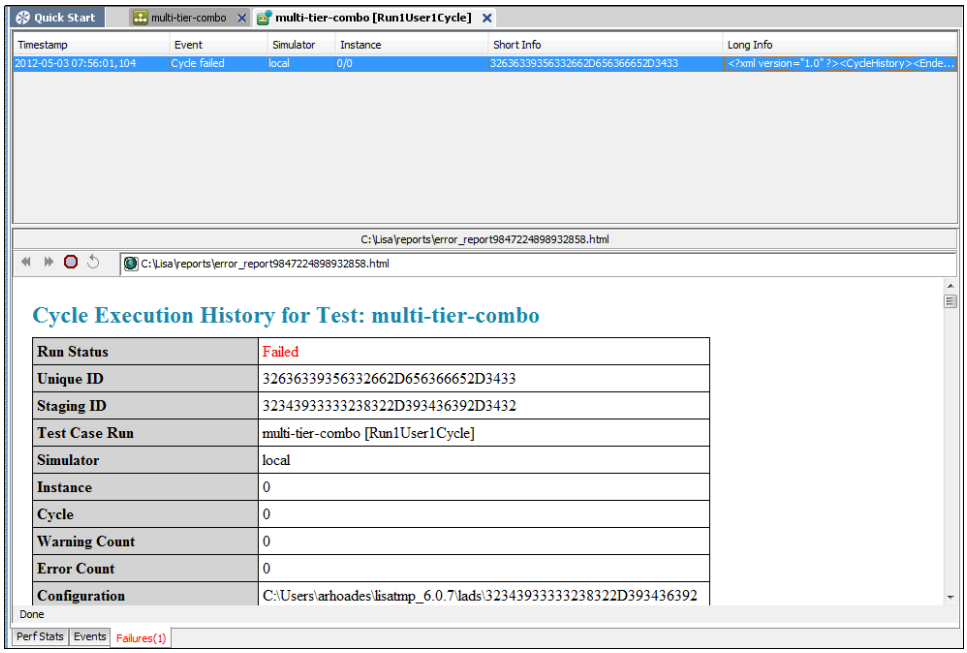
イベントの概要

詳細

イベントの詳細（使用可能な場合）

[失敗]タブ - クイック テスト

ステージングしたテストの実行時にエラーが発生した場合は、ウィンドウの下部の［失敗］タブを確認します。



失敗したテストのレポートを表示するには、ウィンドウの右側にある［詳細］フィールドをダブルクリックします。

クイック テストの開始と停止

テスト モニタ ウィンドウが開いている状態で、メイン ツールバーにあるアイコンをクリックすると、クイック テストを開始および停止できます。

クイック テストを開始するには、メイン ツールバーの[再生] をクリックします。

クイック テストが開始されると、［再生］アイコンは［停止］アイコンに変わります。

クイック テストを停止するには、メイン ツールバーの［停止］ をクリックします。

再度［停止］をクリックすると、テスト ランを強制終了するかどうかを尋ねられます。

- テストを停止することは、「新しいインスタンスを起動しない」ことを意味します。新しいインスタンスは起動されず、実行中のテスト ケースは先頭にループ バックされません。実行中のテストはステップをすべて完了し、終了ステップに移動します。
- テストを強制終了することは、「実行中のすべてのインスタンスをできるだけ早く停止する」ことを意味します。新しいインスタンスは起動されず、実行中のテスト ケースは次のステップに移動しません。 終了ステップは実行されません。 テストが終了ステップを実行しないため、レポートにはテストがまだ実行中であることが示されます。

クイック テストを再生するには、メイン ツールバーの[再生] をクリックします。

クイック テストを閉じるには、メイン ツールバーの［閉じる］ をクリックします。

［アクション］メニュー

テストを実行し、テストを開始するために［再生］をクリックすると、以下のオプションが［アクション］メニューに追加されます。

- メトリック収集の一時停止
- メトリック収集の一時停止の解除
- 最近のメトリックを XML ドキュメントに保存
- 最近のメトリックを CSV ドキュメントに保存
- 間隔データを CSV ドキュメントに保存
- 間隔データを XML ドキュメントに保存
- 最近のイベントを CSV ドキュメントに保存
- このテストをステージング
- テストを停止
- テストを再起動（ドキュメントを再ロード）

テスト ケースのステージング

DevTest ワークステーション からテスト ケースを実行するには、設定、ステージング ドキュメント、コーディネータ サーバを指定します。

DevTest ワークステーション のメイン ツールバーには［ラボ ステータス］アイコンがあります。 クラウドベースのラボにテスト ケースをステージングする場合、このアイコンはラボがプロビジョニング中であること、そして準備ができたことを表します。

次の手順に従ってください:

1. DevTest ワークステーション でテスト ケースを開きます。
2. メイン メニューから［アクション］-［テストのステージング］を選択します。

   ［テスト ケースのステージング］ダイアログ ボックスが表示されます。
3. ［設定］ドロップダウン リストから設定 (P. 116) を選択します。

   このフィールドを空白のままにすると、デフォルトの設定が使用されます。
4. ［ステージング ドキュメント］ドロップダウン リストからステージング ドキュメント (P. 252) を選択します。
5. ［コーディネータ サーバ］ドロップダウン リストから、コーディネータ サーバ (P. 14) または（使用可能な場合） クラウドベースのラボ (P. 393) を選択します。

   コーディネータ サーバの名前には関連するラボが含まれます。たとえば、Coordinator@Default は、Default ラボが使用されることを表します。クラウドベースのラボの場合、ラボが開始され、そのラボのコーディネータに対してテスト ケースがステージングされます。
6. ［ステージング］をクリックします。
   - コーディネータ サーバを選択すると、テスト モニタ (P. 341) ウィンドウが表示されます。 ここで、モニタするメトリックおよびイベントを選択し、テスト ケースを開始できます。

- クラウドベースのラボを選択すると、ラボのプロビジョニングにある程度の時間がかかることを示すメッセージが表示されます。プロセスが完了すると通知されます。[OK]をクリックします。[プロビジョニング完了]メッセージが表示されたら、[OK]をクリックします。テスト モニタ (P. 341) ウィンドウが表示されます。ここで、モニタするメトリックおよびイベントを選択し、テスト ケースを開始できます。

テスト モニタ ウィンドウ - テスト ケース

テスト ケースをステージング (P. 340)すると、DevTest ワークステーションにテスト モニタ ウィンドウが表示されます。テスト ケース用のテスト モニタ ウィンドウは、クイック テスト用のテスト モニタ ウィンドウと同様です。

最初にテストをステージングします。まだテストの実行は開始されていません。

テストの実行時にモニタするメトリックおよびイベントを選択できます。

テスト モニタは、以下の 2 つのタブで構成されています。

- [パフォーマンス統計 (P. 342)] タブ では、メトリックを選択し、それを時間の関数として表示できます。
- [イベント (P. 345)] タブでは、イベントを選択して、そのイベントがテストの実行時に発生したときに表示されるようにすることができます。

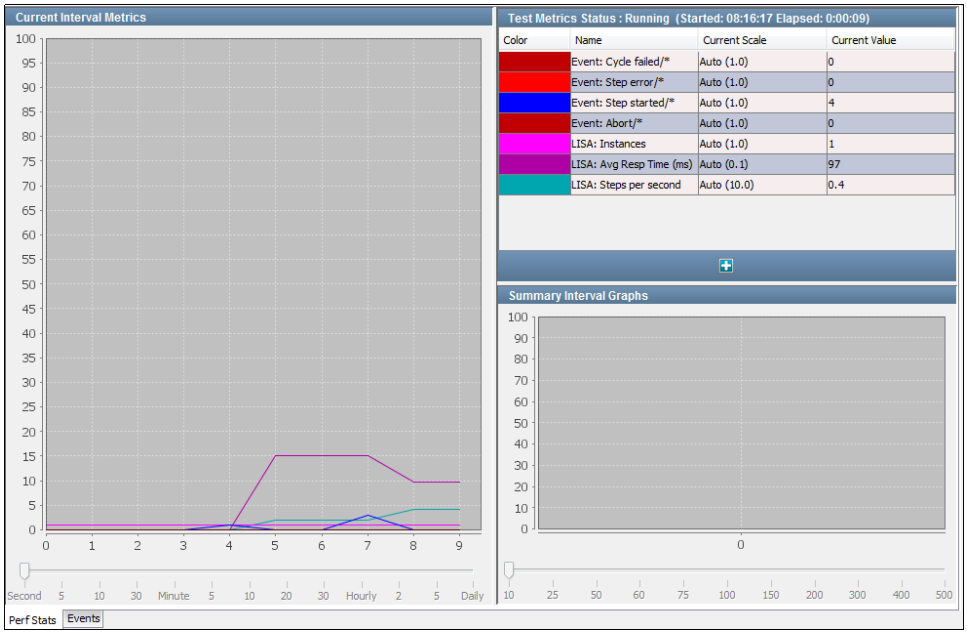
テスト ケースを実行し、テストが失敗した場合は、[失敗 (P. 347)] という 3 つ目のタブも表示されます。

ベースライン テスト ケースを実行する場合は、[統合ベースライン] タブも表示されます。詳細については、「*CA Continuous Application Insight の使用*」を参照してください。

local.properties ファイルに **lisa.coord.failure.list.size** プロパティを追加することで、失敗イベントのサイズを制限できます。

[パフォーマンス統計]タブ - テスト ケース

[パフォーマンス統計] タブでは、テストの実行時にモニタするメトリックおよびイベントを追加できます。



[パフォーマンス統計] タブは以下のパネルで構成されています。

- テスト メトリック ステータス
- 現在の間隔メトリック
- サマリ間隔グラフ

テスト メトリック ステータス

[テスト メトリック ステータス] パネルには、現在モニタされているメトリックのリストが表示されます。

いくつかのメトリックはデフォルトで表示されています。 メトリックは識別しやすくするために色分けされています。

[テスト メトリック ステータス] パネルには、以下の列があります。

色

グラフの色分けに使用される色。

名前

メトリックの名前。 短い名前を使用してメトリックをフィルタしている場合、［名前］フィールドには、スラッシュの後に短い名前が表示されます。 アスタリスクは、メトリックのイベント名のみを使用していることを意味します。

現在のスケール

グラフで使用される縦方向のスケール。 任意のメトリックの［現在の間隔メトリック］グラフのグラフ スケールを調節できます。 ［現在のスケール］セルをダブルクリックしてドロップダウン リストから新しい値を選択することにより、スケールを調節します。 ［現在の間隔メトリック］グラフで、メトリックのスケールが変更されます。

注: ［現在のスケール］のデフォルト設定は、自動スケールの 100 です。() 内の数値は、自動スケールが同じ 0 ～100 のグラフにデータをすべて対応させるために決定したスケールの値を示します。 この値とスケールを乗算したものがグラフの Y 座標になります。 スケールを変更しても現在の値は変わりません。 ただし、グラフで使用される Y 座標を決定する計算では、異なる Y 座標が算出される場合があります。 座標が 100 を超えている場合は、より大きな値に対応できるように Y 軸の制限も増加させる必要があります。

現在の値

メトリックの現在の値。

メトリックはサンプリングに基づいています。たとえば、インスタンス メトリックは、サンプルを収集したときにテストを実行していた仮想ユーザの数です。 テストの実行を待機中のインスタンスの数は、より高い数に設定できます。 たとえば、ステージング ドキュメントでペースが設定されている場合、インスタンス メトリックは何人の仮想ユーザがテストを実行しているかを表わします。 この値は、ステージング ドキュメントで割り当てられている仮想ユーザの数より少ない場合があります。 定常的な仮想ユーザが 250 人いると仮定します。インスタンスの数が 100 の場合、その他の 150 人の仮想ユーザは待機中です。これはペースによってそれらが保留状態になっているためです。

メトリックの詳細については、「*CA Application Test の使用*」の「メトリックの生成 (P. 286)」を参照してください。

現在の間隔メトリック

［テスト メトリック ステータス］パネルの各メトリックに対して、［現在の間隔メトリック］パネルに実行時の指定された時間間隔での値が表示されます。

時間間隔を指定するには、パネルの下部にあるスライダを使用します。実行が開始された後は、値は調節できません。

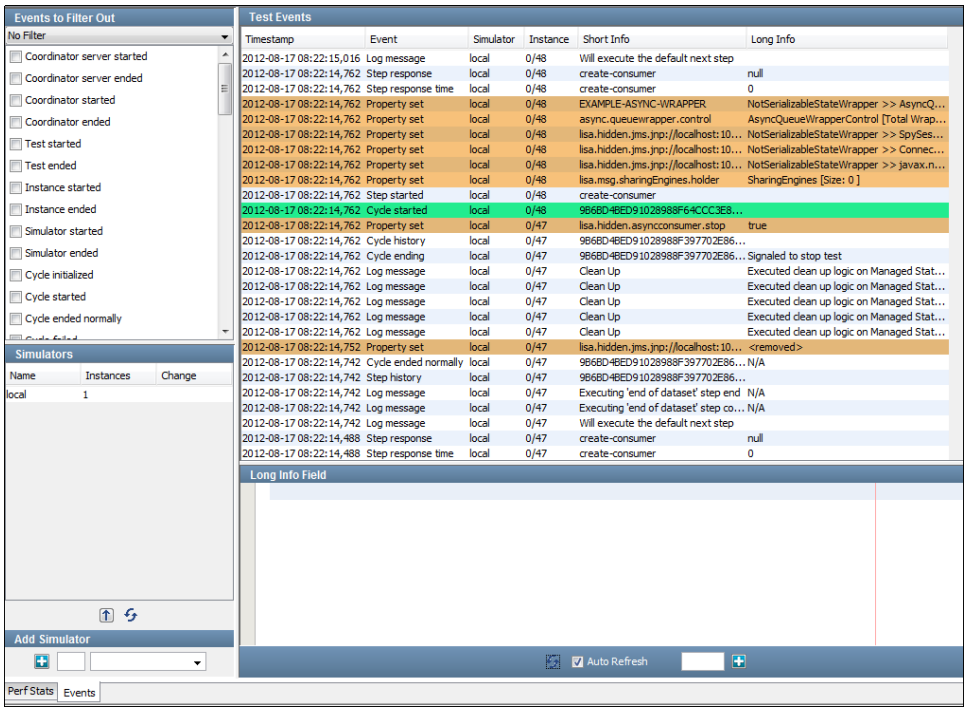
サマリ間隔グラフ

［サマリ間隔グラフ］パネルには、［テスト メトリック ステータス］パネルの各メトリックの値を指定された時間間隔で平均したものが表示されます。

時間間隔あたりのサンプル数を指定するには、パネルの下部にあるスライダを使用します。 実行が開始された後は、値は調節できません。

[イベント]タブ - テスト ケース

［イベント］タブには、実行時に生成されるイベントが表示されます。



［イベント］タブは以下のパネルで構成されています。

- フィルタで除外するイベント
- シミュレータ
- テスト イベント

フィルタで除外するイベント

［フィルタで除外するイベント］パネルでは、実行時に表示されるイベントを制限することができます。 このパネルには、イベントのリストが表示されます。 イベントに対するチェック ボックスをオンにすると、そのイベントは実行時に表示されなくなります。

このパネルにはドロップダウン リストがあります。このオプションでは、事前定義済みのイベントのセットを含めてすべてのイベントをフィルタしないか、またはすべてのイベントをフィルタすることを指定できます。オプションは以下のとおりです。

- フィルタなし
- 簡易イベント セット
- 共通イベント セット
- 詳細イベント セット
- テスト セットのロード
- カスタム イベント セット
- すべてのイベントをフィルタ

注: 負荷テストでは、これらの UI エレメントは無効になります。 DevTest が決定するテスト ケースが負荷テストである場合は、個別のステップ応答などを送信するテストを変更できません。 基本的なイベント以外を送信すると、負荷テストの値が低下します。

シミュレータ

［シミュレータ］パネルには、使用中のシミュレータのステータスが表示されます。

テスト イベント

［テスト イベント］パネルには、イベントが表示されます。 最新のイベントが最上部に表示されます。 最も古いイベントが最下部に表示されます。

パネルの下部にある［自動リフレッシュ］チェック ボックスをオンにすると、イベントをリアルタイムでリスト表示できます。また［自動リフレッシュ］ボックスがオフの状態で［リフレッシュ］アイコンをクリックすると、リストを手動でリフレッシュできます。 フィルタされていないイベントのみが表示されます。

各イベントに対して、以下の情報が表示されます。

タイムスタンプ

イベントの時刻

イベント

イベントの名前

シミュレータ

イベントが生成されたシミュレータの名前。

インスタンス

インスタンス ID および実行番号（スラッシュ区切り）

概要

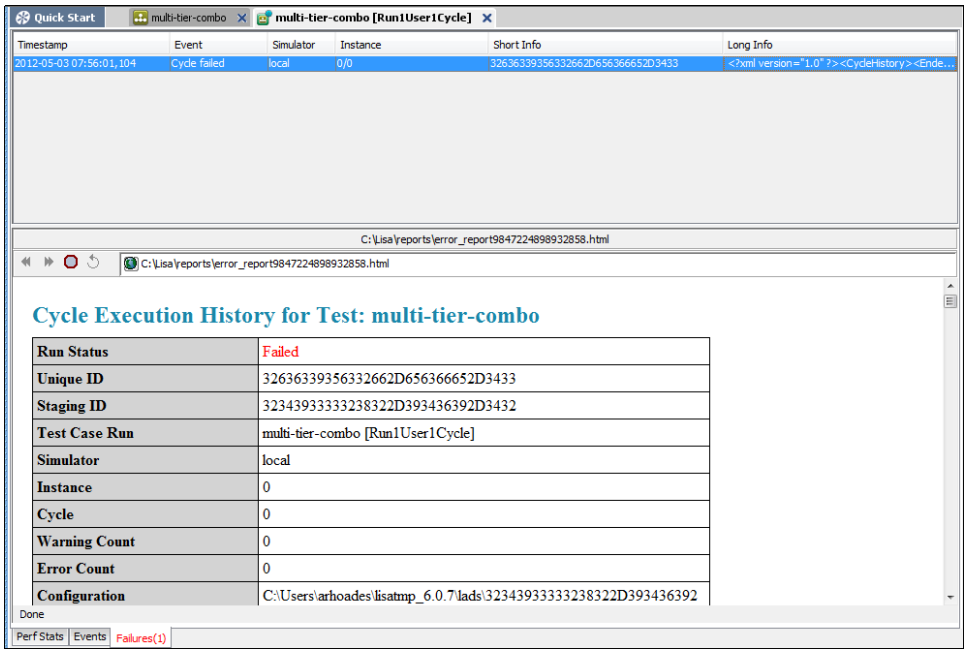
イベントの概要

詳細

イベントの詳細（使用可能な場合）

[失敗]タブ - テスト ケース

ステージングしたテストの実行時にエラーが発生した場合は、ウィンドウの下部の［失敗］タブを確認します。



失敗したテストのレポートを表示するには、ウィンドウの右側にある［詳細］フィールドをダブルクリックします。

テスト ケースの開始と停止

テスト モニタ ウィンドウが開いている状態で、メイン ツールバーにあるアイコンをクリックすると、テスト ケースを開始および停止できます。

テスト ケースを開始するには、メイン ツールバーの［再生］ をクリックします。

テスト ケースが開始されると、［再生］アイコンは［停止］アイコンに変わります。

テスト ケースを停止するには、メイン ツールバーの［停止］ をクリックします。

再度［停止］をクリックすると、テスト ランを強制終了するかどうかを尋ねられます。

- テストを停止することは、「新しいインスタンスを起動しない」ことを意味します。新しいインスタンスは起動されず、実行中のテスト ケースは先頭にループ バックされません。実行中のテストはステップをすべて完了し、終了ステップに移動します。
- テストを強制終了することは、「実行中のすべてのインスタンスをできるだけ早く停止する」ことを意味します。新しいインスタンスは起動されず、実行中のテスト ケースは次のステップに移動しません。 終了ステップは実行されません。 テストが終了ステップを実行しないため、レポートにはテストがまだ実行中であることが示されます。

テスト ケースを再生するには、メイン ツールバーの［再生］ をクリックします。

テスト ケースを閉じるには、メイン ツールバーの［閉じる］ をクリックします。

テスト スイートの実行

テスト スイートでは、関連するテスト ケースやテスト スイートをグループ化して、単一のテストとして実行できます。 テスト スイート ドキュメントでは、スイートのコンテンツ、生成するレポート、および収集するメトリックを指定します。 これらのレポートおよびメトリックは、スイート全体に関連付けられます。 スイート内の各テストは、引き続きそれぞれのレポートやメトリックを生成します。 また、各テストはそれ自体のステージング ドキュメント、設定、監査ドキュメントを使用する機能を引き続き保持します。 そのため、スイート内のテストは分散環境で実行できます。

スイートにスイートが含まれている場合、含まれているスイート内の個々のテストはスイートから抽出され、現在のスイートで個々のテストとして実行されます。 含まれているスイートのデフォルト、および起動や切断の設定は無視されます。

DevTest ワークステーション では、テスト スイートの設定およびステージング、テストの実行時のモニタ、テスト終了時のレポートの表示を行えます。

次の手順に従ってください:

1. DevTest ワークステーション でテスト スイートを開きます。
2. メイン メニューから［アクション］-［実行］を選択します。

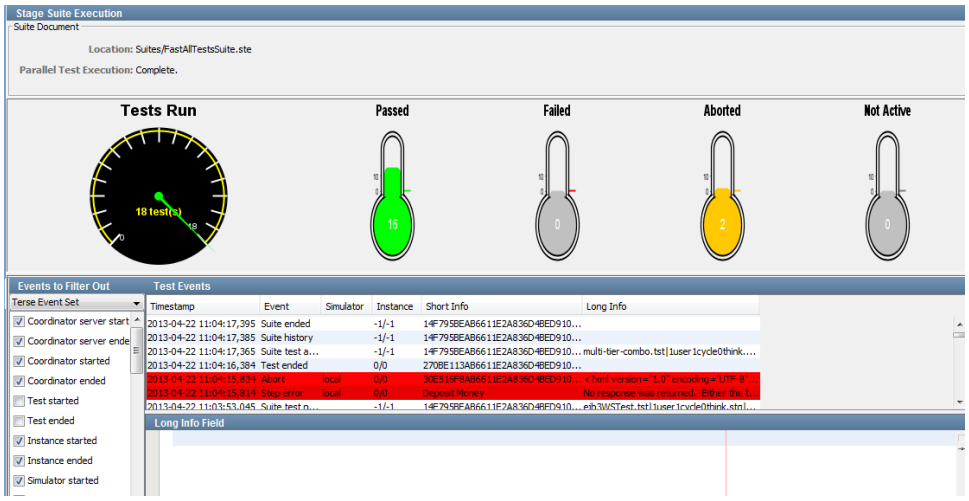
   この操作によってメニューが表示され、このメニューからローカルで実行するか、レジストリを指定して実行するかを選択できます。

   ［スイートをローカルで実行］ダイアログ ボックスが表示されます。
3. テスト スイートの名前を入力します
4. ドロップダウン リストから設定を選択します。
5. ［ステージング］をクリックします。

［スイートのステージングの実行 (P. 350)］ウィンドウが表示され、テストが開始されます。

スイートのステージングの実行

DevTest ワークステーション で[テスト スイートの実行](P. 349)中に［ステージング］をクリックすると、［スイートのステージングの実行］タブが表示され、テストが開始されます。



上部のパネルには、以下の情報が表示されます。

- テスト スイート ドキュメントの場所および名前
- 現在のテストの名前

中央のパネルには、［テスト実行］メータと 4 つの温度計が表示されます。

- ［テスト実行］メータには、完了したテストの数とテストの総数が表示されます。
- 温度計には、［**成功**］、［**失敗**］、［**中止**］、［**非アクティブ**］の 4 つのラベルが付けられています。 温度計の色は、事前に設定された数のケースが成功したことを表します スイート内の 50 から 100 のテスト ケースを実行する場合の既知の動作は以下のとおりです。
  - 成功したテストの数が 50 を超えている場合、温度計はオレンジ色です。
  - 成功したテストの数が 75 を超えている場合、温度計は赤色です。
  - 成功したテストの数が 100 を超えている場合、温度計は灰色です。

下部のパネルには、以下のタブがあります。

- [［イベント］タブ](P. 352)： 要求したイベントが発生したときに表示します

- ［結果］タブ (P. 354)： 個々のテストのステータスを表示します

スイートのステージングの実行 - [イベント]タブ

［イベント］タブには、タブの左側のパネルから選択したイベントがリスト表示されます。

フィルタで除外するイベント

［フィルタで除外するイベント］領域には、使用可能なイベントのリストが表示されます。イベントにはそれぞれチェック ボックスがあります。モニタしないイベントを選択するには、このリストを使用します。

1 つの方法は、フィルタで除外するイベントを手動で選択することです。もう 1 つの方法は、ドロップダウン リストから以下のいずれかのイベント セットを選択した後、必要に応じて選択内容をカスタマイズすることです。

- 簡易イベント セット
- 共通イベント セット
- 詳細イベント セット
- テスト セットのロード

［フィルタなし］を選択すると、すべてのチェック ボックスをオフにできます。［すべてのイベントをフィルタ］を選択すると、すべてのチェック ボックスをオンにできます。

テスト イベント

イベントが発生すると、［テスト イベント］領域に表示されます。最新のイベントがリストの最上位に表示されます。 パネルの下部にある［自動リフレッシュ］チェック ボックスをオンにすると、イベントをリアルタイムでリスト表示できます。 ［自動リフレッシュ］チェック ボックスがオフの状態で［リフレッシュ］アイコンをクリックすると、リストを手動でリフレッシュできます。

各イベントに対して、以下の情報が表示されます。

タイムスタンプ

イベントの時刻

イベント

イベントの名前

シミュレータ

イベントが生成されたシミュレータの名前。

インスタンス

インスタンス ID および実行番号（スラッシュ区切り）

概要

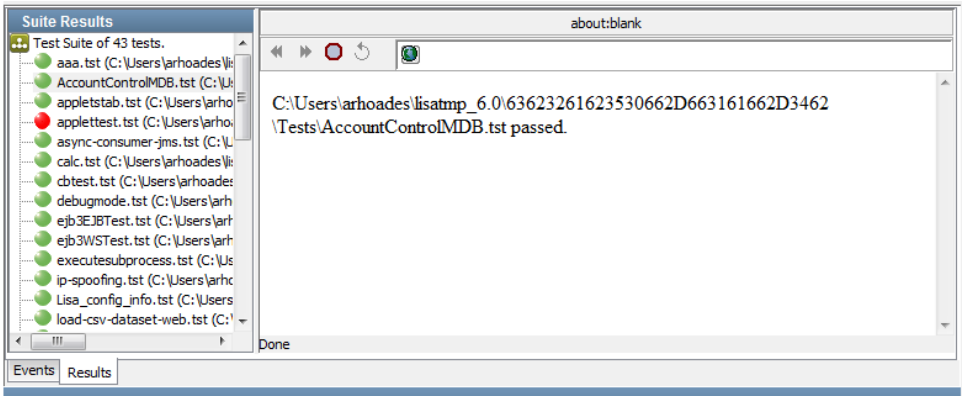
イベントの概要

詳細

イベントの詳細（使用可能な場合）

スイートのステージングの実行 - [結果]タブ

［結果］タブでは、テスト スイート内の個々のテストのステータスが表示されます。



スイート結果

［スイート結果］領域には、スイート内のすべてのテストのリストがアイコンと一緒に表示されます。

- 緑のアイコンは、テストが完了し、成功したことを表しています。
- 赤のアイコンは、テストが完了し、失敗したことを表しています。
- 青のアイコンは、テストがまだ実行中であることを表しています。

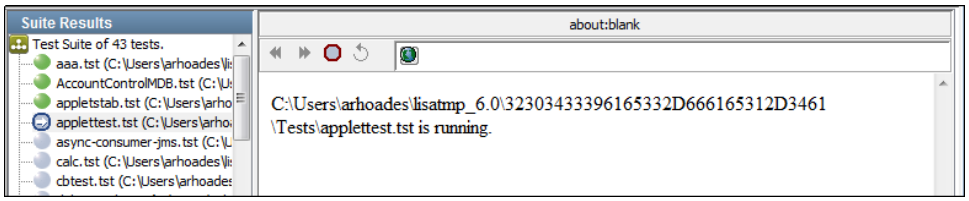
まだ実行中のテストのテスト モニタを表示するには、パネルの下部にある［テストの表示］  をクリックします。

完了したテストについては、テスト名を選択すると、右側のテキスト領域でステータスを確認できます。 ステータスを確認することは、テストが失敗した理由を確認するのに最も有効です。

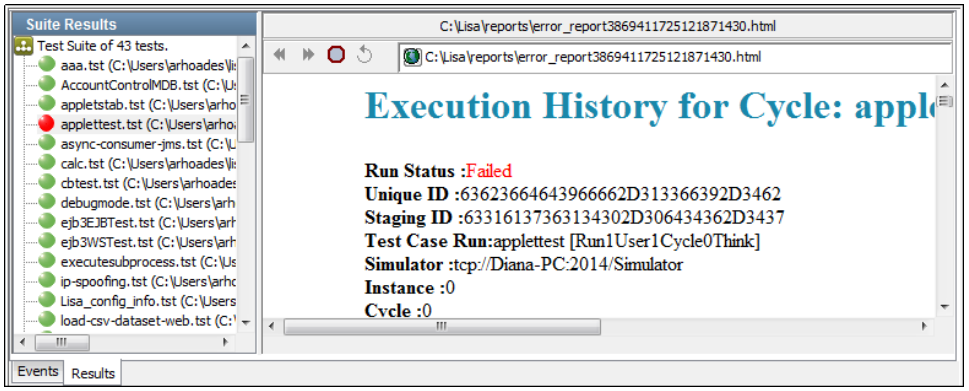
ステータス ウィンドウ

ステータス ウィンドウには、左側で選択されているテストのステータスが表示されます。

以下のテストは実行中です。



以下のテストは失敗しています。



失敗したテストのステータスには、テストが失敗した理由についての情報が表示されます。

［スイートのステージングの実行］パネルの下部にはツールバーがあります。

アイコンの 1 つ目のセットでは、スイート内の個々のテストを管理します。これらのアイコンを使用するには、まず［結果］タブのテスト リスト セクションでテストを選択します。 選択したテストに対しては、以下の機能を使用できます。

	テストを停止	テストが次の論理的な終了に到達するか失敗した後、テストを停止します。 新しいサイクルは開始されません。
	テストの強制終了	現在のステップが完了した直後にテストを停止します。
	テストの表示	このツールは現在実行中のテストに対してのみ機能します。選択したテストのテスト モニタを起動します。

アイコンの 2 つ目のセットでは、テスト スイート自体、またはスイートの実行を管理します。

	スイートの実行	スイートのテストを開始/再起動します。
	閉じる	［スイートのステージングの実行］ウィンドウを閉じます。
	テストの表示	選択したテストのテスト モニタを起動します。

テスト モニタの詳細については、「レジストリ モニタの使用」を参照してください。

ロード テスト オプティマイザの使用

ロード テスト オプティマイザはレジストリ モニタに表示され、テスト中のシステムに対して簡単な負荷テストを実行します。 負荷テストでは、テスト中のシステムでサポートできるユーザの数を判定します。

ロード テスト オプティマイザでは、テスト中のシステムは継続的に実行されます。 特定のターゲットに到達するまで、さらに多くのシミュレートされたユーザがアクセスし続けます。 このターゲットは通常、事前に定義された平均応答時間です。

ロード テスト オプティマイザは、テスト中のシステムで何人のユーザをサポートできるかの判定を支援します。 オプティマイザには、平均応答時間などの任意の DevTest メトリックを設定できます。オプティマイザには、SNMP、JMX、Windows Perfmon などの外部ソースから取得されるメトリックも設定できます。 ユーザの数は事前設定された間隔で増加し、事前設定されたメトリックのしきい値に到達すると通知されます。 たとえば、平均応答時間が 2 秒に達するまで、10 秒ごとにインスタンスを 5 つ追加して、テスト インスタンスの数を増加させることができます。

テストを最適化するには、実行中である必要があります。

オプティマイザを設定および起動する方法

1. レジストリ モニタ で、実行中のテストのリストからテストを選択し、［テストの最適化］  をクリックします。

   ［オプティマイザ］パネルが表示され、ステージング ドキュメントの名前が最上部に表示されます。

2. 以下のパラメータを設定します。

   メトリック

   最適化に使用するメトリック。 プルダウン リストから選択します。

   シミュレータ

   テスト インスタンスを生成するために使用するシミュレータ。 プルダウン リストから選択します。

   最小しきい値

   システムが追加の仮想ユーザを必要としていることをオプティマイザがレポートするときのメトリック値。 数値を入力します。

   最大しきい値

システムがこれ以上の仮想ユーザをサポートできないことをオプティマイザがレポートするときのメトリック値。 数値を入力します。

増分（インスタンス数）

毎回の更新時にシステムに追加する仮想ユーザの数。 数値を入力します。

更新頻度（ミリ秒）

追加を行う間隔（ミリ秒）。

3. ［オプティマイザの開始］  をクリックします。

オプティマイザによって仮想ユーザの数が増やされると、［オプティマイザ］セクションとレジストリ モニタの［テスト］タブの［インスタンス］列の両方に仮想ユーザの数が表示されます。

オプティマイザが実行されたら、パラメータを変更できます。 変更した情報でオプティマイザを更新するには、［更新］  をクリックします。

4. オプティマイザを閉じるには、［閉じる］  をクリックします。

Selenium 統合テスト ケースの実行

Selenium 統合テスト ステップでは、Selenium Builder から DevTest Solutions に Web ベースのユーザ インターフェースのテスト スクリプトをインポートできます。 これらのテスト スクリプトのレコーディングには、Selenium Builder が必要です。 Selenium Builder は Firefox でのみサポートされています。DevTest にテストをインポートした後、Mozilla Firefox、Google Chrome、または Internet Explorer 8.0 以降でテストを実行できます。また、ローカルまたはリモートのブラウザでテストを実行することもできます。

Selenium 統合テストは、DevTest でその他のテスト ケースと同様に実行できます。 ただし、Firefox 以外のブラウザでこれらのテストを実行するには、追加の前提条件タスクが必要です。 テスト ケースの実行に関する一般情報については、「テスト ケースおよびスイートの実行 (P. 315)」を参照してください。

Google Chrome での Selenium 統合テストの実行（ローカル）

このトピックでは、ローカル コンピュータ上の Google Chrome で Selenium 統合テスト ケースを実行する方法について説明します。

次の手順に従ってください:

1. Selenium Chrome ドライバをダウンロードし、ローカル ディレクトリに保存します。
   a. http://www.seleniumhq.org/download/ にアクセスします。
   b. ［**Third Party Browser Drivers NOT DEVELOPED by seleniumhq**］セクションで Chrome ドライバを検索してダウンロードします。
2. Chrome でテスト ケースを実行するために使用するプロジェクト設定ファイルに、以下のプロパティを追加します。
   - **キー：** selenium.browser.type

     値： Chrome
   - **キー：** selenium.chrome.driver.path

     値： ダウンロードした Chrome ドライバのフル パス。 たとえば、C:¥lisa-se¥chromedriver.exe です。

   注: これらのプロパティは、その他のプロジェクト設定ファイルに追加する前に、project.config に追加します。
3. プロジェクト パネルで選択したプロジェクト設定ファイルを右クリックし、［アクティブ化］を選択します。
4. テストを実行します。

Microsoft Internet Explorer での Selenium 統合テストの実行（ローカル）

このトピックでは、ローカル コンピュータ上の Internet Explorer で Selenium 統合テスト ケースを実行する方法について説明します。

次の手順に従ってください:

1. Selenium 32 ビット Windows IE ドライバをダウンロードし、ローカル ディレクトリに保存します。
   a. http://www.seleniumhq.org/download/ にアクセスします。
   b. ［**Internet Explorer Driver Server**］セクションで 32 ビット Windows IE ドライバを検索してダウンロードします。

   注: 64 ビット ドライバのパフォーマンスでの既知の問題のため、64 ビットのシステムを使用している場合でも、32 ビット バージョンをインストールすることを推奨します。
2. Internet Explorer でテスト ケースを実行するために使用するプロジェクト設定ファイルに、以下のプロパティを追加します。
   - **キー：** selenium.browser.type

     値： IE
   - **キー：** selenium.ie.driver.path

     値： ダウンロードした Internet Explorer ドライバのフル パス。例：C:¥lisa-se¥IEDriverServer.exe

   注: これらのプロパティは、その他のプロジェクト設定ファイルに追加する前に、project.config に追加します。
3. プロジェクト パネルで選択したプロジェクト設定ファイルを右クリックし、［アクティブ化］を選択します。
4. Internet Explorer のセキュリティ設定を変更します。
   a. Internet Explorer を開始します。
   b. ［ツール］ - ［インターネット オプション］をクリックします。
   c. ［セキュリティ］タブをクリックします。
   d. ［保護モードを有効にする］チェック ボックスが、以下のゾーンに対して同一の設定（オンまたはオフ）であることを確認します。この設定に整合性がない場合、Selenium は開始しません。
      - インターネット
      - ローカル イントラネット

- 信頼済みサイト
- 制限付きサイト

e. ［OK］をクリックして変更を保存し、［インターネット オプション］ウィンドウを閉じます。

5. テストを実行します。

リモートのブラウザでの Selenium 統合テストの実行

このトピックでは、リモートのブラウザで Selenium 統合テスト ケースを実行する方法について説明します。リモートのブラウザは、Mozilla Firefox、Google Chrome、または Internet Explorer 8.0 以降です。

次の手順に従ってください:

1. Selenium Server をダウンロードし、ローカル ディレクトリに保存します。
   a. http://www.seleniumhq.org/download/ にアクセスします。
   b. ［**Selenium Server (formerly the Selenium RC Server)**］セクションで Selenium Server のスタンドアロン .jar ファイルを検索してダウンロードします。
2. 使用するブラウザ用のドライバがリモート コンピュータで使用可能であることを確認します。
3. リモート コンピュータで、コマンド プロンプトから以下のコマンドを実行します。
   ```
   java -jar selenium-server-standalone-2.xx.0.jar -role hub
   ```
4. リモート コンピュータで、新しいコマンド プロンプトから以下のコマンドを実行します。
   ```
   java -jar selenium-server-standalone-2.xx.0.jar -role node -hub
   http://localhost:4444/grid/register
   -Dwebdriver.chrome.driver=c:¥lisa-se¥chromedriver.exe
   -Dwebdriver.ie.driver=c:¥lisa-se¥IEDriverServer.exe
   ```
5. ローカル コンピュータで、リモートのブラウザでテスト ケースを実行するために使用するプロジェクト設定ファイルに以下のプロパティを追加します。
   - **キー：** selenium.broswer.type

     値： IE、Firefox、または Chrome
   - **キー：** selenium.remote.url

     値： リモート Selenium Server ハブの URL たとえば、**http://your_remote_hostname:4444/wd/hub** などです。

   複数のブラウザで Selenium 統合テストを実行する場合は、ブラウザ タイプごとにプロジェクト設定ファイルを作成します。 その後、各テスト ランで異なる設定ファイルをアクティブにすることにより、複数のブラウザでテストを実行できます。 設定ファイルの詳細については、「設定 (P. 116)」を参照してください。

注: これらのプロパティは、その他のプロジェクト設定ファイルに追加する前に、project.config に追加する必要があります。

6. プロジェクト パネルで選択したプロジェクト設定ファイルを右クリックし、［アクティブ化］を選択します。
7. テストを実行します。

   リモート コンピュータ上の選択されたブラウザでテストが実行されます。

   注: グリッド構成で Selenium Server を使用する方法の詳細については、https://code.google.com/p/selenium/wiki/Grid2 を参照してください。

負荷テストの判定

デフォルトでは、DevTest は、実行時にテスト ケースまたはスイートのさまざまな特性を検証し、負荷テストが実行されていると判定する場合があります。負荷テストを実行していると DevTest が判定した場合、DevTest は自動的に再設定されます。

自動設定は、**lisa.load.auto.reconfigure**=false プロパティを設定することで変更できます。

負荷テスト用の再設定では、いくつかの設定が行われます。 最も重要なのは、テストを実行しているシミュレータから送信されるイベントが制限されることです。 具体的には、シミュレータは LoadTest フィルタに設定されたイベントのみを送信します。 ステップの開始、ステップ応答、ステップ応答時間、サイクルの開始、ログ、プロファイル、およびその他の類似のイベントは、コーディネータに送信されません。したがって、テストを開始した DevTest コンポーネント（DevTest ワークステーション またはテスト ランナー）にも送信されません。

負荷テストでこの制限が行われる理由は、イベントを送信するオーバーヘッドが負荷を生成する処理をすぐに上回ってしまうためです。 コーディネータは、すぐに過負荷になります。これは、反応時間がゼロのテストで顕著に現れます。

負荷テストを実行していると DevTest が見なした場合、コーディネータのログ ファイルに以下のようなメッセージが記録されます。

```
INFO  com.itko.lisa.coordinator.CoordinatorImpl - 負荷テスト用に設定しています（仮想ユーザ >= 150）
INFO  com.itko.lisa.coordinator.CoordinatorImpl - 負荷テスト用に設定しています
INFO  com.itko.lisa.net.RemoteEventDeliverySupport - 負荷テスト用に設定しています
```

ステージング ドキュメントで CAI を無効にするように促す追加のメッセージが表示される場合があります。

テストがスイートの一部である場合、通常、テスト モニタ ウィンドウに情報は表示されません。 負荷テストでは、ユーザ インターフェースに表示されるよりも速くイベントが生成されるため、一部のイベントは無視されます。 これらの実行時メトリックを表示するには、以下のプロパティを変更します。

```
lisa.load.auto.reconfigure=false
```

CA Application Test が負荷テストを判定するパラメータを調整するには、以下のプロパティを調整します。

```
lisa.loadtest.aggressive.detection=false
lisa.coordinator.step.per.sec.load.threshold=100
lisa.coordinator.vuser.load.threshold=150
```

ステージング ドキュメントにデフォルトのレポート ジェネレータが含まれている場合、そのテストは負荷テストではないと判定されます。

1 秒当たりの非クワイエット ステップの数が 100 を超えるか、仮想ユーザの数が 150 を超える場合、そのテストは負荷テストと判定されます。

lisa.load.auto.reconfigure プロパティを ture に設定する場合、0% の反応時間を含むステージング ドキュメントによるテストまたは 0 の反応時間のステップで全体が構成されるテストは、負荷テストと見なされます。

負荷テストと判定された場合のその他の動作は以下のとおりです。

- サーバ コンソールのテスト名の隣にアスタリスクが表示されます。
- レポート コンソールに、通常よりも詳細が表示されなくなります。
- 負荷テストが自動的にオンにされた場合、その変更を記述するイベントが生成されます。

テスト ランナー

テスト ランナー コマンドライン ユーティリティは、同じ機能であるもののユーザ インターフェースのない DevTest ワークステーション の「ヘッドレス」バージョンです。 つまり、スタンドアロン アプリケーションとして実行できます。

テスト ランナーでは、テストをバッチ アプリケーションとして実行できます。 テストをリアルタイムでモニタすることはできませんが、レポートを要求して後で確認することができます。

- Windows では、テスト ランナーは **LISA_HOME¥bin** ディレクトリにある Windows 実行可能ファイル **TestRunner.exe** です。
- UNIX では、テスト ランナーは UNIX 実行可能ファイルの **TestRunner** および UNIX スクリプトの **TestRunner.sh** です。

テスト ランナーでは、DevTest テストを継続的なビルド ワークフローに組み込むことができます。また、テスト ランナーと JUnit を一緒に使用して、Ant やその他のビルド ツールで標準的な JUnit テストを実行できます。

テスト ランナーには、以下のオプションがあります。

```
TestRunner [-h] [[-r StagingDocument] [-t TestCaseDocument] [-cs
CoordinatorServerName]] | [-s TestSuiteDocument] [-m TestRegistryName] [-a]
[-config configurationFileName]
```

テスト ランナーのヘルプ情報を表示するには、**-h** または **--help** オプションを使用します。

```
TestRunner -h
```

バージョン番号を表示するには、**--version** オプションを使用します。

自動化されたビルドの一部として

DevTest テスト ケースは、自動的なビルドおよびテスト プロセスに組み込むことができます。 DevTest では、Java Ant、Java JUnit、およびサンプルの Ant ビルド スクリプトを使用するために必要な追加のソフトウェアを用意しています。

DevTest テスト ケースは、ネイティブ JUnit テストとして実行およびレポートされます。

テスト ランナーは、カスタム Java クラスを使用して標準的な JUnit テストで使用できます。 詳細については、「*管理*」の「Ant および JUnit による DevTest Solutions の実行」を参照してください。

このセクションには、以下のトピックが含まれます。

- テスト ランナーで MAR を実行 (P. 367)
- テスト ランナーでテスト ケースを実行 (P. 368)
- テスト ランナーでスイートを実行 (P. 369)
- その他のテスト ランナーのオプション (P. 370)
- テスト ランナーの複数のインスタンス (P. 371)
- テスト ランナーのログ ファイル (P. 371)

テスト ランナーで MAR を実行

テスト ランナーでモデル アーカイブ（MAR） (P. 300)を実行するには、以下のオプションを指定します。

- **-mar** または **--mar** MAR ファイルの名前

例

以下の例では、**test1.mar** という名前の MAR ファイルを実行します。

```
TestRunner -mar C:¥test1.mar
```

テスト ランナーでテスト ケースを実行

テスト ランナーで単一のテスト ケースを実行するには、以下のオプションを指定します。

- **-t** または **--testCase** テスト ケース ドキュメントの名前
- **-r** または **--stagingDoc** ステージング ドキュメントの名前

ステージングをリモートで行う場合は、以下のオプションも指定します。

- **-cs** または **--coordinatorService** コーディネータ サーバの名前

  コーディネータ サーバをレジストリから取得するために **-cs** を指定しない場合は、MAR/MARI ファイルで指定されているコーディネータが使用されます。 コーディネータも指定しない場合は、レジストリで指定されているデフォルトのコーディネータが使用されます。デフォルトのコーディネータが存在しない場合は、テストはローカルで実行されます。「-cs local」を使用すると、テストはローカルで実行されます。
- **-m** または **--testRegistry** レジストリの名前

その他のオプションについては、「その他のテスト ランナーのオプション (P. 370)」を参照してください。

例

以下の例では、**examples** プロジェクトにある multi-tier-combo テスト ケースを実行します。

```
TestRunner -t ../examples/Tests/multi-tier-combo.tst
-r ../examples/StagingDocs/Run1User1Cycle.stg -a
```

テスト ランナーでスイートを実行

テスト ランナーでスイートを実行するには、以下のオプションを指定します。

- **-s** または **--testSuite** スイート ドキュメントの名前

監査は実行されません。

ステージングをリモートで行うには、以下のオプションも指定します。

- **-m** または **--testRegistry** レジストリの名前

リモート ステージング プロジェクト ドキュメントについての重要な制限は、すべてのプロジェクトに一意の名前を付けることです。 この制限はプロジェクト スイートだけでなく、プロジェクト内のほかのアセット（データ セットなど）を参照するリモートでステージングするプロジェクト テスト ケースにも適用されます。

その他のオプションについては、「その他のテスト ランナーのオプション (P. 370)」を参照してください。

例

以下の例では、**examples** プロジェクトにある AllTestsSuite スイートを実行します。

```
TestRunner -s ../examples/Suites/AllTestsSuite.ste
```

以下の例では、レジストリが別のコンピュータで実行されていると仮定しています。

```
TestRunner -s ../examples/Suites/AllTestsSuite.ste -m
somecomputer/Registry
```

その他のテスト ランナーのオプション

このトピックでは、モデル アーカイブ（MAR）、テスト ケース、またはスイートをテスト ランナーで実行する際に使用できるその他のオプションについて説明します。

ファイル名を指定する場合には、プロパティを使用できます。 ただし、この内容では、プロパティ ファイル（**lisa.properties**、**local.properties**、**site.properties**）で定義されているシステム プロパティおよびプロパティのみが機能します。

テストが完了したら、レポートを表示できます。

ACL セキュリティ情報の指定

-u *userName* または **--username**=*userName*、および**-p** *password* または **--password**=*password* オプションは、ACL セキュリティ情報を DevTest へ渡します。

テストを自動的に開始

-a または **--autoStart** オプションは、テストを自動的に開始します。 したがって、テストをステージングした後に Enter キーを押す必要がありません。

設定の指定

-config または **--configFile** オプションを使用すると、テスト ランに使用する設定を指定できます。

テスト ランナーでは DevTest プロジェクトを解釈できないため、設定ファイルの完全修飾パスを指定する必要があります。 以下に例を示します。

```
-config ¥path¥to¥lisa¥home¥examples¥Configs¥project.config
```

HTML レポートの生成

-html または **--htmlReport** オプションを使用すると、テスト ランナーにテスト ケースについての HTML サマリ レポートを作成するよう指示できます。 このオプションは、スイートには使用できません。

このオプションの後のパラメータは完全修飾ファイル名である必要があります。 以下に例を示します。

```
-html ¥some¥directory¥MyReport.html
```

更新間隔の変更

-u または **--update** オプションを使用すると、更新間隔をデフォルト値の 5 秒から変更することができます。この間隔は、テスト ランナーがログ ファイルにステータス メッセージを書き込む頻度に相当します。

```
-u 10
```

テスト ランナーの複数のインスタンス

1 台のワークステーションから DevTest サーバに対してテスト ランナーの複数のインスタンスをステージングできます。

各インスタンスに 512 MB が必要。

テスト ランナーのログ ファイル

ログの出力は **trunner.log** ファイルに書き込まれます。 このファイルの場所の詳細については、「*管理*」の「ログ ファイルの概要」を参照してください。

使用されるログ レベルは、**LISA_HOME¥logging.properties** ファイルで設定されるレベルと同じです。

ログ レベルを変更するには、**logging.properties** ファイルの **log4j.rootCategory** プロパティを以下のものから

```
{{log4j.rootCategory=INFO,A1}}
```

以下のように変更してください。

```
{{log4j.rootCategory=DEBUG,A1}}
```

LISA Invoke

LISA Invoke は、以下のタスクを URL で実行できる、REST のような Web アプリケーションです。

- テスト ケースの実行
- スイートの実行
- モデル アーカイブ（MAR）の実行

これらのタスクを同期または非同期で実行できます。

応答は XML ドキュメントで構成されます。

lisa.properties ファイルには、LISA Invoke の以下の設定プロパティが含まれます。

```
lisa.portal.invoke.base.url=/lisa-invoke
lisa.portal.invoke.report.url=/reports
lisa.portal.invoke.server.report.directory={{lisa.tmpdir}}{{lisa.portal.invoke.report.url}}
lisa.portal.invoke.test.root={{LISA_HOME}}
```

local.properties ファイルのこれらのプロパティは上書きできます。

LISA Invoke のホーム ページを表示するには、http://*hostname*:1505/lisa-invoke/ にアクセスします。この *hostname* は、レジストリが実行されているコンピュータです。

LISA Invoke によるテスト ケースの実行

LISA Invoke を使用してテスト ケースを実行する構文は以下のとおりです。

```
/lisa-invoke/runTest?testCasePath=testCasePath&stagingDocPath=s
tagingDocPath&[configPath=configPath]&[async=true]&[coordName=c
sName]
```

