DevTest Solutions

はじめに

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: DevTest ポータル

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。

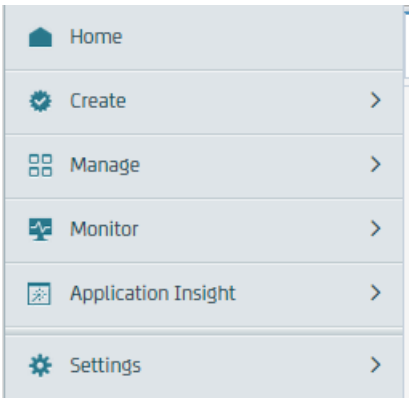
DevTest ポータルのナビゲート

DevTest ポータルを参照します。 ホスト名を、次のプロセスまたはサービスが実行されている DevTest サーバ で置き換えます。レジストリ、ポータル、ブローカ、および VSE。

`http://hostname:1507/devtest`

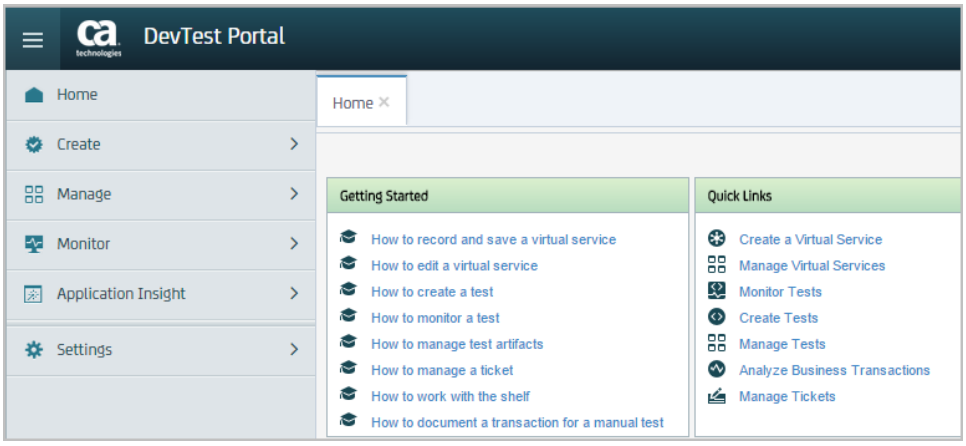
ユーザ ID およびパスワードを使用して DevTest ポータルにログインします。 指定されていない場合は、標準ユーザとしてログインします。 詳細については、管理者に確認してください。

DevTest ポータル ホーム ページを使用して、主要なアクティビティおよび対応するドキュメントのページにアクセスします。 ［ナビゲーション］ペインの任意のセクションを展開して、関連するタスクへのリンクを表示することができます。 リンクをクリックすると、選択したタスクの UI、通常はポータル上のタブが表示されます。 一部のリンクは、DevTest コンソールにリダイレクトされます。



最も頻繁に使用するリンクについては、対応する［Quick Links］オプションをクリックできます。 関連するドキュメントについては、対応する［Getting Started］リンクをクリックします。 ［Getting Started］ペインおよび［Quick Links］ペインが［Home］タブに表示されます。

注：［Getting Started］リンクを初めてクリックすると、ログイン ダイアログ ボックスが表示されます。 ドキュメントを表示するログインは、DevTest ポータル上のセッション終了まで持続します。

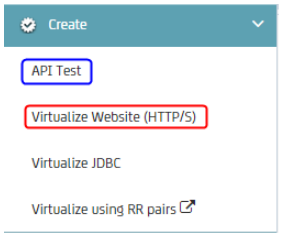


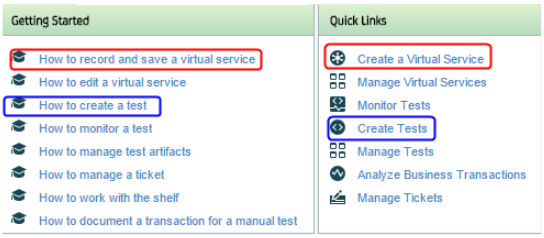
［Quick Links］オプションおよび［Getting Started］オプションからナビゲーション メニュー オプションへのマッピングの詳細が、以下に続きます。