パラメータは以下のとおりです。

testCasePath

呼び出すテスト ケースのパス。

stagingDocPath

ステージング ドキュメントのパス。 指定しない場合は、デフォルトのステージング ドキュメントが作成されます。

configPath

設定のパス。 指定しない場合は、プロジェクト用の project.config ファイルが使用されます。

async

true の場合、応答にはコールバック キーが含まれます。 指定しない場合、このパラメータは false に設定されます。

coordName

コーディネータのパス。指定しない場合は、デフォルトのコーディネータ名が使用されます。

例: 同期呼び出し

以下の URL では、**examples** プロジェクトの **AccountControlMDB** テスト ケースの同期呼び出しを実行します。

```
http://localhost:1505/lisa-invoke/runTest?testCasePath=examples
/Tests/AccountControlMDB.tst&stagingDocPath=examples/StagingDoc
s/1user1cycle0think.stg
```

以下の XML 応答は、テスト ケースが成功したことを示しています。

```
<?xml version="1.0" encoding="UTF-8"?>
<invokeResult>
  <method name="RunTest">
    <params>
```

```
      <param name="stagingDocPath"
value="examples/StagingDocs/1user1cycle0think.stg" />
      <param name="coordName" value="Coordinator" />
      <param name="configPath" value="" />
      <param name="testCasePath"
value="examples/Tests/AccountControlMDB.tst" />
      <param name="callbackKey"
value="643435336537373l2D343765312D3439" />
    </params>
  </method>
  <status>OK</status>
  <result>
    <status>ENDED</status>

<reportUrl><![CDATA[http://localhost:1505/index.html?lisaPortal=
reporting/printPreview_functional.html#Idstr=61653261643936342D
613636392D3435&curtstr=T]]></reportUrl>
    <runId>61653261643936342D613636392D3435</runId>
    <pass count="1" />
    <fail count="0" />
    <warning count="0" />
    <error count="0" />
    <message>AccountControlMDB,Run1User1Cycle0Think</message>
  </result>
</invokeResult>
```

例: 非同期呼び出し

以下の URL では、**examples** プロジェクトの **AccountControlMDB** テストケースの非同期呼び出しを実行します。

```
http://localhost:1505/lisa-invoke/runTest?testCasePath=examples
/Tests/AccountControlMDB.tst&stagingDocPath=examples/StagingDoc
s/1user1cycle0think.stg&async=true
```

以下の XML 応答は、結果エレメント内のコールバック キーを示しています。

```
<?xml version="1.0" encoding="UTF-8"?>
<invokeResult>
  <method name="RunTest">
    <params>
      <param name="stagingDocPath"
value="examples/StagingDocs/1user1cycle0think.stg" />
      <param name="coordName" value="" />
      <param name="configPath" value="" />
```

```
      <param name="testCasePath"
value="examples/Tests/AccountControlMDB.tst" />
      <param name="callbackKey"
value="61663038653562382D663566372D3432" />
      <param name="async" value="true" />
    </params>
  </method>
  <status>OK</status>
  <result>
    <callbackKey>61663038653562382D663566372D3432</callbackKey>
    <message>The LISA test
'examples/Tests/AccountControlMDB.tst' was launched
asynchronously at Mon Mar 26 16:05:39 PDT 2012.</message>
  </result>
</invokeResult>
```

LISA Invoke によるテスト スイートの実行

LISA Invoke を使用してテスト スイートを実行する構文は以下のとおりです。

```
/lisa-invoke/runSuite?suitePath=suitePath&[configPath=configPath]&[async=true
]
```

パラメータは以下のとおりです。

suitePath

呼び出すスイートのパス。

configPath

設定のパス。

async

true の場合、応答にはコールバック キーが含まれます。 指定しない場合、このパラメータは false に設定されます。

LISA Invoke によるモデル アーカイブの実行

LISA Invoke を使用してモデル アーカイブ（MAR）を実行する構文は以下のとおりです。

```
/lisa-invoke/runMar?marOrMariPath=[marOrMariPath]&[async=true]
```

パラメータは以下のとおりです。

marOrMariPath

呼び出す MAR または MAR 情報ファイルのパス。

async

true の場合、応答にはコールバック キーが含まれます。 指定しない場合、このパラメータは false に設定されます。

LISA Invoke によるコールバック サービスの呼び出し

LISA Invoke でコールバック サービスを使用するには、テスト ケースまたはスイートの非同期呼び出しを実行済みである必要があります。 XML 応答には、コールバック キーが含まれています。

コールバック サービスを呼び出すための構文は以下のとおりです。

```
/lisa-invoke/callback?testOrSuitePath=[testOrSuitePath]&callbackKey=[callback
Key]&command=[status|kill|stop]
```

パラメータは以下のとおりです。

testOrSuitePath

テスト ケースまたはスイートのパス。

callbackKey

非同期呼び出しに対する応答に含まれていたコールバック キー。

command

実行するアクション： status、kill、または stop。

LISA Invoke の応答

このセクションでは、LISA Invoke から返される XML ドキュメントに表示されるエレメントについて説明します。

method エレメントは、どのタイプの実行が行われたかを示します。

status エレメントには、実行のステータス（OK または ERROR）が含まれます。

result エレメントには、以下の子エレメントが 1 つ以上含まれます。

status

テスト ケースのステータス（RUNNING または ENDED）。

reportURL

レポート コンソール内のレポートの URL。

runId

実行の一意の識別子。

pass

成功したテストの数。

fail

失敗したテストの数。

warning

警告が発生したテストの数。

error

エラーが発生したテストの数。

message

実行のタイプに固有の情報。

tc

スイートに含まれているテスト ケースの名前。

callbackKey

テスト ケースまたはスイートに対して追加アクションを実行するために使用できる文字列。

REST API

REST API では、DevTest タスクを実行するために REST アーキテクチャ スタイルを使用できます。

API は以下のカテゴリに分類されます。

- コーディネータ サーバ
- シミュレータ サーバ
- ラボ
- VSE サーバ

完全な API ドキュメントはここから入手できます。API ドキュメントには、さまざまな RESTful サービスと対話する方法の詳細が記載されています。

注: API ドキュメントにアクセスするには、Google Chrome または Mozilla Firefox を使用する必要があります。

Web インターフェースからの REST API の呼び出し

DevTest Solutions には、REST API を確認するために使用できるシンプルな Web インターフェースが含まれています。

注: Web インターフェースにアクセスするには、Google Chrome、Internet Explorer、または Mozilla Firefox を使用する必要があります。

デフォルトでは、Web インターフェースは URL に **localhost** を使用します。**localhost** 以外から Web インターフェースにアクセスするには、**local.properties** ファイルに以下のプロパティを設定します。

```
lisa.invoke2.swagger.basepath=http://<public_hostname_or_ip_address>:1505/api/Dcm
```

次の手順に従ってください:

1. レジストリが実行されていることを確認します。
2. サポートされているブラウザに **http://localhost:1505/api/swagger/** を入力します。 レジストリがリモート コンピュータ上にある場合は、**localhost** をそのコンピュータの名前または IP アドレスに置き換えます。

   Web インターフェースが表示されます。
3. カテゴリの［Show/Hide］をクリックします。
4. API のパスの 1 つをクリックします。
5. ［Response Content Type］フィールドを使用して、応答が XML 形式か、または JSON 形式かを指定します。
6. ［Parameters］セクションが表示される場合は、各パラメータに必要な値を入力します。
7. ［Try it out!］をクリックします。

   要求 URL および応答が表示されます。

HP ALM - Quality Center プラグインの使用

HP ALM - Quality Center プラグインを使用すると、HP ALM - Quality Center スイートから Quality Center テストとして DevTest テスト ケースをロードおよび実行できます。 DevTest テストを Quality Center にインポートして、実行できます。 この統合によって、DevTest テストを活用しながら、すべての Quality Center 機能を利用することができます。Quality Center に DevTest テスト ケースをロードすることによって、DevTest テストをリアルタイムで実行できます。また、テスト中のシステムからテスト結果の完全なキャプチャおよび DevTest コールバックも取得します。 DevTest テストは Quality Center のワークフロー内で実行可能で、テスト プロセスのコンテキストおよびステータスを保持するために結果をレポートします。

注: HP ALM - Quality Center のインストールについては、「*インストール*」を参照してください。

このセクションには、以下のトピックが含まれます。

HP ALM - Quality Center での DevTest テストの設定

このプラグインでは、HP ALM - Quality Center と統合するために VAPI-XP を使用しています。

この手順で参照する JavaScript および VBScript のテンプレートを表示するには、［スタート］－［すべてのプログラム］－［DevTest］－［Quality Center Plugin］の順にクリックしてください。 テンプレートの機能は同等です。

DevTest テストには、テスト ケース ファイル、MAR ファイル、MAR 情報ファイルを指定できます。

テスト ケースの場合は、ステージング ドキュメントや設定ファイルを添付することもできます。 ステージング ドキュメントを添付しないと、ステージング ドキュメントが以下の特性で自動的に作成されます。

- 1 ユーザ
- 1 サイクル
- 反応時間ゼロ

MAR ファイルおよび MAR 情報ファイルの場合は、追加ファイルを添付できません。

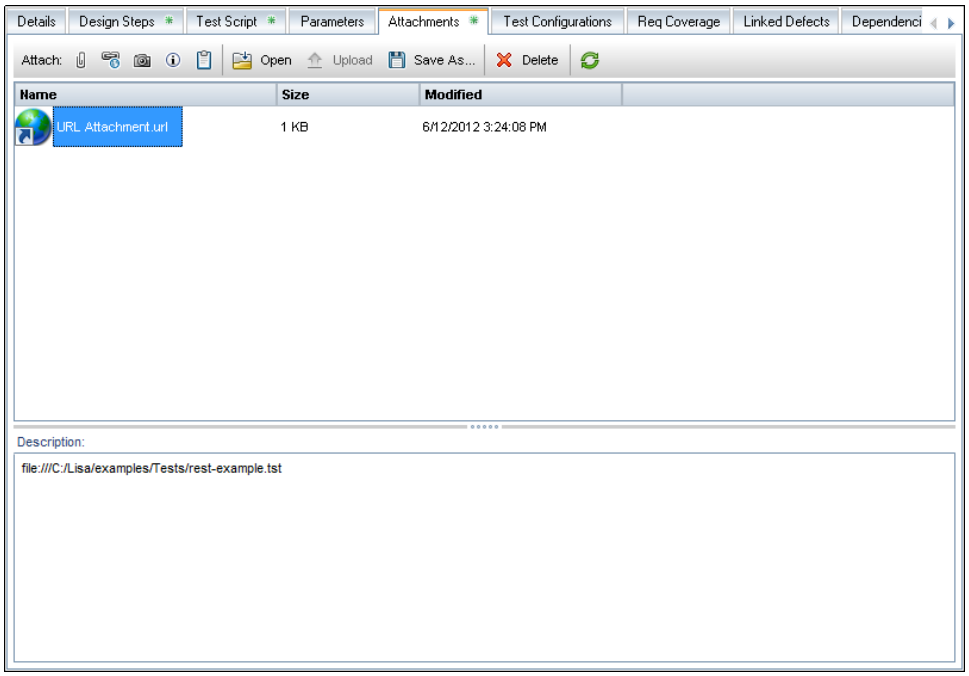
この手順では、1 つ以上の URL 添付ファイルを作成します。 テスト ケース ファイルの URL の例を以下に示します。

```
file:///C:/Lisa/examples/Tests/rest-example.tst
```

MAR 情報ファイルの URL の例を以下に示します。

```
file:///C:/Lisa/examples/MARInfos/rest-example.mari
```

以下の図は、［Attachments］タブを示しています。 テスト ケース ファイルの URL が追加されています。



次の手順に従ってください:

1. Quality Center で VAPI-XP テストを作成します。
2. ［Test Script］タブを選択し、デフォルトの内容をプラグインに含まれている JavaScript または VBScript テンプレートの内容に置き換えます。
3. ［Attachments］タブを選択し、テスト ケース ファイル、MAR ファイル、または MAR 情報ファイルの URL を追加します。
4. （オプション）ステージング ドキュメントおよび設定ファイルの URL を追加します。これらのファイルはテスト ケースの場合は追加できますが、MAR ファイルまたは MAR 情報ファイルには追加できません。
5. VAPI-XP テストを保存します。

HP ALM - Quality Center での DevTest テストの実行

HP ALM - Quality Center で DevTest テストを設定したら、Test Plan モジュールまたは Test Lab モジュールからテストを実行できます。

テスト ケースの場合

- コーディネータが実行されている場合は、テスト ケースを実行するために使用されます。
- コーディネータが実行されていない場合は、テスト ケースをローカルに実行するためにテスト ランナー ユーティリティが使用されます。

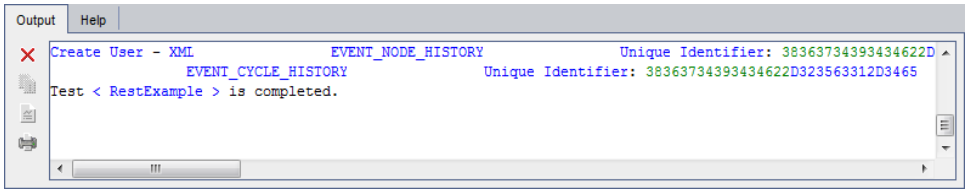
MAR ファイルおよび MAR 情報ファイルの場合

- ファイルでコーディネータが指定されている場合は、コーディネータが実行されている必要があります。
- ファイルでコーディネータが指定されていない場合は、プラグインによってローカルのコーディネータが作成および起動されます。

デフォルトでは、各テスト ランで実行するために Reload コマンドが設定されます。 このコマンドでは、テスト ランを成功させるために必要な、テストのための設計ステップを入力します。 テストが実行される前に、テストに対するすべての変更が更新されます。テストによっては、再ロード プロセスに時間がかかる場合があります。ベースとなる DevTest テスト ファイルが変更されていないことがわかっている場合は、スクリプト ファイルでその行をコメント アウトして、ステップがスキップされるようにすることができます。

Test Plan モジュールからテストを実行するには、［Test Script］タブで緑の矢印をクリックします。 出力ウィンドウに出力が表示されます。

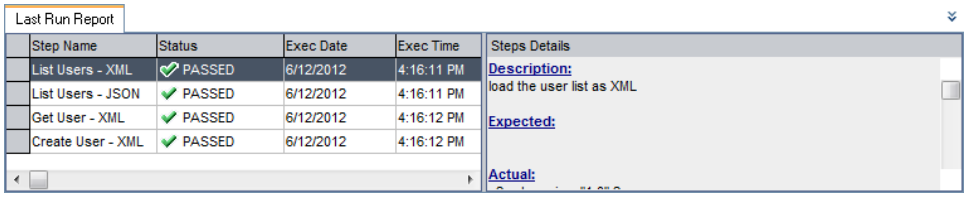
以下の図は、Test Plan での **rest-example** テスト ケースの実行結果を示しています。



Test Lab モジュールからテストを実行するには、テスト セットにテストを追加します。 そこから、単一のテストまたはテスト セット全体を実行できます。

テストの構造によって、テスト ランからは異なる結果が導き出されます。テストに実行されるサイクルが複数ある場合、サイクルの履歴の結果と、成功または失敗のステータスが表示されます。テストのサイクルが 1 つだけの場合は、そのテストのステップのリストが表示されます。

以下の図は、Test Lab での **rest-example** テスト ケースの実行結果を示しています。



LisaQCRunner コマンドライン インターフェース

LISA_HOME¥bin ディレクトリにある **LisaQCRunner** 実行可能ファイルを使用すると、Quality Center の DevTest テストをコマンドラインから実行できます。 結果は、Quality Center のデータベースに保存できます。

この実行可能ファイルの形式は以下のとおりです。

```
LisaQCRunner [-h ホスト] [-P ポート] [-u ユーザ] [-p パスワード] [-D ドメイン] [-l プロジェクト] run|debug|reload テスト名|all
```

デフォルトのホストは localhost です。 デフォルトのポートは 8080 です。 デフォルトのユーザは admin です。 デフォルトのパスワードは admin です。 デフォルトのドメインは DEFAULT です。 デフォルトのプロジェクトは Test です。

プロジェクトの引数は、DevTest プロジェクトではなく、Quality Center プロジェクトです。

run

引数として指定した名前のテストを実行します。 コマンド ウィンドウに出力が表示されます。 結果は Quality Center のデータベースに保存されます。

debug

引数として指定した名前のテストを実行します。 コマンド ウィンドウに出力が表示されます。 結果は Quality Center のデータベースに保存されません。

reload

引数として指定した名前のテスト、またはすべての DevTest テスト（引数が all の場合）を再ロードします。

以下の例は、**rest-example** テスト ケースの実行結果を示しています。

```
LisaQCRunner -h machine.example.com -P 8080 -u admin -p mypassword -D DEFAULT -l
myproject run RunWithTestandStage

Connecting...
接続済み
Running RunWithTestAndStage with the following parameters:
Test Doc: file:///c:/lisa/examples/tests/rest-example.tst
Staging Doc: file:///c:/lisa/examples/stagingdocs/1user1cycle0think.stg
Config:
Mar:
```

```
List Users - XML     EVENT_NODE_HISTORY     Unique Identifier:
63366561643365642D643834302D3461 ...
List Users - JSON     EVENT_NODE_HISTORY     Unique Identifier:
63366561643365642D643834302D3461 ...
Get User - XML     EVENT_NODE_HISTORY     Unique Identifier:
63366561643365642D643834302D3461 ...
Create User - XML     EVENT_NODE_HISTORY     Unique Identifier:
63366561643365642D643834302D3461 ...
Disconnecting...
Disconnected
```

HP ALM - Quality Center プラグインのトラブルシューティング

パフォーマンスを向上させるために、DevTest ブリッジは常に DevTest COM サーバへの参照を維持しています。 したがって、各 API コールに対してサーバはインスタンス化されません。 ブリッジをホストするプロセスが終了すると、ホストが Web ブラウザのようなネイティブ アプリケーションでない場合、この参照は解放されます。そのため、**LisaQCRunner.exe** プロセスは引き続き有効です。何かが望ましくない状態になったとき（たとえば、突然の終了などが原因の場合）、唯一の問題はこの動作です。その場合は、操作を続行する前に、残っている **LisaQCRunner.exe** プロセスを手動で終了することを検討してください。

テスト ディレクタのテスト ケースの実行の失敗

1. ［添付］タブで URL を確認し、PATH およびメンバ名が正しいことを確認します。 テスト ディレクタのログ ファイルには、メンバが見つからなかったことを示す例外が含まれている場合があります。
2. より複雑な問題については、DevTest ワークステーション を起動し、テストに移動して ITR モードでテストを実行します。 テスト ケースを失敗させる例外を監視します。
3. XML 応答および WSDL が変更されていないことを確認します。DevTest テスト ケースは、XPATH コマンドを使用して XML から特定の値を検索します。 XML が変更されている場合、XPATH コマンド（フィルタおよびアサーション）は指定されたターゲットを検索していない場合があり、テスト ケースが失敗します。

テスト ディレクタのテスト ケース - 権限の拒否

テストを実行しようとして、「このアクションを実行するために必要な権限がありません」という意味のエラー メッセージが表示された場合、以下を確認します。

1. HPQC 管理者からテストを実行する ID 権限が付与されている。
2. テストの実行元のコンピュータに DevTest ワークステーション がインストールされている。 DevTest ワークステーション がインストールされている場合は、Quality Center プラグインもインストールされていることを確認します。
3. ブラウザ バージョンが HPQC 認定バージョンと互換性がある。
4. HPQC の .DLL が C:¥Program Files¥Common¥Mercury Interactive¥Quality Center フォルダに正しくインストールされている。

5. QC Test Plan および QC Test Lab の両方からテスト ケースを実行して、エラー結果が同一であることを確認する。

HTTP および SSL デバッグ ビューア

HTTP および SSL デバッグ ビューアでは、DevTest ワークステーション 内の HTTP および SSL アクティビティの詳細を確認できます。 この機能は、診断を行う場合に役立ちます。

このビューアにアクセスするには、メイン メニューから［ヘルプ］-［HTTP/SSL デバッグ］を選択します。

左側の縦棒は、各行のカテゴリを示しています。

- 緑は HTTP 要求に使用されます。
- 紺は HTTP 応答に使用されます。
- 斜線のストライプ模様は SSL に使用されます。 SSL ハンドシェイク サマリに対してはさらに紫のバーが表示されます。

行は以下のように色分けされています。

- 緑は HTTP 要求ヘッダに使用されます。
- 紺は HTTP 応答ヘッダに使用されます。
- 黒は通常の出力に使用されます。
- 灰色は重要でない出力に使用されます。
- 青緑は関心を引く必要のある出力に使用されます。
- 赤紫は重要な出力に使用されます。
- 暗いオレンジは警告の出力に使用されます。
- 赤はエラーの出力に使用されます。
- 紫はサマリの出力に使用されます。

各行の先頭にあるかっこ内の数字はスレッド識別子です。 複数のスレッドが同じ接続に対して動作する場合があります。

ビューアでは、デバッグ ログ内のメッセージを解析して SSL 出力が作成されます。

SSL 出力には、ハンドシェイク プロセスのサマリ (P. 391)が含まれます。SSL の問題を診断する場合は、まずハンドシェイクのサマリを確認します。より詳細な情報が必要な場合は、サマリの前に表示される出力を確認します。ビューアには問題を解決するために必要なすべてのデータがあるとは限りません。

出力をコピーし、プレーン テキストとして別のウィンドウへ貼り付けることができます。 貼り付けたバージョンには、色分けは含まれません。

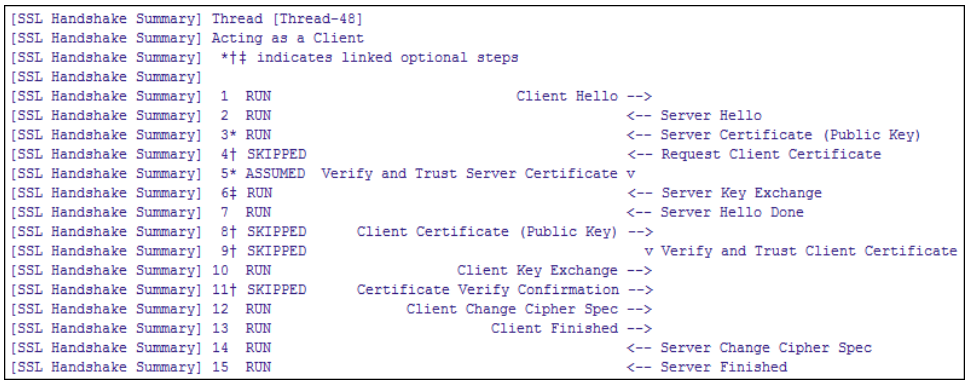
このビューアでは、すべての行、HTTP の行のみ、または SSL の行のみのいずれを表示するかを指定できます。

SSL ハンドシェイク サマリ

HTTP および SSL デバッグ ビューア (P. 389)の SSL 出力には、ハンドシェイク プロセス中に発生したイベントのサマリが含まれます。SSL の問題を診断する場合は、まずハンドシェイクのサマリを確認します。

注: このトピックは、SSL またはその後継である TLS の基本的な知識があることを前提としています。

以下の図は、サマリの例を示しています。



最初の行にはスレッド名が表示されます。

2 行目には、ビューアが使用する SSL デバッグ ログがクライアントとサーバのどちらとして機能しているかが示されます。 セッションが再開されると、2 行目には対応するメッセージも表示されます。

残りの行には、ハンドシェイク プロセスのステップが表示されます。

ハンドシェイク プロトコルのオプションのステップも含め、すべての表示可能なステップがサマリに表示されます。 オプションのステップでは、記号がステップ番号の右側に表示されます。 互いに関連するオプションのステップには、それぞれ異なる記号が使用されます。 たとえば、ステップ 3 とステップ 5 にはアスタリスクが使用されており、どちらもサーバ証明書に関連しています。

ステップはそれぞれ、以下のいずれかのステータスを持ちます。

- RUN： ステップは実行されています。
- SKIPPED： ステップは実行されていません。
- ASSUMED： ステップは、発生したその他のイベントに基づいて実行されたと想定されています。

- UNKNOWN： すべてのステップの初期ステータス。 ステータスが変更されなかった場合に最も考えられるのは、ステップが実行されていない場合です。

各ステップには、クライアントまたはサーバが実行したアクションの簡単な説明が含まれます。 たとえば、最初のステップでは、クライアントがサーバに Hello メッセージを送信したことが示されています。 アクションにメッセージの送信が含まれている場合、左矢印または右矢印はメッセージ フローの方向を示します。 アクションにメッセージの送信が含まれていない場合は、下向きの矢印が表示されます。

SSL で問題が発生した場合、サマリは原因の特定に役立つガイドとなります。 以下の例は、テスト ステップが 非 SSL ポートに HTTPS 要求を行った場合に表示される出力を示しています。

```
SEND TLSv1 ALERT:  fatal, description = handshake_failure
javax.net.ssl.SSLHandshakeException: Remote host closed connection during handshake
```

サーバがセキュア（セキュアでないサーバに SSL を通して接続）であり、正しいポートに接続していることを確認してください

第 14 章: クラウドの DevTest ラボ

クラウドベースのインフラストラクチャを使用して、開発環境やテスト環境をプロビジョニングできます。

このセクションには、以下のトピックが含まれています。

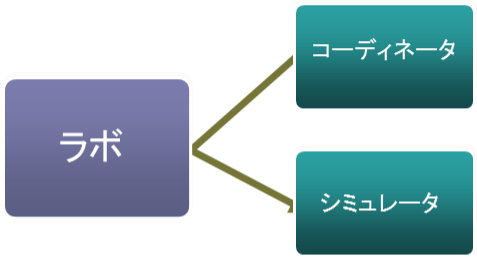
ラボとラボ メンバ

*ラボ*は、1 つ以上のラボ メンバの論理コンテナです。

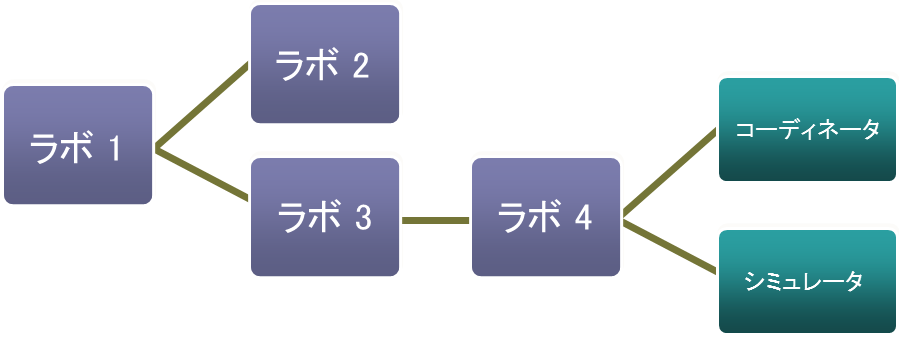
*ラボ メンバ*になれるのは、DevTest サーバまたは非 DevTest サーバです。

- DevTest サーバの有効なタイプは、コーディネータ、シミュレータ、および仮想サービス環境です。
- 非 DevTest サーバの例としては、データベースや Web サーバなどがあります。

以下の図は、2 つのメンバ（コーディネータとシミュレータ）を持つラボを示しています。



ラボは 1 つ以上の子ラボを持つことができます。以下の図は、ラボの階層の特性を示しています。 ラボ 1 は ラボ 2 および ラボ 3 の親です。 ラボ 3 は ラボ 4 の親です。



子ラボの完全修飾名では、区切り文字としてスラッシュを使用します。たとえば、上記の図の ラボ 4 の完全修飾名は ラボ 1/ラボ 3/ラボ 4 です。

DevTest には、**Default** という名前のラボがあります。 コーディネータ、シミュレータ、または仮想サービス環境をラボを指定しないで起動した場合は、Default ラボが使用されます。

ラボ メンバは、以下のいずれかのステータスになります。

- 未初期化
- 起動中
- 実行中
- 不明

ラボ メンバが DevTest サーバである場合、ステータスは［未初期化］から［起動中］、そして［実行中］へと移行します。

ラボ メンバが非 DevTest サーバである場合、ステータスは、［未初期化］から［実行中］に直接移行します。

仮想ラボ マネージャ（VLM）

ラボ (P. 394)が実行されるクラウドベースの環境は、仮想ラボ マネージャ（VLM） と呼ばれます。

以下の VLM プロバイダがサポートされています。

- VMware vCloud Director 1.0 および 1.5

VLM プロバイダにはそれぞれ一意のプレフィックスがあります。 以下の表に、プレフィックスのリストを示します。

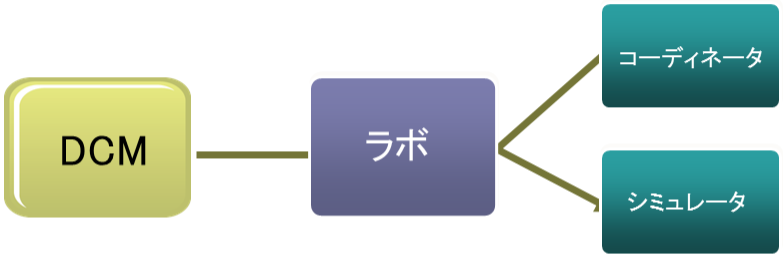
VLM プロバイダ	プレフィックス
VMware vCloud Director	VCD

DevTest ワークステーション では、使用されている VLM プロバイダを示すために、完全修飾ラボ名の先頭にプレフィックスとコロンが付加されます。たとえば、以下のようになります。

VCD:MyOrganization/MyCatalog/MyLab

DevTest Cloud Manager

DevTest Cloud Manager （DCM）は、ラボ (P. 394)と通信するための DevTest Solutions コンポーネントです。

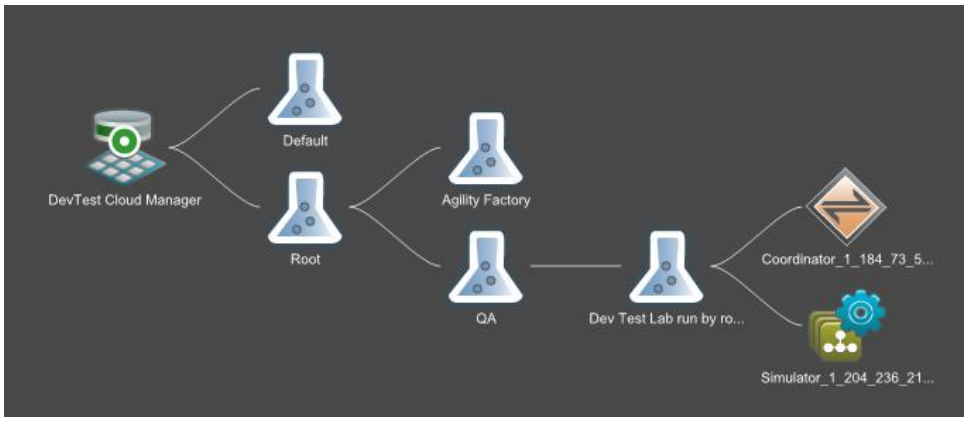


DCM には、以下のような役割があります。

- クラウド関連のプロパティをラボに送信する
- ラボを開始する必要があることを仮想ラボ マネージャ（VLM）プロバイダに通知する
- モデル アーカイブ （MAR） (P. 300) をラボに送信する

DCM を使用すると、サーバ コンソールからラボを表示、開始、モニタ、シャットダウンすることができます。

サーバ コンソールには、ネットワーク グラフの一部として DCM が表示されます。 以下の図は、DCM が Default ラボおよび Root という名前のラボと通信する様子を示しています。 Root ラボには、Agility Factory と QA という 2 つの子ラボがあります。 QA ラボには、Dev Test Lab という名前の子ラボがあります。 Dev Test Lab の完全修飾名は、Root/QA/Dev Test Lab です。



DCM プロパティの設定

開発ラボおよびテスト ラボのプロビジョニングを行えるようにするには、レジストリがあるコンピュータでプロパティを設定します。

一部のプロパティは、すべての仮想ラボ マネージャ（VLM）プロバイダに適用できます。 その他のプロパティは、VLM プロバイダに固有です。

- 一般的なプロパティ（P. 399）
- vCloud Director のプロパティ（P. 401）
- DCM のプロパティの例（P. 402）

DCM の一般的なプロパティ

このセクションでは、すべての VLM プロバイダに適用されるプロパティについて説明します。

以下のプロパティは、**local.properties** ファイルで設定する必要があります。

- lisa.net.bindToAddress

以下のプロパティは、**site.properties** ファイルで設定する必要があります。

- lisa.dcm.labstartup.min
- lisa.dcm.lisastartup.min
- lisa.dcm.lisashutdown.min
- lisa.dcm.lab.cache.sec
- lisa.dcm.lab.factories
- lisa.net.timeout.ms

lisa.net.bindToAddress

レジストリがあるコンピュータの完全修飾ドメイン名または IP アドレス。 このドメイン名または IP アドレスは、レジストリが存在するネットワークの外部のネットワークに接続しているすべてのコンピュータが解決できる必要があります。

構文：

```
lisa.net.bindToAddress=<完全修飾ドメイン名または IP アドレス>
```

lisa.dcm.labstartup.min

VLM プロバイダがラボを開始する際に DevTest が待機する時間（分）。

構文：

```
lisa.dcm.labstartup.min=<整数>
```

lisa.dcm.lisastartup.min

VLM プロバイダが DevTest 用にラボを開始した後、正しく初期化されるまで DevTest が待機する時間（分）。

構文：

```
lisa.dcm.lisastartup.min=<整数>
```

lisa.dcm.lisashutdown.min

テストの実行が完了した後、ラボがシャットダウンされるまで DevTest が待機する時間（分）。

構文：

```
lisa.dcm.lisashutdown.min=<整数>
```

lisa.dcm.lab.cache.sec

DevTest では、VLM プロジェクト設定のキャッシュを保持します。 このプロパティは、キャッシュを更新する必要があるかどうか（たとえば、環境が追加された可能性があるかどうか）を確認するために、DevTest が VLM プロバイダにアクセスする頻度を指定します。 値は秒単位です。

構文：

```
lisa.dcm.lab.cache.sec=<整数>
```

デフォルト： 180

lisa.dcm.lab.factories

特定の VLM プロバイダ用のクラウド サポート ロジックが含まれるオブジェクトの名前。 有効な値は、**com.itko.lisa.cloud.vCloud.vCloudDirectorCloudSupport** のみです。

構文：

```
lisa.dcm.lab.factories=<オブジェクト名>
```

lisa.net.timeout.ms

基盤となるメッセージング システムによって使用されるタイムアウト値（ミリ秒）。 このプロパティは、クラウドに固有のものではありません。一部の操作にデフォルトより長く時間がかかる場合があるため、クラウド統合に対する値を変更します。

構文：

```
lisa.net.timeout.ms=<整数>
```

デフォルト： 30

vCloud Director のプロパティ

このセクションでは、VMware vCloud Director に固有のプロパティについて説明します。

以下のプロパティを **site.properties** ファイルで設定します。

- lisa.dcm.vCLOUD.baseUri

また、以下のプロパティも設定します。

- lisa.dcm.vCLOUD.userId
- lisa.dcm.vCLOUD.password

DevTest はカスタム権限の **lisa.dcm.vCLOUD.userId** と **lisa.dcm.vCLOUD.password** を作成し、これらを各ロールに割り当てます。これらのカスタム権限の初期値を指定するには、**site.properties** ファイルで **lisa.dcm.vCLOUD.userId** プロパティと **lisa.dcm.vCLOUD.password** プロパティを設定します。 初期値を指定しない場合は、サーバ コンソールから値を設定します。 「管理ガイド」の「ロールの追加、更新、および削除」を参照してください。

lisa.dcm.vCLOUD.baseUri

vCloud API のログイン URL。

構文：

```
lisa.dcm.vCLOUD.baseUri=<URL>
```

lisa.dcm.vCLOUD.userId

vCloud Director にログインできるユーザ名。 通常、この形式は、ユーザ名の後に @ （アンパサンド）と組織名を付加します。

構文：

```
lisa.dcm.vCLOUD.userId=<ユーザ名>@<組織名>
```

lisa.dcm.vCLOUD.password

lisa.dcm.vCLOUD.userId プロパティで定義したユーザ名のパスワード。

構文：

```
lisa.dcm.vCLOUD.password=<パスワード>
```

DCM のプロパティの例

以下の例では、vCloud Director ベースの設定を示します。

以下のプロパティは、**local.properties** ファイル内にあります。

```
lisa.net.bindToAddress=myserver.example.com
```

以下のプロパティは、**site.properties** ファイル内にあります。

```
lisa.dcm.labstartup.min=6
lisa.dcm.lisastartup.min=4
lisa.dcm.lisashutdown.min=2
lisa.dcm.lab.cache.sec=240
lisa.dcm.lab.factories=com.itko.lisa.cloud.vCloud.vCloudDirectorCloudSupport

lisa.dcm.vCLOUD.baseUri=https://www.example.com/api/versions
lisa.dcm.vCLOUD.userId=administrator@System
lisa.dcm.vCLOUD.password=mypassword

lisa.net.timeout.ms=60000
```

vCloud Director の設定

VMware vCloud Director を仮想ラボ マネージャ（VLM）プロバイダとして使用して、開発ラボやテスト ラボのプロビジョニングを行うには、vCloud Director の設定手順を実行します。

このトピックでは、以下の vCloud Director の概念を使用します。

- カタログ
- vApp
- vApp テンプレート

vCloud Director の vApp および vApp テンプレートは DevTest のラボに相当します。

設定手順の中で、以下の DevTest サーバ コンポーネントの仮想マシン イメージを作成します。

- コーディネータ
- シミュレータ
- 仮想サービス環境

ラボにコーディネータとシミュレータのみが含まれるようにする場合は、VSE のイメージを作成する必要はありません。

ラボに仮想サービス環境のみが含まれるようにする場合は、コーディネータとシミュレータのイメージを作成する必要はありません。

各サーバ コンポーネントは、**remoteInit** モードで起動されるよう設定する必要があります。このモードでは、サーバ コンポーネントを起動した後、DevTest Cloud Manager （DCM）が必要な初期化設定を送信するまで待機します。

vCloud Director を設定する方法

1. VMware Workstation などのアプリケーションを使用して、ラボに含める各サーバ コンポーネントの仮想マシン イメージを作成します。
   - サーバ コンポーネントを **remoteInit** モードで起動するために、各イメージを設定します。
   - コーディネータの名前には **Coordinator** という単語が含まれる必要があります。

- シミュレータの名前には **Simulator** という単語が含まれる必要があります。
- 仮想サービス環境の名前には **VSE** という単語が含まれる必要があります。

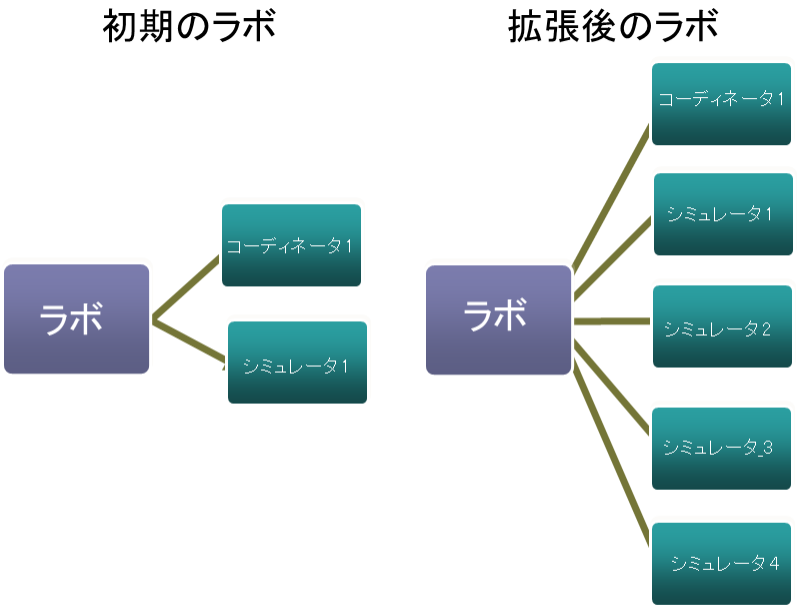
2. vCloud Director Web コンソールに管理者としてログインします。
3. vApp テンプレートとして作成した各仮想マシン イメージをインポートします。
4. vApp テンプレートから vApp を作成します。
5. vApp をカタログに追加します。

テスト ラボの動的な拡張

DevTest では、テストの実行に必要な追加容量を判断し、自動的にラボを拡張することができます。

DCM 配分パターンによる動的シミュレータ スケーリング機能を持つステージング ドキュメントを使用します。詳細については、「配分の選択 (P. 262)」を参照してください。

以下の図に例を示します。 左側の部分は、1 台のコーディネータと 1 台のシミュレータがあるラボの初期状態です。 右側の部分は、拡張が行われた後、1 台のコーディネータと 4 台のシミュレータがあるラボの状態です。



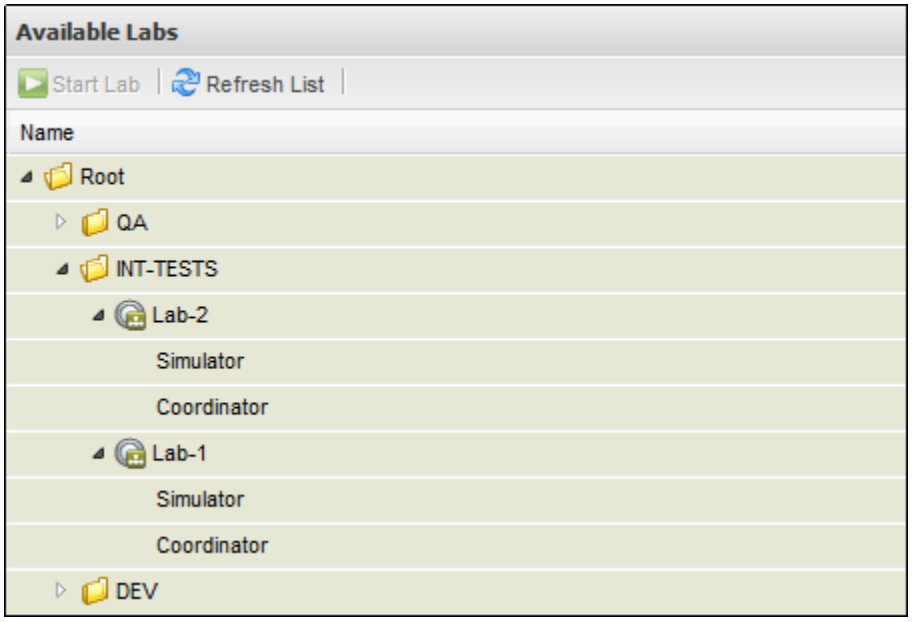
利用可能なラボのリストの表示

サーバ コンソールまたは LISA Invoke を使用すると、クラウド環境で利用できる開発ラボやテスト ラボのリストを表示できます。

サーバ コンソールからの利用可能なラボのリスト表示

サーバ コンソールにアクセスする手順については、「*管理*」の「サーバ コンソールを開く」を参照してください。

以下の図は、サーバ コンソールでの利用可能なラボのリストを示しています。 2 つの子ラボを表示するためにツリー構造が展開されています。



利用可能なラボのリストを表示する方法

1. サーバ コンソールで、［ネットワーク］パネルを表示します。
2. ［DevTest Cloud Manager］ノードを展開します。
3. ［利用可能なラボ］ノードをクリックします。

   右側のパネルに利用可能なラボが表示されます。 ツリー構造は、最初は折りたたまれています。

LISA Invoke による利用可能なラボのリストの表示

LISA Invoke (P. 372) を使用して利用可能なラボのリストを表示できます。構文は以下のとおりです。

```
/lisa-invoke/listRunnableLabs
```

応答は XML ドキュメントです。 各ラボについて、以下の情報が提供されます。

- ラボの名前
- ラボ キー
- 各ラボ メンバの名前

ラボ キーとは、**startLab**、**getLab**、および **killLab** 操作で必要となるパラメータです。

例

以下の URL は、利用可能なラボのリストを要求します。

```
http://localhost:1505/lisa-invoke/listRunnableLabs
```

以下の応答には、利用可能な 3 つのラボのリストが含まれています。

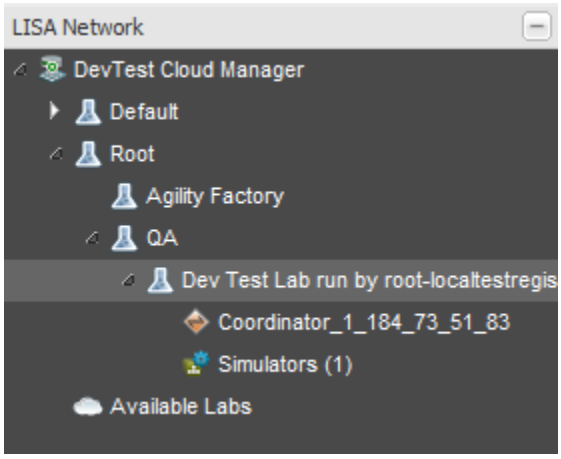
```
<?xml version="1.0" encoding="UTF-8" ?>
<invokeResult>
  <method name="ListRunnableLabs">
    <params />
  </method>
  <status>OK</status>
  <result>
    <lab name="MM-Test" key="VCD:7">
      <labMember name="Simulator" />
      <labMember name="Coordinator" />
    </lab>
    <lab name="MM-Test run by rich-47" key="VCD:47">
      <labMember name="Simulator_1_184_72_204_206" />
      <labMember name="Simulator_2_107_21_199_227" />
      <labMember name="Coordinator_1_184_73_83_115" />
    </lab>
    <lab name="VSE-Cluster run by rich-46" key="VCD:46">
      <labMember name="V_S_E_Member_1_23_21_11_148" />
      <labMember name="V_S_E_Member_2_23_21_3_45" />
      <labMember name="V_S_E_Member_4_184_73_65_217" />
      <labMember name="V_S_E_Member_3_174_129_88_179" />
    </lab>
  </result>
```

```
</invokeResult>
```

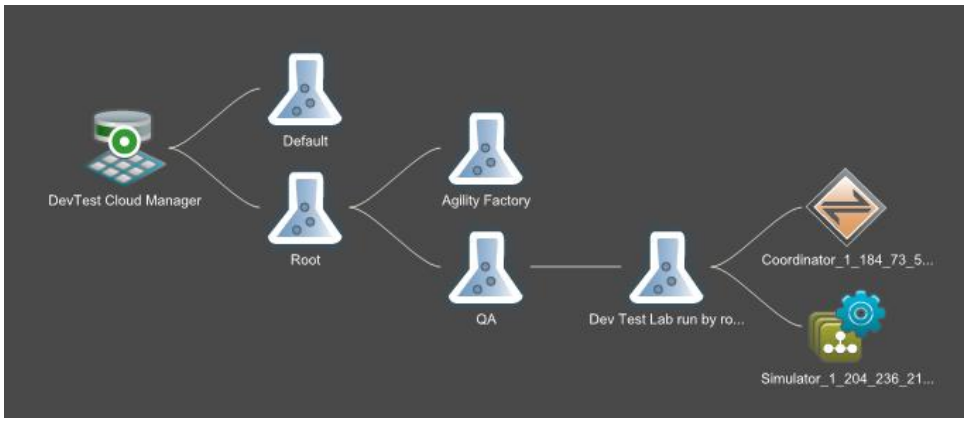
ラボの開始

クラウドベースのラボを開始する場合は、ラボのインスタンスを起動します。 1 つのラボが複数のインスタンスを持つことができます。それらはすべて互いに独立しています。

サーバ コンソールでは、開始されたラボが［ネットワーク］パネルにツリー構造で表示されます。 以下の図は、［ネットワーク］パネルを示しています。



右側のパネルでは、開始されたラボがネットワーク グラフに表示されます。 以下のグラフは、ネットワーク グラフを示しています。



lisa.dcm.labstartup.min プロパティは、仮想ラボ マネージャ（VLM）プロバイダがラボを開始するのを DevTest が待機する時間（分）を制御します。詳細については、「DCM プロパティの設定 (P. 398)」を参照してください。

注: Default ラボを明示的に開始する必要はありません。

ラボは、以下の方法で開始できます。

- コマンドラインから (P. 410)
- サーバ コンソールから (P. 410)
- LISA Invoke から (P. 411)

コマンドラインからのラボの開始

DevTest サーバ 実行可能ファイルのいずれかを呼び出し、サーバ名とラボ名を指定することでラボを開始できます。

たとえば、以下のコマンドは、MyLab という名前のラボを開始します。これは、MyParentLab の子です。MyLab ラボにはラボ メンバが 1 つあります。Dev-Coord という名前のコーディネータです。

```
CoordinatorServer -n Dev-Coord -l MyParentLab/MyLab
```

サーバ コンソールからのラボの開始

この手順では、サーバ コンソールで利用可能なラボのリストをすでに表示 (P. 406)していると仮定しています。

次の手順に従ってください:

1. 利用可能なラボのリストからラボを選択します。
2. ［ラボを開始］をクリックします。
3. ラボが開始されるのを待ちます。

   スタートアップ シーケンスが完了すると、ラボが開始されたことを示すメッセージが表示されます。

   ［ネットワーク］パネルには、開始されたラボがツリー構造で表示されます。右側のパネルでは、開始されたラボがネットワーク グラフに表示されます。
4. ラボ メンバが起動されるのを待ちます。

   ラボ メンバのスタートアップ シーケンスが完了すると、メンバのステータスが［実行中］に変わります。また、ラボ メンバに関する統計が［コンポーネント稼働状況サマリ］タブに表示され始めます。

LISA Invoke からのラボの開始

LISA Invoke (P. 372) を使用してラボを開始することができます。 構文は以下のとおりです。

```
/lisa-invoke/startLab?labKey=LAB:key
```

利用可能なラボ キーは、**listRunnableLabs** 操作に対する応答に含まれています。 詳細については、「利用可能なラボのリストの表示 (P. 406)」を参照してください。

応答は、ステータス メッセージが含まれる XML ドキュメントです。

例

以下の URL は、キーが **VCD:49** であるラボの開始を要求します。

```
http://localhost:1505/lisa-invoke/startLab?labKey=VCD:49
```

以下の応答は、ラボが開始されたことを示しています。

```
<?xml version="1.0" encoding="UTF-8" ?>
<invokeResult>
  <method name="StartLab">
    <params />
  </method>
  <status>OK</status>
  <result>
    <lab name="MM-Test run by rich-49" key="VCD:49" />
  </result>
</invokeResult>
```

ラボに関する情報の取得

LISA Invoke (P. 372) を使用して、実行中のラボに関する基本情報を取得することができます。 構文は以下のとおりです。

```
/lisa-invoke/getLab?labKey=LAB:key
```

利用可能なラボ キーは、**listRunnableLabs** 操作に対する応答に含まれています。 詳細については、「利用可能なラボのリストの表示 (P. 406)」を参照してください。

応答は XML ドキュメントです。 以下の情報を取得できます。

- ラボの名前
- ラボ キー
- 各ラボ メンバの名前、サービス名、および IP アドレス。

ラボが開始中の場合は、一部の情報はまだ取得できません。

ラボが見つからない場合は、応答にエラー メッセージが含まれます。

例

以下の URL は、キーが **VCD:50** であるラボの情報の取得を要求します。

```
http://localhost:1505/lisa-invoke/getLab?labKey=VCD:50
```

以下の応答には、ラボの情報が含まれています。

```
<?xml version="1.0" encoding="UTF-8" ?>
<invokeResult>
  <method name="GetLab">
    <params />
  </method>
  <status>OK</status>
  <result>
    <lab name="MM-Test run by rich-50" key="VCD:50">
      <labMember name="Coordinator_1_50_16_15_3"
serviceName="tcp://50.16.15.3:2011/Coordinator_1_50_16_15_3" ip="50.16.15.3"
/>
      <labMember name="Simulator_2_23_20_80_144"
serviceName="tcp://23.20.80.144:2014/Simulator_2_23_20_80_144"
ip="23.20.80.144" />
      <labMember name="Simulator_1_23_21_5_69"
serviceName="tcp://23.21.5.69:2014/Simulator_1_23_21_5_69" ip="23.21.5.69" />
```

```
    </lab>
  </result>
</invokeResult>
```

ラボへの MAR の展開

開始された (P. 409)ラボに対して、テスト ケースまたはスイートのモデル アーカイブ（MAR）を展開できます。 ラボには、コーディネータと（ほとんどの場合）シミュレータが含まれている必要があります。

次の手順に従ってください:

1. サーバ コンソールの［ネットワーク］パネルで、コーディネータをクリックします。

   コーディネータの詳細ウィンドウが右側のパネルに表示されます。

2. 右側のパネルで、［MAR の展開］をクリックします。

   ［MAR の展開］ダイアログ ボックスが表示されます。

3. ［参照］をクリックして .mar ファイルを選択します。

4. ［展開］をクリックします。

   モデル アーカイブが正常に展開されたことを示すメッセージが表示されます。 テスト ケースまたはスイートが実行されます。

5. ［OK］をクリックします。

ラボの停止

サーバ コンソールまたは LISA Invoke を使用して、現在実行中のラボを停止することができます。 この操作は永続的であり、元に戻すことはできません。

ラボが子ラボである場合、親ラボは停止されません。

注: Default ラボは削除できません。

サーバ コンソールからのラボの停止

ネットワーク グラフでラボを右クリックし、［停止］を選択します。

操作が完了すると、ラボが停止したことを示すメッセージが表示されます。

LISA Invoke からのラボの停止

LISA Invoke (P. 372) を使用してラボを停止することができます。 構文は以下のとおりです。

```
/lisa-invoke/killLab?labKey=LAB:key
```

利用可能なラボ キーは、**listRunnableLabs** 操作に対する応答に含まれています。 詳細については、「利用可能なラボのリストの表示 (P. 406)」を参照してください。

応答は、ステータス メッセージが含まれる XML ドキュメントです。

例

以下の URL は、キーが **VCD:49** であるラボの停止を要求します。

```
http://localhost:1505/lisa-invoke/killLab?labKey=VCD:49
```

以下の応答は、ラボが停止されたことを示しています。

```
<?xml version="1.0" encoding="UTF-8" ?>
<invokeResult>
  <method name="KillLab">
    <params />
```

```
  </method>
  <status>OK</status>
  <result>
    <message>Lab with key VCD:49 has been deleted.</message>
  </result>
</invokeResult>
```

第 15 章: 継続的検証サービス（CVS）

継続的検証サービス（CVS）では、長期間にわたって定期的に実行するテストおよびテスト スイートをスケジュールできます。

CVS には、スケジュールされたテスト（サービス）のリストやそれぞれのステータスを保守するためのダッシュボードがあります。 CVS ダッシュボードからテストを選択し、各テスト ランをモニタすることができます。

モニタには、単一のテストまたはテスト スイート全体が含まれます。1 つのサービスには 1 つ以上のモニタが含まれます。

コーディネータは、シミュレータで実行されるテストを管理します。 状態は、DevTest レジストリ上のデータベースで管理されます。

CVS ダッシュボードからは、その中のサービスまたは個々のモニタの実行を選択できます。

前提条件

CVS は DevTest サーバ 環境で実行されます。

DevTest レジストリを使用して登録および実行されているコーディネータ サーバおよびシミュレータ サーバが必要です。

DevTest サーバ 環境の設定の詳細については、「*インストール*」を参照してください。

このセクションには、以下のトピックが含まれています。

CVS ダッシュボードを開く

CVS ダッシュボードは、DevTest ワークステーション または Web ブラウザから開くことができます。

DevTest ワークステーション から CVS ダッシュボードを開く方法

メイン メニューから［表示］-［CVS ダッシュボード］を選択します。

Web ブラウザから CVS ダッシュボードを開く方法

1. レジストリ (P. 11) が実行されていることを確認します。
2. Web ブラウザに「**http://localhost:1505/**」と入力します。

   レジストリがリモート コンピュータ上にある場合は、**localhost** をそのコンピュータの名前または IP アドレスに置き換えます。

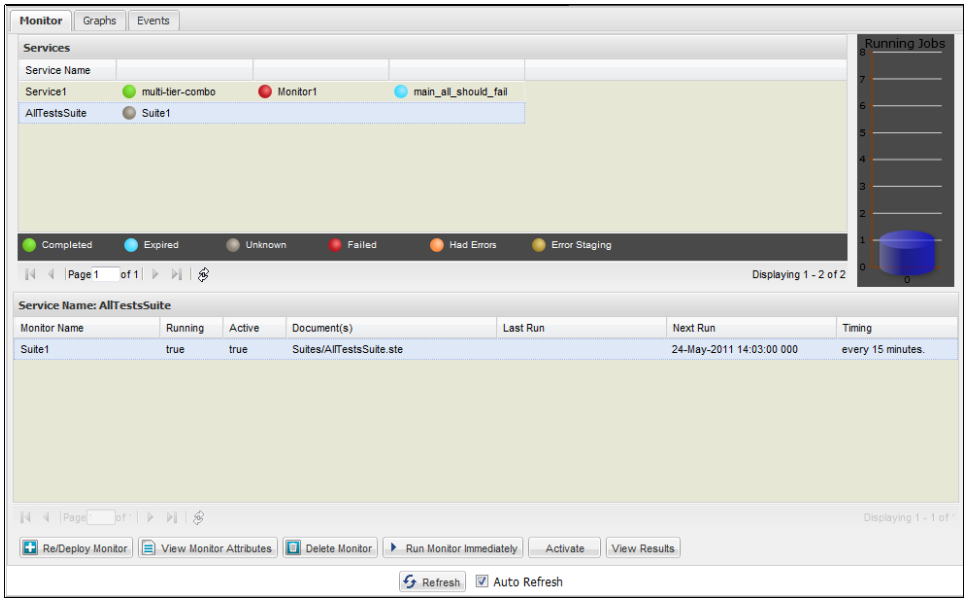
   DevTest コンソールが表示されます。
3. ［継続的検証サービス］をクリックします。

注: CVS ダッシュボードまたは DevTest ワークステーション を閉じることは、CVS のスケジュールされたタスクには影響しません。 レジストリに再接続すると、ダッシュボードに現在のデータおよびステータス情報が表示されます。

CVS ダッシュボードの概要

CVS ダッシュボードには、以下のタブがあります。

- モニタ（P. 420）：CVS ダッシュボードに追加されたすべてのモニタ（テストまたはテスト スイート）のリストが表示されます。 CVS ダッシュボードには、自分が開始したものだけでなく、接続している DevTest レジストリで実行されているモニタがすべてリストされます。
- グラフ（P. 424）： ダッシュボードのステータスがグラフィカルに表示されます。 このタブには、成功または失敗したテストのパーセンテージも表示されます。
- イベント（P. 426）： モニタが記録するステータス イベントが表示されます。



CVS ダッシュボードに表示されるリストをリフレッシュするには、ウィンドウの下部にある［リフレッシュ］をクリックします。

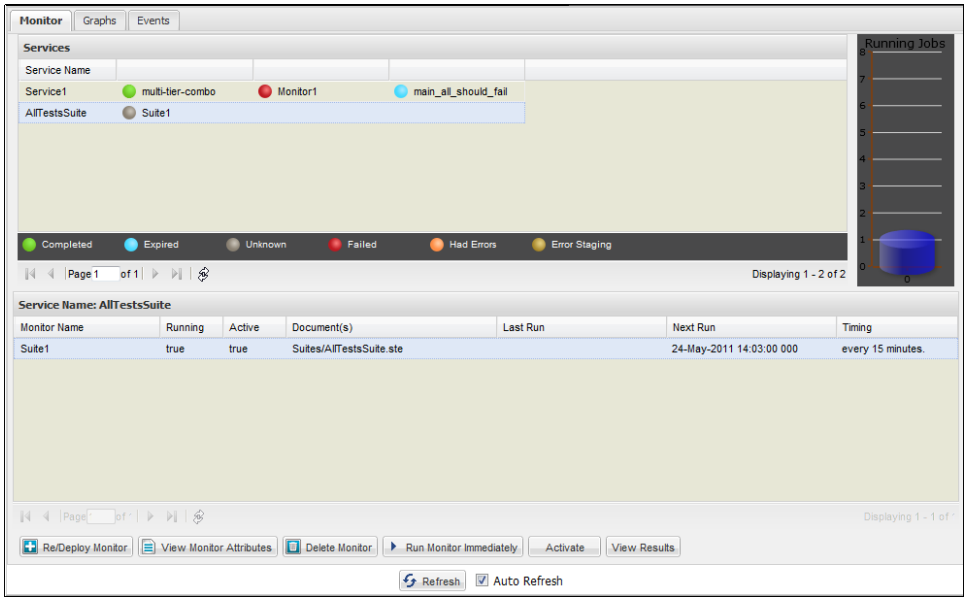
CVS ダッシュボード - モニタ

CVS ダッシュボードは、［モニタ］タブがアクティブな状態で表示されます。

1 つまたは多数のサービスで実行する多数のモニタを追加できます。 1 つのサービスには 1 つ以上のモニタが含まれます。

サービスの実行をスケジュールできます。 そのサービス内のモニタは、スケジュールされた時間にすべて実行されます。サービスは、スケジュールされた間隔でバックグラウンドで実行されます。

1 つのサービスにモニタを追加することも（以下の例の **Service1**）、または異なるサービスにモニタを追加することも（以下の例の **Service1** および **AllTestsSuite**）できます。 1 つのサービスに追加されたすべてのモニタは（たとえば、**Service1**）、そのサービス名の後に追加されて表示されます。



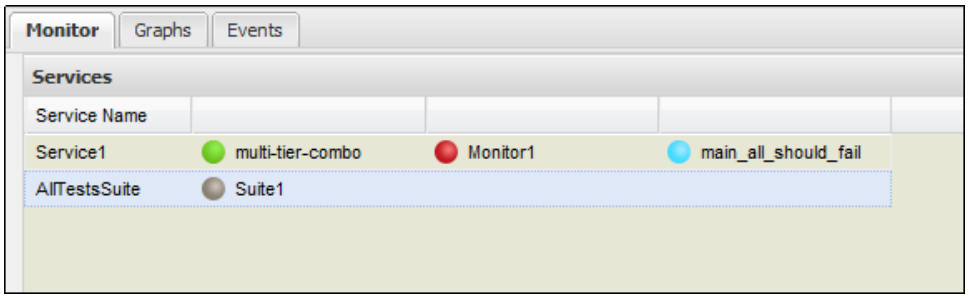
上部パネル

実行された特定のサービス内のテストは、すべて色付きのボール形で示されます。

- **緑：** 完了したテスト。
- **赤：** 失敗したテスト。

- **黄：** ステージング ドキュメントにエラーがあったテスト。

アイコンは、テスト ランの完了、失敗、エラー、ステージングのエラー、不明なエラーを表します。 これらのアイコンは、実行後のモニタの状態を示しています。

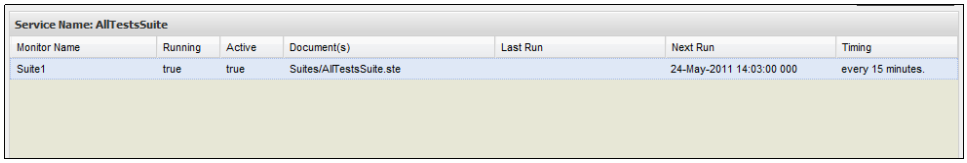


上部パネルの右側には、現在実行されているジョブがグラフで表示されます。 たとえば、以下のグラフは、現在ジョブが 1 つ実行されていることを示しています。

下部パネル

下部パネルには、サービスで実行されている各モニタのステータスが表示されます。

下部パネルには、実行中または最後の実行が完了したモニタがすべて表示されます。 実行が完全に終了した（再度実行されることがスケジュールされてない）モニタは、このリストには表示されません。

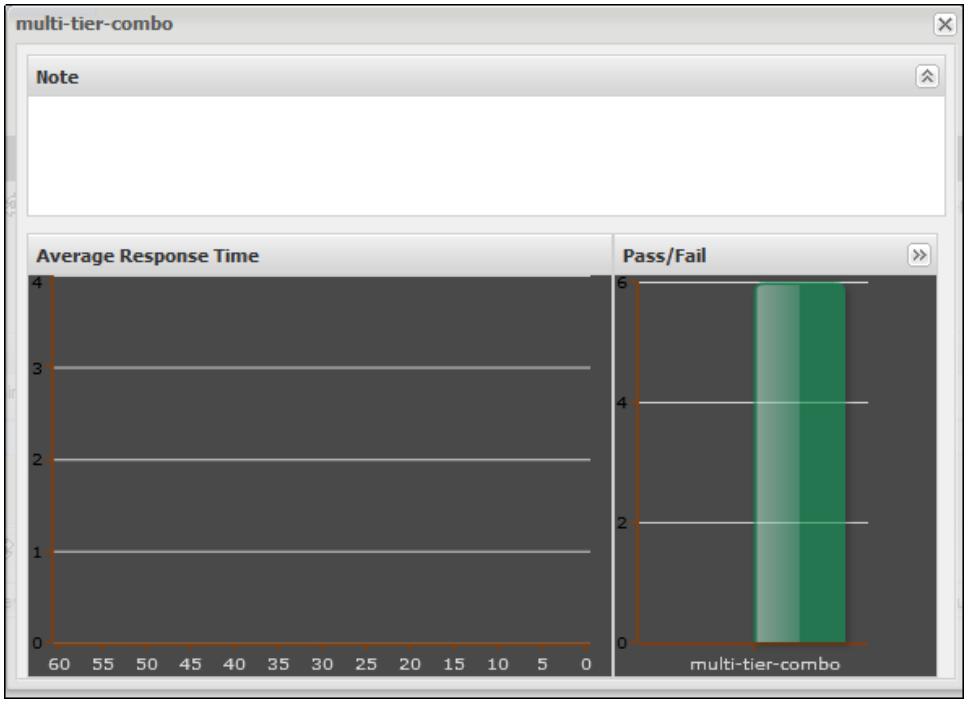


このテーブルには、以下の情報が表示されます。

- **モニタ名：** モニタに付けられた名前
- **実行中：** このモニタが実行中かどうか（true/false）
- **アクティブ：** このモニタがアクティブかどうか（true/false）
- **ドキュメント：** ドキュメントの名前（このモニタと関連付けられているテスト ケース、ステージング ドキュメント、およびスイート ドキュメントなど）
- **最後の実行：** このモニタの最後の実行日時
- **次回の実行：** このモニタの次にスケジュールされている開始時刻
- **間隔：** このモニタのスケジュールの詳細

列は並べ替えて表示することや、表示または非表示にすることができ、カスタマイズが可能です。

モニタに関連する情報を表示するには、モニタをダブルクリックします。



ツールバー

ツールバー タブには、以下のボタンのあるツールバーが表示されます。

- **モニタを展開/再展開：** ダッシュボードにモニタを展開または再展開します。 「CVS へのモニタの展開 (P. 427)」を参照してください。
- **モニタ属性を表示：** モニタに関連する情報を表示します。
- **モニタを削除：** ダッシュボードからモニタを削除します。
- **直ちにモニタを実行：** モニタをすぐに実行します。
- **非アクティブ化：** モニタを非アクティブにする場合にクリックします。非アクティブとは、リストには残るものの実行されないことを意味します。 また、以前に非アクティブ化されたモニタを再度アクティブにする場合にもこのボタンをクリックします。
- **結果を表示：** 実行結果をレポート ビューアで表示する場合にクリックします。

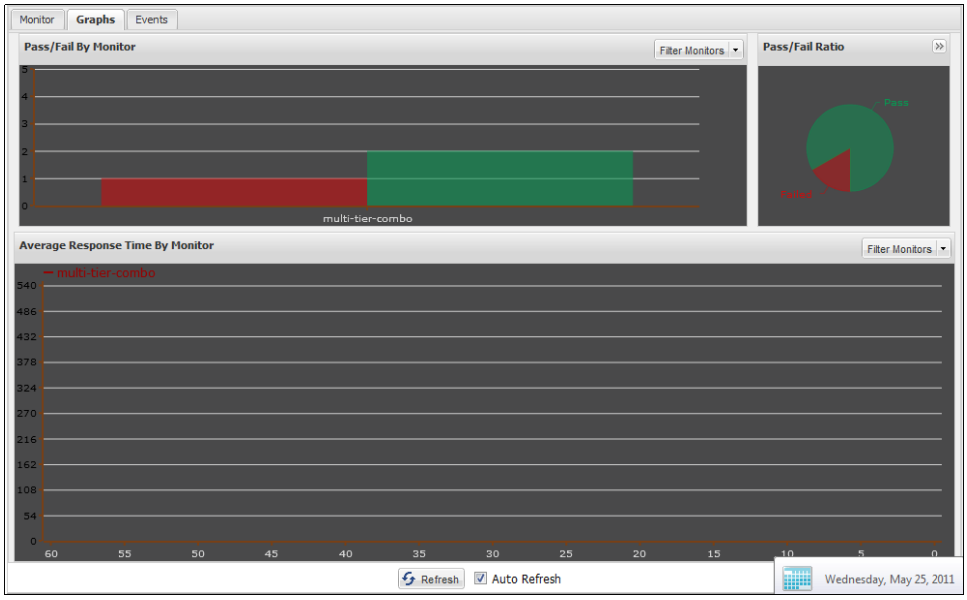
DevTest ワークステーション から CVS ダッシュボードを直接実行 (DevTest からダッシュボードを表示することを選択) した場合は、以下のボタンも表示されます。

-  ダッシュボード上のリストをリフレッシュします。
-  一定期間後にダッシュボード上のリストを自動的にリフレッシュします。

ブラウザから CVS ダッシュボードを直接実行した場合は、これらのボタンは表示されません。

CVS ダッシュボード - グラフ

［グラフ］タブには、CVS ダッシュボード内のテストがグラフィカルな形式で表示されます。このタブには、成功または失敗したテストのグラフィカルなサマリが表示されます。



［グラフ］タブは、3 つのグラフィカルな表示で構成されています。これらは、直前の 60 分間のアクティビティを表示および追跡します。

［フィルタ モニタ］では、表示する特定のモニタを選択できます。

次の手順に従ってください:

1. ウィンドウの上部にある［フィルタ モニタ］をクリックします。

   利用可能なモニタのリストが表示されます。

2. 確認するモニタを選択します。

   以下のセクションに詳細が表示されます。

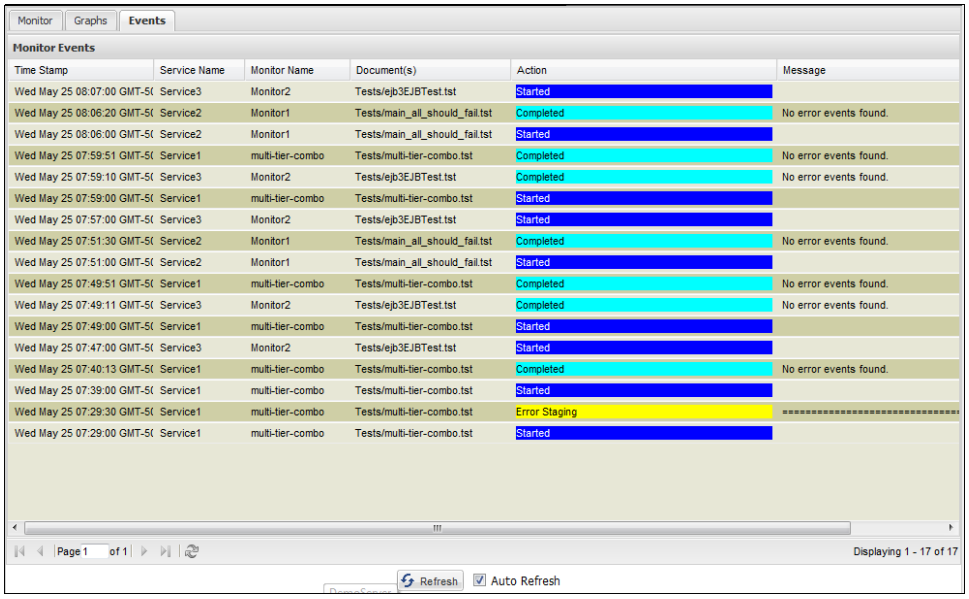
   - モニタ別の成功/失敗： ［モニタ別の成功/失敗］グラフには、各モニタの成功/失敗の結果のヒストグラムを重ねたものが表示されます。 ヒストグラムにカーソルを置くと、上部の［モニタ別の成功/失敗］パネルにテキスト形式で結果が表示されます。
   - 成功/失敗率： ［成功/失敗率］グラフには、成功したテスト（緑）の総数と失敗したテスト（赤）の総数のパーセンテージが円グラフとして表示されます。

- モニタ別平均応答時間：［モニタ別平均応答時間］グラフには、直前の 60 分間のアクティビティに対する各モニタの平均応答時間が示されます。 各モニタは色分けされています。 モニタにカーソルを置くと、ツールヒントに平均応答時間が表示されます。

［モニタ別の成功/失敗］では、テストやスイートをグラフィカルな形式で表示することを選択できます。

CVS ダッシュボード - イベント

［イベント］タブには、テスト ランの開始、終了、失敗など、モニタの実行の段階に対応するさまざまなイベントが表示されます。



以下の情報が表示されます。

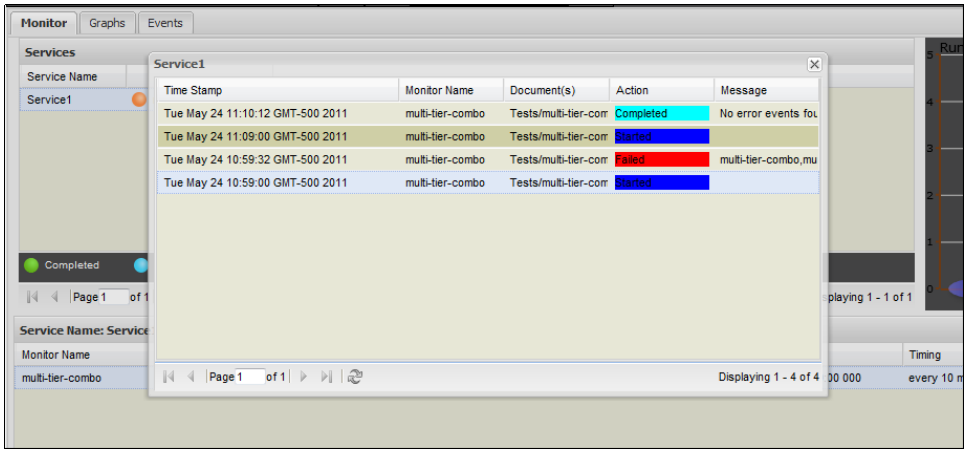
- タイム スタンプ： イベントが発生した時刻。
- サービス名： モニタが存在するサービスの名前。
- モニタ名： イベントが発生したモニタの名前。
- ドキュメント： イベントが発生したモニタに関連付けられているドキュメントのリスト。
- アクション： イベントによってレポートされたアクション。開始イベントは青、ステージング エラー イベントは黄、完了イベントは緑、失敗したイベントは赤で表示されます。
- メッセージ： イベントによってレポートされたメッセージ。テキストがメッセージのセルより長い場合は、セルを右クリックして拡張ビュー ウィンドウを呼び出すことができます。

パネルの下部にある［自動リフレッシュ］チェック ボックスをオンにすると、イベントをリアルタイムで表示できます。また［自動リフレッシュ］ボックスがオフの状態で［リフレッシュ］アイコンをクリックすると、リストを手動でリフレッシュできます。

すべての列がウィンドウに表示されるように、列のサイズを調節できます。［メッセージ］フィールドをダブルクリックすると、メッセージのテキスト全体を参照できます。

テスト サイクルのエラー

時にはテスト サイクルが失敗することがあります。 サイクルにエラーがある場合は、［アクション］リストに別の色で表示されます。 イベントをダブルクリックすると、展開されたウィンドウに関連するメッセージが表示されます。



CVS へのモニタの展開

以下のトピックが含まれます。

DevTest ワークステーション によるモニタの展開 (P. 428)
CVS ダッシュボードによるモニタの展開 (P. 428)
cvsMonitors ディレクトリによるモニタの展開 (P. 429)
CVS マネージャによるモニタの展開 (P. 429)

DevTest ワークステーション によるモニタの展開

この方法の唯一の前提条件は、テスト ケースまたはスイートが存在することです。

次の手順に従ってください:

1. DevTest ワークステーション の左側にあるプロジェクト パネルからテスト ケースまたはスイートを右クリックし、［モニタとして CVS に展開］を選択します。

   モニタ情報を表すタブの集合が表示されます。これは「*CA Application Test の使用*」の「モニタ MAR 情報ファイルの作成 (P. 307)」で必要な情報と同じです。

2. ［展開］をクリックして、モニタを展開します。

CVS ダッシュボードによるモニタの展開

この手順を実行するには、モニタのモデル アーカイブが作成 (P. 312)されている必要があります。

次の手順に従ってください:

1. CVS ダッシュボードの［モニタ］タブで、［モニタを展開/再展開］をクリックします。

   新しいモニタのための［モニタを展開/再展開］ウィンドウが表示されます。

2. ［モデル アーカイブ］フィールドに、MAR ファイルの名前を入力します。

3. MAR が以前に展開されている場合は、［モニタが存在する場合は置換］チェック ボックスをオンまたはオフにします。

4. ［展開/再展開］をクリックします。

   モニタが CVS ダッシュボードに追加されます。

cvsMonitors ディレクトリによるモニタの展開

この手順を実行するには、モニタのモデル アーカイブが作成 (P. 312)されている必要があります。

次の手順に従ってください:

1. **LISA_HOME¥cvsMonitors** ディレクトリに移動します。
2. このディレクトリに MAR ファイルを追加します。

   この後、同じディレクトリに MAR ファイルの新しいバージョンを配置すると、モニタを更新できます。

CVS マネージャによるモニタの展開

CVS モニタを展開するためのコマンドライン オプションの詳細については、「CVS マネージャ (P. 432)」を参照してください。

直ちにモニタを実行

上部パネルにサービスを追加すると、それらはスケジュールされた時間間隔で自動的にバックグラウンドで実行されます。

特定のモニタをすぐに実行する場合は、ツールバーを使用できます。

次の手順に従ってください:

1. モニタを展開します。
2. モニタが下部パネルにリストされている場合は、それを選択します。
3. ［直ちにモニタを実行］をクリックして、モニタをすぐに実行します。

   下部パネルでは、実行中のモニタは確認できません。
4. 実行が完了したモニタを確認するには、CVS ダッシュボードの［イベント］タブに移動して、**-now** というサフィックスが付いたモニタを確認します（例： **Test 3-now**）。

テストの詳細の表示

テスト ランの詳細は、CVS ダッシュボードで表示できます。

次の手順に従ってください:

以下のいずれかの操作を実行します。

- サービスに関連する詳細を表示するには、サービスをダブルクリックします。
- 上部パネルで、詳細を表示するモニタをダブルクリックします。

テストに関連する詳細のウィンドウが表示されます。

また、CVS ダッシュボードの［イベント］タブを表示することもできます。すべての列が表示されるように、列のサイズを調節できます。 メッセージのテキスト全体を表示するには、［メッセージ］フィールドをダブルクリックします。

スケジュールされたテストの実行中に生成されるレポートは、すべてレポート ビューアで表示できます。

電子メール通知の設定

CVS ユーティリティで新しいテストをスケジュールするたびに、CVS ダッシュボードにモニタを展開します。

モニタ ウィンドウでは、電子メール アドレスも設定できます。 電子メール アドレスを設定すると、指定した期間にモニタが実行されるたびに電子メール通知を受信できます。

電子メール通知を設定するには、**lisa.properties** ファイルを更新します。

また、メール サーバ設定を変更し、以下の例のようにメール ホストを入力する必要があります。

```
# If you use performance monitoring alerts, this is the "from" email
address of those alerts
# lisa.alert.email.emailAddr=lisa@itko.com

# And this is the email server we will attempt to route emails with
(SMTP server)
# lisa.alert.email.defHosts=localhost
```

lisa.properties ファイルでこの情報のコメントを外すには、ファイルの最初の列から **#** を削除します。

DevTest を再起動して、メール サーバ設定を設定します。

- 通知電子メール: テスト ランの結果を電子メールで通知するための電子メール アドレスを入力します。

テストが実行されるごとに、電子メールを受信します。

CVS Manager

CVS マネージャ コマンドライン ユーティリティでは、継続的検証サービス (P. 417)に展開したモニタのセットを管理できます。 このユーティリティは、**LISA_HOME¥bin** ディレクトリにあります。

このユーティリティの形式は以下のとおりです。

```
CVSManager [-h] [-m レジストリの指定] [-d アーカイブ ファイル] [-r アーカイブ ファイル] [-l] [-D] [-A] [-e] [x] [-X] [-s 名前] [-n 名前] [-u ユーザ名] [-p パスワード] [--version]
```

-h、--help

ヘルプ テキストを表示します。

-m *レジストリの指定*、--registry=*レジストリの指定*

接続するレジストリを定義します。

-d *アーカイブ ファイル*、--deploy=*アーカイブ ファイル*

指定したモデル アーカイブを CVS にモニタとして展開します。アーカイブで定義されているモニタは、存在しないモニタおよびサービス名の組み合わせを参照している必要があります。

-r *アーカイブ ファイル*、--redeploy=*アーカイブ ファイル*

指定したモデル アーカイブを CVS にモニタとして再展開します。アーカイブで定義されているモニタは、存在するモニタおよびサービス名の組み合わせを参照している必要があります。

-l、--list

現在展開されているモニタとそれぞれの情報をリスト表示します。

-D、--pause

指定したモニタのスケジュールされている実行を一時停止します。

-A、--resume

指定したモニタのスケジュールされている実行を再開します。

-e、--execute-now

スケジュールにかかわらず、指定したモニタをすぐに実行します。 この操作は、モニタのスケジュールされている実行には影響しません。

-x、--remove

CVS からモニタを削除します。サービス名とモニタ名の引数を使用して、削除するモニタを指定します。

-X、--remove-all

CVS からモニタをすべて削除します。サービス名を指定すると、そのサービス名で定義されたモニタだけが削除されます。

-s *名前*、--service-name=*名前*

影響するモニタのサービス名を指定します。

-n *名前*、--monitor-name=*名前*

影響するモニタ名を指定します。

-u *ユーザ名*、--username=*ユーザ名*

DevTest セキュリティのユーザ名を指定します。 このオプションは必須です。

-p *パスワード*、--password=*パスワード*

DevTest セキュリティのパスワードを指定します。 このオプションは必須です。

--version

バージョン番号を出力し、終了します。

例： モニタの展開

この例では、CVS にモニタを展開します。

```
CVSManager -d monitor.mar -u user -p password
```

例： モニタの削除

この例では、同じモニタを削除します（サービスおよびモニタの名前を仮定）。

```
CVSManager -x -s OrderManager -n CheckOrders -u user -p password
```

この例では、OrderManager サービス内のモニタをすべて削除します。

```
CVSManager -X -s OrderManager -u user -p password
```

この例では、モニタをすべて削除します。

```
CVSManager -X -u user -p password
```

第 16 章: レポート

以下の 3 つの領域のいずれかのレポート パラメータを指定することにより、レポート用に収集するデータを指定します。

- クイック テスト (P. 330)
- ステージング ドキュメント (P. 251)
- テスト スイート (P. 287)

実行する各テスト ケースまたはテスト スイートについて収集する特定のイベントまたはメトリックを指定できます。 データベースまたは XML ファイルのどちらかにレポート データを格納するレポート ジェネレータを 3 つの中から選択します。 また、レポート データを保持する期間も指定できます。

レポートを生成した後、それらを表示および管理したり、同僚と共有したり、ほかの場所にエクスポートしたりすることができます。

このセクションには、以下のトピックが含まれています。

レポート ジェネレータのタイプ

ステージング ドキュメント エディタ (P. 253)またはテスト スイート エディタ (P. 288)の［レポート］タブでは、以下のタイプのレポート ジェネレータを選択できます。

- デフォルトのレポート ジェネレータ (P. 436)
- ロード テスト レポート ジェネレータ (P. 437)
- XML レポート ジェネレータ (P. 437)

デフォルトのレポート ジェネレータ

デフォルトのレポート ジェネレータは、機能およびメトリックの情報をキャプチャし、そのデータをレジストリによって参照されるレポート データベースにパブリッシュします。 レポート ポータルは、このレポート データベースを使用します。

このレポート ライタは、負荷テストやパフォーマンス テストに対してはサポートされていません。 負荷テストに対しては、ロード テスト レポート ジェネレータを使用してください。

注: このレポート ジェネレータが選択されている場合、CA Application Test は負荷テストに関して自動設定しません。

第 17 章: ロード テスト レポート ジェネレータ

ロード テスト レポート ジェネレータは、何千もの仮想ユーザを想定した負荷テスト用に設計されています。

このレポートでは負荷メトリックがキャプチャされますが、ステップ レベルのメトリックはキャプチャされません。 ステップ レベルのメトリックをキャプチャすると、データ量が多くなり過ぎ、レポート データベースによってテストのスピードが遅くなるおそれがあります。

ほとんどのイベントは無効にされています。そのため、レポート コンソールには、実行の少数の情報が表示されます。 主なフィードバックは、DevTest ワークステーション の［テスト ラン］パネルに表示されます。

［パラメータ］領域では、テストが自動的に停止されるエラーの数を設定できます。 デフォルト値は 100 です。

デフォルトのレポート ジェネレータは、負荷テストやパフォーマンス テストに対してはサポートされていません。

XML レポート ジェネレータ

このレポート ジェネレータは、キャプチャ可能なすべてのデータを使用して XML ファイルを作成します。 キャプチャされるデータは、テスト スイート エディタまたはステージング ドキュメント エディタのレポート オプションを使用して制限できます。 このレポートを表示するには、レポート ポータルにファイルをインポートします。

このテーブル内のデータは、カスタム レポートのニーズに合わせて使用できます。

テスト、スイート、またはその両方が完了した後、レポート コンソールでそのレポート データを表示できます。 XML データをファイルにエクスポートする方法については、「レポートのエクスポート (P. 474)」を参照してください。

レポート ポータルを開く

レポート ポータルは、DevTest ワークステーション または Web ブラウザから開くことができます。

DevTest ワークステーション から レポート ポータルを開く方法

メイン メニューから［表示］-［レポート コンソール］を選択します。

Web ブラウザからレポート ポータルを開く方法

1. レジストリ (P. 11)が実行されていることを確認します。
2. Web ブラウザに「**http://localhost:1505/**」と入力します。

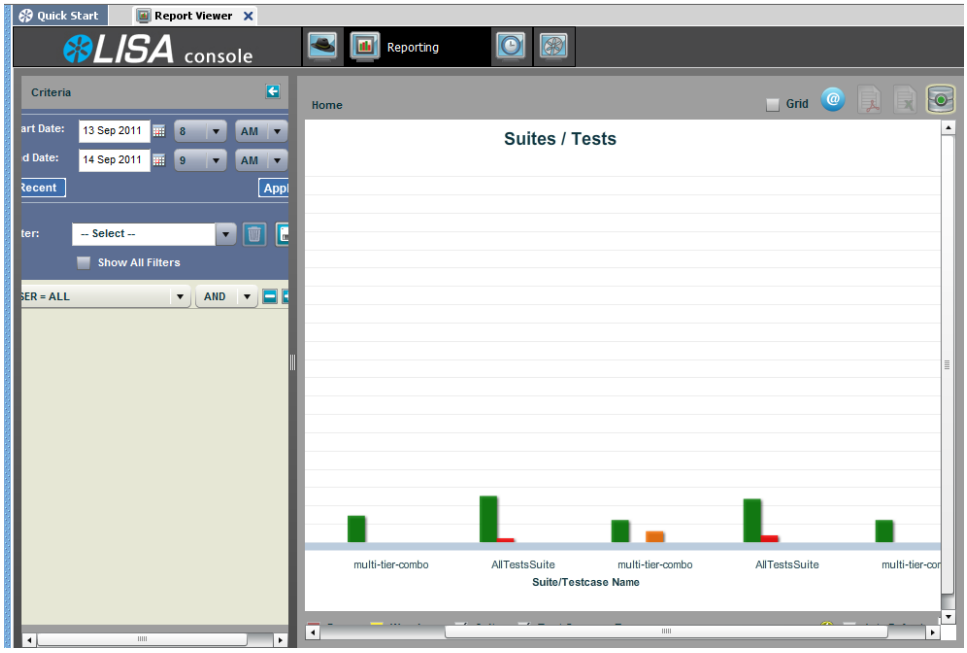
   レジストリがリモート コンピュータ上にある場合は、**localhost** をそのコンピュータの名前または IP アドレスに置き換えます。

   DevTest コンソールが開きます。
3. ［Reporting Dashboard］をクリックします。

レポート ポータルのレイアウト

レポート ポータルでは、DevTest ワークステーション でこれまでに実行されたレポートをすべて表示できます。ステージング ドキュメント、クイック テスト、テスト ケースの実行、またはテスト スイートからのレポートを設定できます。

このセクションでは、multi-tier-combo テスト ケースの実行によって生成されるレポートを確認します。これは examples ディレクトリにあります。



レポート ポータル - 条件

左側のパネルでは、日時の条件と使用するフィルタを選択できます。

開始日/終了日

カレンダ をクリックして、開始日/終了日を選択します。デフォルトでは、開始日および終了日は過去 1 時間の範囲です。1 ～ 12 および 午前/午後 ドロップダウンをクリックして、開始時刻および終了時刻を選択します。 開始日と終了日を入力したら、［適用］をクリックして変更を適用します。

最近使用したもの

日時の条件を自動的にリセットして過去 1 時間に最後に実行されたテスト/スイートを検索する場合は、このボタンをクリックします。

フィルタ

ここでは、任意のフィルタを作成できます。 フィルタの名前を入力して、［保存］をクリックします。 また、［削除］をクリックするとフィルタを削除できます。 作成したフィルタだけでなく、すべてのフィルタを表示するには、［すべてのフィルタを表示］を選択します。

条件に AND/OR 演算子を選択し、［追加］ または［削除］ をクリックします。

レポート ポータル - 右側のパネル

右側のパネルは、選択した条件によって描画されるグラフで構成されています。 詳細については、「レポート - グラフィカル ビュー (P. 445)」を参照してください。

右側のパネルの上部にはレポート ツールバーが表示され、以下のことを実行できます。

アイコン	機能
	レポート ビューをデフォルトのチャート ビューからグリッド ビューに、またはその逆に変更します。詳細については、「レポート - グリッド ビュー (P. 461)」を参照してください。

	表示しているレポートの URL をオペレーティング システムのクリップボードにコピーして、その他のユーザとレポートを共有できるようにします。
	レポート データを PDF ファイルまたは Excel ファイルにエクスポートして、それらの形式でレポートを表示します。 詳細については、「レポートのエクスポート (P. 474)」を参照してください。
	レポート データベースに関する情報を表示します。 LISA Database Connection Successful --- DBUrl: jdbc:derby://localhost:1528/database/lisa.db;create=true Username: rpt Database: Apache Derby Version: 10.6.2.1 - (999685) Driver: Apache Derby Network Client JDBC Driver Version: 10.6.2.1 - (999685) Suite Runs: 4 Test Runs: 71

成功または失敗したテスト ケースの結果のレポートをフィルタできます。また、エラーや警告を表示または非表示にすることや、テスト スイートやテスト ケースを表示または非表示にすることを選択できます。 詳細については、「レポートのフィルタ (P. 442)」を参照してください。

［ズーム］スライダでは、レポートの表示サイズをカスタマイズできます。

［リフレッシュ］ボタン  は表示をリフレッシュします。 自動リフレッシュを設定するには、［自動リフレッシュ］チェック ボックスをオンにして、自動リフレッシュの間隔を設定します。

レポートのフィルタ

レポート ポータル (P. 439)の右側のパネルには、レポートが表示されます。

条件を指定してレポートをフィルタできます。条件を選択すると、レポート ビューアには選択した条件を満たすグラフだけが表示されます。


成功

成功したテスト ケース/スイートが表示されます。


失敗

失敗したテスト ケース/スイートが表示されます。


中止

中止したテスト ケース/スイートが表示されます。


エラー

エラーを生成したテスト ケース/スイートが表示されます。


警告

警告を生成したテスト ケース/スイートが表示されます。

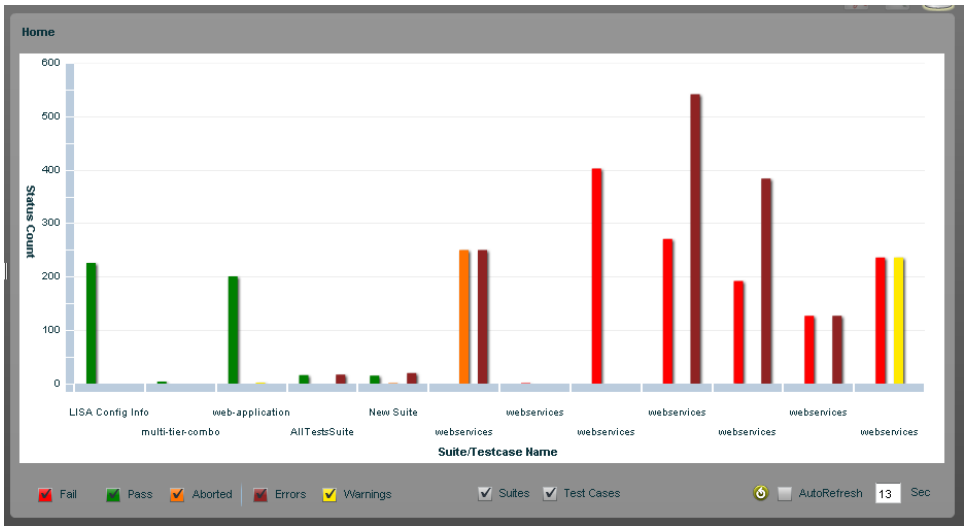

スイート

テスト スイートの結果が表示されます。


テスト ケース:

テスト ケースの結果が表示されます。

レポート ポータルには、実行されて現在データベースにあるすべてのテスト ケースおよびテスト スイートのレポートが表示されます。



上記のレポートでは、フィルタがすべて選択されているため、検索条件は指定されていません。 そのため、レポートには、成功したもの、失敗したもの、中断されたもの、エラーのあるもの、警告のあるもの、それらすべてのテスト ケースおよびテスト スイートが表示されます。

注: より詳しく指定するために、**結果**の条件と**テスト ラン**の条件を組み合わせることができます。 たとえば、［失敗］と［エラー］と［テスト ケース］を選択して、失敗したテスト ケースまたはエラーで完了したテスト ケースだけを表示できます。

レポートをリフレッシュする方法

［リフレッシュ］ をクリックします。

自動的にレポートをリフレッシュする方法

1. ［自動リフレッシュ］チェック ボックスをオンにします。
2. レポートがリフレッシュされるまでの秒数を入力します。

   たとえば、「**15**」を入力すると 15 秒ごとにレポートがリフレッシュされます。

レポートの表示

レポート ビューアでは、レポートは以下の 2 つの形式で表示できます。

- グラフィカル ビュー（P. 445）： レポート ポータルのデフォルトのビュー。 すべてのレポートをグラフィカルな形式で表示できます。
- グリッド ビュー（P. 461）： グリッド ビューを選択すると、グリッド形式でデータを配置できます。

詳細:

標準レポート (P. 463)
レポートの解釈 (P. 473)

レポート - グラフィカル ビュー

デフォルトでは、レポートがグラフとして表示されます。

以下のサンプル レポートには、3 つのテスト ケースが含まれています。

- multi-tier-combo
- main_all_should_fail
- ejb3EJBTest

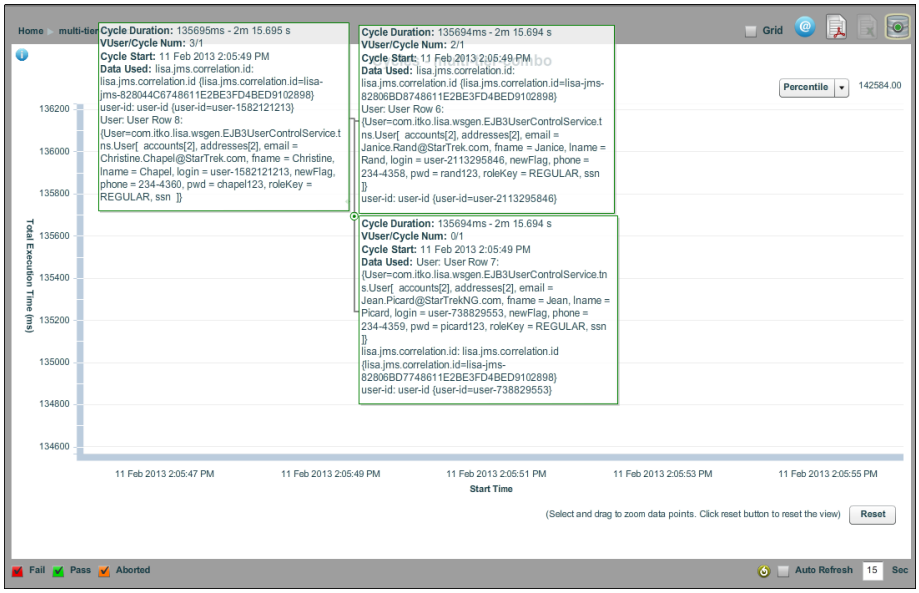
フィルタ条件のチェック ボックスの一部が選択されていないため、成功、失敗、または中断したテストケースだけがレポートに表示されています。

グラフィカル ビューでは、以下の機能が使用できます。

- テスト ケースにマウス カーソルを置くと、テスト ケース名、実行の日付、およびテストの実行の詳細を示す情報ボックスが表示されます。

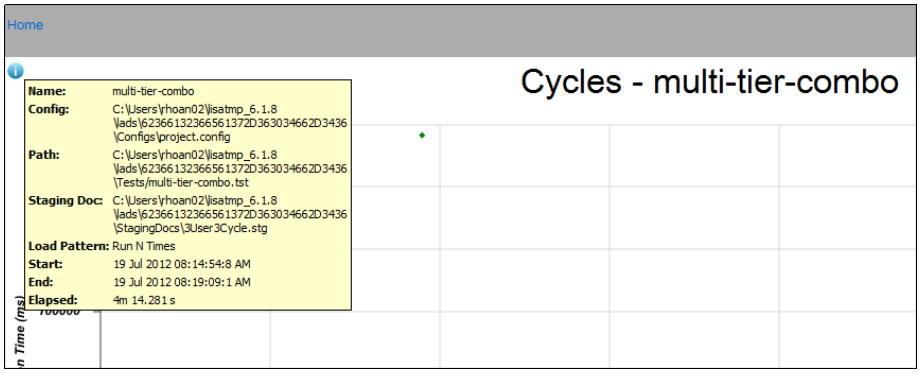
- グラフ内の成功したテスト ケースをダブルクリックすると、テスト ケースのサイクル図が生成されます。

  グラフィカル ビューにサイクル図を表示した場合、［P パーセンタイル］ドロップダウンからオプションを選択して、グラフィカルな形式（図の横向きの線）でパーセンタイル値を表示するように指定できます。パーセンタイル値は、特定のパーセンテージのステップ実行の完了にかかった最大ミリ秒数を示します。たとえば、ステップ A の 75 番目に長い実行サイクルが 7.9 秒で完了した場合、75 番目のパーセンタイルは 7900 以上です。

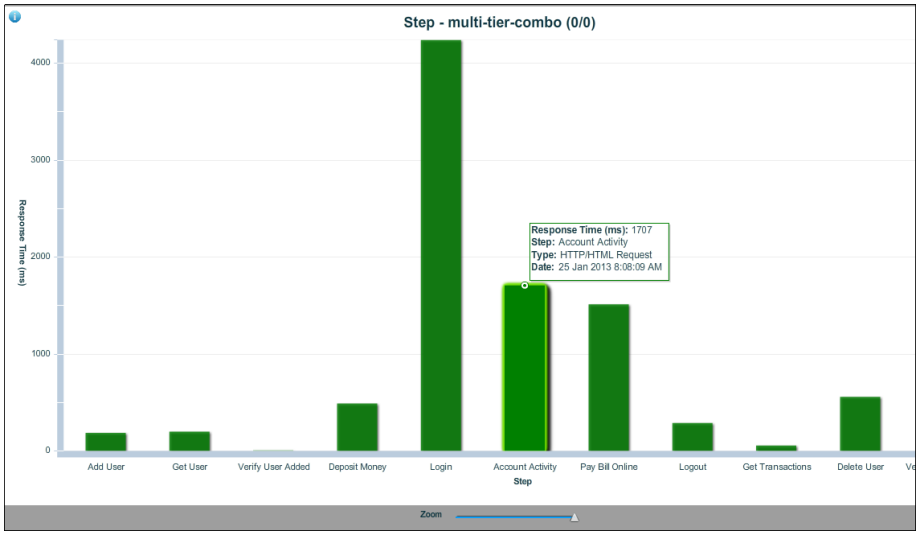


これらのドットにマウス カーソルを置くと、各ドットがテスト ケースのサイクルを表していることがわかります。また、各サイクルの詳細には、応答時間（ミリ秒）、サイクル/インスタンス番号、サイクルの日付、および使用されたデータが表示されます。 サイクルは、開始から終了までのテスト ケース全体を表します。

- ウィンドウの左上の［情報］ボタンをクリックすると、テスト ケース環境に関する詳細情報が表示されます。



- サイクルのサブセットをより詳しく検査するには、ドットのグループを選択してズームし、サイクルのサブセットのみを表示します。
- 完全なサイクル ビューに戻るには、［リセット］ボタンをクリックします。
- テスト ケースをダブルクリックすると、応答時間に関する情報とともに、ケースの各ステップのビューが表示されます。



- 折れ線グラフとして表示されるレポートには、自動でスケールが設定されます。［現在のスケール］のデフォルト設定は、自動スケールの 100 です。 () 内の数値は、自動スケールが同じ 0 ～100 のグラフにデータをすべて対応させるために決定したスケールの値を示します。この値とスケールを乗算したものがグラフの Y 座標になります。 スケールを変更しても現在の値は変わりません。 ただし、グラフで使用される Y 座標を決定する計算では、異なる Y 座標が算出される場合があります。 それが 100 を超えている場合は、より大きな値に対応できるように Y 軸の制限も増加させる必要があります。

レポート - レポート メニュー

テスト ケースまたはスイートを右クリックすると、以下のオプションが使用できます。 レポートの例の表示する方法については、「レポート - グラフィカル ビューの例 (P. 449)」を参照してください。

分析	■ **最長トランザクション上位 10 件**： 実行時間の長い上位 10 件のトランザクションの円グラフ ■ **平均トランザクション応答**： トランザクション応答時間の折れ線グラフ ■ **メトリック**： DevTest イベント メトリックの時間ごとの折れ線グラフ ■ **リクエスト数/秒**： 1 秒あたりの要求数の時間ごとの折れ線グラフ ■ **パフォーマンス サマリ**：ステップごとの平均応答時間および標準偏差の棒グラフ ■ **サイクル パフォーマンス サマリ**： ミリ秒単位のサイクル時間およびサイクル実行時間の棒グラフ ■ **HTTP 詳細**： グリッド形式の名前ごとの HTTP トラフィックの詳細 ■ **HTTP サマリ**： 累積 HTTP トラフィック サマリの折れ線グラフ
エラー レポートを表示	■ 失敗の詳細 ■ エラーの詳細 ■ 警告の詳細 ■ 中止の詳細
起動プロパティを表示	タイム スタンプ、プロパティ名、およびプロパティ値と共に、起動プロパティをすべてグリッド形式で表示します。
履歴を表示	ウィンドウの上半分へ現在のグラフを縮小表示し、2 つの追加のレポートを表示します。 ■ 実行されたテスト ケースおよびそれらの結果のグリッド形式のリスト ■ 実行時間の履歴を示す折れ線グラフ。 履歴レポートには、同じテスト ケースの前回の実行についての情報が表示されます。
削除	選択したテスト ケースまたはスイートをレポートから削除します。

ピン	テスト ケースまたはスイートが 30 日後に自動削除されないようにします。 ピンが解除された 30 日より古いテストは自動削除されます。 詳細については、「管理ガイド」の「自動レポート保守」を参照してください。
インポート	XML ファイルからテスト ケースまたはスイートの情報をインポートします。
エクスポート	XML ファイルにテスト ケースまたはスイートの情報をエクスポートします。
画像の保存	.png ファイルとしてグラフィカル イメージを保存します。 ステップのメニューを表示するには、ステップのバーを右クリックします。

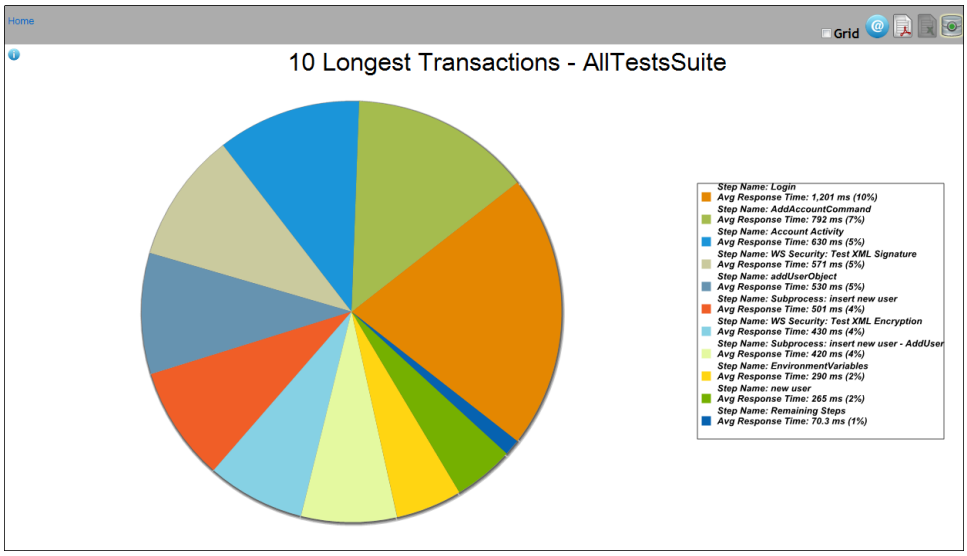
レポート - グラフィカル ビューの例

このセクションでは、以下のレポート メニュー オプションの例を示します。

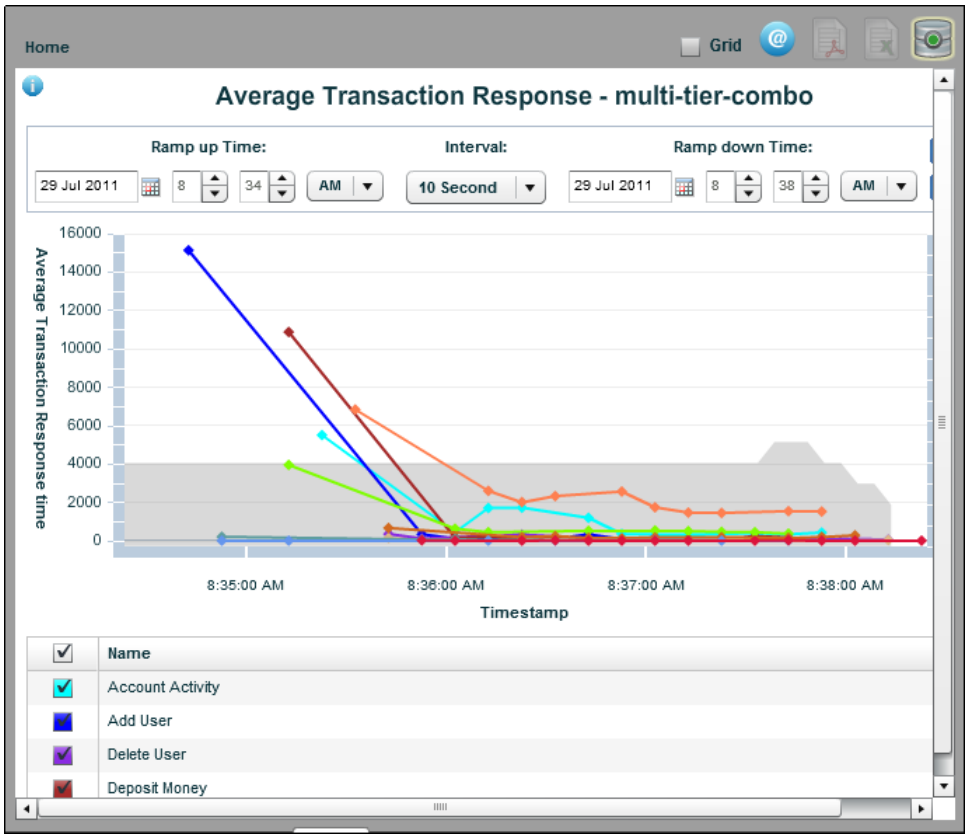
- 分析 (P. 450)
- エラー レポートを表示 (P. 456)
- 履歴を表示 (P. 460)

レポートの例の分析

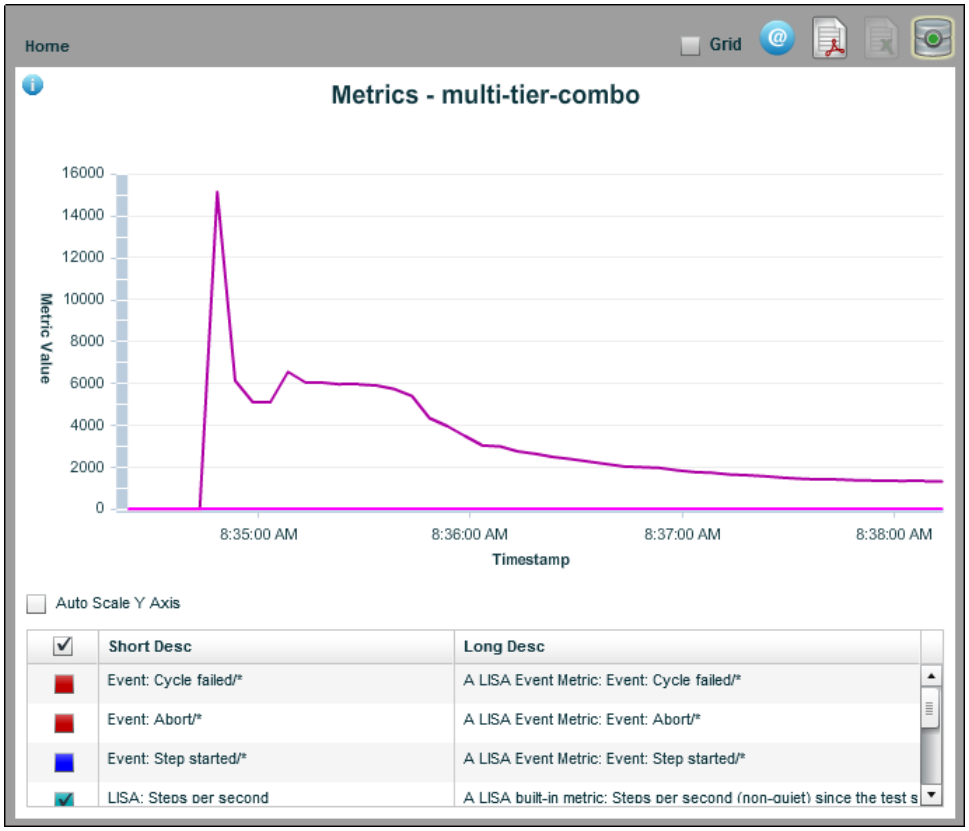
最長トランザクション上位 **10** 件



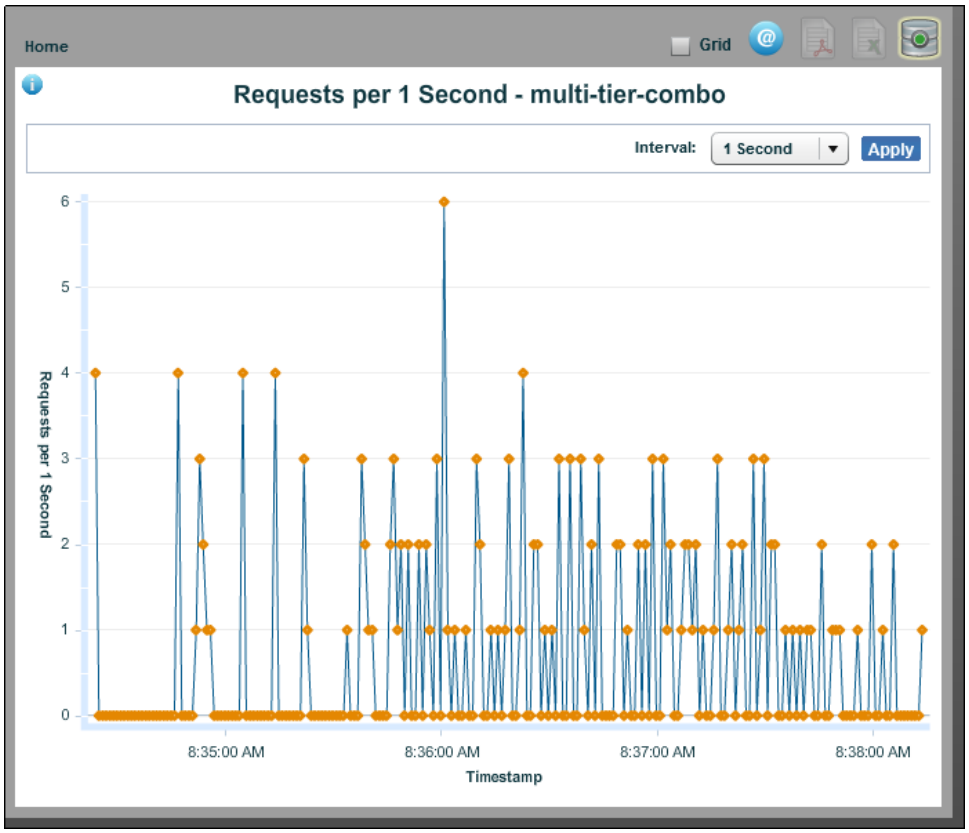
平均トランザクション応答


Home
Grid
Average Transaction Response - multi-tier-combo
Ramp up Time:
Interval:
Ramp down Time:
29 Jul 2011
8
34
AM
10 Second
29 Jul 2011
8
38
AM
16000
14000
12000
10000
8000
6000
4000
2000
0
Average Transaction Response time
8:35:00 AM
8:36:00 AM
8:37:00 AM
8:38:00 AM
Timestamp
Name
Account Activity
Add User
Delete User
Deposit Money

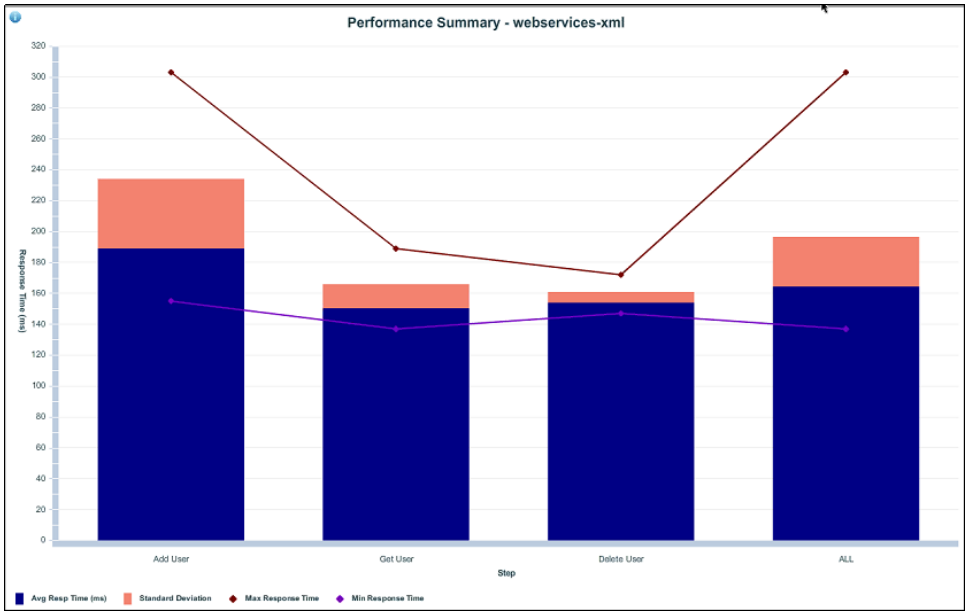
メトリック


Home
Grid
Metrics - multi-tier-combo
16000
14000
12000
10000
8000
6000
4000
2000
0
Metric Value
8:35:00 AM
8:36:00 AM
8:37:00 AM
8:38:00 AM
Timestamp
Auto Scale Y Axis
Short Desc
Long Desc
Event: Cycle failed/*
A LISA Event Metric: Event: Cycle failed/*
Event: Abort/*
A LISA Event Metric: Event: Abort/*
Event: Step started/*
A LISA Event Metric: Event: Step started/*
LISA: Steps per second
A LISA built-in metric: Steps per second (non-quiet) since the test s

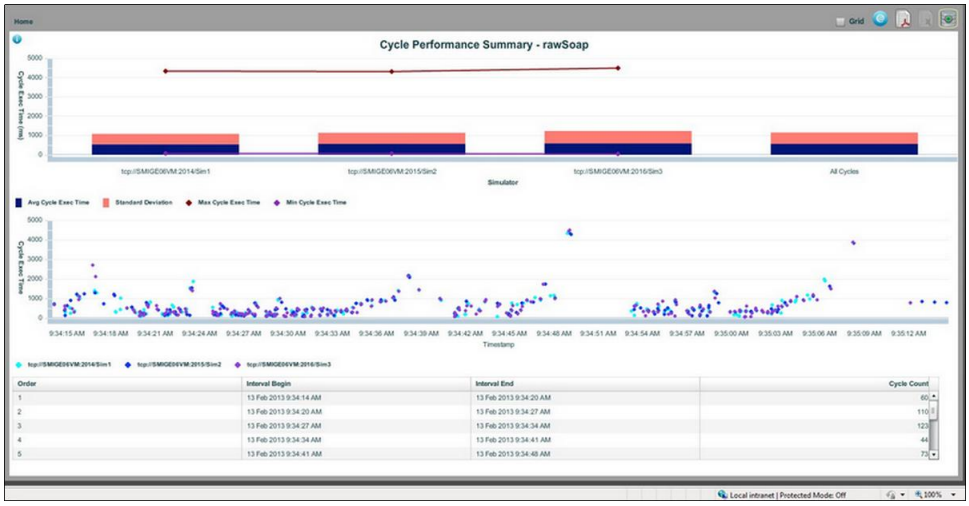
リクエスト数/秒


Home
Grid
Requests per 1 Second - multi-tier-combo
Interval:
1 Second
Apply
Requests per 1 Second
0
1
2
3
4
5
6
8:35:00 AM
8:36:00 AM
8:37:00 AM
8:38:00 AM
Timestamp

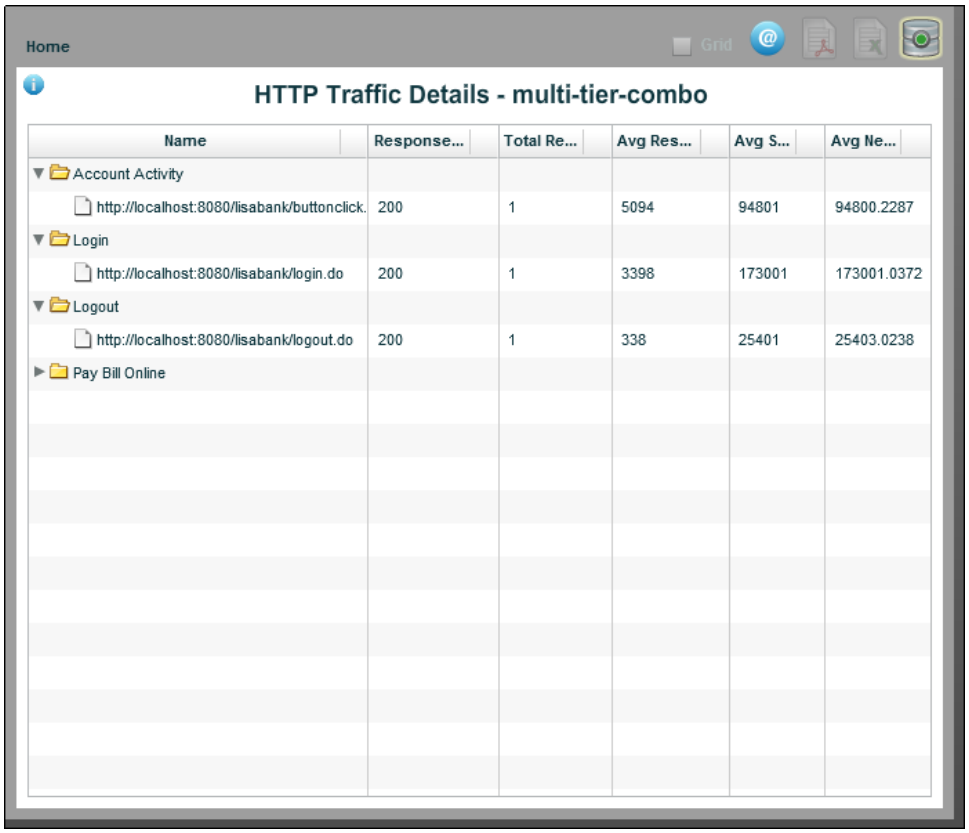
パフォーマンス サマリ


Performance Summary - webservices-xml
Response Time (ms)
Add User
Get User
Delete User
ALL
Step
Avg Resp Time (ms)
Standard Deviation
Max Response Time
Min Response Time

サイクル パフォーマンス サマリ

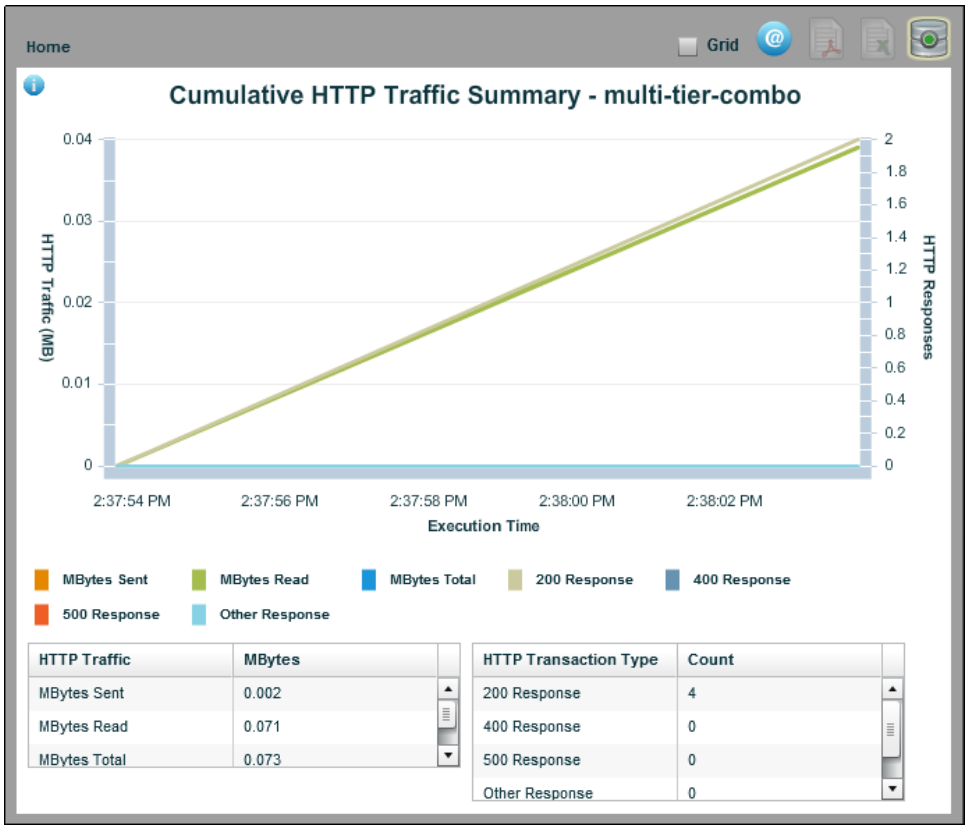


HTTP 詳細



HTTP サマリ

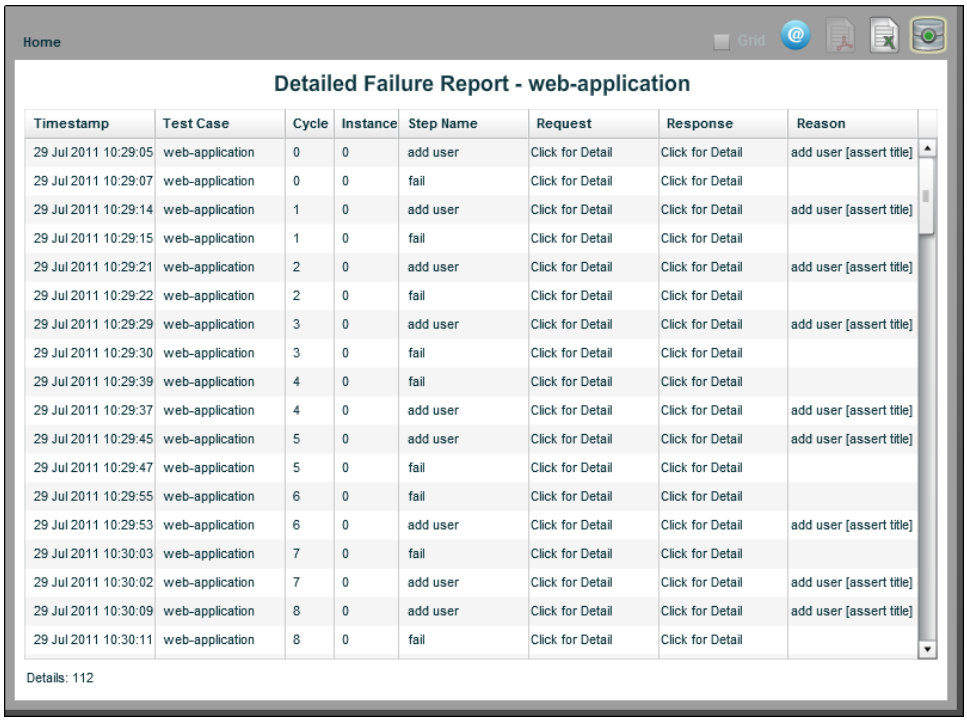
累積 HTTP トラフィック サマリ レポートを表示するには、local.properties または site.properties のいずれかで次のプロパティを設定する必要があります： **lisa.commtrans.ctstats= true**。



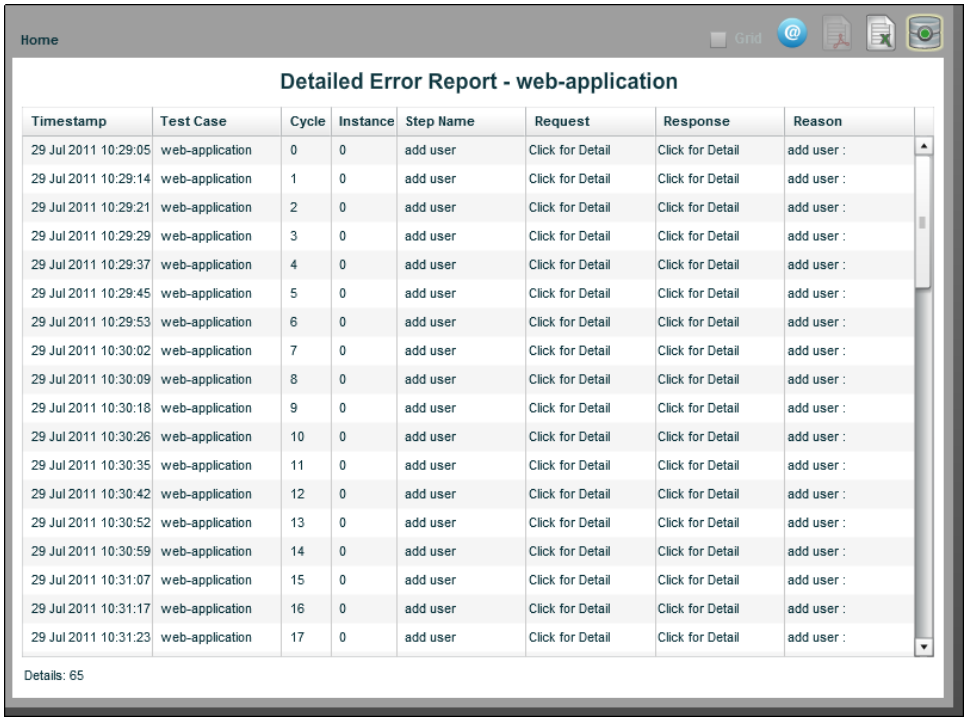
[エラー レポートを表示]の例

エラー レポートは、エラーなしに完了したもの以外のステップからのエラーおよび警告をレポートします。 以下の図は、以下のエラー レポートの例です。

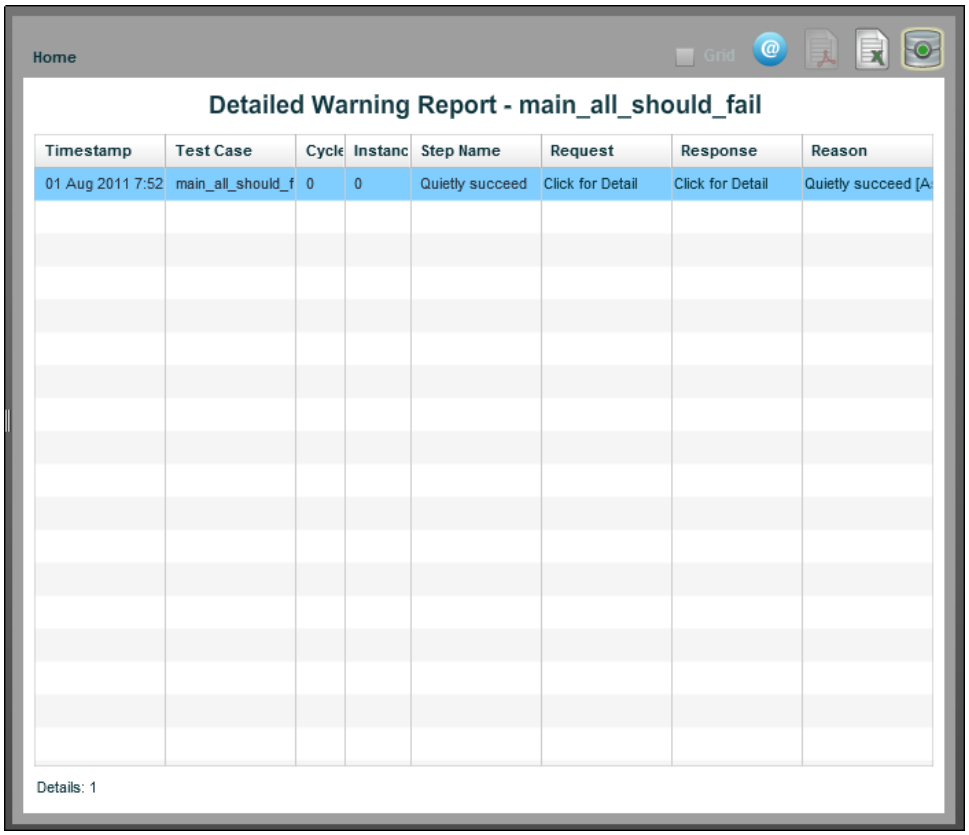
失敗の詳細



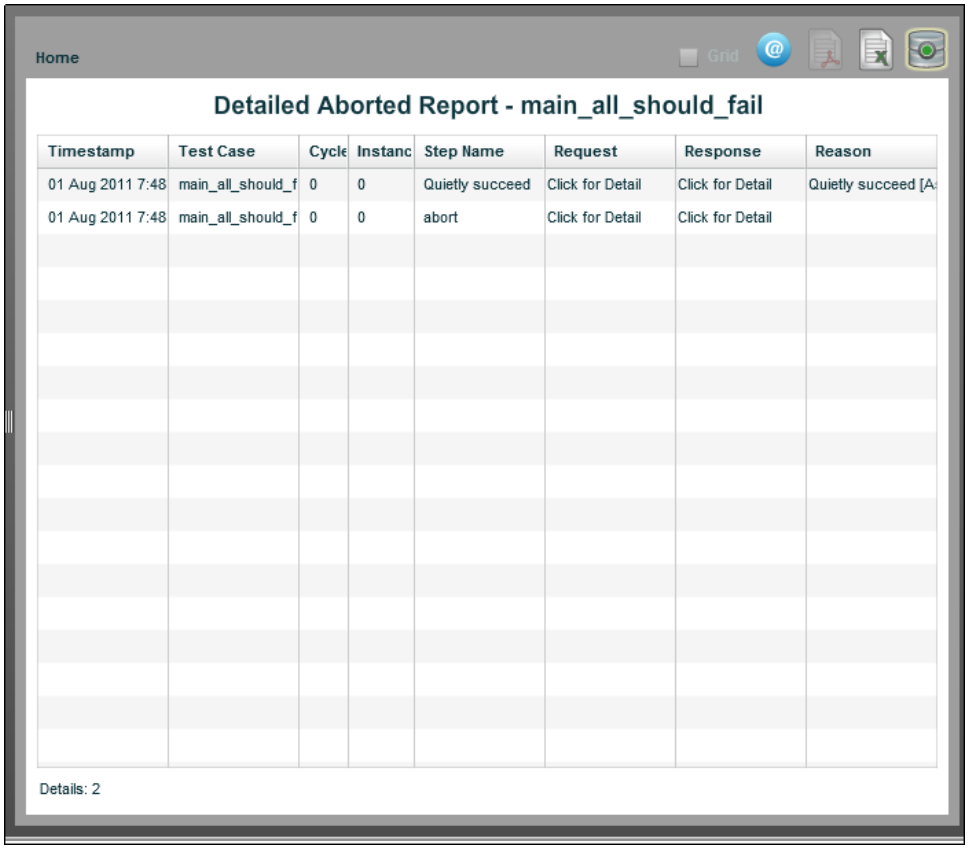
エラーの詳細


Home
Grid
Detailed Error Report - web-application
Timestamp | Test Case | Cycle | Instance | Step Name | Request | Response | Reason
29 Jul 2011 10:29:05 | web-application | 0 | 0 | add user | Click for Detail | Click for Detail | add user :
29 Jul 2011 10:29:14 | web-application | 1 | 0 | add user | Click for Detail | Click for Detail | add user :
29 Jul 2011 10:29:21 | web-application | 2 | 0 | add user | Click for Detail | Click for Detail | add user :
29 Jul 2011 10:29:29 | web-application | 3 | 0 | add user | Click for Detail | Click for Detail | add user :
29 Jul 2011 10:29:37 | web-application | 4 | 0 | add user | Click for Detail | Click for Detail | add user :
29 Jul 2011 10:29:45 | web-application | 5 | 0 | add user | Click for Detail | Click for Detail | add user :
29 Jul 2011 10:29:53 | web-application | 6 | 0 | add user | Click for Detail | Click for Detail | add user :
29 Jul 2011 10:30:02 | web-application | 7 | 0 | add user | Click for Detail | Click for Detail | add user :
29 Jul 2011 10:30:09 | web-application | 8 | 0 | add user | Click for Detail | Click for Detail | add user :
29 Jul 2011 10:30:18 | web-application | 9 | 0 | add user | Click for Detail | Click for Detail | add user :
29 Jul 2011 10:30:26 | web-application | 10 | 0 | add user | Click for Detail | Click for Detail | add user :
29 Jul 2011 10:30:35 | web-application | 11 | 0 | add user | Click for Detail | Click for Detail | add user :
29 Jul 2011 10:30:42 | web-application | 12 | 0 | add user | Click for Detail | Click for Detail | add user :
29 Jul 2011 10:30:52 | web-application | 13 | 0 | add user | Click for Detail | Click for Detail | add user :
29 Jul 2011 10:30:59 | web-application | 14 | 0 | add user | Click for Detail | Click for Detail | add user :
29 Jul 2011 10:31:07 | web-application | 15 | 0 | add user | Click for Detail | Click for Detail | add user :
29 Jul 2011 10:31:17 | web-application | 16 | 0 | add user | Click for Detail | Click for Detail | add user :
29 Jul 2011 10:31:23 | web-application | 17 | 0 | add user | Click for Detail | Click for Detail | add user :
Details: 65

警告の詳細


Home
Grid
Detailed Warning Report - main_all_should_fail
Timestamp
Test Case
Cycle
Instanc
Step Name
Request
Response
Reason
01 Aug 2011 7:52
main_all_should_f
0
0
Quietly succeed
Click for Detail
Click for Detail
Quietly succeed [A
Details: 1

中止の詳細

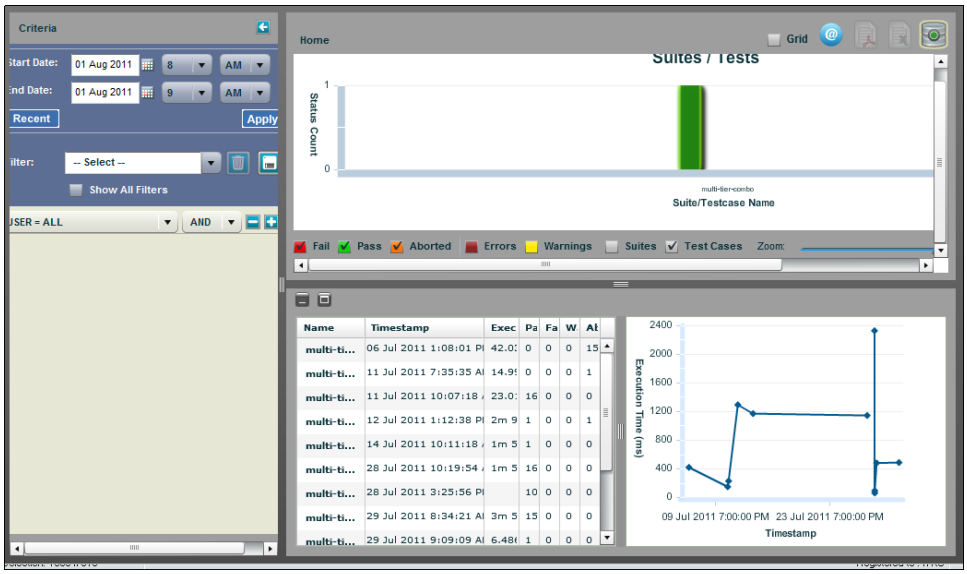

Home
Grid
Detailed Aborted Report - main_all_should_fail
Timestamp
Test Case
Cycle
Instanc
Step Name
Request
Response
Reason
01 Aug 2011 7:48
main_all_should_f
0
0
Quietly succeed
Click for Detail
Click for Detail
Quietly succeed [A
01 Aug 2011 7:48
main_all_should_f
0
0
abort
Click for Detail
Click for Detail
Details: 2

[履歴を表示]レポートの例

［履歴を表示］レポートは、現在のグラフをウィンドウの上半分へ縮小表示し、2 つの追加のレポートを表示します。

- 実行されたテスト ケースおよびそれらの結果のグリッド形式のリスト
- 実行時間の履歴を示す折れ線グラフ。

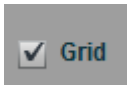
このレポートには、同じテスト ケースの前回の実行についての情報が表示されます。



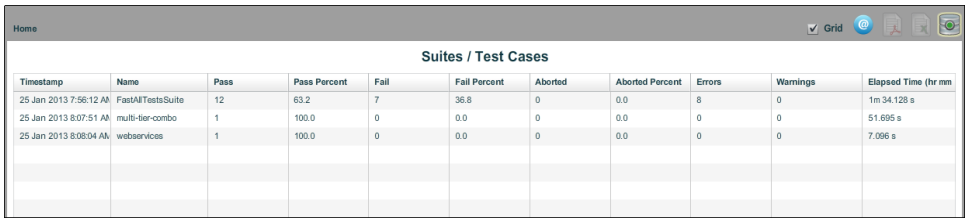
レポート - グリッド ビュー

グリッド ビューには、グラフィカル ビューと同じ情報が表示されます。違いは、それがグリッド形式で表示されるということのみです。 レポート情報のフィルタはすべて同じように動作します。

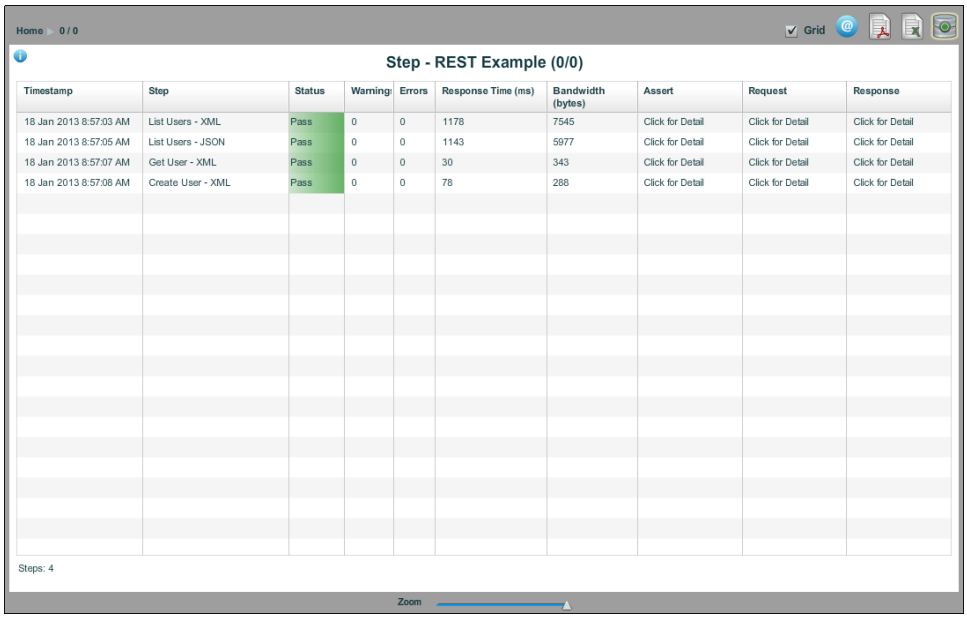
注: Excel にレポートをエクスポートするには、グリッド ビューである必要があります。

結果をグリッドで表示するには、ウィンドウの上部の［グリッド］ 

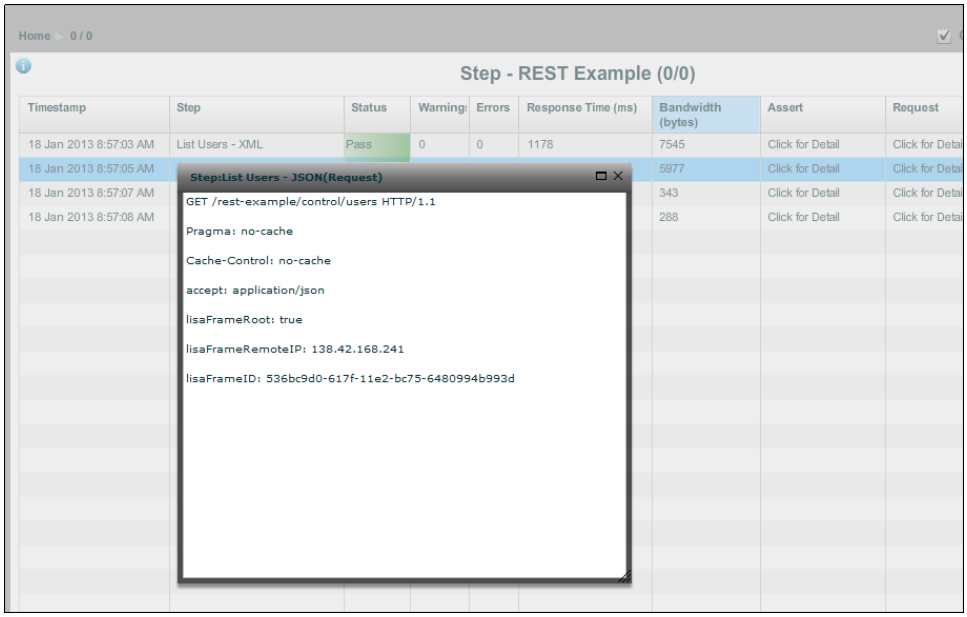
をオンにします。



レポート内のテスト ケースを選択して、テスト ケースの詳細を確認することができます。



［アサート］、［要求］、または［応答］列の［クリックして詳細を表示］をクリックすると、ステップの各コンポーネントの詳細情報が表示されます。



このビューでクリックし、要求および応答のデータを表示します。 データが XML である場合は、［整形済み XML］タブを選択して整形されたテキストを表示することができます。 XML 以外のデータの場合、［整形済み XML］タブには「Text is not valid XML」（テキストは有効な XML ではありません）と表示されます。

パフォーマンス サマリ レポートのグリッド ビューで、［カスタマイズ］ドロップダウンからオプションを選択して、レポートの最後の列にパーセンタイル値を表示するように指定できます。 パーセンタイル値は、特定のパーセンテージのステップ実行の完了にかかった最大ミリ秒数を示します。 たとえば、ステップ A の 75 番目に長い実行サイクルが 7.9 秒で完了した場合、75 番目のパーセンタイルは 7900 以上です。

標準レポート

事前にフォーマット済みの以下の 4 つのレポートを生成し、PDF にエクスポートできます。

- 機能テスト レポート（P. 464）
- パフォーマンス テスト レポート（P. 465）
- スイート サマリ レポート（P. 467）
- メトリック レポート（P. 469）

重要な定義：

- **サンプル サイズ**は、ステージング ドキュメントに基づきます。デフォルトでは、1 秒ごとにサンプリングし、10 秒ごとに平均します。 サンプル サイズは、サンプル サイズを変更するためのスライダを移動させることによって変更できます。
- **サイクル**は、テスト ケース全体の完全な実行です。
- **平均サイクル時間**は、テストを最初から終わりまで（サイクル）実行するためにかかった時間の平均です。

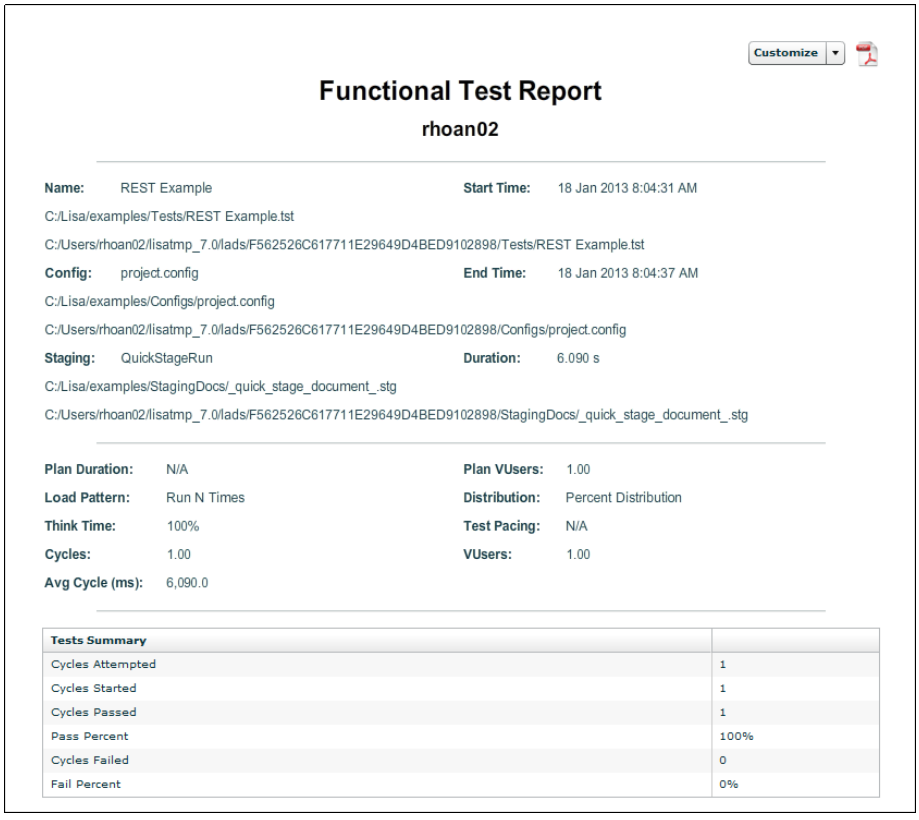
機能テスト レポート

機能テスト レポートを生成する方法

1. ステップのレポートを表示するには、テストをダブルクリックします。
2. ステップを選択し、［PDF にエクスポート］  を選択します。

   ［レポートを選択してください］ウィンドウが表示されます。
3. テストの使用可能なレポートのリストから［機能テスト レポート］を選択し、［OK］をクリックします。

   ［カスタマイズ］ドロップダウンを使用すると、このレポートの詳細なサマリを作成できます。

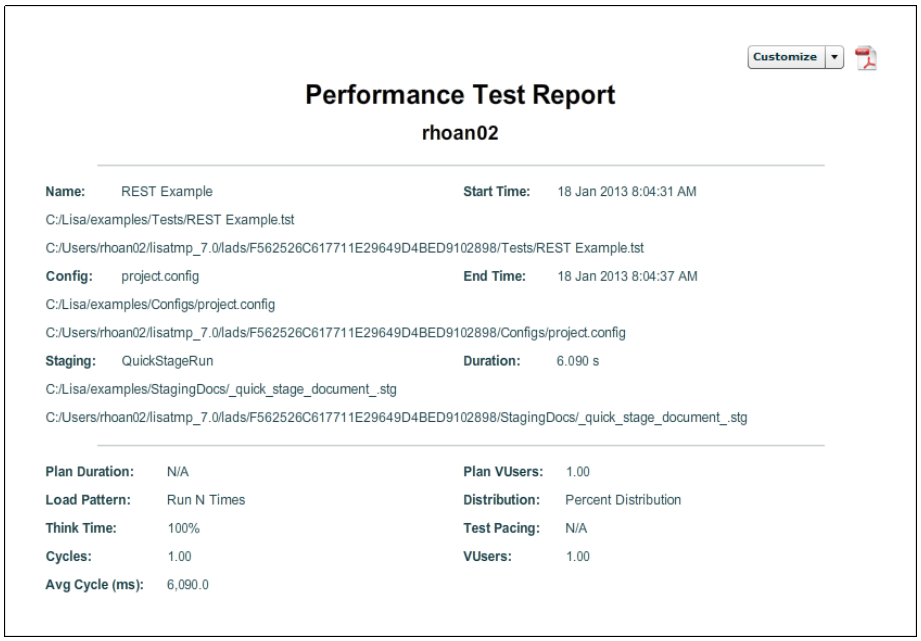


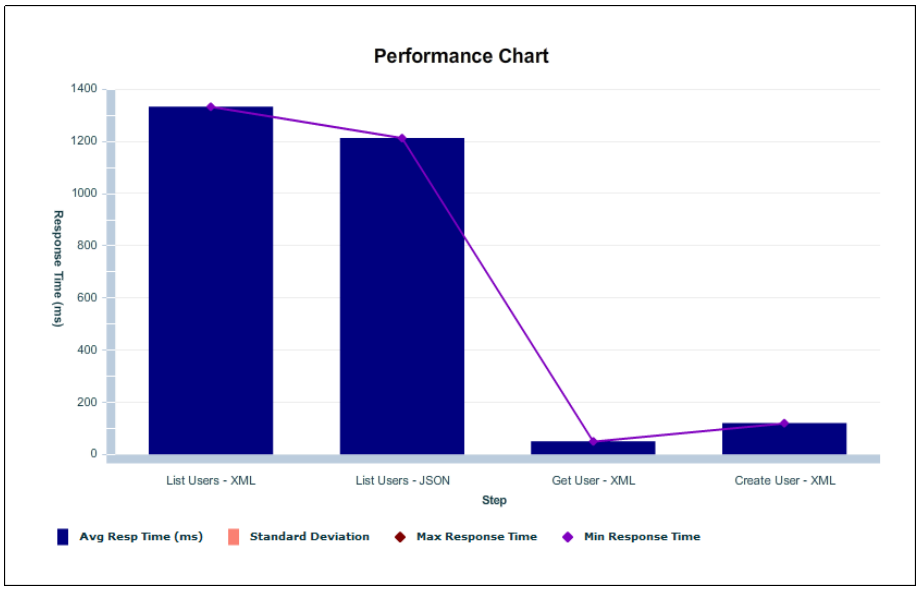
パフォーマンス テスト レポート

パフォーマンス テスト レポートを生成する方法

1. テスト ステップを右クリックします。
2. ［PDF にエクスポート］  をクリックします。

   ［レポートを選択してください］ウィンドウが表示されます。
3. 使用可能なレポートのリストから［パフォーマンス レポート］を選択し、［OK］をクリックします。





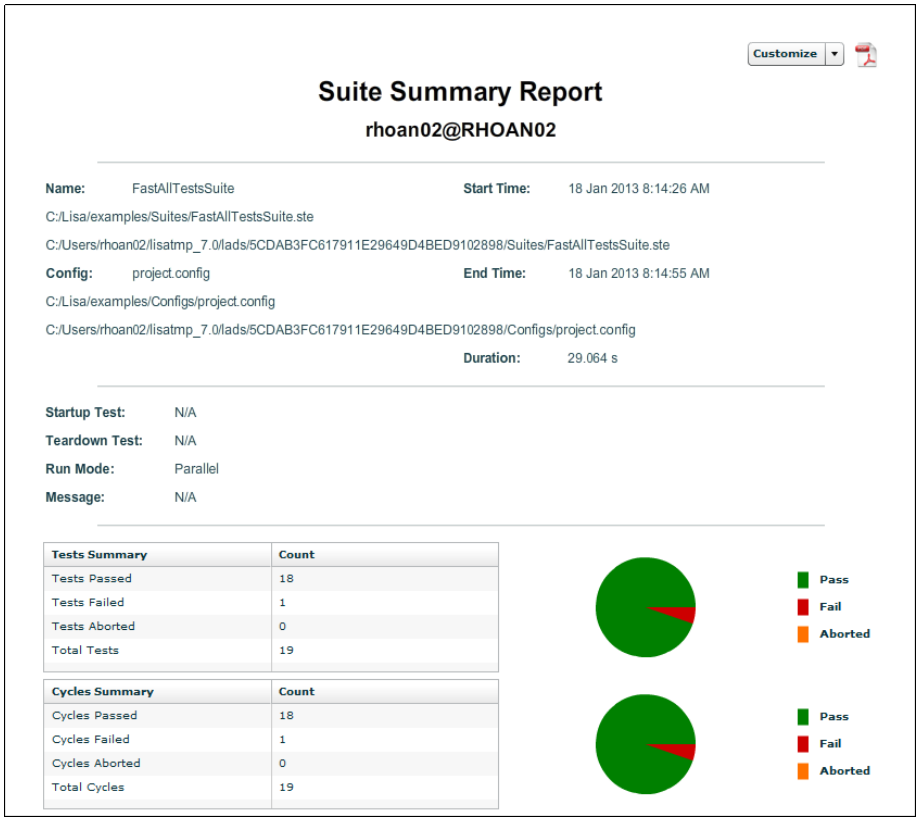
Step	Total Time (ms)	Sample Size	Min (ms)	Max (ms)	Avg (ms)	Median (ms)	Std Dev	90th percentile
List Users - XML	1,332	1	1,332	1,332	1,332.00	1,332.00	0.0	1,332.0
List Users - JSON	1,212	1	1,212	1,212	1,212.00	1,212.00	0.0	1,212.0
Get User - XML	50	1	50	50	50.00	50.00	0.0	50.0
Create User - XML	120	1	120	120	120.00	120.00	0.0	120.0

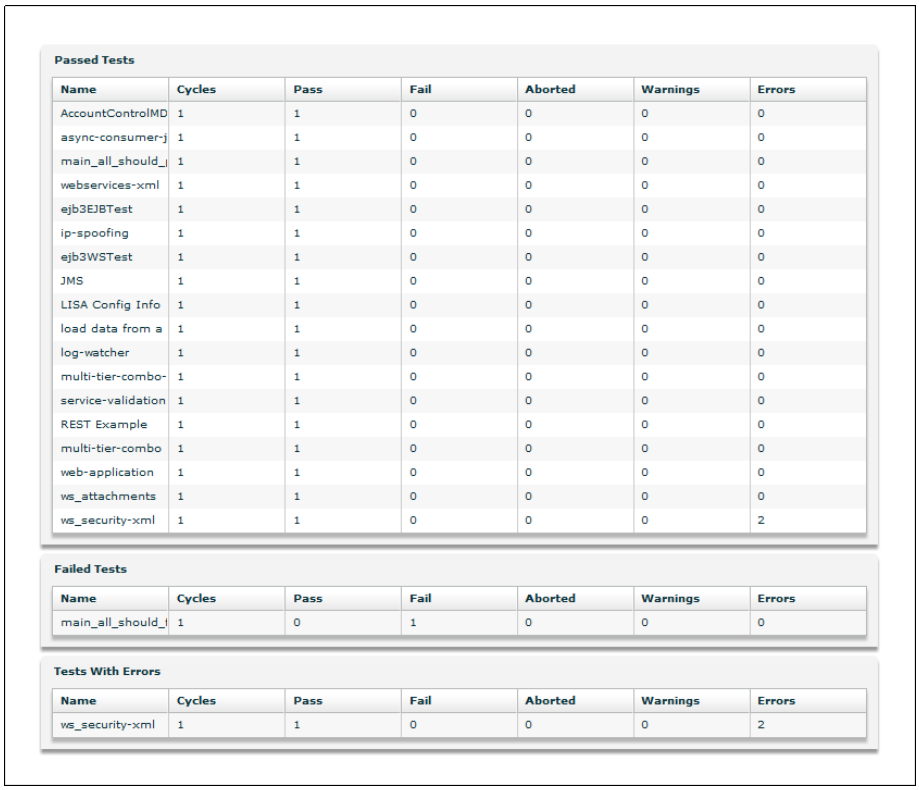
スイート サマリ レポート

スイート サマリ レポートを生成する方法

1. テスト スイート レポートを開きます。
2. ［PDF にエクスポート］  をクリックします。
3. ［レポートを選択してください］ウィンドウが表示されます。
4. 使用可能なレポートのリストから［スイート サマリ レポート］を選択し、［OK］をクリックします。

   ［カスタマイズ］ドロップダウンを使用すると、失敗したテストのタイプについての詳細を表示できます。すべてのテストが成功した場合、詳細ビューはサマリ ビューと同じです。





Passed Tests

Name	Cycles	Pass	Fail	Aborted	Warnings	Errors
AccountControlMD	1	1	0	0	0	0
async-consumer-j	1	1	0	0	0	0
main_all_should_	1	1	0	0	0	0
webservices-xml	1	1	0	0	0	0
ejb3EJBTest	1	1	0	0	0	0
ip-spoofing	1	1	0	0	0	0
ejb3WSTest	1	1	0	0	0	0
JMS	1	1	0	0	0	0
LISA Config Info	1	1	0	0	0	0
load data from a	1	1	0	0	0	0
log-watcher	1	1	0	0	0	0
multi-tier-combo-	1	1	0	0	0	0
service-validation	1	1	0	0	0	0
REST Example	1	1	0	0	0	0
multi-tier-combo	1	1	0	0	0	0
web-application	1	1	0	0	0	0
ws_attachments	1	1	0	0	0	0
ws_security-xml	1	1	0	0	0	2

Failed Tests

Name	Cycles	Pass	Fail	Aborted	Warnings	Errors
main_all_should_	1	0	1	0	0	0

Tests With Errors

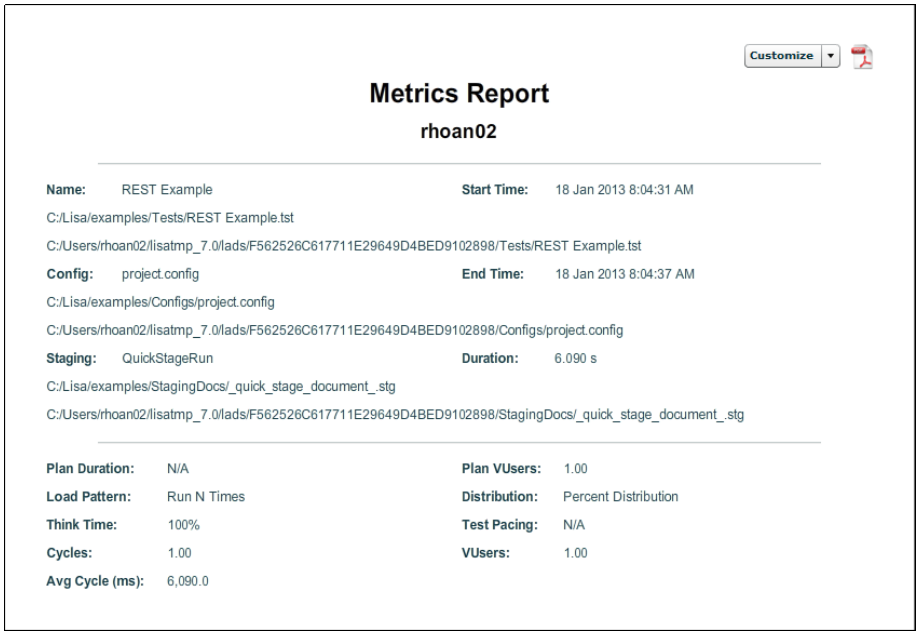
Name	Cycles	Pass	Fail	Aborted	Warnings	Errors
ws_security-xml	1	1	0	0	0	2

メトリック レポート

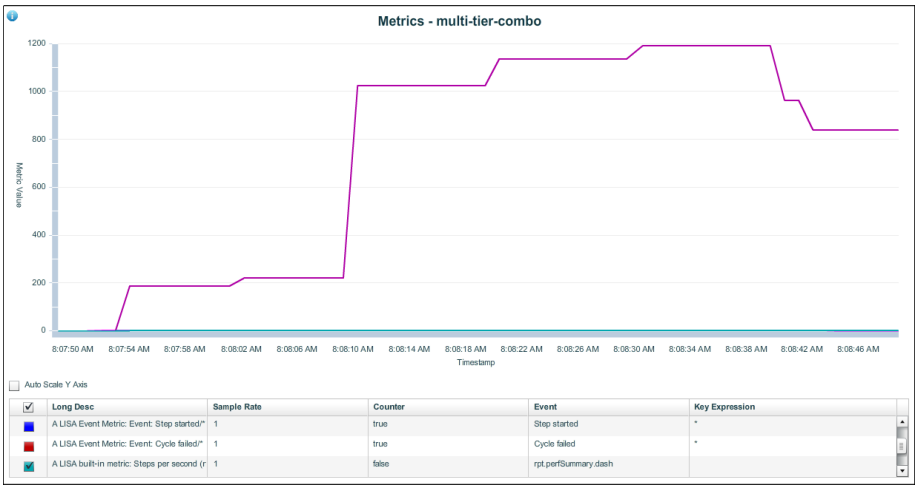
メトリック レポートを生成する方法

1. テスト/スイート レポートで、テストまたはスイートを右クリックします。
2. ［分析］-［メトリック］を選択して、メトリック チャートまたはグリッドを表示します。
3. そのレポートから、［PDF にエクスポート］  をクリックします。

   ［レポートを選択してください］ウィンドウが表示されます。
4. 使用可能なレポートのリストから［メトリック レポート］を選択し、［OK］をクリックします。



メトリック チャートをウィンドウで表示するために右クリックすると、凡例がインタラクティブに表示され、イベントおよび自動スケール パラメータを選択できます。 PDF レポートでは、［カスタマイズ］ドロップダウンを使用して選択します。

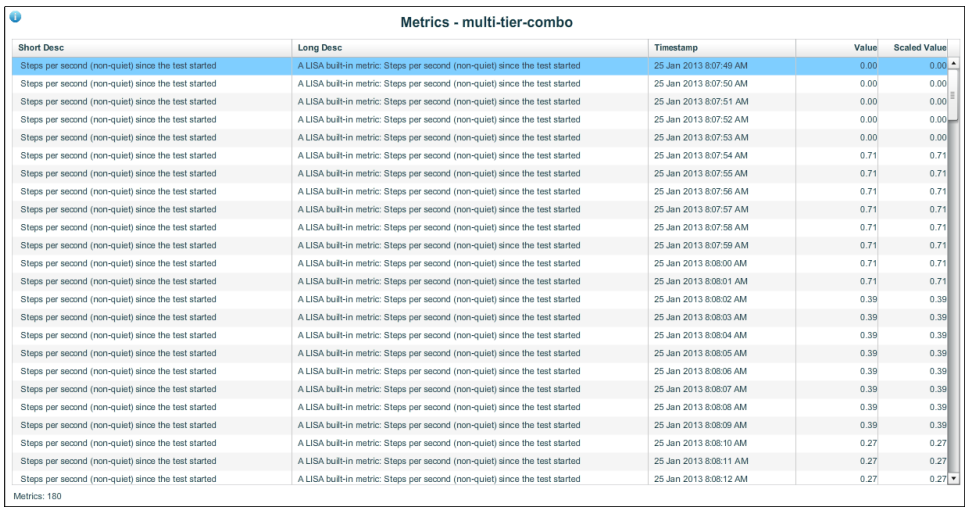


［カスタマイズ］ボタンでは、以下のオプションを選択できます。

- サマリの詳細
- Y 軸を自動スケール
- すべてのメトリック
- メトリックなし
- メトリック（選択）

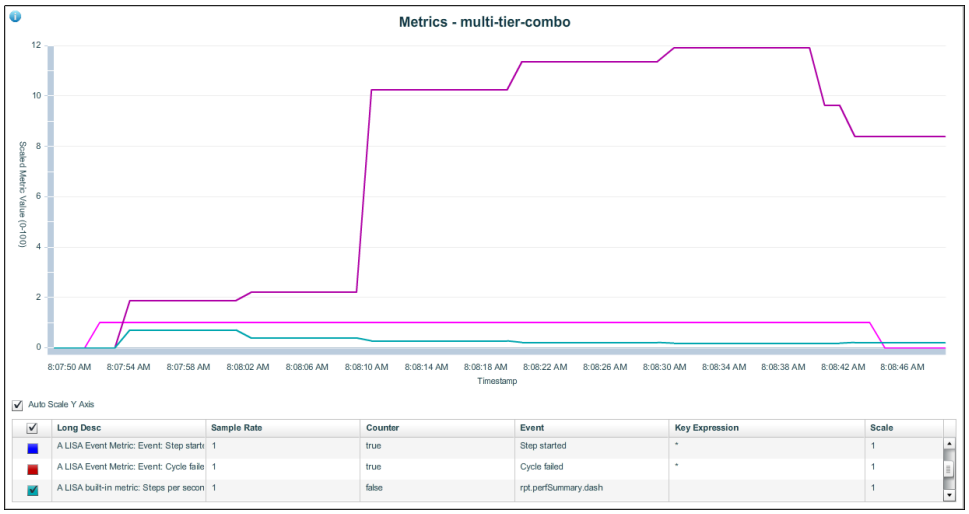
以下に、カスタマイズされたレポートの例を示します。

サマリの詳細



Y 軸を自動スケール

Y 軸の自動スケールを理解するため、このレポートを前記の同じレポートと比較してください。



すべてのメトリック

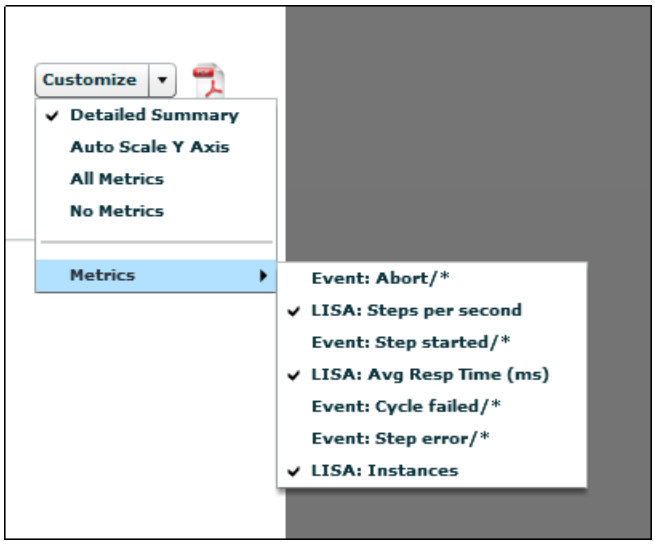
［すべてのメトリック］は、メトリック レポートのデフォルトです。［カスタマイズ］ドロップダウンで［メトリック］または［メトリックなし］を選択している場合は、［すべてのメトリック］を選択して、レポートに使用可能なメトリックをすべて表示できます。

メトリックなし

［カスタマイズ］ドロップダウンから［メトリックなし］を選択した場合は、メトリック チャートからメトリックがすべてクリアされます。

メトリック

［メトリック］ドロップダウンの使用可能な個別のメトリックを選択することによって、メトリック チャートに表示されるメトリックを選択できます。


Customize
Detailed Summary
Auto Scale Y Axis
All Metrics
No Metrics
Metrics
Event: Abort/*
LISA: Steps per second
Event: Step started/*
LISA: Avg Resp Time (ms)
Event: Cycle failed/*
Event: Step error/*
LISA: Instances

レポートの解釈

レポートの結果が予期するものと異なることがあります。そのため、レポート データが正しくないように見える場合があります。

テストのループ

テスト ケース内のループによって、レポート エンジンに予期しない結果が生じる可能性があります。

ワークフロー エンジンは、開始ステップから終了ステップまで、ループなしに流れるように設計されています。 このステップのチェーンの実行は、成功、失敗、または中止イベントのいずれかと見なされます。 ループは、ステージング ドキュメントを使用して、テスト ケースの外部から開始されるように設計されています。 10 人の仮想ユーザを指定してあるテスト ケースを 1 回実行するステージング ドキュメントを使用すると、10 個の成功/失敗/中止イベントが発生します。 ループがテスト ケースで実行される場合、成功/失敗/中止は、最後のステップに到達するまで発生しません。したがって、成功/失敗/中止イベントが 1 つだけ作成されます。

反応時間およびレポート

反応時間とは、DevTest がテスト ステップの実行の開始を待機する時間です。 反応時間の目的は、任意のシステムで実際のユーザの通信をシミュレートすることです。 反応時間は、ステップ エディタまたはステージング ドキュメントで設定できます。

メイン レポート パネルには、テストの開始から終了までの実行時間の合計が表示されます。 これは期間であるため、設計による反応時間が含まれ、テスト全体が実行されるのにどれくらいかかったかを示します。 反応時間を除外するには、ステージング ドキュメントまたはテスト ステップで反応時間を 0 に設定します。

パフォーマンスの数値を確認している場合は、反応時間が含まれない各ステップの応答時間が得られるパフォーマンス レポートを生成します。これはパフォーマンス テストに有益なレポートです。 反応時間は、平均ステップ パフォーマンス時間の一部ではありません。

レポートのエクスポート

レポート ビューアでは、レポートを以下の 3 つの形式でエクスポートできます。

- XML
- Microsoft Office Excel
- Adobe Acrobat PDF

XML へのレポートのエクスポート

テスト スイートまたはステージング ドキュメントで XML レポートを指定している場合、XML ファイルにレポート データを直接エクスポートできます。 エクスポートするには、テストまたはスイートを右クリックし、［エクスポート］を選択します。 詳細については、「レポート - グラフィカル ビュー (P. 445)」を参照してください。 エクスポートされたデータを保存した後で、整形または操作できます。

XML にレポート データをエクスポートすると、あるレジストリから別のレジストリにレポート データを移動できます。

Excel へのレポートのエクスポート

Excel スプレッドシートにレポート データをすべてエクスポートできます。

次の手順に従ってください:

1. エクスポートするレポートを開きます。
2. ［グリッド］チェック ボックスをオンにします。

   Excel にエクスポートするには、レポートがグリッド ビューである必要があります。
3. ［Excel にエクスポート］をクリックします。
4. Excel ファイルの名前を指定します。

   レポート データが保存した Excel ファイルに格納されます。

PDF へのレポートのエクスポート

PDF にもレポート データをエクスポートできます。

次の手順に従ってください:

1. レポートのグラフィカル ビューを開きます。
2. ［PDF にエクスポート］アイコンをクリックします。
   ［レポートを選択してください］ウィンドウが表示されます。
3. エクスポートするレポートを選択します。

レポート データベースの変更

別のデータベースに接続するように DevTest を設定する方法を、**lisa.properties** または **site.properties** で設定できます。LISA データベース コンポーネントは、以下のように、定義されたプール設定に各コンポーネントを割り当てることによって設定します。

```
lisadb.reporting.poolName=common
lisadb.vse.poolName=common
lisadb.legacy.poolName=common
lisadb.acl.poolName=common
```

デフォルトでは、すべての DevTest データベースは、ここで定義されるような共通の Derby データベース接続プールを共有しています。

```
lisadb.pool.common.driverClass=org.apache.derby.jdbc.ClientDriv
er
lisadb.pool.common.url=jdbc:derby://localhost:1528/database/lis
a.db;create=true
lisadb.pool.common.user=rpt
lisadb.pool.common.password=rpt
```

たとえば、レポート データベースが Oracle に接続するように変更する場合、まず新しいデータベース プール設定を定義します。

```
lisadb.pool.mypool.driverClass=oracle.jdbc.OracleDriver
lisadb.pool.mypool.url=jdbc:oracle:thin:@//myhost:1521/orcl
lisadb.pool.mypool.user=rpt
lisadb.pool.mypool.password=rpt
```

その後、レポート コンポーネントがそのプールを使用するように設定します。

```
lisadb.reporting.poolName=mypool
```

レポート ポータルに入る場合は、ウィンドウの右上隅のデータベース アイコンにマウス カーソルを置き、使用するデータベースの詳細を取得します。

レポートのパフォーマンスのトラブルシューティング

負荷テストを実行し、テスト ケースのボトルネックがレポート データベースへのイベントの書き込みであると判明した場合は、レポート ジェネレータからメトリック以外のすべてを削除することを検討します。

通常、問題は負荷テストの実行時に DevTest が記録しようとするデータが多すぎることです。 テスト スイート エディタの［レポート］タブで、以下の順にイベントをオフにしてみます。

設定/参照されたプロパティ

このイベントは、データベース内の大半の行を生成し、負荷テストにほとんど役立ちません。 各ステップの実行では、プロパティの取得および設定がすぐに 10 以上になることを考慮してください。 負荷テストでは、このイベントによりデータベースに行が追加され、短期間に数百万行にもなります。 たとえば、テスト ケースのステップが 5 つあり（各ステップに 5 つの取得および 5 つの設定）、100 ユーザを 10000 サイクル実行すると、データベース内の行数は 5000 万行になります。

すべてのイベントを記録

このイベントは、基本的にワークフロー エンジンのレコーディングです。 このイベントでも、ステップごとに容易に 5 行以上の行が生成されることがあります。

要求/応答

このイベントは余分な行を作成しませんが、ペイロードが大きい可能性があるので CLOB として格納されます。 最大量のデータを作成する可能性がありますが、テーブルおよびインデックスが同じように急速に増加するとは限らないため、データベースへの影響は小さくなります。

それでもレポート エンジンがボトルネックと考えられる場合は、**lisa.reporting.useAsync**=false プロパティを有効にすることを検討します。このプロパティは、JMS を使用してレポート イベントを送信するように DevTest に指示します。 シミュレータおよびコーディネータのバックグラウンド スレッドは、データベースにイベントを非同期で書き込みます。これは、すべてのイベントがデータベースに書き込まれる前に負荷テストが終了することを意味します。そのため、しばらくの間、レポートは表示されません。 その時間は、テスト ケースと生成されたイベント数によって異なります。 シミュレータのキューは、通常、フラッシュに最も時間がかかります。シミュレータのログには、完了したパーセンテージを示す INFO レベルのメッセージが書き込まれます。

非同期レポート機能は、データベースへの書き込み速度を低下させないため、シミュレータの実行速度を速くすることができます。その代わりに、データは JMS キューに入れられて後で書き込まれます。

第 18 章: レコーダおよびテスト ジェネレータ

DevTest ワークステーション のコードなしのテスト環境を使用すると、QA、開発、およびその他で、迅速な設計や機能テスト、ユニット テスト、レグレッション テスト、負荷テストを動的な Web サイト（RIA）に対して実行できます。

豊富なブラウザおよび Web ユーザ インターフェース、および UI の下に存在する多くの構成要素およびデータをテストするためにこの製品を使用できます。 DevTest では、要件が満たされていることを確認するために、チームが機能テストに必要とするすべてのデータおよび実装レイヤに対して、分析、呼び出し、検証が可能です。

このセクションには、以下のトピックが含まれています。

Web サービスの生成

Web サービス（XML）テスト ケースを作成できます。 Web サービス実行（XML）ステップを使用して、テスト ケース内の Web サービス操作をコールし、応答および要求をテストします。 これらの Web サービス操作は、チュートリアル 7 で使用した EJB のメソッド コールと同等の機能です。 Web サービス実行（XML）ステップの詳細については、「*CA Application Test の使用*」の「Web サービス実行（XML）ステップ」を参照してください。

このステップを使用するには、デモ サーバを実行していることを確認します。

Web サービスの生成には、以下の手順が含まれます。

1. Web サービス（XML）ステップの作成（P. 480）
2. Web サービス クライアントの作成（P. 480）
3. テスト ケースの実行（P. 480）

Web サービス（XML）ステップの作成

次の手順に従ってください:

1. DevTest ワークステーション でモデル エディタを開きます。
2. モデル エディタ ウィンドウで右クリックし、［ステップの追加］-［Web/Web サービス］-［Web サービス実行（XML）］を選択します。

   Web サービス ステップが追加されます。
3. ［ステップ情報］領域の **addUser** ステップの名前を変更します。
4. addUser ステップをダブルクリックします。
5. Web サービス実行エディタが開きます。
6. XML ドキュメントを作成するには、［新規ドキュメント］をクリックします。

Web サービス クライアントの作成

次の手順に従ってください:

1. ［WSDL URL］フィールドに WSDL の場所を入力します。

   `http://WSSERVER:WSPORT/itko-examples/services/UserControlService?wsdl`
2. ［サービス名］フィールドに「**UserControlServiceService**」と入力します。

   Web サービスの名前には、スペースを使用しないでください。
3. ［ポート］フィールドに「**UserControlServiceService**」と入力します。
4. ［操作］リストから、テストする操作を選択します。
5. テストのエラー時に実行されるアクションに、［エラー時］リストから［テストを中止する］を選択します。

テスト ケースの実行

次の手順に従ってください:

1. ［実行］  をクリックします。

   テストが実行され、要求および応答が表示されます。
2. 実行時の要求を表示するには、［要求］タブをクリックします。
3. 実行時の応答を表示するには、［応答］タブをクリックします。

Web サイトの記録

DevTest ワークステーション には、プロキシ レコーダを使用して Web サイト テスト ケースをテストする HTTP レコーダが用意されています。

これは、Web サイトに対するパスの追跡を支援し、レコーディング中に生成される各 HTTP 要求のテスト ステップを自動的に作成します。

注: レコーディングの前には、ブラウザのキャッシュを無効化またはフラッシュして、Web レコーダがマルチバイト エンコーディング システムのサーバ応答パケットからクライアント要求パラメータ エンコーディングを取得できるようにしてください。 Web レコーダが応答パケットをキャプチャできない場合、デフォルトでは ISO-8859-1 でパラメータをデコードまたはエンコードするため、キャプチャされたパラメータが文字化けする可能性があります。

レコーディングを開始する方法

1. ［レコーディング］  をクリックします。
2. ［ユーザ インターフェース用テスト ケースを記録］-［Web レコーダ（HTTP プロキシ）］をクリックします。

   また、メイン メニューから［アクション］-［ユーザ インターフェース用テスト ケースを記録］-［Web レコーダ（HTTP プロキシ）］を選択することもできます。

   これにより、HTTP テストを記録および再生するために使用するブラウザを起動できます。

   - アクティブなタブにすでに開いているテスト ケースがある場合、ブラウザでテストを再生するかどうかを尋ねられます。
   - DevTest ワークステーション 内に開いているテスト ケースがない場合、記録する Web サイトの名前を入力する［テスト レコーダ］ウィンドウが表示されます。
3. テストする Web ページの URL を入力します。

   注: たとえば、「localhost」と入力します。 **http://localhost:8080/lisabank/** は、HTTP プロキシ レコーディングでは動作しません。URL では、「localhost」の代わりにホスト名または IP アドレスを使用します。
4. 以下から基本設定を選択します。

   - HTML 応答のみ： HTML 応答のみをキャプチャします。

- 外部ブラウザを使用： 外部ブラウザ ウィンドウを開きます。

ポートの使用方法

デフォルトでは、DevTest プロキシ レコーダはポート 8010 をレコーディングに使用します。

このポートを使用しない場合、**lisa.editor.http.recorderPort=8010** プロパティを使用して、**lisa.properties** ファイルの設定を上書きできます。

レコーディングを開始するには、［レコーディングを開始］をクリックして Web レコーダを開始するか、または［プロキシ設定］をクリックしてプロキシを設定します。

以下のトピックが含まれます。

- プロキシの設定 (P. 483)
- レコーディングの開始 (P. 484)
- 記録されたトランザクションの表示 (P. 485)
- ITR における表示 (P. 485)

プロキシの設定

［プロキシ設定］ウィンドウを開くには、［テスト レコーダ］ウィンドウの［プロキシ設定］をクリックします。 以下のプロキシ設定を設定できます。

プロキシを使用

プロキシ設定を使用するには、このオプションを選択します。 このオプションは、プロキシ設定を入力してプロキシを必要に応じて使用する場合に役立ちます。

Web プロキシ サーバ

プロキシ サーバのホスト名（サーバ IP）とそのポート番号を入力します。

プロキシ サーバを使用しないホストおよびドメイン

プロキシを使用しないサーバのホスト名およびドメインを入力します。 プロキシ サーバを使用せずに直接接続するホストのパイプ（|）区切りリストを入力します。 アスタリスク（*）を一致のワイルドカード文字として使用できます（例：「*.foo.com|localhost」）。

セキュア Web プロキシ サーバ

セキュア プロキシ設定を入力します。

プロキシ サーバを使用しないホストおよびドメイン

プロキシを使用しないホスト名およびドメインを入力します。

単純なホスト名を除外

単純なホスト名（たとえば、**192.168.1.1** や **server1.company.com** ではなく **localhost** や**サーバ名**）を除外する場合に選択します。

プロキシ サーバ認証

以下の認証情報を入力します。

- ドメイン
- ユーザ名
- パスワード

事前送信

［チャレンジを待機］、［ベーシックを送信］、または［NTLM を送信］を選択します。

注: 認証が必要な場合、プロキシ サーバは通常、すべての要求でチャレンジを送信します。プロキシ サーバがチャレンジを送信しない場合、このフィールドを設定すると最初の要求に認証ヘッダを強制的に設定します。

変更を保存してプロキシ設定を設定するには、［OK］をクリックします。

レコーディングの開始

次の手順に従ってください:

1. テストのレコーディングを開始するには、テスト ジェネレータの［レコーディングを開始］をクリックします。

   ［テスト レコーダ］ウィンドウが開き、ロードした Web ページの URL が表示されます。

   注: Unicode 文字を記録するには、外部ブラウザを使用してください。

2. ユーザと同じように情報を入力することで、Web ページをテストできます。
3. 入力の完了後、ブラウザ アクティビティのレコーディングを停止するには、［テスト レコーダ］ウィンドウの下部の［レコーディングの停止］をクリックします。

   ［記録されたエレメント］ウィンドウが表示されます。

4. ［編集内容をコミット］をクリックして、レコーディングを完了します。

記録されたトランザクションの表示

レコーディングを停止した後、記録されたトランザクションはすべて［記録されたエレメント］タブに表示されます。

すべてのトランザクションが左側のパネルにリスト表示されます。 ステップ詳細および応答は、右側のタブに表示されます。

次の手順に従ってください:

1. ［応答］タブを選択して、記録された HTML 応答を表示します。
2. ［テスト レコーダ］内のこれらのトランザクションをコミットするには、［編集内容をコミット］をクリックします。

   ［Web レコーディングのパラメータ］パネルが表示されます。
3. テストに必要なパラメータを入力します。
4. テスト ステップとしてトランザクションを追加するには、［テスト レコーダ］ウィンドウの下部の［テストに追加して閉じる］をクリックします。

   テスト ケースは、DevTest ワークフローの HTTP 要求に基づいて作成されます。 テスト ケースの各ステップは、記録された HTTP 要求を表しています。

ITR における表示

ITR でこのテスト ケースを実行すると、これらすべてのトランザクションを再度表示できます。

記録されたステップの詳細を表示するには、［表示］-［ソース］-［DOM ツリー］タブを参照します。

モバイル テスト ケースの記録

次の手順に従ってください:

1. 目的のモバイル アセットを定義する設定ファイルがアクティブであることを確認します。
2. テスト ケースを作成 (P. 234) します。
3. ［レコーディングまたはテンプレートでステップを作成］  をクリックします。
4. ［ユーザ インターフェース用テスト ケースを記録]-[モバイル レコーダ］をクリックします。

   ［テスト レコーダ］ウィンドウが表示されます。

   注: モバイル シミュレータを使用して記録する場合、モバイル シミュレータ ウィンドウも表示されます。
5. 複数のモバイル アセットを定義した場合、［Choose Mobile asset （モバイル アセットの選択）］ドロップダウン リストから接続するアセットを選択し、［OK］をクリックします。
6. レコーダ ウィンドウの下部にある［レコーディングを開始］をクリックします。
7. ［テスト レコーダ］ウィンドウで、テスト ケースに対して記録するアクションを実行します。

   重要: モバイル シミュレータでこれらのアクションを直接実行しないでください。 DevTest ワークステーション は、［テスト レコーダ］で実行するアクションをシミュレータに自動的に送信します。

   ［テスト レコーダ］では、ユーザが対話する各画面エレメントが強調表示されます。赤枠は画面位置を示します。緑枠はより詳細な画面エレメントを示します。 特定の画面エレメントをポイントすると、その画面エレメントのツールヒント ウィンドウが表示されます。ボタンまたはコンポーネントに複数の可能な動作がある場合、ツールヒントにクリックした時点のヒントが表示されます。
8. レコーディング中にアクションまたはジェスチャを手動で追加するには、［テスト レコーダ］内の画面または特定のエレメントを右クリックし、挿入するアクションを選択します。

たとえば、レコーディング プロセス中に「シェイク」の手動挿入、向きの変更、特定のアサーションの挿入を実行できます。 使用可能なアクションの詳細については、「モバイル テスト ステップの変更」を参照してください。

注: モバイル テスト ケースの 1 つのステップには、ステップを完了するために必要な数のジェスチャ（タップ、スワイプ）が含まれます。テスト ステップをダブルクリックすると、これらのサブステップ（アクション）が［アクション］テーブルに表示されます。

9. テスト用のアクションをすべてキャプチャしたら、［レコーディングの停止］をクリックします。

   レコーダ ウィンドウおよびモバイル シミュレータが閉じます。 新しいテスト ケースには、レコーディング中にキャプチャされたモバイル アクションを表すテスト ステップが生成されます。

10. 各ステップの詳細を表示するには、以下を実行します。

    a. 確認するテスト ステップをクリックします。各テスト ステップには、実行するアクションのスクリーンショットが含まれます。

    b. 詳細を展開するには、右側のエレメント ツリーの［モバイル テスト ステップ］をクリックします。

       ［モバイル テスト ステップ］タブが開き、テスト アプリケーションのスクリーンショットが表示されます。タブの上部の［アクション］セクションには、テスト ステップで実行される個別のアクションが表示されます。 スクリーンショット内の関連するエレメントを強調表示するには、アクションをクリックします。

    c. 特定のアクションと関連付けられたスクリーンショットを表示するには、［アクション］セクション内のアクションをクリックします。

       記録されたテスト ステップの変更の詳細については、「モバイル テスト ステップの変更」を参照してください。

       テスト ステップへのアサーションの追加の詳細については、「アサーションのモバイル テスト ステップへの追加 (P. 167)」を参照してください。

ITR でのモバイル テスト ケースの実行

次の手順に従ってください:

1. 目的のモバイル アセットが含まれる設定ファイルがアクティブであることを確認します。
2. 実行するテスト ケースを開きます (P. 235)。
3. ツールバーの［新しい ITR を開始］  をクリックします。

   注: 新しい ITR 実行を開始するか、前の ITR 実行を開くか（ITR を以前に実行している場合）を選択します。

   ［対話型テスト ラン（ITR）］ウィンドウが表示されます。
4. ITR ウィンドウの下部にある［実行］  をクリックします。

   モバイル シミュレータ ウィンドウが開き、DevTest はテスト ステップを実行します。テストが実行されている間、シミュレータ上にモバイル アプリケーションを表示できます。

   テストが完了したことを示すメッセージが表示されます。シミュレータ ウィンドウが閉じます。
5. ［OK］をクリックします。

   ITR の使用方法の詳細については、［テスト ケースおよびスイートの実行 (P. 315)］を参照してください。

モバイル テスト ケースのリモートでの実行

次の手順に従ってください:

1. 予期されるモバイル アセットを定義する設定ファイルがアクティブであることを確認します。
2. 実行するテスト ケースを開きます (P. 235)。
3. ツールバーの［ITR］アイコン  をクリックします。

   注: 新しい ITR 実行を開始するか、前の ITR 実行を開くか（ITR を以前に実行している場合）を選択します。

   ［対話型テスト ラン（ITR）］ウィンドウが表示されます。
4. ITR ウィンドウの下部にある［実行］ をクリックします。

   モバイル シミュレータ ウィンドウが開き、DevTest はテスト ステップを実行します。

   テスト ケースの詳細を表示するには、クラウド プロバイダ（SauceLabs.com など）にログインします。テストが SauceLabs エミュレータ上で実行されているときに、ライブ スクリーンキャストとしてテストを表示できます。

   テストが完了すると、メッセージが DevTest に表示されます。
5. ［OK］をクリックします。

   ITR の使用方法の詳細については、［テスト ケースおよびスイートの実行 (P. 315)］を参照してください。

第 19 章: モバイル テスト

このセクションには、以下のトピックが含まれます。

モバイル テストの概要

このセクションには、以下のトピックが含まれます。

モバイル テストの概要

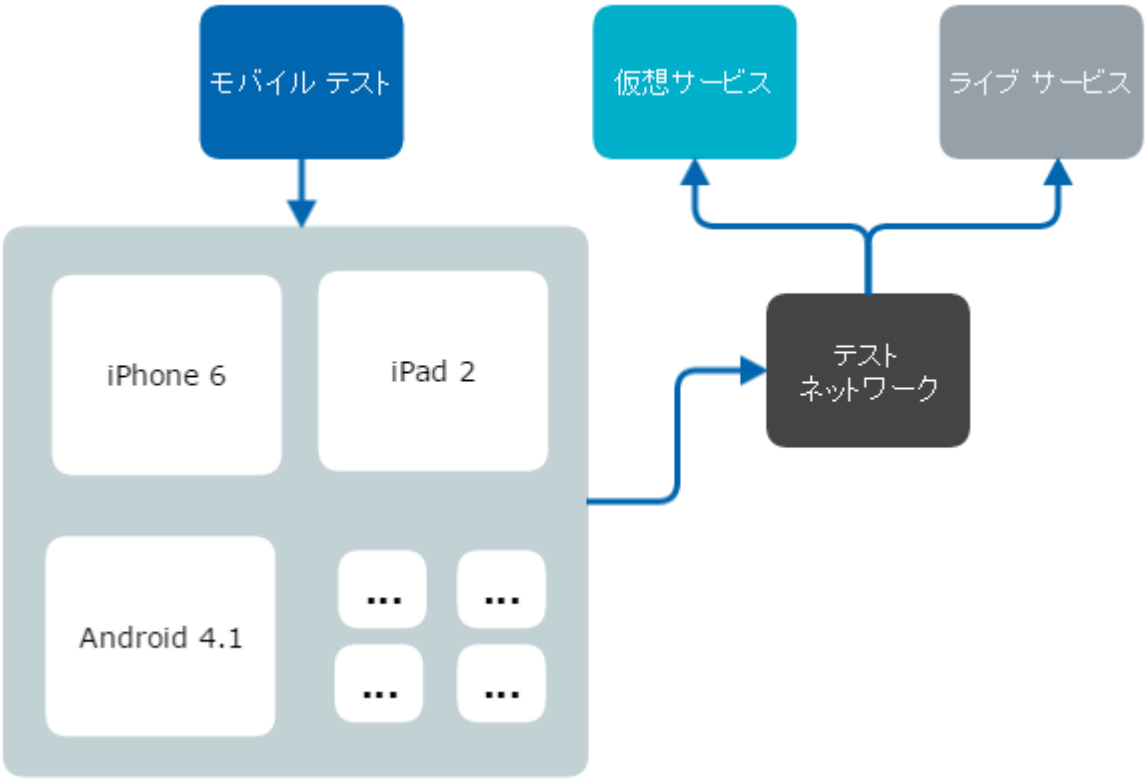
モバイル テストは、テスト ケースを開発、ステージング、モニタする基本的な DevTest の機能を、モバイル iOS、Android、およびハイブリッド アプリケーションのテストへと拡張します。 モバイル アプリケーションとのインタラクションを記録することにより、容易にテスト ケースを作成できます。 その後、個別のアクション、アサーション、およびフィルタを各ステップに追加することにより、その他の DevTest テスト ケースと同様にこれらのテストを変更できます。

モバイル テスト レコーダでは、以下の方法で記録できます。

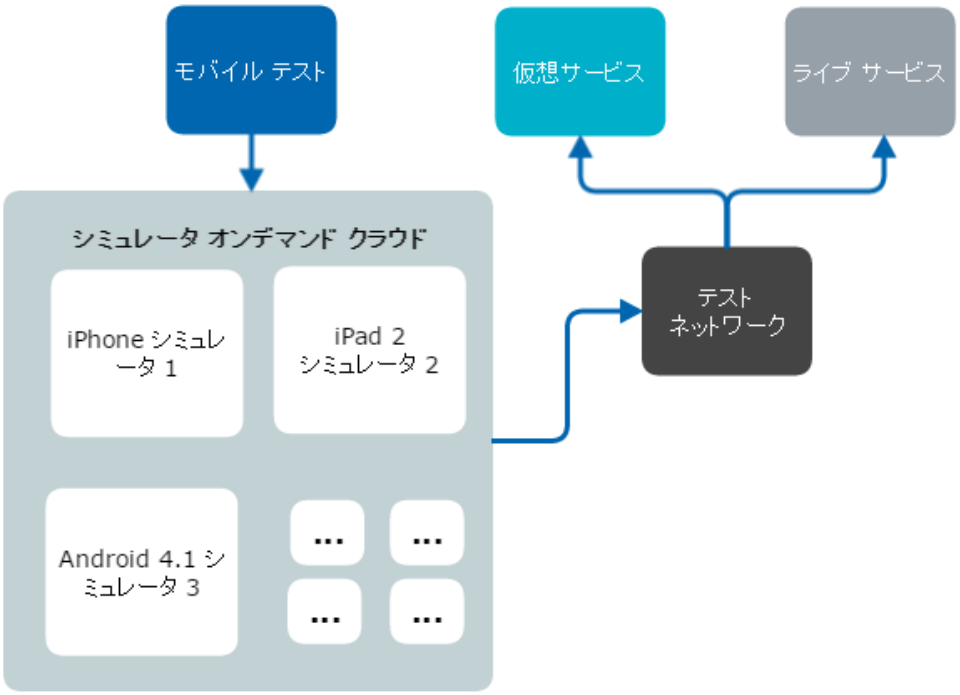
- USB ポート経由でユーザのマシンに接続されたデバイスから直接記録します。
- インストールされている Android および iOS シミュレータを使用して記録します。
- より拡張性の高いソリューションでは、クラウド ベースの Android および iOS シミュレータを使用して記録します。

モバイル テストでは、複数のデバイスに対してテストできる単一のテスト ケースを作成できます。 この機能を使用可能な各種シミュレータと組み合わせると、さまざまなモバイル デバイス用アプリケーションの開発およびテスト プロセスが大幅に簡略化されます。

以下の図は、接続デバイスおよびクラウド ベースのシミュレータを使用するモバイル テスト環境を示しています。



以下の図は、インストール済みおよびクラウド ベースのシミュレータを使用するモバイル テスト環境を示しています。



注: モバイル テストのセットアップの詳細については、「モバイル テスト環境の設定」を参照してください。

サポートされているモバイル アクションおよびジェスチャ

DevTest Solutions は以下のモバイル ジェスチャをサポートしています。

モバイル ジェスチャ	iOS ネイティブ アプリ	Android ネイティブ アプリ	iOS ハイブリッド アプリ	Android ハイブリッド アプリ
タップ	はい	はい	はい	はい
ロング タップ	はい	はい	はい	はい
エレメントのスワイプ	はい	はい	はい	はい
画面のスワイプ	はい	はい	はい	いいえ
ピンチ	はい	はい	はい	いいえ
ズーム	はい	はい	はい	N/A
回転	はい	はい	はい	N/A
スクロール	はい	はい	はい	N/A

向きの変更	はい	はい	はい	はい
シェイク	はい	はい	はい	N/A
戻る	いいえ	はい	いいえ	はい
バックグラウンドに移動	はい	N/A	N/A	N/A

モバイル テスト ケースを作成および再生する方法

モバイル テスト ケースを作成および再生するには、以下のタスクを完了させます。

次の手順に従ってください:

1. 以下のいずれかのアセットを作成します。

   SauceLabs セッション アセット

   モバイル クラウド テスト用の SauceLabs アカウント情報を定義します。

   シミュレータ セッション アセット

   テストに使用されるモバイル シミュレータを定義します。

   接続デバイス セッション アセット

   テストに使用されるモバイル デバイスを定義します。 このアセットは、ユーザのネットワークに接続される物理モバイル デバイスに使用されます。

   これらの各アセットの詳細については、「モバイル アセット」を参照してください。

   アセットの作成に関する一般情報については、「アセットの作成 (P. 130)」を参照してください。

2. モバイル テスト ケースを記録します (P. 486)。
3. 必要に応じてテスト ステップを変更します。
4. モバイル テスト ケースを実行します。
   - テスト ケースの実行に関する一般情報については、「テスト ケースおよびスイートの実行 (P. 315)」を参照してください。
   - ITR でのモバイル テスト ケースの実行の詳細については、「ITR でのモバイル テスト ケースの実行 (P. 488)」を参照してください。
   - モバイル テスト ケースのリモートでの実行の詳細については、「モバイル テスト ケースのリモートでの実行 (P. 489)」を参照してください。

リモート テスト

リモート テストを使用して、ある場所にあるコンピュータに接続される（その後、別のコンピュータが別の場所から接続する）物理 iOS および Android モバイル デバイスをテストします。リモート デバイスの可用性により、さまざまなオペレーティング システムおよびデバイスの組み合わせを使用できるテスト戦略を作成できます。

リモート テストでは、以下のようにさまざまなオペレーティング システムの組み合わせでモバイル デバイス テストを実行できます。

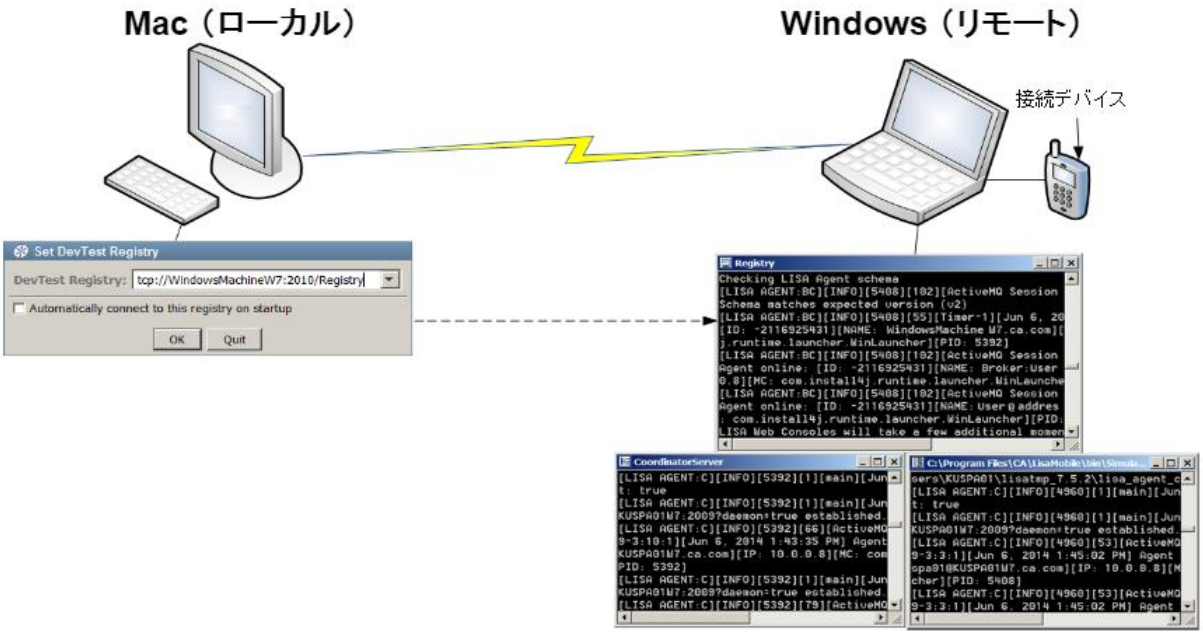
- Mac から Windows
- Windows から Mac
- Windows から Windows
- Mac から Mac

DevTest ワークステーション でのリモート テストは以下のプロセスを使用します。

1. DevTest ワークステーション をローカル マシンから、物理モバイル デバイスが接続されているリモート マシンで実行されているレジストリ、シミュレータ、およびコーディネータに接続します。
2. リモート テストを実行する前に、ローカル マシンにモバイル アセットを作成し、確認します。
3. ローカル マシンにテスト ケースを作成します。
4. 接続先のリモート マシンのレジストリ、シミュレータ、およびコーディネータに対して実行される、ローカル マシン上のテスト ケースをステージングします。

注: Windows マシンからは Android デバイスのテストのみを実行できます。Mac マシンからは Android および iOS の両方のテストを実行できます。

以下の手順では、Android デバイスで実行されるネイティブ アプリケーションのリモート テストを実行する方法について説明します。 この例では、Android デバイスは、レジストリ、シミュレータ、およびコーディネータを実行する Windows マシンに接続されます。 Mac マシン上の DevTest ワークステーション は、以下に示されているように、Windows マシン（「リモート マシン」） 上のレジストリ、シミュレータ、およびコーディネータに接続します。



次の手順に従ってください:

1. Windows マシン上のレジストリ、シミュレータ、およびコーディネータを起動します。
2. Mac マシン上の DevTest ワークステーション を起動します。
3. Windows マシン上で実行されているレジストリに接続します。 ［LISA レジストリの設定］ダイアログ ボックスに Windows マシン レジストリのアドレスを入力します。
4. Mac マシン上の DevTest ワークステーション に新しいプロジェクトを作成します。

5. Mac マシン上の LISA プロジェクトの Data フォルダに、.apk ファイル（ネイティブ Android アプリケーション）をコピーします。 アセットの［アプリケーション］フィールドに .apk ファイルの場所を指定します。 以下に例を示します。

   アプリケーション： *<LISA_PROJ_ROOT>*/Data/ApiDemos.apk

6. ローカル コンピュータに正常に記録されたテスト ケースおよび検証済みのアセットが含まれていることを確認します。 コンピュータにすでにこれらのアイテムが含まれている場合は、手順 10 に進みます。
7. Mac マシン上で、ApiDemos 用の Android Mac ネイティブの実デバイス アセットを作成します。
8. アセットをすべて検証します。
9. テストをローカルで記録および再生し、両方のアクションが手順 7 で作成したデバイス アセットを使用してローカル マシン上で正常に実行されることを検証します。
10. テスト ケースを右クリックし、［テストのステージング］をクリックします。
11. テストをステージングするには、［再生］をクリックします。

また、AllTestSuite 機能を使用してリモート ステージングを実行することもできます。

次の手順に従ってください:

1. ［AllTestSuite］を右クリックし、［DCM tcp://*<Connected Registry>* で実行］を選択し、テストをリモートでステージングします。

   ［スイートを tcp://*<Connected Registry>*/ を介して実行］ダイアログ ボックスが表示されます。

2. ［ステージング］をクリックします。

   テストが Mac マシン上で開始され、テスト結果が表示されます。 Mac 上の DevTest ワークステーション は、Windows マシン上で実行されるリモート レジストリ、シミュレータ、およびコーディネータに接続します。

   注: AllTestSuite ファイルにステージングしようとしているアイテム（実際のテスト、MAR ファイル、スイート）が含まれていることを確認してください。

Web ブラウザ テスト

Web テストを記録するため、DevTest では、ビルトイン Web ブラウザを備えたアプリケーションが提供されます。

次の手順に従ってください：

1. Web コンテンツをテストするプラットフォームとデバイスの組み合わせが含まれる、新しいアセットを作成します。
2. ［Mobile Session］ダイアログ ボックスの［アプリケーション］フィールドに、埋め込み Web ブラウザが含まれる .app ファイルまたは .apk ファイルをロードします。

   注: DevTest では、iOS および Android の両方用の埋め込み Web ブラウザ（Mobile Browser.app および MobileBrowser.apk）が含まれるアプリケーションがインストールされます。［アプリケーション］フィールドで、以下に移動します。
   `LISA_HOME¥examples¥Data¥mobile`
3. コンテンツが正しく見えることを確認するために、アプリケーションのテスト ケースを記録 (P. 486) します。

モバイル ラボ

DevTest モバイル ラボでは、デバイスのグループを取得できます。また、ローカル ネットワーク内のシミュレータ/エミュレータによって、CA Application Test でそれらのデバイスをすべて使用して対話できます。

環境で実行されている単一のコンピュータには、複数のデバイスが接続されている可能性があります。 モバイル ラボは、これらのデバイスをテスタで使用できるようにします。USB またはシミュレータによって接続される、最大 50 個の物理デバイスをテストして管理できます。 iOS および Android の両方と、複数のデバイス バージョンがサポートされています。

モバイル ラボには、DevTest Cloud Manager (P. 397) と同じ機能が含まれるほか、接続されたモバイル デバイスおよびエミュレータ/シミュレータを識別する機能が追加されました。

モバイル ラボの開始

仮想モバイル ラボを開始する場合は、ラボのインスタンスを起動します。

仮想モバイル ラボでは、モバイル アプリケーション テストを専用のハードウェア プール(たとえば、iOS アプリケーション テスト専用 OSX サーバ)に隔離できます。 仮想モバイル ラボを起動するには、DevTest において、テスト タスクを実行する 1 つ以上のシミュレータ サーバ プロセスと組み合わされたコーディネータ サーバ プロセスが、仮想ラボ インスタンスあたり少なくとも 1 つ必要です。1 つのサーバに 1 つのシミュレータ インスタンスとすることをお勧めします。コーディネータ サーバ プロセスは実際のテストを実行しませんが、正しくテストをスケジュールして結果をレポートするために必要です。

詳細については、「ラボの開始 (P. 409)」を参照してください。

DevTest サーバ 実行可能ファイルを呼び出し、サーバ名とラボ名を指定することでラボを開始できます。

次の手順に従ってください:

1. コマンド ラインから、モバイル ラボ コーディネータ サーバを起動します。

   `CoordinatorServer -l LabName`

   ここで、*LabName* はユーザ定義のラボ名を示します。1 つのラボに 1 つのコーディネータ プロセスとすることをお勧めします。

2. モバイル ラボ シミュレータを起動します。

   `Simulator -l LabName -n LabServerName`

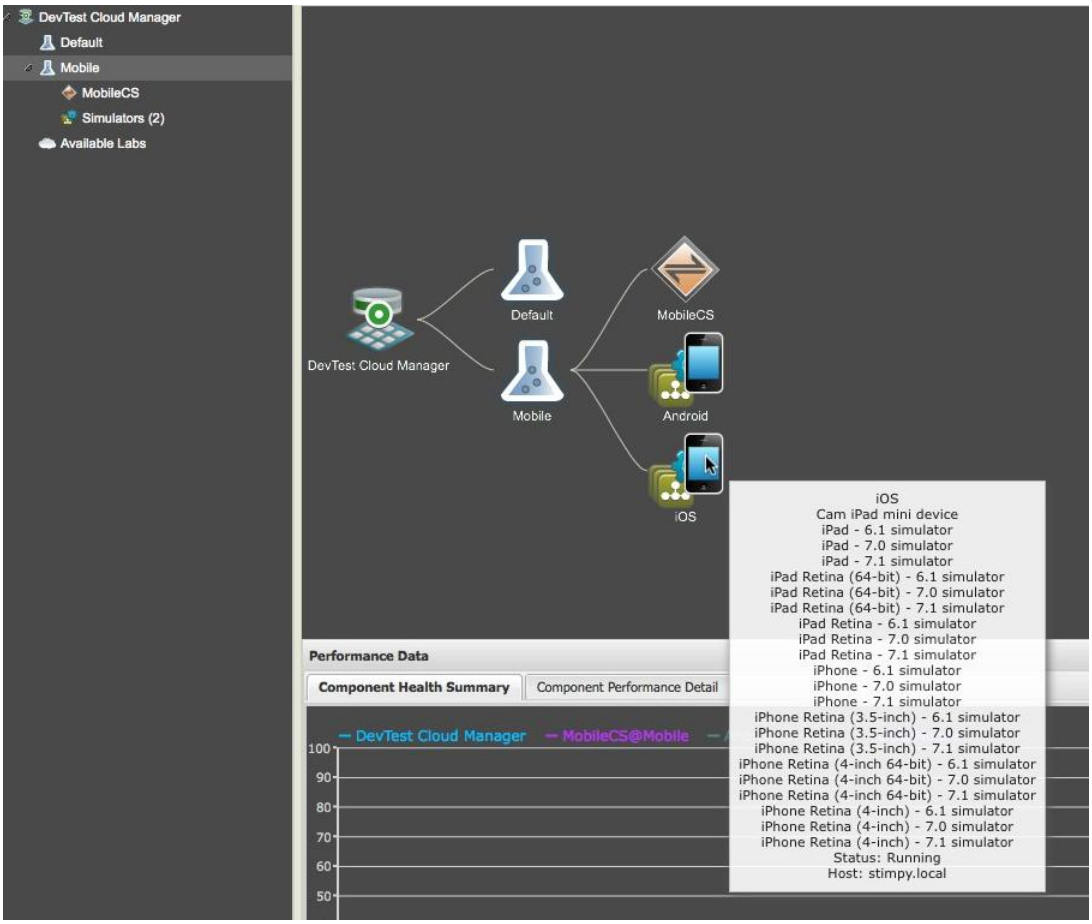
   ここで、*Labname* はユーザ定義のラボ名を示し、*LabServerName* はシミュレータの名前を示します。 1 つのラボ サーバに 1 つのシミュレータとすることをお勧めします。

   注: これらの実行可能ファイルは LISA_HOME¥bin ディレクトリにあります。

モバイル ラボの確認

サーバ コンソールによって、モバイル ラボの存在を確認できます。

マウス カーソルをシミュレータ アイコンに置くことにより、ラボ内の個別マシンのモバイル機能を確認します。機能セットでは、接続デバイス、iPhone シミュレータ イメージ、および Android AVD のリストが表示されます。



モバイル ラボでのテストのステージング

次の手順に従ってください:

1. プロジェクト パネルでテストを見つけます。
2. テストを右クリックし、［StageTest］を選択します。

   注: ラボでのテストのステージングを行うための、ラボ コーディネータ サーバを選択していることを確認します。

モバイル機能を有するラボのコーディネータのみが、あらかじめ構築されたコーディネータ リストにあるテストを使用できます。

モバイル ラボでのテストのスイートのステージング

次の手順に従ってください:

1. モバイル ラボにテストのスイートをステージングする前に、スイートに割り当てられたデフォルト コーディネータが存在することを確認します。 存在しない場合、スイートをステージングすると設定を求められます。
   a. エディタでテスト スイートを開きます。
   b. ［デフォルト］タブをクリックします。
   c. コーディネータのリストからラボ コーディネータを選択します。
2. スイートをステージングするには、プロジェクト パネルでスイートを右クリックします。
3. ［DCM で実行］を選択します。
4. ［ステージング］をクリックします。

Voyager による自動テスト

Voyager は、モバイル アプリケーションを調査してテスト ケースを作成する、アプリケーション エクスプローラ（すなわち「クローラ」）です。Voyager では、アプリケーション ファイルが検査されてから、アプリケーション全体にわたって、ファイルがすべてのベクター（ページ、リンク、ジェスチャ、および入力）に分割されます。 その後、これらのベクターが調査、実行されて、データが生成されます。

Voyager により、以下のことが可能です。

- **一連の自動テストの作成**

  Voyager を使用することで、テストをレコーダで記録し、変更し、テスト スイートに格納する時間を節約します。 Voyager によって、アプリケーション全体にわたってすべてのベクターが生成され、テスト ケースが作成されて自動化されます。

- **アプリケーションの特定のページに関するテスト ケースの作成**

  Voyager のデータの具体的な領域を特定して、その領域のテスト ケースを生成することができます。

- **Voyager データによる既存のテスト ケースへのステップの追加**

  Voyager データを取得し、そのデータを使用して既存のテスト ケースにステップを追加することができます。

テストの生成

Voyager を使用して、テスト ケースを自動的に生成できます。

次の手順に従ってください：

1. ［Actions］メニューで、［Launch Voyager］をクリックします。

   ［Voyager configuration］ダイアログ ボックスが表示されます。

2. ［App］フィールドで、Voyager がアクセスするアプリケーションを入力するか、または選択します。

3. ［App Graph］フィールドで、Voyager データが含まれるデータ ファイルの名前を入力するか、または選択します。

   注: テスト用のデータ ファイルは、プロジェクト パネルの Data フォルダにあります。

4. モバイル テスト ジェネレータをアクティブにするには、［Auto Generate Tests］をクリックします。

   注: 新しいページが見つかるたびに、テストが作成されます。Voyager が停止すると、データベースが保存されます。

5. ［Output Folder］フィールドで、.appgraph ファイルの場所を入力するか、またはデフォルトの場所を受け入れます。

6. SauceLabs アカウントまたはアクセス キーが変更された場合は、［Reconfigure SauceLabs］をクリックします。 SauceLabs が設定されていない場合、Voyager の起動前に［SauceLabs］ダイアログ ボックスが表示されます。

7. ［OK］をクリックし、Voyager を起動します。

   Voyager ダッシュボードが表示されます。ページがクロールされると、現在のページが表示されます。 一意のページの数が一番上に表示されます。

8. ［停止］をクリックし、Voyager を停止します。

ページ固有のテスト ケースの作成

ボイジャーを起動し、データ ファイルを作成してアプリケーション データをキャプチャすると、データの具体的な領域を特定して、その領域のテスト ケースを生成することができます。

次の手順に従ってください：

1. 新規モバイル テスト ケース (P. 496) を作成します。
2. テスト エディタで、［レコーディングまたはテンプレートでステップを作成］  をクリックします。
3. ［ユーザ インターフェース用テスト ケースを記録］-［Generate Mobile Steps］をクリックします。

   ［Page Selection］ダイアログ ボックスが表示されます。

   注: モバイル シミュレータを使用して記録する場合、モバイル シミュレータ ウィンドウも表示されます。 この状況は、Voyager を実行している場合にのみ発生します。 この場合、すでに記録されたデータからステップが生成されます。

   このダイアログ ボックスに入力するには、Voyager を最低 1 回実行しておく必要があります。
4. ドロップダウン リストから、アプリケーションとして［UICatalog］を選択します。
5. Voyager データベースとして、［Data/Voyager/UICatalog.appgraph］を選択します。
6. Voyager から、テストに含めるページを選択します。

既存のテスト ケースから Voyager ファイルへのモバイル ステップの追加

既存のモバイル テスト ケースから、モバイル テスト ステップを Voyager ファイルに追加できます。

次の手順に従ってください：

1. プロジェクト パネルのプロジェクト ツリーで、データが含まれる .appgraph ファイルを右クリックします。
2. ［Merge Test Case］を選択します。
3. 新しいステップを追加するテスト ケースを選択します。
4. ［OK］をクリックします。

   テストデータは、Voyager .appgraph ファイルにマージされます。

既存の Voyager ファイルからテスト ケースへのモバイル ステップの追加

Voyager のデータを使用して、既存のモバイル テスト ケースにステップを追加することができます。

次の手順に従ってください：

1. モバイル テスト ケースを開きます。
2. ［ユーザ インターフェース用テスト ケースを記録］-［Generate Mobile Steps］をクリックします。

   ［Page Selection］ダイアログ ボックスが表示されます。

   注: このダイアログ ボックスに入力するには、Voyager を最低 1 回実行しておく必要があります。
3. Voyager で記録したアプリケーション ファイルを選択します。
4. テストに追加するステップが含まれる、.appgraph ファイルを選択します。
5. 記録されたページをクリックします。
6. ページがデータベースに追加されたことを確認します。

モバイル テスト ケースのトラブルシューティング

- より多くのエラー メッセージを取得するには、DevTest ワークステーション の左下隅の［システム メッセージ］をクリックします。 このペインでは、問題を解決できる追加情報が提供されていることがよくあります。
- 追加のデバッグ情報を取得するには、LISA_HOME¥logging.properties で以下のプロパティを設定します。

  log4j.logger.com.itko.lisa.mobile=DEBUG

第 20 章: 高度な機能

このセクションには、以下のトピックが含まれています。

DevTest での BeanShell の使用

BeanShell （http:// www.beanshell.org/）は、フリーでオープン ソースの軽量な Java スクリプト言語です。 BeanShell は、リフレクション API を使用して Java ステートメントおよび式を動的に実行する Java アプリケーションです。 BeanShell の使用によって、クラス ファイルをコンパイルする必要がなくなります。

BeanShell では、コマンドラインに標準的な Java 構文（ステートメントおよび式）を入力して、結果をすぐに確認できます。 また、Swing GUI も使用可能です。 BeanShell は Java クラスからもコールできます。これは、この製品での使用方法です。

BeanShell は複数の場所で使用されます。

- プロパティ式を解釈するため
- JavaScript テスト ステップのインタープリタ フレームワークとして
- スクリプト実行によるアサート アサーションのインタープリタ フレームワークとして

詳細:

BeanShell スクリプト言語の使用

BeanShell Java とコンパイルされた Java の主な違いは、型システムにあります。 Java は強く型付けされますが、BeanShell はスクリプト環境で型付けを緩くできます。 ただし、必要に応じて、BeanShell での型付けを厳密に強制することができます。

BeanShell では、型付けが自然な方法で緩められているため、標準的な Java メソッド コードのように見える BeanShell スクリプトを作成することができます。ただし、Java 構文フレームワークを維持しつつ、Perl や JavaScript などの従来のスクリプト言語のように見えるスクリプトを作成することもできます。

変数が型付けされている場合、BeanShell は型をサポートおよびチェックします。変数が型付けされていない場合、BeanShell は変数の実際の型を誤って使用しようとした場合にエラーを返すだけです。

以下の Java フラグメントはすべて BeanShell で有効です。

```
foo = "Foo";
four = (2+2) * 2 / 2.0;
print(foo + " = " + four);
」を参照してください。
」を参照してください。
```

```
hash = new Hashtable();
date = new Date();
hash.put("today", date);
」を参照してください。
」を参照してください。
```

BeanShell では、メソッドを宣言し、その後にメソッドを使用できます。引数および戻り値型も緩く型付けできます。

型付け

```
int addTwoNumbers(int a, int b){
return a + b;
}
```

緩い型付け

```
add (a,b){
return a + b;
}
```

2 番目の例では、以下は正しく動作します。

```
sumI = add (5,7);
sumS = add("DevTest " , "Rocks");
sumM = add ("version ", 2);
```

BeanShell は、その使用を容易にするコマンドのライブラリも提供しています。

これらのコマンドの例を以下に示します。

- **source()：** BeanShell （bsh）スクリプトを読み込みます。
- **run()：** bsh スクリプトを実行します。
- **exec()：** ネイティブ アプリケーションを実行します。
- **cd()**、**copy()** など： UNIX ライクなシェル コマンド。
- **print()：** 文字列として引数を出力します。
- **eval()：** コードとして文字列の引数を評価します。

詳細については、http://www.postgresql.org にある BeanShell のユーザー ガイドを参照してください。

また、同じ場所で、BeanShell、ソース コード、および完全な Javadoc を取得できます。

スタンドアロンでの BeanShell の使用

BeanShell はスタンドアロン インタープリタとして使用可能なため、DevTest の外部で試行できます。 BeanShell は、www.beanshell.org からダウンロードできます。 このダウンロードは、bsh-xx.jar （xx はバージョン番号。現在は 2.0）という名前の単一の小さな JAR ファイルです。 この JAR ファイルをクラスパスに追加します。

以下の設定で BeanShell を使用することができます。

- **コマンドラインから：** java bsh.Interpreter [スクリプト名] [引数]
- **BeanShell GUI から：** java bsh.Console
- **Java クラスから：**

  ```
  Import bsh.Interpreter;
  ```
  」を参照してください。
  」を参照してください。

```
Interpreter I = new Interpreter();
i.set ("x",5);
i.set("today", new Date());
Date d = (Date)i.get("date");
i.eval("myX = x * 10");
System.out.println(i.get("myX"));
```

DevTest での BeanShell の使用

BeanShell インタープリタは、JavaScript 実行ステップおよびスクリプト実行によるアサートアサーションで使用されます。 これらのエレメントは、どちらも DevTest Java オブジェクトと現在の DevTest の状態（プロパティ）を公開します。 これにより、カスタム機能を追加するための強力な環境が実現します。 公開された Java オブジェクトは、テストの現在の状態の確認と変更の両方に使用できます。 たとえば、スクリプト内の DevTest プロパティの読み取り、変更、作成ができます。

出発点として、DevTest の **TestExec** クラスを理解してください、TestExec およびその他の多くのクラスの情報については、「*SDK の使用*」を参照してください。

等号が存在する場合、DevTest は BeanShell 内部のプロパティ表記を使用します。 以下に例を示します。

```
{{= new Date()}}
```

このプロパティ式は、BeanShell を使用して解釈されます。

日付ユーティリティの使用

com.itko.util.DateUtils クラスには、静的メソッドとして多くの日付ユーティリティ関数が含まれています。 これらのすべての関数は、文字列として整形された日付を返します。 これらの関数は、パラメータ式または JavaScript ステップで使用できます。

```
com.itko.util.DateUtils.formatDate(Date date, String format)
com.itko.util.DateUtils.formatCurrentDate(String format)
com.itko.util.DateUtils.formatCurrentDate(int offsetInSec, String format)
com.itko.util.DateUtils.rfc3339(Date date)
com.itko.util.DateUtils.rfc3339()
com.itko.util.DateUtils.rfc3339(int offsetInSec)
com.itko.util.DateUtils.samlDate(Date date)
com.itko.util.DateUtils.samlDate()
com.itko.util.DateUtils.samlDate(int offsetInSec)
```

たとえば、整形された日付文字列をとる Web サービス コールがあり、サーバが 2 分遅れている場合、以下を使用できます。

```
=com.itko.util.DateUtils.formatCurrentDate(-120,"yyyy-MM-dd'T'HH:mm:ss.SSSZ")
```

これは、「2007-11-22T13:30:37.545-0500」という文字列（これらのガイドラインに従って整形された現在の時刻から 120 秒を引いた値）を生成します。

RFC 3339 は、デフォルトの Java 日付フォーマッタが生成する日付とはわずかに異なります。 厳密な RFC 3339 の日付を必要とする場合、rcf3339 関数を使用できます。

```
=com.itko.util.DateUtils.rfc3339()
```

これは、「2007-11-22T13:30:37.545-05:00」という文字を生成します。

SAML の日付は、「**yyyy-MM-dd'THH:mm:ss'Z**」という形式を使用して整形されます。 samlDate 関数は単なるヘルパーです。したがって、formatDate API を使用するときは、その文字列の形式を記憶している必要はありません。

詳細については、以下を参照してください。

http://download.oracle.com/javase/1.5.0/docs/api/java/text/SimpleDateFormat.html

http://tools.ietf.org/html/rfc3339#section-5.6

クラス ローダ サンドボックスの例

以下の Java クラスは、静的な変数にアクセスして変更を行うため、マルチスレッドまたはマルチユーザで実行できないクラスの簡単な例です。

```
public class NeedsASandbox ¥{

static ¥{

System.out.println("This is my static initializer. You will see
this many times.");

□

static String s;

□

public NeedsASandbox() ¥{¥};

□

public void setS(String s)¥{

this.s = s;

public String getS() ¥{

return s;
```

このクラスは、クラス ローダ サンドボックスで実行する必要があります。

たとえば、DevTest でこのクラスを実行し、10 人のユーザの負荷テストを作成すると仮定します。クラス ローダ サンドボックスを使用しない場合、System.out.println のテキストは 1 回しか表示されず、s の値は正しくありません。 これは、すべてのユーザが 1 つのクラス ローダで実行されているためです。 この場合、この特定のクラスは失敗します。 これがアプリケーションの正しい機能であると仮定し、これを適切に動作させるためにクラス ローダ サンドボックスのサポートを使用します。

クラス ローダ サンドボックス コンパニオンを作成すると、DevTest が 10 人のユーザをステージングした場合、コードのテキストが 10 回表示されます。DevTest は 10 個の個別のクラス ローダを構築して、このクラスを 10 回インスタンス化します。つまり、クラス変数「static string s」の 10 個の個別のインスタンスがあります。この機能によって、スレッド セーフではないアプリケーション ロジックを同時ユーザ テストで実行できます。

以下の 3 つの条件が存在する場合にのみ、クラス ローダ サンドボックス コンパニオンは役立ちます。

- DevTest で POJO をテストしている。
- POJO に静的なメンバがある。
- 複数の仮想ユーザを使用してテストしている。

コンテナ内テスト（ICT）

DevTest でリモート プロキシを使用するコンテナ内テスト（ICT）は、Enterprise JavaBean 実行および動的 Java 実行テスト ステップで使用可能です。

この機能によって、ローカルの EJB および任意の Java オブジェクト（テスト対象のアプリケーションのコンテナのクラスパスで利用可能な任意のオブジェクト）のコンテナ内テストが可能になります。

RMI および EJB は、両方ともリモート オブジェクト プロトコルとしてサポートされています。

コンテナ内テスト（ICT）は 2 つの接続モード（EJB および RMI）を使用します。

このセクションには、以下のトピックが含まれます。

- EJB を使用するアクセス（J2EE コンテナ環境）（P. 516）
- RMI を使用するアクセス（カスタム Java サーバおよびアプリケーション環境）（P. 518）
- DevTest ワークステーション からの ICT インストールのテスト（P. 519）

EJB を使用するアクセス

標準の J2EE コンテナのコンテナ内オブジェクトをテストするには、ステートフル セッション Enterprise JavaBean（EJB）が使用されます。この EJB は、LISA_HOME¥incontainer¥ejb （LISA_HOME は DevTest がインストールされている場所）の **lisa-remote-object-manager.ear** という名前の展開形式の EAR ディレクトリ、およびスタンドアロン JAR ファイル **LISARemoteObjectManagerEJB.jar** として、インストーラにバンドルされています。

この EJB は J2EE コンテナ内の J2EE アプリケーションで展開され、「**LISARemoteObjectManagerEJB**」という名前を使用して JNDI によりアクセス可能である必要があります。EJB の展開は使用されている J2EE コンテナによって異なります。また、ICT EJB の展開に問題がある場合は、ベンダーから提供されたドキュメントが役立つことがあります。

J2EE アプリケーションが分離されたクラスローダを使用する場合、ICT EJB およびその依存関係が含まれるように XML 展開記述子を変更することにより、アプリケーションへ ICT EJB を組み込む必要があります。

JBoss

1. **$LISA¥incontainer¥ejb¥lisa-remote-object-manager.ear ディレクトリを $JBOSS¥server¥default¥deploy** またはその他の適切な展開ディレクトリにコピーすることで、展開形式の ICT EAR が正常に展開されたかどうかをテストします。
2. JBoss は新しい EAR を認識し、エラーなしでそれを展開します。DevTest ワークステーション から EJB に接続できることを確認します。
3. スタンドアロン設定で EAR が正常に展開された後、ICT EJB を J2EE アプリケーションに統合します。 これは、**$LISA¥incontainer¥ejb¥lisa-remote-object-manager.ear** の内容をコピーして既存のアプリケーション EAR に含めるだけです。 または、これらのファイルと組み合わされた J2EE アプリケーションが含まれる新しい EAR を作成します。

アプリケーション展開記述子を変更して ICT とその依存関係を含める方法については、アプリケーション XML 展開記述子 **application.xml** を参照してください。

<module> XML エレメントは、ICT EJB と Java JAR ファイルの依存関係の存在を示すために使用されます。

WebLogic

1. WebLogic での最初のテストは、展開形式の ICT EAR を WebLogic に展開することです。これには、例えば WebLogic の管理コンソール GUI などを使用します。 WebLogic Server が開発モードである場合、このサーバの **autodeploy** ディレクトリに **$LISA¥incontainer¥ejb¥lisa-remote-object-manager.ear** ディレクトリをコピーすることもできます。WebLogic は自動的に EAR を展開します。EAR がエラーなしで展開される様子を観察できます。
2. DevTest ワークステーション から EJB に接続できることを確認します。
3. スタンドアロン設定で EAR が正常に展開された後、ICT EJB を J2EE アプリケーションに統合します。 これは、**$LISA¥incontainer¥ejb¥lisa-remote-object-manager.ear** の内容をコピーして独自のアプリケーション EAR に含めるだけです。 または、これらのファイルと組み合わされた J2EE アプリケーションが含まれる新しい EAR を作成します。

RMI を使用するアクセス

カスタム Java アプリケーションの場合は、アプリケーションを ICT と統合するようにソース コードを変更できます。 ICT テストは、**LISARemoteObjectManagerRMIServer** リモート サーバ オブジェクトを RMI レジストリにバインドすることにより実行できます。 このオブジェクトは、ICT を実行するために DevTest ワークステーション からの接続を許可します。

たとえば、以下のコードをカスタム アプリケーションに含めることで、RMI によって ICT リモート サーバ オブジェクトをバインドできます。

```
LISARemoteObjectManagerRMIServer remoteObjectManagerServer = new
LISARemoteObjectManagerRMIServer();Registry registry =
LocateRegistry.createRegistry(port);registry.bind("LISARemoteObjectManager",
remoteObjectManagerServer);
```

注: ICT が正しく動作するためには、RMI 名「**LISARemoteObjectManager**」を正確に入力する必要があります。

DevTest ワークステーション は、RMI URL **rmi://***hostname*:*port*/**LISARemoteObjectManager** を使用してアプリケーションに接続しようとします。

hostname および port は変数で、テスト アプリケーションで変更して DevTest ワークステーション で設定できます。

サンプルのサーバ アプリケーションは、$LISA/incontainer/rmi/example にあります。 コンソール ウィンドウを使用して「java -jar ExampleServer.jar」コマンドを実行すると、サンプル サーバを実行できます。

注: このコマンドを正常に実行するには、**$LISA/incontainer/rmi/example** ディレクトリから実行する必要があります。**$LISA/incontainer/rmi/example/src** ディレクトリのサンプル ソース コードを参照してください。

ICT インストールのテスト

ICT のサーバ側の部分が起動された後で、DevTest ワークステーション から接続できることをテストする必要があります。

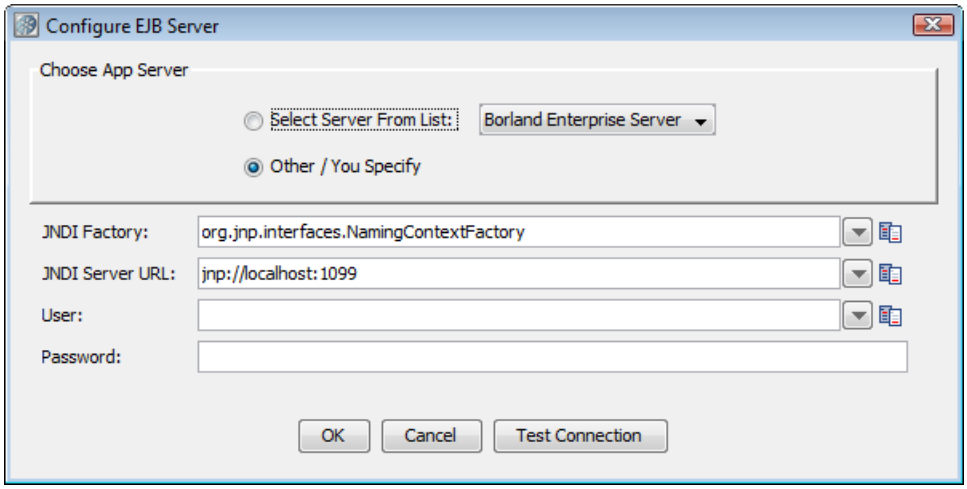
実行方法

1. DevTest ワークステーション を開きます。
2. **ICT Connection Test** という名前の新しいテスト ケースを作成します。
3. このテスト ケースの内部に、新しい動的 Java 実行テスト ステップを作成します。
4. 新しいテスト ステップ用のエディタで、［リモート］オプションを選択します。
5. ［リモート コンテナ タイプ］ドロップダウン リストを接続プロトコル（EJB または RMI のいずれか）に設定します。

リモート コンテナ タイプが EJB の場合

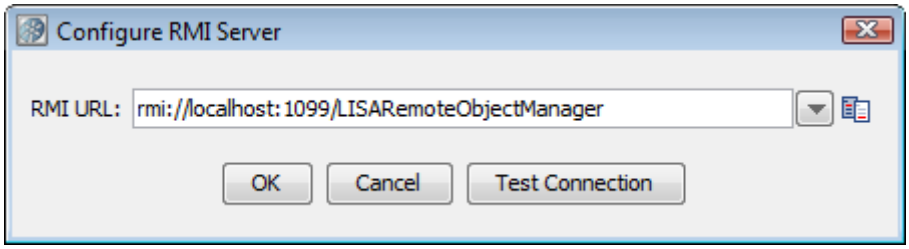
接続プロトコルとして EJB を使用する場合は、［設定］をクリックしてプロトコルの設定を入力します。

たとえば、ICT を実行する J2EE コンテナのホスト名または IP アドレスおよびポート番号を［EJB サーバの設定］ダイアログ ボックスに入力します。



リモート コンテナ タイプが RMI の場合

接続プロトコルとして RMI を使用する場合、RMI の設定に［RMI サーバの設定］ダイアログ ボックスを使用する例を以下に示します。



ICT サーバへの接続テストに成功した後、［次のクラスの新規オブジェクトを作成］フィールドに「**java.util.Date**」と入力し、［オブジェクトの構築/ロード］をクリックして java.util.Date クラスの新しいコンテナ内インスタンスを作成します。

これでインストールは完了し、ICT を使用する準備が整います。

依存関係

ICT サーバ側クラスは、以下のサードパーティ ライブラリに依存しています。

- BeanShell 2.0b4 (bsh-2.0b4-lisa-remote-object-manager.jar).

  注: .bsh スクリプトが削除されるように、元の JAR ファイルが変更されています。 これらのスクリプトは、JAR ファイルを検査して自動的にそれらを実行する J2EE コンテナで問題が生じることがわかっています。

- XStream 1.1.2 （xstream-1.1.2.jar）
- Log4j 1.2.13 （log4j-1.2.13.jar）
- Jakarta Commons Logging 1.0.2 （commons-logging.jar）

これらの依存関係は、ICT 統合を簡便化するために DevTest ICT JAR ファイルとは個別のファイルとして意図的に配布されます。 テストするアプリケーションには、これらのライブラリの一部またはすべてがすでにそのクラスパスに含まれている場合があるため、クラスパスへのこれらの依存関係の追加が必要ないことがあります。

DDL の生成

local.properties で以下のプロパティを設定すると、レポートおよび VSE 用の DDL が作成されます。

- **eclipselink.ddl-generation=create-tables**
- **eclipselink.ddl-generation.output-mode=sql-script**
- **eclipselink.target-database=Oracle**

エージェント、CAI、および CVS 用の DDL を作成するには、以下のコマンドを使用します。

- **java -jar LisaAgent.jar -ddl oracle** （CAI 用の Oracle DDL の生成）
- **java -jar LisaAgent.jar -ddl mysql** （CAI 用の MySQL DDL の生成）

付録 A: 付録 A - DevTest プロパティ ファイル（lisa.properties）

lisa.properties プロパティ ファイルには、初期化および設定情報が格納されます。

プロパティ ファイルは DevTest インストール ディレクトリに格納されます。

DevTest ワークステーション でこのファイルの内容を表示できます。

注: このファイルにカスタム プロパティを追加しないでください。このファイルは、CA Technologies の判断で差し替えられる可能性があります。

lisa.properties ファイルを開くには、メイン メニューから［システム］-［プロパティの編集］を選択します。

このセクションには、以下のトピックが含まれています。

VSEasy プロパティ (P. 574)

Javadoc およびソース コードのパスのカンマ区切りリスト

これらのパスは、クラスおよびパラメータ ドキュメントを示すために使用されます。ドキュメント パスには、Javadoc へのベース パスであるディレクトリおよび URL を使用できます。以下の例には、Web サイトの JDK ドキュメントが含まれています。ただし、遅延のため、Web サイトは推奨されていません。

lisa.java.docPath=LISA_HOME¥examples¥javadoc,[http://java.sun.com/j2se/1.3/docs/api/

lisa.java.docPath=LISA_HOME¥examples¥javadoc

lisa.java.sourcePath=LISA_HOME¥examples¥src

このソース パスには、ベース パスおよびソースの JAR または ZIP ファイルとしてディレクトリを使用できます。

lisa.axis.compiler.version=1.4

これは、lisa.axis.compiler.version 1.4 です。

システム プロパティ

file.encoding

DevTest が読み取りまたは書き込みを行うファイルのエンコーディング。

デフォルト： UTF-8

lisa.supported.html.request.encodings

HTTP/HTML 要求ステップでサポートするエンコーディングを含めるには、カンマ区切りリストを変更します。 また、基盤となる JVM は、このリストのエンコーディングをすべてサポートする必要があります。 Web ページがリストでサポートされていないエンコーディングを使用している場合、エンコーディングは DevTest のデフォルトのエンコーディング（**lisa.properties** 内の file.encoding キー）に置き換えられます。 また、HTTP/HTML 要求ステップを作成するときにエンコーディングが選択されていない場合は、DevTest のデフォルトのエンコーディングが使用されます。

デフォルト： ISO-8859-1, UTF-8, Shift_JIS, EUC-JP

lisa.supported.jres

デフォルト： 1.7

org.jfree.report.LogLevel

デフォルト： Error

javax.xml.parsers.DocumentBuilderFactory

デフォルト： org.apache.xerces.jaxp.DocumentBuilderFactoryImpl

javax.xml.parsers.SAXParserFactory

デフォルト： org.apache.xerces.jaxp.SAXParserFactoryImpl

javax.xml.transform.TransformerFactory

デフォルト： org.apache.xalan.processor.TransformerFactoryImpl

サーバ プロパティ

lisa.net.bindToAddress

リスンする IP アドレス。デフォルトでは、LAN 内のすべての IP アドレスをリスンします。 この値は特定の IP アドレスに制限できます。 その他のコンピュータの接続を許可せず、ローカル接続を許可するには、値を *127.0.0.1* または *localhost* に設定します。

OS X プロパティ

apple.awt.brushMetalLook

デフォルト： false

apple.awt.brushMetalRounded

デフォルト： false

apple.laf.useScreenMenuBar

デフォルト： true

apple.awt.showGrowBox

デフォルト： true

com.apple.mrj.application.growbox.intrudes

デフォルト： true

com.apple.macos.smallTabs

デフォルト： true

com.apple.mrj.application.apple.menu.about.name

デフォルト： LISA

com.apple.mrj.application.live-resize

デフォルト： true

更新通知

lisa.update.every=1

より新しいバージョンがダウンロード可能かどうかを DevTest が確認する頻度を制御します。 確認を無効にするには、値を空白に設定します。 その他の有効な値は以下のとおりです。

- 0 （すべての起動時に確認）
- 1 （1 日あたり 1 回確認）
- 2 （2 日ごとに確認）、など。

lisa.update.URL=http://www.itko.com/download/ga/

基本的なデフォルト

lisa.testcase=test.xml

com.itko.lisa.test.TestCase クラスからテストを直接実行する場合のデフォルト。

lisa.registry=registry.xml

com.itko.lisa.test.TestCase クラスからテストを直接実行する場合のデフォルト。

lisa.runName=Ad-hoc Run

com.itko.lisa.test.TestCase クラスからテストを直接実行する場合のデフォルト。

lisa.registryName=registry

接続するレジストリのデフォルト名、および名前を指定せずに起動する場合のレジストリのデフォルト名。

lisa.coordName=coordinator

明示的に名前を指定せずに起動され、かつテスト ランナーがコーディネータ サーバを必要とし、コマンドラインでコーディネータ サーバを指定しない場合のコーディネータ サーバのデフォルト名。

lisa.simulatorName=simulator

コマンドラインでシミュレータ デーモンを指定しない場合のシミュレータ デーモンのデフォルト名。

lisa.vseName=VSE

明示的に名前を指定せずに起動された場合の仮想環境サーバのデフォルト名。

lisa.defaultRegistry.pulseInterval=30

レジストリのステータス ログ間隔。 デフォルト値は 30 秒です。

lisa.coordinator.pulseInterval=30

コーディネータのステータス ログ間隔。 デフォルト値は 30 秒です。

lisa.simulator.pulseInterval=30

シミュレータのステータス ログ間隔。 デフォルト値は 30 秒です。

lisa.vse.pulseInterval=30

VSE のステータス ログ間隔。 デフォルト値は 30 秒です。

lisa.server.projectmap.refresh.pulseInterval=600

DevTest サーバ（コーディネータおよびシミュレータ）がファイル パスへのプロジェクト名のマップをリフレッシュする時間間隔。

lisa.defaultRegistryConnectionTimeoutSeconds=90

DevTest レジストリに接続する場合にコーディネータおよびシミュレータが使用するタイムアウト値（秒）。 0 の値は、無制限のタイムアウトを示します。つまり、接続しようとして永久に待機します。

lisa.regex.helper.tutorial.url=http://download.oracle.com/javase/tutorial/essential/regex/

正規表現ヘルパー ウィンドウで正規表現のチュートリアルを表示するための URL。

lisa.hooks=com.itko.lisa.files.SampleHook

DevTest にフックを登録します。これらの値はカンマ区切りです。

lisa.project.ignore=CVS|SCCS|RCS|rcs|¥.DS_Store|¥.svn|vssver¥.scc|vssver2¥.scc|¥.sbas|¥.IJI¥..*|.*¥.pyc|.*¥.pyo|¥.git|.*¥.hprof|_svn|¥.hg|.*¥.lib|.*~|__pycache__|¥.bundle|.*¥.rbc

このプロパティは、プロジェクト ツリーで表示する必要がないファイルのタイプを非表示にできる Java 正規表現です。 デフォルトでは、サードパーティ製品のさまざまな既知の制御ファイルが非表示です。デフォルトで非表示にされているファイルを表示するには、正規表現を変更します。

HTTP ヘッダ キー プロパティ

HTTP サポートでのヘッダ キーのデフォルト値を以下のリストに示します。変更の利点を活かすには、すべてのテストに対してこれらを削除または変更します。 単一のテストまたは単一の HTTP トランザクションの実行に対してこれらを変更するには、テスト ノードに固有のヘッダ ディレクティブを使用します。

ice.browser.http.agent

値： compatible、MSIE 6.0、Windows NT 5.0

デフォルト： Mozilla/4.0

lisa.http.header.0.key

デフォルト： Pragma

lisa.http.header.0.value

デフォルト： no-cache

lisa.http.header.1.key

デフォルト： Cache-Control

lisa.http.header.1.value

デフォルト： no-cache

lisa.http.header.0.key

デフォルト： Accept

lisa.http.header.0.value

デフォルト： image/gif, image/x-xbitmap, image/jpeg

lisa.http.header.1.key

デフォルト： Accept-Language

lisa.http.header.1.value

デフォルト： en

lisa.http.header.2.key

デフォルト： Accept-Charset

lisa.http.header.2.value

デフォルト： iso-8859-1,*,utf-8

lisa.http.header.3.key

デフォルト： User-Agent

lisa.http.header.3.value

デフォルト： Mozilla/4.0, MSIE 6.0; Windows NT 5.0

HTTP フィールド エディタ プロパティ

HTTP フィールド エディタに表示されるフィールドのデフォルトを、以下のリストに示します。 エディタによって自動的に追加されるため、「Authentication」を含めないでください。

lisa.gui.http.fieldNames

値： Accept,Accept-Language,User-Agent,Connection

lisa.webrecorder.textMIMEs

値： html,text,magnus-internal,application/pdf

lisa.webrecorder.notTextMIMEs

値： css,script

lisa.webrecorder.alwaysIgnore

値： .gif,.jpg,.jpeg,.css,.js,.ico

lisa.web.ntlm

テスト ランナーでの NTLM 認証を有効にします。

デフォルト： true

テスト ケースの実行パラメータ

lisa.hotDeploy=C¥Projects¥Lisa¥custom_classes

この設定は、カスタム クラスを探す場所を、DevTest に組み込まれているクラスローダに指示します。デフォルトは $LISA_HOME¥hotDeploy です。

lisa.overloadThreshold=1000

テスト ノードは、予期された反応時間の長さではなく、反応時間内のスリープした実際の時間の長さを確認することにより、シミュレータがスラッシングしているかどうかを判断しようとします。 この設定は、シミュレータが過負荷状態であるという警告テスト イベントを送信するまでに許容できる反応時間の追加の「スリップ」の長さです。 デフォルトの 1000 は、シミュレータが予期された時間よりも 1 秒多くスリープした場合（コンピュータの CPU の容量不足が原因と推定される場合）、テスト イベントが生成されることを意味します。

lisa.webservices.encode.empty.xmlns=true

一部の Web サービス サーバ スタックは、SOAP 要求（jbossWs など）に空の xmlns 文字列を必要とします。 Amazon などのその他の SOAP 要求は、空の xmlns では動作しません。 コールするスタックに合わせてこのプロパティを変更します。

lisa.webservices.encode.version=1.1

クライアント スタブの生成に対して強制するエンコーディング バージョンを設定します。 デフォルトは 1.1 です。

lisa.tm.sys.min.millis=0

lisa.tm.sys.max.millis=0

新しいシステム ステップのデフォルトの反応時間（ミリ秒）。 システム ステップには、サブプロセス、続行、続行（クワイエット）、失敗、終了が含まれます。

lisa.numFilters.warning=100

lisa.numAsserts.warning=100

場合によって、フィルタおよびアサーションがテスト ステップに動的に追加されます。 負荷テストでは、これが何千ものアサート/フィルタを意味する可能性があります。 上記の 2 つの数値は、ログが WARN レベル メッセージを生成するまでのしきい値です。

lisa.exception.on.num.exceeded=true

しきい値を超えた場合に TestDefException（テストの強制終了）を生成するかどうか。

lisa.urltrans.encode.queryparams=false

このプロパティを false に設定すると、ステップの実行時に HTTP/HTML URL 内のクエリ パラメータはデコード/エンコードされません。

lisa.generic.url.decoder=true

7.0.0 以前で作成された HTTP/HTML 要求ステップがあるテスト ケースを DevTest 7.1.1 以降で開く場合に、このプロパティを true に設定します。

テスト イベント処理のカスタマイズ

lisa.perfmon.snmp.port=1161

StatKeeper は、Windows Perfmon、JMX、SNMP などのプラットフォーム固有のモニタをラップまたは実装する Perfmon 統合クラスをロードできます。DevTest には、Windows パフォーマンス モニタまたは SNMP エージェントのいずれか（両方ではない）への統計出力の生成をサポートする Perfmon DLL クラスおよび SNMP クラスが用意されています。詳細については、SNMP のドキュメントを参照してください。SNMP 用の通常のポートは 161 です。ただし、それを使用するには、root である必要があります。

lisa.perfmon.class=com.itko.lisa.stats.snmp.SnmpPerfmon

パフォーマンス モニタにネイティブ OS データを提供できるツールがある場合、そのツールに DevTest データをプッシュできるクラスを実装し、ここにそのクラス名を指定します。

lisa.perfmon.dll=/c:/Projects/Lisa/PerfmonJNI/LISAPerfmonJNI/Debug/LISAPerfmonJNI.dll

StatKeeper への Windows Perfmon 統合には、Windows （ネイティブ）実装用の「DLL」設定があります。 ここに、フル パス/ファイルを指定します。 PerfmonStatKeeperWindows を使用する場合は、この指定のみが必要です。

シミュレータは、RMI の通信を減らすため、個別のスレッドおよびキューを使用してコーディネータにテスト イベントを送信します。 これらは、デーモン スレッドにイベントをプッシュさせるしきい値です。 最小サイズまたは最大待機時間を使用します（長くかかりすぎるか、または多すぎる場合、それらをポストします）。

lisa.eventPoolPoll=250

キュー サイズまたは最大待機時間を超えたかどうかの確認を行う頻度（ミリ秒）。

lisa.eventPoolSize=64

送信する前に取得する最大サイズ。

lisa.eventPoolMaxWait=1000

イベントを転送しないままにしておく時間の長さ。

テスト マネージャ/エディタ プロパティ

gui.show.memory.status

デフォルト： false

lisa.screencap.delay.seconds

デフォルト： 6

lisa.screencap.dir

デフォルト： LISA_HOME¥screens

lisa.screen.cap.prefix

デフォルト： lisa-screencap-

lisa.earsubdir.endingnamepart

デフォルト： -contents

lisa.model.editor.inspector.scale

このプロパティは、モデル エディタのモデルおよびステップ インスペクタのアイテムの「スケール」（主にフォント サイズ）を設定します。1.0 は 12 ポイントです。したがって、このプロパティの値は、設定したいポイント数を 12 で割ることによって定義します。たとえば、11 ポイントは 11/12 = 0.92、10 ポイントは 10/12 = 0.83 （デフォルト）、14 ポイントは 14/12 = 1.17 です。

デフォルト： 0.83

lisa.stats.decimalFormat

一部の組み込みのメトリック（1 秒あたりのステップ数）では、浮動小数点値を表示するために Java DecimalFormat を使用します。 小数点を使用しない場合は、この値を「######」にするか、http://download.oracle.com/javase/6/docs/api/java/text/DecimalFormat.html を参照して一連の表示から選択してください。

デフォルト： ###,###.#

lisa.editor.custJavaNodeEditor.classes

これは、テスト ケースに含めるすべてのカスタム Java テスト ノードのカンマ区切りリストです。

lisa.editor.combined.report.type

lisa.gui.log4jfmt

これは、［システム メッセージ］ウィンドウのメッセージの形式を制御します。

デフォルト： %-5p - %m%n

lisa.editor.http.recorderPort

HTTP レコーダはこのポートにバインドされます。

デフォルト： 8010

lisa.editor.proxy.webProxySupport

デフォルト： オン

J2EE サーバ パラメータ

lisa.prefill.jndiNames=	JBOSS=org.jnp.interfaces.NamingContextFactory Weblogic=weblogic.jndi.WLInitialContextFactory Websphere=com.ibm.websphere.naming.WsnInitialContextFactory Borland Enterprise Server=com.inprise.j2ee.jndi.CtxFactory iPlanet/Sun AS=com.sun.jndi.cosnaming.CNCtxFactory
lisa.prefill.jndiUrlPrefix=	JBOSS=jnp:// Weblogic=t3:// Websphere=iiop:// Borland Enterprise Server=iiop:// iPlanet/Sun AS=iiop://
lisa.prefill.jndiDefPort=	JBOSS=1099 Weblogic=7001 Websphere=2809 Borland Enterprise Server=1099 iPlanet/Sun AS=1099
lisa.prefill.jndiNeedsClass=	JBOSS=false Weblogic=false Websphere=true Borland Enterprise Server=true iPlanet/Sun AS=true
lisa.prefill.jndiFactories=	org.jnp.interfaces.NamingContextFactory weblogic.jndi.WLInitialContextFactory com.ibm.websphere.naming.WsnInitialContextFactory com.webmethods.jms.naming.WmJmsNamingCtxFactory com.tibco.tibjms.naming.TibjmsInitialContextFactory com.inprise.j2ee.jndi.CtxFactory com.sun.jndi.cosnaming.CNCtxFactory fiorano.jms.runtime.naming.FioranoInitialContextFactory

lisa.prefill.jndiServerURLs=	jnp://SERVER:1099&t3://SERVER:7001 iiop://SERVER:PORT tibjmsnaming://SERVER:7222&iiop://SERVER:PORT iiop://localhost:9010&wmjmsnaming://Broker #1@SERVER:PORT/JmsAdminTest
lisa.editor.URLTransEditor.protos=	http,https
lisa.editor.URLTransEditor.hosts=	
lisa.editor.URLTransEditor.ports=	80,443
lisa.editor.URLTransEditor.files=	
lisa.prefill.jdbc.names=	Oracle SQL Server WLS Oracle JDataStore Sybase DB2 MySQL Derby
lisa.prefill.jdbc.jdbcDrivers=	oracle.jdbc.driver.OracleDriver com.microsoft.sqlserver.jdbc.SQLServerDriver com.microsoft.jdbc.sqlserver.SQLServerDriver weblogic.jdbc.oci.Driver com.borland.datastore.jdbc.DataStoreDriver com.sybase.jdbc.SybDriver com.ibm.db2.jcc.DB2Driver org.gjt.mm.mysql.Driver org.apache.derby.jdbc.ClientDriver
lisa.prefill.jdbc.jdbcConnectionURLs=	jdbc:oracle:thin:@SERVER:1521:SIDNAME jdbc:sqlserver://SERVER:PORT;databasename=DBNAME jdbc:microsoft:sqlserver://SERVER:PORT jdbc:weblogic:oracle:TNSNAME jdbc:borland:dslocal:DBNAME jdbc:sybase:Tds:SERVER:PORT/DBNAME jdbc:db2://SERVER:PORT/DBNAME jdbc:mysql://SERVER:PORT/DBNAME jdbc:derby://DBSERVER:DBPORT/DBNAME

内部レンダリングに使用するネイティブ ブラウザ情報

lisa.internal.browser.on=yes

lisa.internal.browser.win=com.itko.lisa.web.ie.IEUtils

lisa.internal.browser.osx=com.itko.lisa.web.jxbrowser.JxBrowserUtils

lisa.internal.browser.linux=com.itko.lisa.web.jxbrowser.JxBrowserUtils

lisa.internal.browser.sol-sparc=null

lisa.internal.browser.imgs=false

lisa.internal.browser=msie

WR タイプ： mozilla、safari、msie

lisa.internal.browser.swing.heavy=false

lisa.internal.browser.usejsinviews=yes

lisa.example.wsdls=http://localhost:8080/itko-examples/services/UserControlService?wsdl

テスト マネージャ/モニタ プロパティ

monitor.events.maxrows

テスト実行時にテスト マネージャで保持されるイベント行の最大数。保持されるオブジェクトの数は、この数の 2 倍です。

デフォルト： 500

lisa.tm.sysmess.size

システム メッセージ ウィンドウの最大サイズ。 消費されるメモリはこの値の 2 倍です。

デフォルト： 10240

ビルトイン文字列ジェネレータ パターン

lisa.patterns.stringgenerator.types=&Phone=(DDD)DDD-DDDD, &SSN=DDD-DD-DDDD, &Date=Lll-DD-DDDD, &Zip=D*(5)

JMX 情報

lisa.jmx.types	com.itko.lisa.stats.jmx.JSE5Connection com.itko.lisa.stats.jmx.TomcatConnection com.itko.lisa.stats.jmx.JBossConnection com.itko.lisa.stats.jmx.JSR160RMIConnection com.itko.lisa.stats.jmx.WeblogicConnector com.itko.lisa.stats.jmx.Weblogic9Connector com.itko.lisa.stats.jmx.WebsphereSOAPConnection com.itko.lisa.stats.jmx.ITKOAgentConnection com.itko.lisa.stats.jmx.OracleASConnector
lisa.jmx.typeprops	com.itko.lisa.stats.jmx.JSE5Connection=LISA_JMX_JSE5 com.itko.lisa.stats.jmx.TomcatConnection=LISA_JMX_TOMCAT5 com.itko.lisa.stats.jmx.JBossConnection=LISA_JMX_JBOSS3240 com.itko.lisa.stats.jmx.JSR160RMIConnection=LISA_JMX_JSR160RMI com.itko.lisa.stats.jmx.Weblogic9Connector=LISA_JMX_WLS9 com.itko.lisa.stats.jmx.WeblogicConnector=LISA_JMX_WLS6781 com.itko.lisa.stats.jmx.OracleASConnector=LISA_JMX_OC4J com.itko.lisa.stats.jmx.WebsphereSOAPConnection=LISA_JMX_WASSOAP5X com.itko.lisa.stats.jmx.ITKOAgentConnection=LISA_JMX_ITKOAGENT

通常、これらのパラメータに対して必要なものはありません。

LISA_JMX_JSR160RMI

デフォルト： LISA_HOME/lib/mx4j.lib

LISA_JMX_ITKOAGENT

デフォルト： LISA_HOME/lib/mx4j.lib

LISA_JMX_JSE5

JSE 5 を実行する場合、このプロパティを変更する必要はありません。バージョンが正しいという前提で、必要なものが含まれる jbossall-client を出荷しています。

LISA_JMX_JBOSS3240

デフォルト：
LISA_HOME/lib/mx4j.lib{{path.separator}}LISA_HOME/hotDeploy/jbossall-client.jar

LISA_JMX_TOMCAT5

このパラメータは Tomcat 用です。

デフォルト：
LISA_HOME/lib/mx4j.lib{{path.separator}}LISA_HOME/hotDeploy/mx4j-tools.jar

LISA_JMX_WLS9

デフォルト：
LISA_HOME/hotDeploy/weblogic.jar{{path.separator}}LISA_HOME/hotDeploy/wljmxclient.jar

LISA_JMX_WLS6781

このパラメータは、使用する weblogic.jar の場所に変更する必要があります。 wlclient.jar はすべて機能します。

デフォルト： LISA_HOME/hotDeploy/weblogic.jar

Oracle AS

LISA_JMX_OC4J

デフォルト：
LISA_HOME/lib/oc4jclient.jar{{path.separator}}LISA_HOME/lib/adminclient.jar

IBM WebSphere

LISA_JMX_WASSOAP5X

DevTest の使用を開始するまでは IBM/WAS クラスパスを使用し、これを空白のままにするほうが簡単です。

デフォルト： LISA_HOME/lib/mx4j.lib

lisa.alert.email.emailAddr

パフォーマンス モニタリング アラートを使用する場合、このパラメータはそれらのアラートの「送信元」の電子メール アドレスです。

形式： lisa@itko.com

lisa.alert.email.defHosts

このパラメータは、電子メールのルーティングを試みる電子メール サーバ（SMTP サーバ）です。

デフォルト： localhost

lisa.rundoc.builtins

値

com.itko.lisa.files.1user1cycle.stg

テスト ケースを 1 人のシミュレートされたユーザで 1 回実行します

com.itko.lisa.files.1user1cycle0think.stg

テスト ケースを 1 人のシミュレートされたユーザで反応時間なしで 1 回実行します

com.itko.lisa.files.1user1min.stg

テスト ケースを 1 人のシミュレートされたユーザで 1 分間実行し、必要に応じてテストを再起動します

com.itko.lisa.files.1user5min.stg

テストを 5 分間ステージングし、必要に応じて再起動します

com.itko.lisa.files.1usernonstop.stg

手動で停止されるまでこのテストを実行します

com.itko.lisa.files.5user1min.stg

このテストを 5 人の仮想ユーザで 1 分間実行します

lisa.auditdoc.builtins

デフォルト： com.itko.lisa.files.DefaultAudit.aud

テスト マネージャ/ITR プロパティ

lisa.tm.itr.max.delay.seconds

デフォルト： 5

外部コマンド シェル

test.cmde.win.shell=cmd /c

test.cmde.unix.shell=sh -c

test.cmde.Windows.NT.(unknown).shell=cmd /c

テスト パラメータ

lisa.props.blankOnMissing=true

キーが正しくない場合に、DevTest にキーを空の文字列に置換させる場合に使用するプロパティ。

lisa.test.custevents=&101=「カスタム イベント 101」、&102=「カスタム イベント 102」

カスタム イベント： 最初に許可されているイベント番号は 101 です。したがって、ここでは例として 2 つの番号を示しました。

lisa.SimpleWebFilter.responseCodeRegEx=[45] d d

lisa.fsss.dateformat=MM/dd/yyyy hh:mm:ss a

Quartz のスケジューラ インスタンスの作成に StdSchedulerFactory が使用するプロパティ

主なスケジューラ プロパティの設定

org.quartz.jobStore.class

デフォルト： org.quartz.simpl.RAMJobStore

org.quartz.threadPool.class

デフォルト： org.quartz.simpl.SimpleThreadPool

org.quartz.threadPool.threadCount

デフォルト： 5

lisa.meta-refresh.max.delay

デフォルト： 5

TM のプラットフォーム固有のパラメータ

lisa.tm.exec.unix

デフォルト： xterm -e {0}

lisa.tm.exec.win

デフォルト： cmd /c start {0}

lisa.tm.exec.osx

デフォルト： open -a /Applications/Utilities/Terminal.app {0}

Swing テストのサポートによって使用されるプロパティ

lisa.swingtest.client.logging.properties.file

SwingTestProgramStarter でカスタム Log4J ログ プロパティ ファイルを使用するには、コメントを外します。

デフォルト： C:/Lisa/swingtestclient-logging.properties

ライセンス設定

laf.request

デフォルト： laf/license.do

laf.default.url

デフォルト： https://license.itko.com

laf.displaysetting

デフォルト： true

レポート プロパティ

lisa.reporting.defaultPageSize

値： A4 | letter

JPA レポート プロパティ

rpt.eclipselink.ddl-generation

デフォルト： create-tables

rpt.eclipselink.ddl-generation.output-mode

デフォルト： database

rpt.eclipselink.validateschema

デフォルト： false

perfmgr.rvwiz.whatrpt.autoExpire

デフォルト： true

perfmgr.rvwiz.whatrpt.expireTimer

期限切れのレポートを確認するには、autoExpire = true に設定します。整数の後に文字（m = 月、w = 週、d = 日、h = 時間）を付加して有効期間を設定します。 デフォルトの有効期間は 30d （30 日）です。

デフォルト： 30d

rpt.hibernate.validateschema

レポート データベースのスキーマを検証します。デフォルトでは、レジストリのみがスキーマを検証します。

デフォルト： false

lisa.0.registry.local.autoshutdown

デフォルト： true

lisa.8.registry.local.autoshutdown

デフォルト： true

lisa.10.registry.local.autoshutdown

デフォルトの動作では、ローカル レジストリが自動的に起動された場合、自動的にはシャットダウンされません。 例外は、DevTest ワークステーション、VSE、および VSE ワークステーションです。

デフォルト： true

lisa.4.registry.local.autoshutdown

デフォルト： false

lisa.5.registry.local.autoshutdown

JUnit およびテスト ランナーでは、レジストリを自動シャットダウンしないでください。

デフォルト： false

Eclipse コネクタに使用されるプロパティ

lisa.eclipse.connector.port

デフォルト： 8546

サンプル テスト スイートに使用されるプロパティ

EXAMPLES_HOME

デフォルト： LISA_HOME/examples

VSE プロパティ

lisa.magic.string.min.length

マジック ストリングを構成するために必要な引数と見なされる、VSE トランザクション要求の引数値の最小長を定義します。

デフォルト： 3

lisa.magic.string.word.boundary.type

マジック ストリングの内容の VSE での検索が単語境界とどのように関連するかを定義します。

値

- **none：** 単語境界は問題ではありません。
- **start：** マジック ストリングの候補は単語境界で始まる必要があります。
- **end：** マジック ストリングの候補は単語境界で終了する必要があります。
- **both：** マジック ストリングの候補は単語全体として見つかる必要があります。つまり、両端が単語境界です。

デフォルト： both

lisa.magic.string.exclusion

マジック ストリングの候補とする文字列から特定の文字列を除外するかどうかを指定します。

値： Yes、YES、yes、No、NO、no、true、True、TRUE、false、False、FALSE、__NULL

デフォルト： Yes, YES, yes, No, NO, no, true, True, TRUE, false, False, FALSE, __NULL

lisa.magic.string.xml.tags

XML タグ内のテキストをマジック ストリングに変更するかどうかを指定します。

デフォルト： false

lisa.vse.server.dir.full.service.name=false

複数の VSE インスタンスが異なるコンピュータ上で同じインストール ディレクトリから実行されるかどうかを指定します。実行される場合、このプロパティのコメントを外して true に設定します。

デフォルト： false

lisa.vse.response.xml.prettyprint

VSE が XML 応答を記録する場合、サービス イメージ内の XML を整形するかどうかを指定します。 デフォルトの false は、DevTest が XML への書式設定、整形、または改行の追加を行わないことを示します。XML が 1 つの長い文字列として受信された場合、DevTest ではそのように表示されます。

デフォルト： false

lisa.vse.remember.execution.mode

VSE ダッシュボードで仮想サービスに設定した実行モードを VSE が記憶するかどうかを指定します。 シャットダウン時に VSE が実行モードを記憶しないようにする場合は、このプロパティをコメント アウトします。

デフォルト： true

lisa.vse.match.event.buffer.size

VSE で一致する関連イベントがバッファされる数を定義する、各 VS モデルのイベントの数を設定します。 このプロパティは、各 VS モデルのイベントの数を設定します。 たとえば、2 つの VS モデルが 100 のデフォルト イベント バッファ サイズで展開されている場合、200 のイベントがバッファされます。 これらのイベントは、VSE ダッシュボードの VS モデル検査ページの［一致］タブのソースです。

デフォルト： 100

lisa.vse.request.event.set.buffer.size

DevTest イベントの完全なセットがバッファされる、VSE のインバウンド要求の数を定義します。 このプロパティは、各 VS モデルのイベントの数を設定します。 たとえば、2 つの VS モデルがデフォルト要求バッファ サイズ（5）で展開されている場合、10 セットのイベント（インバウンド要求ごとにグループ化）がバッファされます。 イベントのこれらのセットは、VSE ダッシュボードの VS モデル検査ページの［イベント］タブのソースです。

性能上の理由で、VSE は 50 のイベントを処理用に保持します。この制限のため、イベントが処理キューから削除され、検査ビューに存在しなくなる可能性があります。イベントの切り捨てを防ぐため、イベント処理キューのサイズを増やすことができます。 ただし、このアクションによってメモリ使用量が増加し、パフォーマンスが低下する可能性があります。

デフォルト： 5

lisa.vse.si.text.editor.order

自動検出を使用する場合に登録済みの VSE テキスト応答エディタがクエリされる順番を定義します。 一意にするために十分なクラス名の「右端」だけが必要です。

値： XMLTextEditor、JSONTextEditor

デフォルト： デフォルトのテキスト エディタ

lisa.tm.def.min.millis

新しい「標準」ステップのデフォルトの最短反応時間（ミリ秒）を定義します。

デフォルト： 500

lisa.tm.def.max.millis

新しい「標準」ステップのデフォルトの最長反応時間（ミリ秒）を定義します。

デフォルト： 1000

lisa.vse.rest.max.optionalqueryparams

WADL ファイルまたは RAML ファイルにおいて 1 つのメソッドで使用する、オプションのクエリ パラメータの最大数を指定します。 WADL または RAML ファイルにおけるオプションのクエリ パラメータの数が、指定された *lisa.vse.rest.max.optionalqueryparams* の値よりも多い場合、最初から数えて *lisa.vse.rest.max.optionalqueryparams* の値に等しい個数のパラメータのみが、トランザクションを作成するために使用されます。

デフォルト： 5

Date-Checker Properties

VSE 日付ユーティリティは以下のプロパティを使用して、どの日付パターンが日付の詳細な変換に有効であると見なすかを決定します。 以下の各エントリは、VSE が日付の一部と見なす正規表現を表しています。

lisa.vse.datechecker.dayregex

((¥[12¥]¥¥d)¥|(3¥[01¥])¥|(0?¥[1-9¥]))

lisa.vse.datechecker.monthnumberregex

((1¥[012¥])¥|(0 ¥¥d)¥|0¥[1-9¥]¥|¥[1-9¥])

lisa.vse.datechecker.monthalpharegex

(¥¥bJAN¥¥b¥|¥¥bFEB¥¥b¥|¥¥bMAR¥¥b¥|¥¥bAPR¥¥b¥|¥¥bMAY¥¥b¥|¥¥bJUN¥¥b¥|¥¥bJUL¥¥b¥|¥¥bAUG¥¥b¥|¥¥bSEP¥¥b¥|¥¥bOCT¥¥b¥|¥¥bNOV¥¥b¥|¥¥bDEC ¥¥b)

lisa.vse.datechecker.yearlongregex

¥¥d¥¥d¥¥d¥¥d

lisa.vse.datechecker.yearshortregex

¥¥d¥¥d

lisa.vse.datechecker.timeregex

(¥¥s?((¥[012¥]? ¥¥d)¥|(2¥[0123¥]))¥:((¥[012345¥] ¥¥d)¥|(60))¥:((¥[012345¥] ¥¥d)¥|(60)))

lisa.vse.datechecker.time.hhmmssregex

((¥[012¥]? ¥¥d)¥|(2¥[0123¥]))¥:((¥[012345¥] ¥¥d)¥|(60))¥:((¥[012345¥] ¥¥ d)¥|(60))

lisa.vse.datechecker.time.millisregex

((¥[012¥]?¥¥d)¥|(2¥[0123¥]))¥:((¥[012345¥]¥¥d)¥|(60))¥:((¥[012345¥]¥¥d)¥|(60))¥¥.((¥¥d ¥¥d ¥¥d)¥|0)

lisa.vse.datechecker.time.millis.zoneregex

((¥[012¥]?¥¥d)¥|(2¥[0123¥]))¥:((¥[012345¥] ¥¥d)¥|(60))¥:((¥[012345¥] ¥¥ d)¥|(60)) ¥¥.((¥¥d ¥¥d ¥¥d)¥|0) ¥¥s(¥[A-Za-z¥]¥[A-Za-z¥]¥[A-Za-z¥])

lisa.vse.datechecker.wstimestampregex

¥¥d¥¥d¥¥d¥¥d-((1¥[012¥])¥|(0¥¥d)¥|0¥[1-9¥]¥|¥[1-9¥])-((¥[12¥]-¥¥-d)¥|(3¥[01¥])¥|(0?¥[1-9¥]))T((¥[012¥]?-¥¥-d)¥|(2¥[0123¥]))¥:((¥[012345¥]-¥¥-d)¥|(60))¥:((¥[012345¥]-¥¥-d)¥|(60))-¥¥-.-¥¥-d-¥¥-d-¥¥-d¥[-+¥]¥¥d¥¥d¥¥d ¥¥d

lisa.vse.datechecker.ddmmmyyyyregex

((¥[12¥]¥¥d)¥|(3¥[01¥])¥|(0?¥[1-9¥]))(JAN¥|FEB¥|MAR¥|APR¥|MAY¥|JUN¥|JUL¥|AUG¥|SEP¥|OCT¥|NOV¥|DEC)¥¥d ¥¥d ¥¥d ¥¥d

lisa.vse.datechecker.mmmddyyyyregex

(JAN¥|FEB¥|MAR¥|APR¥|MAY¥|JUN¥|JUL¥|AUG¥|SEP¥|OCT¥|NOV¥|DEC)((¥[12¥]¥¥d)¥|(3¥[01¥])¥|(0?¥[1-9¥]))¥¥d ¥¥d ¥¥d ¥¥d

lisa.vse.datechecker.yyyyddmmmregex

¥¥d¥¥d ¥¥d ¥¥d((¥[12¥] ¥¥d)¥|(3¥[01¥])¥|(0?¥[1-9¥]))(JAN¥|FEB¥|MAR¥|APR¥|MAY¥|JUN¥|JUL¥|AUG¥|SEP¥|OCT¥|NOV¥|DEC)

lisa.vse.datechecker.yyyymmmddregex

¥¥d¥¥d¥¥d¥¥d(JAN¥|FEB¥|MAR¥|APR¥|MAY¥|JUN¥|JUL¥|AUG¥|SEP¥|OCT¥|NOV¥|DEC)((¥[12¥] ¥¥d)¥|(3¥[01¥])¥|(0?¥[1-9¥]))

lisa.vse.datechecker.ddmmmregex

((¥[12¥]¥¥d)¥|(3¥[01¥])¥|(0?¥[1-9¥]))(JAN¥|FEB¥|MAR¥|APR¥|MAY¥|JUN¥|JUL¥|AUG¥|SEP¥|OCT¥|NOV¥|DEC)

lisa.vse.datechecker.mmmddregex

(JAN¥|FEB¥|MAR¥|APR¥|MAY¥|JUN¥|JUL¥|AUG¥|SEP¥|OCT¥|NOV¥|DEC)((¥[12¥]¥¥d)¥|(3¥[01])|(0?¥[1-9¥]))

lisa.vse.datechecker.ddmmmyyregex

((¥[12¥]¥¥d)¥|(3¥[01¥])¥|(0?¥[1-9¥]))(JAN¥|FEB¥|MAR¥|APR¥|MAY¥|JUN¥|JUL¥|AUG¥|SEP¥|OCT¥|NOV¥|DEC)¥¥d¥¥d

VSE では、以下の各エントリは日付の一部として有効なパターンを表します。

lisa.vse.datechecker.dayformat

形式： dd

lisa.vse.datechecker.monthnumberformat

形式： MM

lisa.vse.datechecker.monthalphaformat

形式： MMM

lisa.vse.datechecker.yearlongformat

形式： yyyy

lisa.vse.datechecker.yearshortformat

形式： yy

lisa.vse.datechecker.timeformat

形式： HH:mm:ss

lisa.vse.datechecker.time.hhmmssformat

形式： HH:mm:ss

lisa.vse.datechecker.time.millisformat

形式： HH:mm:ss.SSS

lisa.vse.datechecker.time.millis.zoneformat

形式： HH:mm:ss.SSS z

lisa.vse.datechecker.wstimestampformat

形式： yyyy-MM-dd'T'HH:mm:ss.SSSZ

以下の日付パターンは、少なくとも日、月、および年を含む上記のパターンの組み合わせを使用しないと構成できません。

lisa.vse.datechecker.mmmddyyyy.separatorformat

形式： MMM*dd*yyyy

lisa.vse.datechecker.mmddyyyy.separatorformat

形式： MM*dd*yyyy

lisa.vse.datechecker.ddmmmyyyy.separatorformat

形式： dd*MMM*yyyy

lisa.vse.datechecker.ddmmyyyy.separatorformat

形式： dd*MM*yyyy

lisa.vse.datechecker.yyyymmmdd.separatorformat

形式： yyyy*MMM*dd

lisa.vse.datechecker.yyyymmdd.separatorformat

形式： yyyy*MM*dd

lisa.vse.datechecker.ddmmmyyyyformat

形式： ddMMMyyyy

lisa.vse.datechecker.mmmddyyyyformat

MMMddyyyy

lisa.vse.datechecker.yyyyddmmmformat

形式： yyyyddMMM

lisa.vse.datechecker.yyyymmmddformat

形式： yyyyMMMdd

lisa.vse.datechecker.ddmmmyyformat

形式： ddMMMyy

lisa.vse.datechecker.ddmmmformat

形式： ddMMM

lisa.vse.datechecker.mmmddformat

形式： MMMdd

lisa.vse.datechecker.separators

日付形式で使用する有効な区切り文字を定義します。

値： -/.

例： **lisa.vse.datechecker.separators=/** は、10/15/2011 のように区切り文字として「/」を設定します。

lisa.vse.datechecker.top.priorityorder=lisa.vse.datechecker.wstimestampformat

日付パターンに一致する順序を定義します。
lisa.vse.datechecker.separators で定義されている区切り文字によって、アスタリスクが置換されます。

例： MM*dd*yy から 4 つの日付パターン、「MM-dd-yyyy」、「MM dd yyyy」、「MM/dd/yyyy」、「MM.dd.yyyy」を生成します。

lisa.vse.datechecker.date.priorityorder

lisa.vse.datechecker.mmmddyyyy.separatorformat&¥
lisa.vse.datechecker.mmddyyyy.separatorformat&¥
lisa.vse.datechecker.ddmmmyyyy.separatorformat&¥
lisa.vse.datechecker.ddmmyyyy.separatorformat&¥
lisa.vse.datechecker.yyyymmmdd.separatorformat&¥
lisa.vse.datechecker.yyyymmdd.separatorformat&¥
lisa.vse.datechecker.mmmddyyyyformat&¥
lisa.vse.datechecker.ddmmmyyyyformat&¥
lisa.vse.datechecker.yyyyddmmmformat&¥
lisa.vse.datechecker.yyyymmmddformat&¥
lisa.vse.datechecker.ddmmmyyformat

lisa.vse.datechecker.time.priorityorder

lisa.vse.datechecker.time.millis.zoneformat&¥
lisa.vse.datechecker.time.millisformat&¥
lisa.vse.datechecker.time.tenthsformat&¥
lisa.vse.datechecker.time.hhmmssformat

lisa.vse.datechecker.bottom.priorityorder

lisa.vse.datechecker.mmmddformat&¥
lisa.vse.datechecker.ddmmmformat

lisa.vse.datechecker.months

値： JAN、FEB、MAR、APR、MAY、JUN、JUL、AUG、SEP、OCT、NOV、DEC

lisa.vse.datechecker.coarse.year.regex

形式：
(?ism).*((¥¥d[./-](19|20)[0-9]{2})|((JAN|FEB|MAR|APR|MAY|JUN|JUL|AUG|SEP|OCT|NOV|DEC)[. /-](19|20)[0-9]{2})|((19|20)[0-9]{2}[. /-][0-9]{2})).*

年に似ている何かがペイロードに含まれるかどうかを判定する、大まかなレベルのテストを定義します。 一致しすぎないようにするために、月情報や標準的な日付区切り文字（-、/、. など）のような、その他の識別情報を付近で探します。

年がその他の 4 桁の数と区別しにくい場合、このパターンは一致しない可能性があります。 その場合、マジック デートは見つかりません。 以下のいずれかの解決方法を試行します。

- この正規表現を調整する
- 以下に示す、より許容度の高い形式を試行する

lisa.vse.datechecker.coarse.year.regex

形式：
(?ism).*((¥¥d[./-]?(19|20)[0-9]{2})|((JAN|FEB|MAR|APR|MAY|JUN|JUL|AUG|SEP|OCT|NOV|DEC)[. /-]?(19|20)[0-9]{2})|((19|20)[0-9]{2}[. /-]?[0-9]{2})).*

年に似ている何かがペイロードに含まれるかどうかを判定する、上記のものより許容度の高い、大まかなレベルのテストを定義します。 日付が認識されていない場合、この正規表現によって日付が見つかる場合があります。 一致しすぎないようにするために、月情報や標準的な日付区切り文字（-、/、. など）のような、その他の識別情報を付近で探します。

ただし、この正規表現はパフォーマンスに影響する可能性があります。 この正規表現に一致する日付が多すぎる場合、適用可能でない状況で上記のパターンが実行されています。 大きなペイロードに対して多数の正規表現パターンを実行すると、時間がかかることがあります。

VSE JMS メッセージング - 無視されるカスタム JMS プロパティ

vse.jms.ignore.proplist

一部の JMS プラットフォームでは、それらが送信する JMS メッセージに余分なカスタム プロパティが含まれています。これらのプロパティは、VSE の作成を妨げる可能性があります。 **vse.jms.ignore.proplist** には、VSE メッセージング レコーダで JMS サービスを記録する場合に無視されるプロパティのリストが含まれます。 その形式は、正規表現のカンマ区切りリストです。デフォルトでは、標準 JMS 拡張プロパティ（JMSX）および JBoss カスタム ブックキーピング プロパティ（JMS_.）を除外します。

デフォルト： JMSX.*, JMS_.*

VSE レコーダ会話バッチ サイズ

lisa.vse.recorder.conversation.batch.size

データベースにコミットする前に処理する会話の数を定義します。 サイズの大きな少数の会話がある場合は、この数を小さくします。 サイズの小さい多数の会話がある場合は、この数を大きくします。 バッチ サイズ <= 0 は、バッチ サイズ 1 と同じです。

デフォルト： 10

VSE サービス イメージ データベース設定

eclipselink.ddl-generation

デフォルト： create-tables

eclipselink.ddl-generation.output-mode=database

サービス イメージのリポジトリとして使用するデータベースを定義します。 VSE は、オープン ソースの EclipseLink JPA プロバイダを使用します。 デフォルトでは、サービス イメージのリポジトリとしてレポート データベースを使用します。

デフォルト： LISA レポート データベース

VSE スキーマの検証

lisa.eclipselink.query.warn.threshold

EclipseLink がクエリのタイミングを調整するように設定されている場合（デフォルトの場合）、この値は、**com.itko.lisa.vse.stateful.model.SqlTimer ロガー**が WARN レベル メッセージを記録する前にデータベース クエリが可能な最大間隔（ミリ秒単位）を定義します。 ロガーでしきい値を DEBUG に設定すると、EclipseLink が発行するすべての SELECT ステートメントのタイミングが返されます。 このログは、VSE のパフォーマンスの問題をデバッグする最初のステップです。 データベースがローカルで、インデックスがすべて作成されている場合、参照に約 20 ミリ秒を超えることはありません。 ただし、ユーザの管理下になく、インデックスがない可能性のあるネットワーク上のデータベースでは、結果を返すために優に 100 ミリ秒以上かかる場合があります。 この場合、正しく展開すれば実際には高速な場合でも、VSE が遅いように見える可能性があります。 ほとんどのサービス イメージは、VSE エンティティ キャッシュに十分対応できるほどサイズが小さいか、または十分に小さいワーキング セットを備えています。 そのため、VSE に十分な容量のヒープがあれば、VSE によって発行される SELECT の数はゼロにまで減少します。 キャッシュはデフォルトでソフト参照を維持します。

デフォルト： 100

VSE 区切り付きコンテンツ検出プロパティ

lisa.vse.delimited.delimiter.check.maxchars

コンテンツ タイプを特定するために確認する最大文字数を定義します。

デフォルト： 1000

lisa.vse.delimited.delimiter.check.maxpercent

コンテンツ タイプを特定するために確認する文字の最大パーセンテージを定義します。 このプロパティが **lisa.vse.delimited.delimiter.check.maxchars** より大きい場合に使用されます。

デフォルト： 2

lisa.vse.delimited.delimiter.list

Delimited Content Detector が区切り文字として探す文字を定義します。

デフォルト： :,¥¥,,;,|,¥u00A6,¥t,¥n

lisa.vse.delimited.delimiter.minimum.threshold

Delimited Content Detector がカウントする区切り文字の最小数を定義します。ペイロードには、少なくともこの数の区切り文字が必要です。

デフォルト： 3

lisa.vse.delimited.namevalue.separator.list

名前と値が区切られたペイロードを検出するために使用される名前/値区切り文字を定義します。 これらの値は **lisa.vse.delimited.delimiter.list** にあってはなりません。

REST データ プロトコル プロパティ

lisa.protocol.rest.editor.observedtraffic.max

URI ルール ウィザードの下部ペインに入力される一致するトラフィックのアイテムの最大数を定義します。 ウィザードが下部ペインに一致するトラフィックを入力するために長時間かかる場合は、このフィールドの値を小さくしてください。

デフォルト： 100

lisa.protocol.rest.editor.unmatchedtraffic.max

一致しないトラフィックのダイアログ ボックスで表示される一致しないトラフィックの項目の最大数を定義します。 ウィザードがこのウィザードに入力するために長時間かかる場合は、このフィールドの値を小さくしてください。

デフォルト： 100

lisa.protocol.rest.idPattern

識別子と見なすことができる HTTP 要求の一部を検出するために、REST データ プロトコルが使用する正規表現を定義します。 このデータ プロトコルは、そのような識別子を動的パラメータに変換します。 特定の形式に従うことがわかっている動的パラメータがある場合、このプロパティを使用してそのパラメータを検出できます。

デフォルト： [a-zA-Z]+[0-9]{5,}[a-zA-Z]*

例： 値 **id12345** でストック項目を示すことができ、後続のルールには {URLPARAM0} が含まれます。

lisa.protocol.rest.maxChanges

この数を超えると変動が大きすぎるのでルールを生成する必要があると認識される、トークンに対して VSE で許可される変更の最大数を定義します。

デフォルト： 1

lisa.protocol.rest.parameterBaseName

REST データ プロトコルがルールのパラメータに使用するプレフィックスを定義します。

デフォルト： URLPARAM

lisa.protocol.rest.startPosition

REST データ プロトコルが変数トークンの検索を開始する URL 内の位置を定義します。

デフォルト： 3

例： **GET /a/b/c/d** という URI では、**GET** は位置 0、**a** は位置 1、**b** は位置 2 にあります。

vse.log.trace.truncate.response.at=2048

DevTest が vse.log に書き込む応答のサイズを制限します。トレース レベルのログ記録が有効な場合にのみ、このプロパティが適用されます。具体的には、**com.itko.lisa.vse.http.Transaction**=TRACE、または **com.itko.lisa.vse.stateful.protocol.http.Coordinator**=TRACE の場合です。

値

- **0**： 全応答がログ記録されます。
- **>0**: ログ記録された応答は、指定された文字で切り捨てられます。

デフォルト： 2048

ネットワーク ポート プロパティ

ネットワーク ポート プロパティは、**lisa.net.APPID.port** の形式です。

これらのデフォルトを変更する必要がある場合は、**site.properties** ファイルで上書きします。 クライアントは、これらの値に基づいてサーバのどのポートに接続するかを想定します。

lisa.net.0.port=2008

DevTest ワークステーション

lisa.net.2.port=2011

コーディネータ

lisa.net.3.port=2010

レジストリ

lisa.net.4.port=2012

JUnit 実行

lisa.net.5.port=2005

テスト ランナー（cmdline）

lisa.net.6.port=2007

その他

lisa.net.8.port=2013

VSE

lisa.net.9.port=2004

VSE マネージャ

lisa.net.11.port=2006

サービス マネージャ

lisa.net.1.port=2014

シミュレータ（この番号からシミュレータを指定することにより、同じコンピュータで多数のシミュレータを持つことが容易です）。

CA Continuous Application Insight プロパティ

lisa.pathfinder.on

デフォルト： true

lisa.pathfinder.broker.host

デフォルト： 0.0.0.0

lisa.pathfinder.broker.port

デフォルト： 2009

lisa.webserver.port

組み込み Web サーバ。

デフォルト： 1505

lisa.webserver.https.enabled

デフォルト： false

devtest.port

デフォルト： 1507

lisa.enable.workstation.pathfinder.browser.ui

デフォルト： true

lisa.portal.enable.pathfinder.browser.ui

デフォルト： true

lisa.webserver.host

組み込み Web サーバのホスト。 *0.0.0.0* はすべてのローカル アドレスにバインドします。

デフォルト： 0.0.0.0

lisa.webserver.acl.authmodule.baseurl

デフォルト： /acl/

これらのプロパティは、DevTest ワークステーション のすべてのポータル アプリケーション ランチャに対して値を設定します。 現在、ホストの値には TR ホスト名が使用されているため、その値を動的に取得します。URL は、システムのブラウザで外部的に起動されます。

lisa.portal.url.prefix

デフォルト： http://* （http://¥*）

lisa.portal.root.base.url

デフォルト： /index.html

lisa.portal.cvsdashboard.base.url

デフォルト： /index.html?lisaPortal=cvsdashboard

lisa.portal.pathfinder.console.base.url

デフォルト： /index.html?lisaPortal=pathfinder

lisa.portal.server.console.base.url

デフォルト： /index.html?lisaPortal=serverconsole

lisa.portal.model.execution.url

デフォルト： /index.html?lisaPortal=serverconsole

lisa.portal.reporting.context

デフォルト： reporting

lisa.portal.reporting.console.base.url

デフォルト： /index.html?lisaPortal=reporting

lisa.portal.defect.capture.url

デフォルト： /pathfinder/lisa_pathfinder_agent/defectcapture

lisa.portal.save.defect.data.url

デフォルト： /pathfinder/lisa_pathfinder_agent/saveDefectData

lisa.portal.invoke.base.url

デフォルト： /lisa-invoke

lisa.portal.invoke.report.url

デフォルト： /reports

lisa.portal.invoke.server.report.directory

デフォルト： lisa.tmpdirlisa.portal.invoke.report.url

devtest.portal.base.url.path

デフォルト： /devtest/#/main

devtest.portal.homepage.url.path

デフォルト： {{devtest.portal.base.url.path}}/dashboard

lisa.portal.pathfinder.explorer.base.url

デフォルト： {{devtest.portal.base.url.path}}/createartifacts

lisa.portal.pathfinder.management.base.url

デフォルト： {{devtest.portal.base.url.path}}/pathManageAgents

lisa.portal.invoke.test.root

デフォルト： LISA_HOME

レポート グラフ ビュー プロパティ

rpt.lisa.graph.view.threshhold

デフォルト： 100

rpt.lisa.graph.view.infomessage

最大しきい値より多い結果数。 詳細を取得するには、フィルタを変更します。

rpt.lisa.graph.scatter.view.threshhold

デフォルト： 10000

rpt.lisa.graph.scatter.view.infomessage

最大しきい値より多い結果数。 詳細を取得するには、フィルタを変更します。

非同期レポートのサポート

lisa.reporting.useAsync

負荷テストを実行し、テスト ケースのボトルネックがレポート データベースへのイベントの書き込みであると判明した場合は、レポート ジェネレータからメトリック以外のすべてを削除することを検討します。 それでもレポート エンジンがボトルネックと考えられる場合は、このプロパティを有効にすることを検討します。 このプロパティは、JMS を使用してレポート イベントを送信します。また、シミュレータおよびコーディネータのバックグラウンド スレッドはデータベースにイベントを非同期で書き込みます。 これは、すべてのイベントがデータベースに書き込まれる前に負荷テストが終了することを意味します。そのため、しばらくの間、レポートが表示されません（時間はテスト ケースと生成されたイベント数によって異なります）。シミュレータのキューは、通常、フラッシュに最も時間がかかります。シミュレータのログには、完了したパーセンテージを示す INFO レベルのメッセージが書き込まれます。

現在のところ、この機能は「高度な利用方法」と考えられており、デフォルトでは無効です。 フィードバックは、http://ca.com/support までお寄せください。

デフォルト： false

lisa.reporting.step.max.propsused.buffersize

デフォルト： 100

lisa.reporting.step.max.propsset.buffersize

これらのプロパティは、テスト ケースの実行時にテスト ステップの統計の収集を制御します。 これらの値は、使用および設定されるプロパティに対して記録される発生の最大数を指定します。 デフォルト値を変更する必要はありません。

デフォルト： 100

lisa.threadDump.generate

デフォルト： true

lisa.threadDump.interval

デフォルト： 30

lisa.threadDump.loggerName

定期的なスレッド ダンプを有効にします。これは有効にしておくことをお勧めします。 それによって、threadDumpLogger が INFO 以下であるかどうかが効率よく確認されます。また、WARN 以上に設定されている場合は、何も行いません。 プロパティは起動時に 1 回だけ読み取られますが、**logging.properties** ファイルでは 10 秒ごとに変更の有無が確認されます。必要なことは、このプロパティをそのままにしておき、threadDumpLogger のログ レベルを INFO に設定し、実行中の DevTest サーバ の定期的なスレッド ダンプが取得されるのを最大で 30 秒間待機するだけです。 パフォーマンスの問題をデバッグする場合に、これらのログは役立ちます。 詳細については、**logging.properties** のコメントを参照してください。

デフォルト： threadDumpLogger

これらのプロパティは、VSE サーバのメトリック収集を制御するために使用されます。 収集されたメトリックは、Web ベースの VSE ダッシュボードで表示できます。

lisa.vse.metrics.collect

メトリック収集をオン（true）またはオフ（false）にする主要なプロパティ。

デフォルト： true

lisa.vse.metrics.txn.counts.level

このプロパティは、トランザクション数が記録されるレベルを制御します。 名前付きのトランザクション数が合計されると、任意の期間のトランザクション数を取得します。

値： none （false）、service、request

- **service：** サービス名トランザクション数の名前。
- **request：** サービス名＋要求操作名トランザクション数。

デフォルト： service

lisa.vse.metrics.sample.interval

このプロパティは、トランザクション レートおよび応答時間がサンプリングされる頻度を制御します。

デフォルト： 5m

lisa.vse.metrics.delete.cycle

デフォルト： 1h

lisa.vse.metrics.delete.age

これらのプロパティは、古いメトリック データをスキャンして削除する頻度、および何を古いと見なすかを制御します。

デフォルト： 30d

lisadb.internal.enabled

レジストリで内部 Derby データベース インスタンスを起動するかどうかを示します。

デフォルト： true

lisadb.internal.host

内部 Derby データベースが使用するネットワーク インターフェース。デフォルト値の 0.0.0.0 は、すべてのインターフェースが使用されることを示します。

デフォルト： 0.0.0.0

lisadb.internal.port

内部 Derby データベースがリスンするポート番号。

デフォルト： 1528

lisa.acl.audit.logs.delete.frequency

デフォルト： 1d

lisa.acl.audit.logs.delete.age

これらのプロパティは、古い ACL 監査ログ データをスキャンして削除する頻度、および何を古いと見なすかを制御します。

デフォルト： 30d

データベース プロパティ

データベースと通信するコンポーネントは、レポート、エージェント ブローカ、VSE、および ACL です。 通常、それらのすべてに同じ接続プールを指定しますが、それぞれに個別のプールを定義したり、組み合わせたりすることもできます。 新しいプールを定義するには、**lisa.db.pool.myPool.url** などのプロパティを設定します。基盤となるプール実装はオープン ソースの c3p0 プールで、さまざまなプロパティが継承されています。 使用可能な設定については、http://www.mchange.com/projects/c3p0/index.html#configuration_properties を参照してください。

lisadb.reporting.poolName

デフォルト： common

lisadb.vse.poolName

デフォルト： common

lisadb.acl.poolName

デフォルト： common

lisadb.broker.poolName

デフォルト： common

lisadb.pool.common.driverClass

デフォルト： org.apache.derby.jdbc.ClientDriver

lisadb.pool.common.url

デフォルト jdbc:derby://localhost:1528/database/lisa.db;create=true

lisadb.pool.common.user

デフォルト： rpt

lisadb.pool.common.password_enc

プロパティ名の末尾の _enc を削除し、*=MyPlaintextPassword* を追加して、パスワードを設定します。 パスワードは、起動時に自動的に暗号化されます。

デフォルト： 76f271db3661fd50082e68d4b953fbee

以下のプール プロパティは、DevTest のアイドル時の接続数を最小限に抑えます。

lisadb.pool.common.minPoolSize

デフォルト： 0

lisadb.pool.common.minPoolSize

デフォルト： 0

lisadb.pool.common.maxPoolSize

デフォルト： 10

lisadb.pool.common.acquireIncrement

デフォルト： 1

lisadb.pool.common.maxIdleTime

デフォルト： 45

lisadb.pool.common.idleConnectionTestPeriod

デフォルト： 5

その他の共通のデータベース設定： 共通のテンプレートから関連するユーザ、パスワード、およびプール サイズ パラメータをコピーして貼り付けます。

lisadb.pool.POOLNAME.driverClass

デフォルト： oracle.jdbc.OracleDriver

lisadb.pool.POOLNAME.url

デフォルト： jdbc:oracle:thin:@HOST:1521:SID

lisadb.pool.POOLNAME.driverClass

デフォルト： com.ibm.db2.jcc.DB2Driver

lisadb.pool.POOLNAME.url

デフォルト： jdbc:db2://HOST:50000/DBNAME

lisadb.pool.POOLNAME.driverClass

デフォルト： com.microsoft.sqlserver.jdbc.SQLServerDriver

lisadb.pool.POOLNAME.url

デフォルト： jdbc:sqlserver://durry;databaseName=LISA

lisadb.pool.POOLNAME.driverClass

デフォルト： com.mysql.jdbc.Driver

lisadb.pool.POOLNAME.url

デフォルト： jdbc:mysql://HOST:3306/DBNAME

lisa.jdbc.asset.pool.size

JDBC 接続アセットを定義する場合、接続プール サイズはこのプロパティによって設定できます。

デフォルト： 5

メインフレーム プロパティ

lisa.mainframe.bridge.enabled

CICS メインフレーム ブリッジを有効にするかどうかを示します。

デフォルト： false

lisa.mainframe.bridge.mode

メインフレーム ブリッジをクライアント モードまたはサーバ モードのどちらで実行するかを示します。

デフォルト： server

lisa.mainframe.bridge.port

メインフレーム ブリッジがサーバ モードで実行されている場合、このポートはメインフレーム ブリッジがリスンするウェルノウン ポートです。

デフォルト： 61617

lisa.mainframe.bridge.server.host

デフォルト： 127.0.0.1

lisa.mainframe.bridge.server.port

メインフレーム ブリッジがクライアント モードで実行されている場合、これらの値は LPAR エージェントの IP アドレスおよびウェルノウン ポートです。

デフォルト： 3997

lisa.mainframe.bridge.connid

各クライアントの 2 文字の一意の ID。

デフォルト： AA

詳細については、「エージェント」の「メインフレーム ブリッジ」を参照してください。

WebSphere MQ プロパティ

lisa.mq.ccsid.default=819

このプロパティは、DevTest から送信されるすべての IBM WebSphere MQ メッセージのデフォルトのコード化文字セット ID（CCSID）を上書きするために使用できます。 この値は、**IBM WebSphere MQ** ステップの［パブリッシャ情報］の［メッセージ プロパティ］で各ケースに対して上書きできます。 詳細については、「*CA Application Test の使用*」を参照してください。 CCSID の全リストについては、http://www-01.ibm.com/software/globalization/ccsid/ccsid_registered.html を参照してください。 米国ロケールのデフォルト値は、819 （ASCII）です。

ローカライズ オプション

lisa.locale.languages=en

このプロパティを使用して、UI でサポートする言語を指定することができます。 有効な値は、**en** （英語）または **ja** （日本語）です。

ここで **en** と **ja** の両方を指定した場合、［メイン］メニューの［システム］-［言語］オプションを使用して、英語と日本語の UI を切り替えることができます。

lisa.supported.html.request.encodings=

有効な値は、ISO-8859-1、UTF-8、Shift_JIS、EUC-JP、Windows-31J です。

Selenium 統合プロパティ

selenium.enable.waitfor=true

Web エレメントを検索する必要があるテスト ケースの各ステップに waitForElementPresent ステップを暗黙的に追加するかどうかを指定します。

値

- **True :** 実際のステップの前に実行され、エレメントが存在し、ページがロードされていることを確認する、暗黙的なステップを追加します。
- **False :** 暗黙的なステップを追加しません。 このプロパティを false に設定した場合、スクリプトが Selenium Builder で正常に実行されても、ページのロードに遅延が発生するために DevTest で失敗する状況が発生する可能性があります。 すでに Selenium Builder で waitForElementPresent ステップを定義している場合は、このプロパティを false に設定できます。

selenium.browser.type

Selenium 統合テスト ケースを実行するためのブラウザのタイプを定義します。 このプロパティは、プロジェクト設定ファイルで使用します。

値

- Chrome
- IE
- Firefox

注: このプロパティが指定されていない場合、テストはデフォルトで Firefox で実行されます。

selenium.ie.driver.path

Microsoft Internet Explorer ブラウザで Selenium 統合テストを実行するために使用される、ローカル コンピュータ上の Selenium ドライバのフル パスを定義します。 このプロパティは、プロジェクト設定ファイルで使用します。

例:

```
C:¥lisa-se¥IEDriverServer.exe
```

selenium.chrome.driver.path

Chrome ブラウザで Selenium 統合テストを実行するために使用される、ローカル コンピュータ上の Selenium ドライバのフル パスを定義します。 このプロパティは、プロジェクト設定ファイルで使用します。

例:

```
C:¥lisa-se¥ChromeDriverServer.exe
```

selenium.remote.url

リモートのブラウザで Selenium 統合テスト ケースを実行するためのリモート Selenium Server ハブの URL を定義します。このプロパティは、プロジェクト設定ファイルで使用します。

例:

```
http://your_remote_hostname:4444/wd/hub
```

注: 同じプロジェクト設定ファイルで、**selenium.chrome.driver.path** および **selenium.ie.driver.path** を定義できます。 **selenium.chrome.driver.path** または **selenium.ie.driver.path** が含まれる設定ファイルに、**selenium.remote.url** も含めることはできません。

selenium.WebDriver.DesiredCapabilities.filePath

Selenium Web ドライバの詳細オプションの設定に使用されたパラメータ ファイルの場所を指定します。

デフォルト： {{LISA_PROJ_ROOT}}/Data/selenium-capabilities.conf

VSEasy プロパティ

VSEasy で HTTP プロトコルを使用して自動設定を使用する場合、ポート プロパティによってポートの動的な割り当てを制御します。

最小のポートが 1024 未満である場合、値は 1024 に設定されます。最大のポートが 65535 を超える場合、値は 65535 に設定されます。最大のポートが最小のポート以下である場合、両方の値はデフォルトに戻ります。

lisa.vseasy.http.min.dynamic.port

デフォルト： 8000

lisa.vseasy.http.max.dynamic.port

デフォルト： 65535

lisa.vseasy.default.group.tag

VSEasy コンソールのデフォルト グループの名前を指定します。

デフォルト： VSEasy

付録 B: カスタム プロパティ ファイル

カスタム プロパティ用に以下の 2 つのプロパティ ファイルを使用できます。

- local.properties
- site.properties

サイト プロパティはテスト サーバに格納できます。これにより、テスト サーバに接続するすべてのワークステーションへ **site.properties** ファイルが自動的にプッシュされます。

プロパティ ファイルは以下の優先順位で評価されます： コマンド ラインおよび vmoptions ファイルは常にプロパティ ファイルよりも優先されます。次に、**local.properties** は **site.properties** よりも優先され、site.properties は **lisa.properties** よりも優先されます。

注: DevTest ワークステーション が起動されると、レジストリに接続するように促されます。 レジストリに接続した後、site.properties はワークステーションに送信されます。 レジストリを Registry1 から Registry2 に変更すると、Registry2 から **site.properties** がワークステーションに送信されます。

カスタム プロパティ ファイルを使用する方法

1. LISA_HOME ディレクトリに移動します。
2. **_site.properties** または **_local.properties** ファイルをコピーし、同じディレクトリにそれを貼り付けます。
3. 貼り付けたファイルの名前を **site.properties** または **local.properties**（先頭のアンダースコアなし）に変更します。

このセクションには、以下のトピックが含まれています。

ローカル プロパティ ファイル (P. 576)
サイト プロパティ ファイル (P. 604)
logging.properties (P. 607)

ローカル プロパティ ファイル

lisaAutoConnect

サイト全体のプロパティを DevTest レジストリから自動的にロードするには、このプロパティのコメントを外し、DevTest レジストリの正しい URL に設定します。 通常、**hostname/lisa.TestRegistry** プロパティは動作します。 このファイル内のプロパティは、サイト レベルで定義されるプロパティを上書きします。

サイト全体のプロパティを DevTest レジストリから自動的にロードするには、このプロパティのコメントを外し、DevTest レジストリの正しい URL に設定します。通常、**hostname/lisa.TestRegistry** は動作します。このファイル内のプロパティは、非 GUI コンポーネントに対してサイト レベルで定義されるプロパティを上書きします。

DevTest ワークステーション はこのプロパティを使用しません。 レジストリ選択のダイアログ ボックスまたはワークステーションの［レジストリの変更］ボタンを使用して、接続するレジストリを選択できます。 レジストリ選択のダイアログ ボックスで［起動時に確認する］チェック ボックスをオフにすると、DevTest ワークステーション は最後に使用したレジストリに接続します。

形式： tcp://somehost/Registry

ライセンス プロパティ

laf.server.url=https://license.itko.com

laf.domain=iTKO/LISA/YOURCO

laf.username=YOURUSERNAME

laf.password=YOURPASSWORD

VSE プロパティ

lisa.vse.deploy.dir

ランタイム ファイルが管理されているディレクトリをオーバーライドできます。 このプロパティには、ディレクトリの絶対パスを設定します。パスの先頭にドライブを指定しない場合は、LISA_HOME からの相対パスになります。 ディレクトリ階層を指定する場合は、C:¥¥Temp¥¥myVSE の形式で指定します。

注: プロパティ ファイルでは、円記号を認識させるために 2 つ並べる（エスケープする）必要があります。

lisa.vse.si.editor.prefer.xml

有効な値は、**typemap.properties** の **vseTextBodyEditors** 名に割り当てられている VSE SI テキスト エディタ クラス名のリストです。 カスタム テキスト エディタを記述し、標準の SDK メソッドを介して使用可能にする場合は、そのクラス名をこのプロパティの値に指定することもできます。

以下のプロパティでは、VSE に新しい I/O （NIO）ではなく、古いソケット I/O を使用させることができます。 古いソケットのサポートを使用する場合は、以下の制限に注意してください。

- VSE が SSL の前に自動的にプロキシ処理を行う機能はサポートできません。 この解決策は、ライブ システム モードの VSM を作成し、SSL を有効にして実際の仮想サービスを指定することです。
- リスン ステップが SSL を使用するように設定されている場合でも、VSE がプレーン テキスト トラフィックを処理する機能はサポートできません。 この解決策は、SSL が設定された仮想サービスと、SSL が設定されていない仮想サービスの 2 つを提供することです。
- 古いソケットのサポートの使用は、VSE が使用するデフォルトのソケットのサポートほど効率的に拡張できません。

lisa.vse.tcp.uses.nio

このプロパティを false に設定すると、プレーン ソケットと SSL ソケットの両方で古い I/O を使用します。

デフォルト： true

lisa.vse.plain.tcp.uses.nio

このプロパティはプレーン ソケットのみを制御します。

デフォルト： sa.vse.tcp.uses.nio

lisa.vse.ssl.tcp.uses.nio

このプロパティは SSL ソケットのみを制御します。

デフォルト： isa.vse.tcp.uses.nio

lisa.vse.execution.mode

- EFFICIENT は、VS モデルを介した最も効率的なパスを使用します。
- TRACK は、VS レベルで VSE アクティビティを記録します。
- LIVE は、VS モデルによって受信された要求をライブ システムにルーティングします（パス スルー モードに似ています）。
- VALIDATION は、応答を決定するために VSE とライブ システムの両方を使用します。 どちらもトラッキング情報として記録され、モデルの修正プロセスを提供します。 ライブ応答は仮想サービスの応答になります。
- DYNAMIC は、その他の 4 つのモードのいずれかに解決する必要があるすべての要求用のスクリプトまたはサブプロセスを呼び出します。 この方法は、仮想サービス モデルのみが参照する要求を追跡するのに役立ちます。
- LEARNING は、ライブ システムからの新しい応答または更新された応答を持つように仮想サービスを自動的に「修復」または修正します。
- STAND_IN は、最初に仮想サービスに要求をルーティングします（［最も効率的］モードと同様）。 ただし、仮想サービスの応答がない場合、要求はライブ システムに自動的にルーティングされます。
- FAILOVER は、最初にライブ システムに要求をルーティングします（［ライブ システム］モードと同様）。 ただし、ライブ システムの応答がない場合、要求は仮想サービスに自動的にルーティングされます。

デフォルト： EFFICIENT

lisa.vse.body.cache.weight

特定のキャッシュに保持できるキャッシュ値のサイズをバイト単位で定義します。

デフォルト： 1024 * 1024 * 2 （2 MB）

lisa.vse.metric.collect

false に設定すると、DevTest は、応答時間、強制された遅延、および 1 秒あたりのトランザクション数を計算する定期的なサンプルの収集を停止します。 VSE_METRICS_TXN_COUNTS テーブルで追跡されるその他のメトリック データは、以下の項目のチャートを生成するために引き続き収集されます。

- サーバ チャート
- ライフタイム トランザクション合計
- 日単位トランザクション数
- サーバの可用性
- サービス チャート
- 1 日あたりの合計トランザクション数

false に設定すると、VSE コンソールのチャート リストの以下の項目には、データが表示されません。

- サービス チャート
- トランザクション スループット
- トランザクションのヒット数およびミス数
- レスポンス時間
- 強制された遅延
- 1 秒あたりのトランザクション数

デフォルト： true

lisa.vse.max.hard.errors

VSE サービスを停止させるエラー数を設定できます。

このプロパティの値に関係なく、エラーは引き続き VSE コンソールで赤い円とエラー数と共にレポートされます。

サービスを停止させるエラー数を指定する場合は、有効な数値（0 以上）を入力します。0 の値はエラーが許可されていないことを意味し、サービスは最初のエラーの後にシャット ダウンされます。

エラー数を無制限に指定するには、負の数値を入力します。

無効な値を入力したか、値を入力していない場合は、デフォルトのエラー数の 3 が使用されます。

デフォルト： 3

注: エラーのタイプおよびモデルによっては、最終的なエラー数が **lisa.vse.max.hard.errors** で設定された値より多くなる場合があります。仮想サービスのシャットダウンには、時間がかかる場合があります。シャットダウン中も、一部のタイプのエラーは引き続きカウントされます。

lisa.tcp.tcprecorder.close.immediately

ソケットを TCP のレコーディングの最後（終了時）にすぐ閉じるかどうかを指定します。

ローカル ソケット（リスン ポート）とリモート ソケットの両方が影響を受けます。

ソケットの読み取りまたは書き込み中（サーバまたはクライアントで長い遅延が発生している場合）にレコーディングが終了した場合に、このプロパティは通信を閉じるか完了させるかを指定します。

長時間の会話（大きなファイルのダウンロードなど）の場合、このプロパティは会話を終了させるか完了するまで実行させるかを指定します。

値： 空白、true、false。 値が空白の場合は、デフォルトの true が使用されます。

デフォルト： true

lisa.vse.body.cache.size

VSE 要求オブジェクトおよび応答オブジェクトの圧縮されていないボディをキャッシュするために設定されるメモリの量を指定します。 単位は MB です。

デフォルト： 10

lisa.vse.body.cache.timeout

VSE 要求および応答の圧縮されていないボディをキャッシュに保持しておく時間を指定します。 単位は秒です。

デフォルト： 60

lisa.vse.argument.match.allow.whitespace

このプロパティは、サービス イメージに一致した場合、受信要求の引数の先頭および末尾の空白を許可するように VSE に指示します。 デフォルト値は *false* です。この値を設定すると、VSE は、要求とサービス イメージを照合する前に受信要求の引数の先頭および末尾の空白を削除します。

デフォルト： false

これらのプロパティは、プロトコルがレコーディングおよび再生に使用する IMS Connect メッセージのパラメータを定義します。

lisa.vse.protocol.ims.encoding

デフォルト： EBCDIC

lisa.vse.protocol.ims.header.length

デフォルト： 80

lisa.vse.session.timeout.ms

既存のセッションが見つからない場合、VSE は要求をイメージの各会話のスタータ トランザクションと照合します。一致が見つかった場合は、新しいセッションが作成され、関連する応答が返されます。 セッションは、VSE によって参照された後、2 分間保持されます。 この動作は、**lisa.vse.session.timeout.ms** を設定することによって変更できます。 会話スタータが一致しない場合、セッションは作成されず、定義された順にステートレス トランザクションのリストが確認されます。一致が見つかった場合は、適切な応答が返されます。 それでも一致が見つからない場合は、「不明な要求」応答が送信されます。

デフォルト： 120000

lisa.vse.tcp.oldio.read.timeout

タイム アウトおよび再試行ロジックの実行までに、TCP 読み込みの完了を待機する期間を制御します。 デフォルト設定の 500 ミリ秒は、「通常」またはネットワーク遅延が問題とならない良く調整されたネットワークには十分な値です。 1/2 秒より長い遅延が発生するネットワークの場合、このプロパティをコードが再試行するまでにより長い時間待機するように設定できます。 再試行ロジックは、スループットを多少犠牲にしても（1 秒あたりのトランザクション数を減らしても）、TCP ソケット上の会話を保持するために十分である必要があります。

lisa.vse.tcp.oldio.read.timeout プロパティを有効にするには、NIO を無効にする必要があります。**lisa.vse.ssl.tcp.uses.nio=false** に設定してください。

デフォルト： 500

lisa.vse.conversational.back.navigation.allowed

VSE は、会話型トランザクションでの再生中に記録されていない後方ナビゲーションをサポートします。このプロパティを *true* に設定すると、すべてのサービスの後方ナビゲーションが有効になります。 会話型セッションは受信した要求を追跡し、任意の次の要求を前の要求の 1 つにすることができます。

デフォルト： false

lisa.vse.tracking.delete.data.age

セッション トラッキング情報を保持する時間（時間単位）を定義します。

デフォルト： 8

lisa.vse.protocol.ims.response.includes.llll

要求ペイロードに 4 バイトの IMS Connect LLLL 長フィールドが存在することを示します。値が ture の場合は、LLLL フィールドが含まれています。値が false の場合は含まれていません。

デフォルト： false

lisa.vse.ims.connect.llzz.request

要求ペイロードに 4 バイトの IMS Connect LLZZ 長フィールドが存在することを示します。値が **ture** の場合は、LLZZ フィールドが含まれています。 値が **false** の場合は含まれていません。

デフォルト： false

lisa.vse.ims.connect.llzz.response

応答メッセージに 4 バイトの IMS Connect LLZZ 長フィールドを生成するプロトコルを示します。値が **ture** の場合は、LLZZ フィールドが作成されます。 値が **false** の場合は作成されません。

デフォルト： true

エンタープライズ ダッシュボード プロパティ

lisa.enterprisedashboard.service.url

エンタープライズ ダッシュボードが配置されているサーバの URL。これにより、レジストリをモニタできます。

デフォルト： tcp://localhost:2003/EnterpriseDashboard

以下は、DevTest がエンタープライズ ダッシュボード データベースに接続するために使用するプロパティです。 デフォルトの接続は、内部 Derby データベースへの接続です。 外部データベースへの接続の詳細については、「外部データベースの設定」を参照してください。

lisadb.dradis.poolName

デフォルト： common

lisadb.pool.common.driverClass

デフォルト： org.apache.derby.jdbc.ClientDriver

lisadb.pool.common.url

デフォルト： jdbc:derby://localhost:1530/database/dradis.db;create=true

lisadb.pool.common.user

デフォルト： app

lisadb.pool.common.password_enc

プロパティ名の末尾の _enc を削除し、*=MyPlaintextPassword* を追加して、パスワードを設定します。 パスワードは、起動時に自動的に暗号化されます。

デフォルト： 76f271db3661fd50082e68d4b953fbee

以下のプール プロパティは、DevTest のアイドル時の接続数を最小限に抑えます。

lisadb.pool.common.minPoolSize

デフォルト： 0

lisadb.pool.common.minPoolSize

デフォルト： 0

lisadb.pool.common.initialPoolSize

デフォルト： 0

lisadb.pool.common.maxPoolSize

デフォルト： 10

lisadb.pool.common.acquireIncrement

デフォルト： 1

lisadb.pool.common.maxIdleTime

デフォルト： 45

lisadb.pool.common.idleConnectionTestPeriod

デフォルト： 5

lisadb.internal.enabled

レジストリが内部 Derby データベースを起動するかどうかを制御します。 これまでのコンポーネントですべて外部データベースを使用するように設定している場合は、このプロパティを *false* に設定することによってリソースを節約できます。

デフォルト： true

lisa.default.keystore

デフォルト： {{LISA_HOME}}webreckeys.ks

lisa.default.keystore.pass

デフォルト： passphrase

DevTest から DevTest への通信の暗号化 通常、ネットワーク トラフィックは暗号化されません。暗号化を使用する場合は、標準で SSL がサポートされています。「tcp」を付けてサーバ エンドポイントを指定する代わりに、「ssl」を付けて指定します（例： ssl://hostname:2010/Registry）。 以下のプロパティは、いずれも設定の必要はありません。ssl を付けてエンドポイント（およびサーバ名）を指定するだけで十分です。 たとえば、以下のようにして新しいシミュレータを開始できます。

```
Simulator -name ssl://thishost:2014/Simulator -labName
ssl://regHost:2010/Registry
```

デフォルトで内部自己署名証明書（**LISA_HOME¥webreckeys.ks** 内）が提供されています。 より強力な証明書を使用するには、**lisa.net.keyStore.password** で **lisa.net.keyStore** プロパティおよびプレーン テキスト パスワードを指定します。

次回の DevTest の起動時に、プレーン テキスト パスワードが の暗号化された文字列（**lisa.net.keyStore.password_enc**）に置換されます。

lisa.net.keyStore

デフォルト： {{LISA_HOME}}lisa.ks

lisa.net.keyStore.password

この場所に識別証明書が保存されます。サーバ インストール環境の場合、クライアント側では信頼されたサーバのリストにこの証明書を追加する必要があります。クライアント インストール環境で相互（クライアント）認証を実行している場合、サーバ側でこの証明書が信頼ストアに追加されている必要があります。 相互認証を使用しないクライアント インストール環境の場合、これを指定する必要はありません。

lisa.net.trustStore

デフォルト： {{LISA_HOME}}lisa.ts

lisa.net.trustStore.password_enc

デフォルト： 079f6a3d304a978146e547802ed3f3a4

この場所に信頼された証明書が保存されます。 主としてクライアント インストール環境である場合は、ここには信頼されたサーバのリストが保存されます。 サーバ インストール環境で相互（クライアント）認証を実行している場合、ここに信頼されたクライアント証明書を配置します。

lisa.net.clientAuth

相互認証を使用するかどうかを指定します。

デフォルト： false

lisa.net.default.protocol

値： SSL または TCP

デフォルト： SSL

デフォルト： ssl

HTTP プロキシ サーバを使用する場合のライセンス プロパティ

laf.usehttpproxy.server

デフォルト： true

laf.httpproxy.server

プロキシ サーバを *my_proxyserver.com* の形式で定義します。

laf.httpproxy.port

プロキシ サーバが認証情報を必要とする場合、空白のままにするとネイティブ NTLM 認証を使用します。

デフォルト： 3128

laf.httpproxy.domain

NTLM に必要な場合に、プロキシ ドメインを定義します。

laf.httpproxy.username

オプション

laf.httpproxy.password

オプション

laf.email.alert

ライセンス エラーがある場合に、電子メールを送信します。以下の例を参照してください。 *mailhost* をメール ホストのホスト名に置換します。 *recipient@mycompany.com* を送信先の電子メール アドレスに置換します。 *from@mycompany.com* を電子メールの送信元である電子メール アドレスに置換します。

例： *mailhost,recipient@mycompany.com,from@mycompany.com*

SDK プロパティ

DevTest SDK サンプルには以下が必要です。

asserts

形式： com.mycompany.lisa.AssertFileStartsWith

com.mycompany.lisa.AssertFileStartsWith

形式：
com.itko.lisa.editor.DefaultAssertController,com.itko.lisa.editor.DefaultAssertEditor

filters

形式： com.mycompany.lisa.FilterFileFirstLine

com.mycompany.lisa.FilterFileFirstLine

形式：
com.itko.lisa.editor.FilterController,com.itko.lisa.editor.DefaultFilterEditor

nodes

形式： com.mycompany.lisa.node.FTPTestNode

com.mycompany.lisa.node.FTPTestNode

形式：
com.mycompany.lisa.node.FTPTestNodeController,com.mycompany.lisa.node.FTPTestNodeEditor

SSL プロパティ

SSL 証明書の検証のデフォルト動作を変更します。

ssl.checkexpiry

「true」の値は、証明書の有効期限を検証することを意味します。

デフォルト： false

ssl.checkcrl

「true」の値は、証明書で指定されている証明書失効リストを検証することを意味します。

デフォルト： false

SSL 用のクライアント証明書およびパスワードを有効にします（HTTP ステップおよび RAW SOAP ステップによって使用されます。また、上書きされなかった場合は Web サービス ステップによっても使用されます）。

ssl.client.cert.path

キーストアへのフル パス。

ssl.client.cert.pass

キーストアのパスワード。 DevTest が実行されると、このパスワードは自動的に暗号化されます。

ssl.client.key.pass

JKS キーストアを使用する場合に、キーにキーストアとは異なるパスワードがあるときのキー エントリ用のオプションのパスワード。 DevTest が実行されると、このパスワードは自動的に暗号化されます。

Web サービス ステップでは、SSL 用のクライアント証明書およびパスワード（ssl.client.cert.path および ssl.client.cert.pass）を上書きします。

ws.ssl.client.cert.path

キーストアへのフル パス。

ws.ssl.client.cert.pass

キーストアのパスワード。 DevTest が実行されると、このパスワードは自動的に暗号化されます。

ws.ssl.client.key.pass

JKS キーストアを使用する場合に、キーにキーストアとは異なるパスワードがあるときのキー エントリ用のオプションのパスワード。DevTest が実行されると、このパスワードは自動的に暗号化されます。

HTTP 認証プロパティ

DevTest が実行されると、これらの認証情報は自動的に暗号化されます。値をリセットするには、暗号化されていないプロパティ名を使用します。ネイティブ NTLM 認証を使用する場合は（Windows のみ）、これらの設定をコメント アウトしておきます。

lisa.http.domain

ドメイン名（このオプションは NTLM に使用します）

lisa.http.user

ユーザ名

lisa.http.pass

パスワード

lisa.http.preemptiveAuthenticationType

チャレンジが送信されるのを待たずに、先に認証情報を送信します。

値： basic、ntlm、negotiate

デフォルト： ntlm

lisa.http.forceNTLMv1

デフォルトでは NTLMv2 が使用されます。ただし、このプロパティを *true* に設定することにより、ネイティブの統合 Windows 認証を使用しない場合に NTLMv1 を使用することを強制できます。

デフォルト： true

Kerberos 認証プロパティ

lisa.java.security.auth.login.config

ログイン設定ファイルの場所。

lisa.java.security.krb5.conf

事前設定された場所を上書きするために使用される Kerberos 設定ファイルの場所。

lisa.http.kerberos.principal

プリンシパルおよびパスワード認証の DevTest でのサポートを使用する場合にログインに使用されるプリンシパルの名前。 このパラメータは、DevTest ワークステーション の起動時に暗号化されます。

lisa.http.kerberos.pass

プリンシパルおよびパスワード認証の DevTest でのサポートを使用する場合にログインに使用されるパスワード。 このパラメータは、DevTest ワークステーション の起動時に暗号化されます。

HTTP プロキシ サーバ プロパティ

lisa.http.webProxy.host

マシン名または IP アドレス。

lisa.http.webProxy.nonProxyHosts

プロキシ認証から除外されるマシン名または IP アドレス。複数の値を入力するには、パイプ記号（|）で値を区切ります。 ワイルドカードとして * を使用します。

デフォルト： 127.0.0.1

lisa.http.webProxy.port

lisa.http.webProxy.ssl.host

マシン名または IP アドレス。

lisa.http.webProxy.ssl.nonProxyHosts

SSL プロキシ認証から除外されるマシン名または IP アドレス。 複数の値を入力するには、パイプ記号（|）で値を区切ります。ワイルドカードとして * を使用します。

デフォルト： 127.0.0.1

lisa.http.webProxy.ssl.port

空白のままにすると、統合 NTLM 認証を使用します

lisa.http.webProxy.host.domain

NTLM 認証に使用されます

lisa.http.webProxy.host.account

lisa.http.webProxy.host.credential

lisa.http.webProxy.nonProxyHosts.excludeSimple

プロキシの使用から単純なホスト名を除外します。

デフォルト： true

lisa.http.webProxy.preemptiveAuthenticationType

チャレンジが送信されるのを待たずに、先に認証情報を送信します。

値： basic、ntlm

デフォルト： ntlm

lisa.http.timeout.connection

HTTP タイムアウト（ミリ秒） - タイムアウトを延長して無期限に待機するには、値をゼロに設定します。

デフォルト： 15000

lisa.http.timeout.socket

HTTP タイムアウト（ミリ秒）。タイムアウトを延長して無期限に待機するには、値をゼロに設定します。

デフォルト： 180000

XML シリアライゼーション設定

lisa.toxml.serialize.mode=NOREFERENCES

NOREFERENCES は、単純なツリーが使用されることを意味します。 循環参照は許可されません。 XPATHREFERENCES は、XPATH 表記が参照に使用されることを意味します。循環参照が使用できます。循環参照がある場合、XPATHREFERENCES にフォールバックします。 ただし、再読み取りであるバージョン 3.6 以前のシリアライゼーションは失敗し、XPATHREFERENCES のデフォルトを設定する必要がある場合があります。

ワークステーション管理

lisa.ui.admin.tr.control=no

lisa.ws.jms.SoapAction.quoted=false

SOAP over JMS と TIBCO BusinessWorks/Active Matrix : BW/AM 要求では、SOAPAction ヘッダを引用符で囲む必要があります。

DevTest が date、time、または dateTime をシリアライズする場合、デフォルトでは以下の形式を使用します。デフォルトの形式/タイム ゾーンを変更するには、これらのプロパティを使用します。 また、テスト ケースで動的に設定することもできます。 形式を XML スキーマの仕様に準拠しないものにも設定できます。 ただし、ネガティブ テスト ケースを除き、推奨されません。

lisa.ws.ser.dateFormat=yyyy-MM-dd

lisa.ws.ser.timeFormat=HH:mm:ss.SSS'Z'

lisa.ws.ser.timeFormat.timeZone=GMT

lisa.ws.ser.dateTimeFormat=yyyy-MM-dd'T'HH:mm:ss.SSS'Z'

lisa.ws.ser.dateTimeFormat.timeZone=GMT

ws.raw.format=true

SOAP 応答を RAW Soap ステップ用に整形する場合は、このプロパティを true に設定します。 この整形も、実行時に動的に実行できます。

lisa.ws.endpoint.fullautoprop=false

WS エンドポイント URL 全体は、単一の DevTest プロパティに自動的に変換されます。 WSSERVER および WSPORT プロパティを使用するために URL を変換する場合は、このプロパティを **false** に設定します。

stats.unix.xml.folder={ {LISA_HOME } }/umetrics

IP スプーフィング

lisa.ipspoofing.interfaces=2-34-56-78-90-AB, eth0, Realtek PCIe GBE Family Controller

インターフェース： IP スプーフィングに使用されるインターフェースのカンマ区切りリスト。これらのインターフェースは、MAC アドレス（JDK 1.6+）、インターフェース名、またはインターフェース表示名を使用して指定できます。

lisa.ui.useNativeFileDialog=true

ネイティブ ファイル ダイアログ ボックスの使用を強制します。

クイック スタートの最近の項目のプロパティ

lisa.prefill.recent=10

lisa.quickstart.recent=5

［Open Recent Quick Start］（最近のクイック スタートを開く）タブ（5 以上）および［Prefill］（事前入力）コンボ ボックス（10 以上）で表示する最近の項目の最大数を指定します。

LISA Invoke プロパティ

lisa.portal.invoke.base.url=/lisa-invoke

lisa.portal.invoke.report.url=/reports

lisa.portal.invoke.server.report.directory=lisa.tmpdirlisa.portal.invoke.report.url

lisa.portal.invoke.test.root=d:/lisatests/

サーバ ホスト名プロパティ

lisa.net.externalIP=localhost

この値は、レジストリにリモートでアクセス可能な外部 IP アドレスまたは DNS 名である必要があります。 このパラメータは、関連する接続プロパティで参照されます。 値を「localhost」のままにすると、接続したアプリケーションに送信されたときに、レジストリ サーバの外部 IP アドレスで自動的に上書きされます。

DevTest から DevTest への通信の暗号化プロパティ

通常、ネットワーク トラフィックは暗号化されません。SSL 暗号化は標準でサポートされています。「tcp」でサーバ エンドポイントを指定する代わりに、「ssl」で指定します（例： **ssl://hostname:2010/Registry**）。以下のプロパティは、どれも設定の必要がありません。ssl でエンドポイント（およびサーバ名）を指定するだけで十分です。たとえば、新しいシミュレータを起動するには、**Simulator -name ssl://thishost:2014/Simulator -labName ssl://regHost:2010/Registry** と指定します。

LISA_HOME¥webreckeys.ks はデフォルトの内部自己署名証明書です。より強力な証明書を使用するには、**lisa.net.keyStore.password** で **lisa.net.keyStore** プロパティおよびプレーン テキスト パスワードを指定します。次に DevTest が起動されるときに、プレーン テキスト パスワードが **lisa.net.keyStore.password_enc** の暗号化された文字列に置換されます。

lisa.default.keystore={{LISA_HOME}}webreckeys.ks

lisa.default.keystore.pass=passphrase

識別証明書が保持される場所。サーバ インストールの場合、この値は信頼できるサーバのリストに追加される証明書です。相互（クライアント）認証を実行するクライアント インストールの場合、この値はサーバ側の信頼ストアに追加される証明書です。相互認証を使用しないクライアント インストールの場合、このパラメータを指定する必要はありません。

lisa.net.keyStore={{LISA_HOME}}lisa.ks

lisa.net.keyStore.password=PlainTextPasswordWilBeConvertedToEncrypted

信頼できる証明書が保持される場所。プライマリ クライアント インストールの場合、このフィールドでは信頼できるサーバのリストが保持されます。相互（クライアント）認証を実行するサーバ インストールの場合、このフィールドは信頼できるクライアント証明書を配置する場所です。

lisa.net.trustStore={{LISA_HOME}}lisa.ts

lisa.net.trustStore.password_enc=079f6a3d304a978146e547802ed3f3a4

相互認証を使用するかどうか。

lisa.net.clientAuth=false

ActiveMQ 接続のデフォルトのプロトコルを定義します。デフォルトは **tcp** です。

lisa.net.default.protocol=tcp

DevTest エージェントおよび CAI プロパティ

lisa.pathfinder.on

エージェントでエラーが発生していて、DevTest エージェントに対してライセンスされていない場合は、ここでオフにすることを検討してください。

デフォルト： false

IBM WebSphere MQ プロパティ

文字列値は、一意コード ジェネレータなどのデータ セットによって提供されます。

lisa.mq.correlation.id=string-value

（実行時の値）送信する MQ メッセージに対して相関 ID を設定します。

lisa.mq.correlation.id.bytes=byte[]

（実行時の値）出力できない値に対して byte[] として相関 ID を設定します。

lisa.mq.message.id=string-value

（実行時の値）送信する MQ メッセージに対してメッセージ ID を設定します。

lisa.mq.message.id.bytes=byte[]

（実行時の値）出力できない値に対して byte[] としてメッセージ ID を設定します。

JMS プロパティ

文字列値は、多くの場合、一意コード ジェネレータなどのデータ セットによって提供されます。

lisa.jms.correlation.id

送信する JMS メッセージに対して JMSCorrelationID を設定する実行時の値。

値： 文字列値

lisa.jms.ttl.milliseconds

送信する JMS メッセージに対して有効期間を設定する実行時の値。

値： 数値

jms.always.close.jndi

true に設定すると、JMS ステップを使用して常に JNDI コンテキストをその実行の最後に閉じます。

値： true、false

その他のプロパティ

lisa.coord.failure.list.size

テストでレポートされるエラー数が多すぎると、コーディネータは使用可能なヒープ スペースをすべて使い果たしてしまいます。 ヒープ スペースが使い果たされた場合は、DevTest サーバ を再起動する必要があります。 複数のテストが実行されている場合、この状況は好ましくありません。テストがステージングされても、エラーがスローされるだけになります。このプロパティは、コーディネータのエラー リストのサイズを制限します。

デフォルト： 15

testexec.lite.longMsgLen

長い応答のテキスト全体を表示するには、このプロパティを応答の最長の長さに設定します。テストを実行し、必要に応じてキャプチャし、正しい応答が確実に返されることを検証したら、local.properties ファイルに追加した行をコメント アウトしてデフォルトの長さに戻します。

デフォルト： 1024

java.rmi.server.hostname

コンピュータの特定の NIC を使用して通信します。

値： 選択した NIC の IP アドレス

lisa.xml.xpath.computeXPath.alwaysUseLocalName

このプロパティは、XPath の生成時に XPath の local-name() 関数を常に使用するかどうかを設定します。デフォルト値は false です。この値は、local-name() が必要な場合にのみ使用されることを意味します。 XML ノードのネームスペースに関係なく動作する XPath を生成するには、このプロパティの値を true に設定します。

デフォルト： false

lisa.commtrans.ctstats

累積 HTTP トラフィック サマリ レポートの生成に必要な情報をキャプチャするには、このプロパティをカスタム プロパティ ファイルのいずれかに追加します。

デフォルト： true

gui.viewxml.maxResponseSize

エディタまたは ITR で、XML 応答のビューを DOM ビューなしのプレーンで表示するサイズを指定するプロパティ。

デフォルト： 5 MB

rpt.cleaner.initDelayMin

デフォルト： 10

rpt.cleaner.pulseMin

これらのプロパティは、レポートのクリーンアップ スレッドをより頻繁に実行するために追加されています。 1 つ目は、クリーナの開始までの初期遅延（分）です。 2 つ目は実行の間隔（分）です。

デフォルト： 60

lisa.LoadHistoryWriteSupport.max.errors

レポートされるエラー数を制限して、レポート ジェネレータがバックアップされるのを防ぎます。

デフォルト： 50

lisa.metric.initialization.timeout

メトリック コレクタの初期化がテスト/スイートのステージング中に発生すると、コレクタが初期化されるまでテスト/スイートは開始されません。 デフォルト値の 90000 は、初期化が 90 秒後にタイム アウトになることを意味します（コレクタ単位）。

デフォルト： 90000

lisa.graphical.xml.diff.report.numberofextralines

このプロパティは、グラフィカルな XML 差分にエラーが表示される場合に、差分のある行の前後に表示される行数を制御します。 デフォルトは、前に 2 行および後に 2 行です。

値： 0、1、および 2。

デフォルト： 2

lisa.gui.dashboard.sleep.ms

テストの終わりにダッシュボードがリフレッシュされる直前には、短い sleep() があります。 この sleep() によって、メトリック コレクタのタイムスライスがその作業を完了できます。 このプロパティでタイムスライスを調整します。

形式： ミリ秒

lisa.scripting.default.language

使用するデフォルトのスクリプト言語を指定します。

値

- applescript （OS X 用）
- beanshell
- freemarker
- groovy
- javascript
- velocity

デフォルト： beanshell

ユーザ セッション ライフタイム プロパティ

さまざまな DevTest モジュールのすべてのユーザ セッション ライフタイムはプロパティで設定可能です。

ACL セッション関連のプロパティおよびそのデフォルト値を以下に示します。

registry.max.user.lifetime.seconds

registry.max.user.lifetime.seconds プロパティは、セッションでユーザによって実行された最後のアクティビティ後にユーザ セッションが有効な期間を決定する主要なセッション ライフタイム プロパティです。

DevTest ワークステーション が DevTest コンソール Web アプリケーション（レポート、CVS、CAI、サーバ コンソールなど）、DevTest ポータル、および DevTest ワークステーション の認証をバイパスするために、SSO セキュリティ トークンがメモリに保持される秒数を指定します。

コンソール Web アプリケーションにアクセスするためのボタンをクリックすると、DevTest は SSO セキュリティ トークンを使用して ACL 認証をバイパスします。その SSO セキュリティ トークンは再利用するためにメモリに格納されます。ただし、DevTest ワークステーション が 20 分間（またはこのプロパティで指定された秒数）アイドル状態である場合、その SSO セキュリティ トークンは期限切れになります。その後、再認証が必要になります。 DevTest ワークステーション を起動すると、これらのボタンをクリックして、ユーザ ID およびパスワードを入力せずに Web アプリケーションに直接移動できます。 ただし、DevTest ワークステーション が 20 分間アイドル状態（アクティビティなし）であった後、これらのボタンをクリックすると、新しいセキュリティ トークンが再度メモリに保持されるように、DevTest ワークステーション はユーザ ID およびパスワードの入力を要求します。

デフォルト： 1200

以下のモジュールにはユーザ セッション ライフタイムがありますが、セッションがまだ有効かどうかを確認するために常にレジストリと通信します。

vse.max.user.lifetime.seconds=1200

console.max.user.lifetime.seconds=1200

coordinator.max.user.lifetime.seconds=1200

simulator.max.user.lifetime.seconds=30

コンソールのリフレッシュ間隔は以下のプロパティで設定できます。

lisa.portal.server.console.polling.interval.seconds=5

現在のセッションが有効で期限切れになっていないかどうかを DevTest コンソールが確認する間隔。

user.session.check.interval.seconds=60

サイト プロパティ ファイル

site.properties 内のプロパティは、レジストリからそれに接続するすべての DevTest アプリケーションまたはサービスに送信されます。これらのプロパティは、**lisa.properties** でローカルに定義されているプロパティよりも優先されます。ただし、これらのプロパティは、**local.properties** で定義されているか、または コマンドラインで -D コマンドライン オプションを使用して定義されたプロパティに対しては優先されません。

レポート、VSE、ACL、およびブローカなどのコンポーネントがデータベースと通信します。 デフォルトでは、これらのコンポーネントは**共通**の接続プールを使用します。ただし、それぞれに個別のプールを定義したり、組み合わせたりできます。 新しい接続プールを定義するには、**lisadb.pool.newpool.url** などのプロパティを追加します。 基盤となるプール実装はオープン ソースの c3p0 プールです。 DevTest ではさまざまなプロパティが継承されています。選択できる c3p0 の設定の詳細については、http://www.mchange.com/projects/c3p0/index.html#configuration_properties を参照してください。

これらのプロパティは、デフォルトの内部 Derby データベースに接続するために DevTest が使用するデフォルトのプロパティです。外部データベースを使用するように DevTest を設定するには、**LISA_HOME¥database** ディレクトリの適切な **site.properties** ファイルを参照します。

lisadb.reporting.poolName

デフォルト： common

lisadb.vse.poolName

デフォルト： common

lisadb.acl.poolName

デフォルト： common

lisadb.broker.poolName

デフォルト： common

lisadb.pool.common.driverClass

デフォルト： org.apache.derby.jdbc.ClientDriver

lisadb.pool.common.url

デフォルト jdbc:derby://localhost:1528/database/lisa.db;create=true

lisadb.pool.common.user

デフォルト： rpt

lisadb.pool.common.password_enc

デフォルト： 76f271db3661fd50082e68d4b953fbee

lisadb.pool.common.minPoolSize

デフォルト： 0

lisadb.pool.common.initialPoolSize

デフォルト： 0

lisadb.pool.common.maxPoolSize

デフォルト： 10

lisadb.pool.common.acquireIncrement

デフォルト： 1

lisadb.pool.common.maxIdleTime

デフォルト： 45

lisadb.pool.common.idleConnectionTestPeriod

デフォルト： 5

lisadb.internal.enabled

レジストリが内部 Derby データベースを起動するかどうかを制御します。

デフォルト： true

lisa.pathfinder.on

デフォルト： true

DCM の設定

lisa.dcm.labstartup.min=6

lisa.dcm.lisastartup.min=4

lisa.dcm.lisashutdown.min=2

lisa.dcm.lab.cache.sec=<デフォルトは 180>

lisa.dcm.lab.factories=com.itko.lisa.cloud.serviceMesh.ServiceMeshCloudSupport;com.itko.lisa.cloud.vCloud.vCloudDirectorCloudSupport;;

lisa.dcm.SERVICEMESH.baseUri=<URL>

lisa.dcm.SERVICEMESH.userId=<ユーザ ID>

lisa.dcm.SERVICEMESH.password=<パスワード>

lisa.dcm.vCLOUD.baseUri=<https://<FQDN>/api/versions>

lisa.dcm.vCLOUD.userId=<ユーザ ID>

lisa.dcm.vCLOUD.password=<パスワード>

lisa.net.timeout.ms=60000

logging.properties

log4j.rootCategory

デフォルト： INFO,A1

クラウド実行の一元化されたログ記録を提供するには、**レジストリ** アペンダを追加し logging.properties ファイル内のレジストリ アペンダ プロパティのコメントを解除します。 以下に例を示します。

```
log4j.rootCategory=INFO,A1,registry
```

以下の行では、DevTest によって使用されるサードパーティ ライブラリのログ レベルを調節します。ログ レベルの指定により、DevTest に無関係なメッセージでログがいっぱいになることがなくなります。

log4j.logger.com.teamdev

デフォルト： WARN

log4j.logger.EventLogger

デフォルト： WARN

log4j.logger.org.apache

デフォルト： ERROR

log4j.logger.com.smardec

デフォルト： ERROR

log4j.logger.org.apache.http

デフォルト： ERROR

log4j.logger.org.apache.http.header

デフォルト： ERROR

log4j.logger.org.apache.http.wire

デフォルト： ERROR

log4j.logger.com.mchange.v2

デフォルト： ERROR

log4j.logger.org.hibernate

デフォルト： WARN

log4j.logger.org.jfree

デフォルト： ERROR

log4j.logger.com.jniwrapper

デフォルト： ERROR

log4j.logger.sun.rmi

デフォルト： INFO

log4j.logger.com.itko.util.ThreadDumper

デフォルト： INFO

log4j.logger.profiler

プロファイル イベントをログに記録する場合、このプロパティを *INFO* に設定します。

デフォルト： OFF

log4j.appender.A1

デフォルト： com.itko.util.log4j.TimedRollingFileAppender

log4j.appender.A1.File

デフォルト： ${lisa.tmpdir}/${LISA_LOG}

log4j.appender.A1.MaxFileSize

デフォルト： 10MB

log4j.appender.A1.MaxBackupIndex

デフォルト： 5

log4j.appender.A1.layout

デフォルト： org.apache.log4j.EnhancedPatternLayout

log4j.appender.A1.layout.ConversionPattern

デフォルト： %d{ISO8601}{UTC}Z (%d{HH:mm}) [%t] %-5p %-30c - %m%n

log4j.logger.VSE

VSE トランザクション一致/不一致イベントに対して、個別のログを保持します。個別ログの使用により、デバッグが非常に簡単になります。

実稼働システムでは、INFO を WARN に変更します。ログ記録によってトランザクション レートが高くなり、システムの速度が低下する場合があります。以下の行をコメント アウトする場合は注意してください。ログ レベルを、INFO ではなく OFF または WARN に明示的に設定します。

デフォルト： INFO、VSEAPP

log4j.additivity.VSE

VSE のログをその他のログ宛先に追加するには、この行をコメントアウトします。

デフォルト： false

log4j.appender.VSEAPP

デフォルト： com.itko.util.log4j.TimedRollingFileAppender

log4j.appender.VSEAPP.File

デフォルト： ${lisa.tmpdir}/vse_matches.log

log4j.appender.VSEAPP.MaxFileSize

デフォルト： 10MB

log4j.appender.VSEAPP.MaxBackupIndex

デフォルト： 20

log4j.appender.VSEAPP.layout

デフォルト： org.apache.log4j.EnhancedPatternLayout

log4j.appender.VSEAPP.layout.ConversionPattern

デフォルト： %d{ISO8601}{UTC}Z (%d{HH:mm})[%t] %-5p - %m%n

アドバイザリ イベントに対して個別のログを保持します。このロガーは、潜在的なメモリ リークなどの、潜在的な設定の問題を警告します。このログは、ノイズを最小化するために、意図的にアプリケーションログとは別に保持します。

log4j.logger.ADVICE

デフォルト： INFO, ADVICE_APP

log4j.additivity.ADVICE

デフォルト： false

log4j.appender.ADVICE_APP

デフォルト： org.apache.log4j.RollingFileAppender

log4j.appender.ADVICE_APP.File

デフォルト： ${lisa.tmpdir}/advice.log

log4j.appender.ADVICE_APP.MaxFileSize

デフォルト： 10MB

log4j.appender.ADVICE_APP.MaxBackupIndex

デフォルト： 20

log4j.appender.ADVICE_APP.layout

デフォルト： org.apache.log4j.EnhancedPatternLayout

log4j.appender.ADVICE_APP.layout.ConversionPattern

デフォルト： %d{ISO8601}{UTC}Z (%d{HH:mm}) %-5p - %m%n

有効な場合、定期的なスレッド ダンプはここに送信されます。 DevTest は INFO レベルでログを書き込みます。 そのため、スレッド ダンプを取得するには、DevTest サーバまたは DevTest ワークステーション が実行されていても、次の行の WARN を INFO に変更します。 このアクションにより、指定されたファイルに 30 秒でスレッド ダンプが出力されます。詳細については、**lisa.properties** 内で「threadDump」を検索してください。 このアクションによって、スレッド ダンプを容易に取得してパフォーマンスの問題をデバッグできるようにもなります。 次の行の WARN を INFO に変更して 1 ～ 2 分待機します。その後、設定を WARN に戻します。

また、LISA_HOME/bin/ServiceManager アプリケーションで特定の時点のスレッド ダンプを生成できます。たとえば、jstack などの標準的な Java ツールを使用するか、または以下のコマンドを発行します。

ServiceManager -threadDump tcp://hostname:2014/Simulator

log4j.logger.threadDumpLogger

デフォルト： WARN, THREAD_DUMPS

log4j.additivity.threadDumpLogger

デフォルト： false

log4j.appender.THREAD_DUMPS

デフォルト： org.apache.log4j.RollingFileAppender

log4j.appender.THREAD_DUMPS.File

デフォルト： ${lisa.tmpdir}/threadDumps/TD_${LISA_LOG}

log4j.appender.THREAD_DUMPS.MaxFileSize

デフォルト： 10MB

log4j.appender.THREAD_DUMPS.MaxBackupIndex

デフォルト： 20

log4j.appender.THREAD_DUMPS.layout

デフォルト： org.apache.log4j.EnhancedPatternLayout

log4j.appender.THREAD_DUMPS.layout.ConversionPattern

デフォルト： %d{ISO8601}{UTC}Z (%d{HH:mm}) [%t] %-5p - %m%n

log4j.appender.registry

DevTest のログ記録を（リモート）レジストリにミラーリングします。

デフォルト： com.itko.lisa.net.LoggingToRegistryAppender

log4j.appender.registry.layout

デフォルト： org.apache.log4j.EnhancedPatternLayout

log4j.appender.registry.layout.ConversionPattern

デフォルト： %d{ISO8601}{UTC}Z [%t] %-5p - %m%n

以下のプロパティでは、vse.log に HTTP ベースのトラフィックの要求および応答を出力します。 この情報は、「一致が見つかりません」応答を返す仮想サービスをデバッグするのに役立ちます。そのような結果は、VS が予期された要求を受信しなかったことを示す場合があります。

このログ メッセージでは、TCPMON が示すものと同じ要求および応答が明らかになります。 仮想サービスとそのクライアント アプリケーションの間に TCPMON を挿入することなく、要求および応答が vse.log に出力されます。 以下のオプションでは、HTTP 要求および応答のみをログ記録します。特に、予期しないメッセージで仮想サービスが応答していて、RAW 応答と RAW 要求を一致させる場合、これらは仮想サービスのデバッグに役立ちます。

log4j.logger.com.itko.lisa.vse.http.Transaction

このクラスを移動する要求を vse.log に出力します。 このログ メッセージは、「Raw playback response」で始まります。

値： TRACE、NULL

デフォルト： Null

log4j.logger.com.itko.lisa.vse.stateful.protocol.http.Coordinator

このクラスを移動する要求を vse.log に出力します。 このログ メッセージは、「Raw request start」で始まります。

値： TRACE、NULL

デフォルト： Null

log4j.logger.com.itko.lisa.vse.stateful.protocol.http.HttpListenStep

このクラスを移動する要求を vse.log に出力します。 このログ メッセージは、「Raw request start」で始まります。

値： TRACE、NULL

デフォルト： Null

用語集

アサーション

*アサーション*は、1 つのステップとそのすべてのフィルタが実行された後に実行されるエレメントです。 アサーションにより、ステップの実行結果が予測と一致することが検証されます。 アサーションは、通常、テスト ケースまたは仮想サービス モデルのフローを変更するために使用されます。 グローバル アサーションは、テスト ケースまたは仮想サービス モデルの各ステップに適用されます。詳細については、「*CA Application Test の使用*」の「アサーション (P. 151)」を参照してください。

アセット

*アセット*は、1 つの論理的な単位にグループ化される設定プロパティのセットです。 詳細については、「*CA Application Test の使用*」の「アセット (P. 123)」を参照してください。

一致許容差

*一致許容差*は、CA Service Virtualization が受信要求をサービス イメージ内の要求と比較する方法を制御する設定です。 オプションは、EXACT、SIGNATURE、および OPERATION です。 詳細については、「*CA Service Virtualization の使用*」の「一致許容差」を参照してください。

イベント

*イベント*は、発生したアクションに関するメッセージです。 テスト ケースまたは仮想サービス モデル レベルでイベントを設定できます。 詳細については、「*CA Application Test の使用*」の「イベントについて (P. 281)」を参照してください。

会話ツリー

*会話ツリー*は、仮想サービス イメージにおいてステートフル トランザクションの会話パスを表すリンクされたノードのセットです。 各ノードは、withdrawMoney などの操作名でラベル付けされます。 getNewToken、getAccount、withdrawMoney、deleteToken は、金融機関システムの会話パスの一例です。 詳細については、「*CA Service Virtualization の使用」を参照してください。*

仮想サービス モデル（VSM）

*仮想サービス モデル*は、実際のサービス プロバイダなしでサービス要求を受信および応答します。 詳細については、「*CA Service Virtualization の使用*」の「仮想サービス モデル（VSM）」を参照してください。

監査ドキュメント

*監査ドキュメント*では、1 つのテスト、またはスイート内の 1 つのテスト セットに対する成功条件を設定できます。 詳細については、「*CA Application Test の使用*」の「監査ドキュメントの作成 (P. 278)」を参照してください。

クイック テスト

*クイック テスト*機能を使用すると、最小のセットアップでテスト ケースを実行できます。 詳細については、「*CA Application Test の使用*」の「クイック テストのステージング (P. 330)」を参照してください。

グループ

グループ、または仮想サービス グループは、VSE コンソールでまとめてモニタできるように、同じグループ タグでタグ付けされている仮想サービスのコレクションです。

継続的検証サービス（CVS）ダッシュボード

*継続的検証サービス（CVS）ダッシュボード*では、長期間にわたって定期的に実行するテスト ケースおよびテスト スイートをスケジュールできます。 詳細については、「*CA Application Test の使用*」の「継続的検証サービス（CVS） (P. 417)」を参照してください。

コーディネータ

*コーディネータ*はテスト ランの情報をドキュメントとして受け取り、1 つ以上のシミュレータ サーバで実行されるテストをコーディネートします。詳細については、「*CA Application Test の使用*」の「コーディネータ サーバ (P. 14)」を参照してください。

コンパニオン

*コンパニオン*は、すべてのテスト ケースの実行の前後に実行されるエレメントです。 コンパニオンは、単一のテスト ステップではなく、テスト ケース全体に適用されるフィルタとして理解できます。 コンパニオンはテスト ケース内で（テスト ケースに対して）グローバルな動作を設定するために使用されます。 詳細については、「*CA Application Test の使用*」の「コンパニオン (P. 184)」を参照してください。

サービス イメージ（SI）

*サービス イメージ*は、CA Service Virtualization で記録されたトランザクションの正規化バージョンです。 各トランザクションは、ステートフル（会話型）またはステートレスです。 サービス イメージを作成する方法の 1 つは、仮想サービス イメージ レコーダを使用することです。 サービス イメージは、プロジェクトに格納されます。 サービス イメージは、*仮想サービス イメージ*（VSI）とも呼ばれます。詳細については、「*CA Service Virtualization の使用*」の「サービス イメージ」を参照してください。

サブプロセス

*サブプロセス*は、別のテスト ケースによってコールされるテスト ケースです。 詳細については、「*CA Application Test の使用*」の「サブプロセスの作成」を参照してください。

シミュレータ

*シミュレータ*は、コーディネータ サーバの管理下でテストを実行します。詳細については、「*CA Application Test の使用*」の「シミュレータ サーバ (P. 17)」を参照してください。

ステージング ドキュメント

*ステージング ドキュメント*には、テスト ケースを実行する方法に関する情報が含まれます。詳細については、「*CA Application Test* の使用」の「ステージング ドキュメントの作成 (P. 251)」を参照してください。

設定

*設定*は、プロパティの名前付きのコレクションであり、通常はテスト中のシステムの環境に固有の値を指定します。 ハードコードされた環境データをなくすことにより、設定を変更するだけで、異なる環境内のテスト ケースまたは仮想サービス モデルを実行できます。 プロジェクトのデフォルト設定の名前は project.config です。 プロジェクトは多数の設定を持つことができますが、一度にアクティブになるのは 1 つの設定のみです。詳細については、「*CA Application Test の使用*」の「設定 (P. 116)」を参照してください。

対話型テスト ラン（ITR）

*対話型テスト ラン（ITR）*ユーティリティを使用すると、テスト ケースまたは仮想サービス モデルをステップごとに実行できます。 テスト ケースまたは仮想サービス モデルを実行時に変更し、結果を確認できます。 詳細については、「*CA Application Test の使用*」の「対話型テスト ラン（ITR）ユーティリティの使用 (P. 317)」を参照してください。

ディセンシタイズ

*ディセンシタイズ*は、機密データをユーザ定義の代替データに変換するために使用されます。 クレジット カード番号や社会保障番号は機密データの例です。 詳細については、「*CA Service Virtualization の使用*」の「データのディセンシタイズ」を参照してください。

データ セット

*データ セット*は、実行時にテスト ケースまたは仮想サービス モデルにプロパティを設定するために使用できる値のコレクションです。 データ セットによって、テスト ケースまたは仮想サービス モデルに外部のテスト データを使用することができます。データ セットは、DevTest の内部または外部（たとえば、ファイルやデータベース テーブル）に作成できます。 詳細については、「*CA Application Test の使用*」の「データ セット (P. 171)」を参照してください。

データ プロトコル

*データ プロトコル*は、データ ハンドラとも呼ばれます。 CA Service Virtualization では、データ プロトコルは、要求の解析処理を行います。 一部のトランスポート プロトコルは、要求を作成するジョブの委任先のデータ プロトコルを許可（または要求）します。 結果として、プロトコルは要求ペイロードを認識する必要が生じます。 詳細については、「*CA Service Virtualization の使用*」の「データ プロトコルの使用」を参照してください。

テスト ケース

*テスト ケース*は、テスト中のシステムのビジネス コンポーネントをテストする方法の仕様です。 各テスト ケースには、1 つ以上のテスト ステップが含まれます。 詳細については、「*CA Application Test の使用*」の「テスト ケースの作成 (P. 85)」を参照してください。

テスト スイート

*テスト スイート*は、順番に実行されるようにスケジュールされたテスト ケース、その他のテスト スイート、またはその両方のグループです。 スイート ドキュメントは、スイートのコンテンツ、生成するレポート、および収集するメトリックを指定します。詳細については、「*CA Application Test の使用*」の「テスト スイートの作成 (P. 287)」を参照してください。

テスト ステップ

*テスト ステップ*は、実行される単一のテスト アクションを表すテスト ケース ワークフローのエレメントです。 テスト ステップの例としては、Web サービス、Java Bean、JDBC、JMS メッセージングなどがあります。 テスト ステップには、フィルタ、アサーション、データ セットなどの DevTest エレメントを含めることができます。 詳細については、「*CA Application Test の使用*」の「テスト ステップの作成 (P. 221)」を参照してください。

トランザクション フレーム

*トランザクション フレーム*は、DevTest Java エージェントまたは CAI Agent Light がインターセプトしたメソッド コールに関するデータをカプセル化します。詳細については、「*CA Continuous Application Insight の使用*」の「ビジネス トランザクションおよびトランザクション フレーム」を参照してください。

ナビゲーション許容差

*ナビゲーション許容差*は、CA Service Virtualization が会話ツリーを検索して次のトランザクションを見つける方法を制御する設定です。 オプションは、CLOSE、WIDE、および LOOSE です。 詳細については、「*CA Service Virtualization の使用*」の「ナビゲーション許容差」を参照してください。

ネットワーク グラフ

ネットワーク グラフは、DevTest クラウド マネージャおよび関連するラボをグラフで表示するサーバ コンソールの領域です。詳細については、「*CA Application Test の使用*」の「ラボの開始 (P. 409)」を参照してください。

ノード

DevTest の内部では、テスト ステップは*ノード*とも呼ばれます。これが、一部のイベントがイベント ID 内にノードを持つ理由です。

パス

*パス*には、Java エージェント がキャプチャしたトランザクションに関する情報が含まれます。詳細については、「*CA Continuous Application Insight の使用*」を参照してください。

パス グラフ

*パス グラフ*には、パスおよびそのフレームのグラフ表示が含まれています。 詳細については、「*CA Continuous Application Insight の使用*」の「パス グラフ」を参照してください。

反応時間

*反応時間*は、テスト ステップを実行する前にテスト ケースが待機する時間です。詳細については、「*CA Application Test の使用*」の「テスト ステップの追加 - 例 (P. 223)」および「ステージング ドキュメント エディタ (P. 253) - ［ベース］タブ」を参照してください。

フィルタ

*フィルタ*は、ステップの前後に実行されるエレメントです。フィルタは、結果のデータを処理、またはプロパティに値を格納する機会を提供します。グローバル フィルタは、テスト ケースまたは仮想サービス モデルの各ステップに適用されます。 詳細については、「*CA Application Test の使用*」の「フィルタ (P. 132)」を参照してください。

プロジェクト

*プロジェクト*は、関連する DevTest ファイルのコレクションです。ファイルには、テスト ケース、スイート、仮想サービス モデル、サービス イメージ、設定、監査ドキュメント、ステージング ドキュメント、データ セット、モニタ、および MAR 情報ファイルなどが含まれます。 詳細については、「*CA Application Test の使用*」の「プロジェクト パネル (P. 54)」を参照してください。

プロパティ

*プロパティ*は、ランタイム変数として使用できるキー/値ペアです。 プロパティには、さまざまなタイプのデータを格納できます。 一般的なプロパティには、LISA_HOME、LISA_PROJ_ROOT、LISA_PROJ_NAME などがあります。 設定は、プロパティの名前付きのコレクションです。 詳細については、「*CA Application Test の使用*」の「プロパティ (P. 101)」を参照してください。

マジック ストリング

*マジック ストリング*は、サービス イメージの作成中に生成される文字列です。 マジック ストリングは、仮想サービス モデルによって応答内で意味のある文字列値が提供されることを確認するために使用されます。 {{=request_fname;/chris/}} は、マジック ストリングの一例です。 詳細については、「*CA Service Virtualization の使用*」の「マジック ストリングとマジック デート」を参照してください。

マジック デート

レコーディング中、日付パーサは要求および応答をスキャンします。 日付表示形式の広範な定義に一致する値は、*マジック デート*に変換されます。 マジック デートは、仮想サービス モデルによって応答内で意味のある日付値が提供されることを確認するために使用されます。 {{=doDateDeltaFromCurrent("yyyy-MM-dd","10");/*2012-08-14*/} は、マジック デートの一例です。詳細については、「*CA Service Virtualization の使用*」の「マジック ストリングとマジック デート」を参照してください。

メトリック

*メトリック*により、テストおよびテスト中のシステムのパフォーマンス/機能面に定量的手法および測定単位を適用できます。 詳細については、「*CA Application Test の使用*」の「メトリックの生成 (P. 286)」を参照してください。

モデル アーカイブ（MAR）

*モデル アーカイブ（MAR）*は、DevTest Solutions における主要な展開アーティファクトです。 MAR ファイルには、プライマリ アセット、プライマリ アセットを実行するために必要なすべてのセカンダリ ファイル、情報ファイル、および監査ファイルが含まれます。 詳細については、「*CA Application Test の使用*」の「モデル アーカイブ（MAR）の操作 (P. 299)」を参照してください。

モデル アーカイブ（MAR）情報

*モデル アーカイブ（MAR）情報*ファイルは、MAR を作成するために必要な情報が含まれるファイルです。詳細については、「*CA Application Test の使用*」の「モデル アーカイブ（MAR）の操作 (P. 299)」を参照してください。

ラボ

*ラボ*は、1 つ以上のラボ メンバの論理コンテナです。 詳細については、「*CA Application Test の使用*」の「ラボとラボ メンバ (P. 394)」を参照してください。

レジストリ

*レジストリ*は、すべての DevTest サーバおよび DevTest ワークステーション コンポーネントの登録を一元的に行うための場所です。 詳細については、「*CA Application Test* の使用」の「レジストリ (P. 11)」を参照してください。

仮想サービス環境（VSE）

*仮想サービス環境（VSE）*は、仮想サービス モデルを展開して実行するために使用する DevTest サーバ アプリケーションです。 VSE は CA Service Virtualization とも呼ばれます。 詳細については、「*CA Service Virtualization の使用*」を参照してください。