作成

共通の作成アクションを実行します。

［Virtualize using RR pairs］の後のアイコンに注目します。 このアイコンは、リンクをクリックすると別の UI にリダイレクトされることを示しています。





API Test

開くもの： ［API Test］タブ

Getting Started：「*CA Application Test の使用*」の［How to create a test］

Quick Links： Create Tests

Virtualize Website (HTTP/S)

開くもの： ［Virtualize Website (HTTP/S)］タブ

Getting Started：「*CA Service Virtualization の使用*」の［How to record and save a virtual service］

Quick Links： Create a Virtual Service

JDBC の仮想化

開くもの： ［Virtualize JDBC］タブ

トピック： 「CA Service Virtualization の使用」の「JDBC」

Virtualize RR Pairs

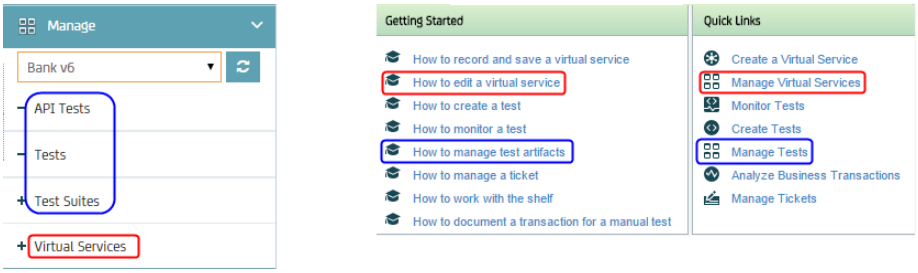
リダイレクト： DevTest コンソール

開くもの： ［Create Virtual Service on VSEasy］タブ

トピック： 「*CA Service Virtualization の使用*」の「VSEasy を使用した仮想サービスの作成および展開」

Manage

テストアーティファクトを管理します。



API Test

DevTest ポータルで作成されたテストを管理します。

Getting Started： 「*CA Application Test の使用*」の「API テストの管理」の章の［How to manage test artifacts］

Quick Links： Manage Tests

テスト： Current Status (Today)

テスト

テスト ケースに関する情報を表示し、DevTest ワークステーションで作成されたテスト ケースを実行します。

Getting Started： 「*CA Application Test の使用*」の「テストの管理」の章の［How to manage test artifacts］

Quick Links： Manage Tests

テスト： Current Status (Today)

Suites

テスト スイートに関する情報を表示し、DevTest ワークステーション で作成されたテスト スイートを実行します。

Getting Started： 「*CA Application Test の使用*」の「テスト スイートの管理」の章の［How to manage test artifacts］

Quick Links： Manage Tests

テスト： Current Status (Today)

Vservices

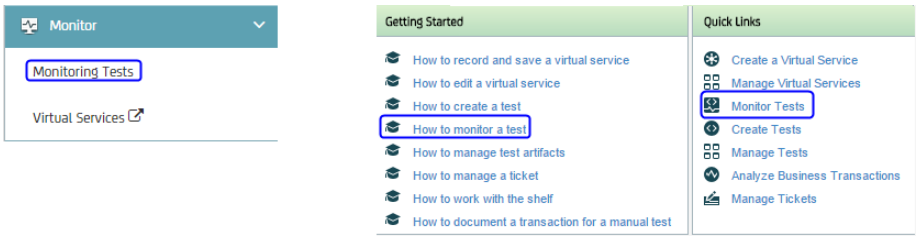
開くタブ： http-rest、WebServicesModel、http

Getting Started： 「*CA Service Virtualization の使用*」の［How to edit a virtual service］

Quick Links： Manage Virtual Services

Monitor

テストまたは仮想サービスをモニタします。選択条件に適合するテストまたはスイートがない場合、［Create a Test］または［Execute a Test］をクリックできます



Monitoring Tests

［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。

検索オプションを切り替えるには、［フィルタ］ボタンをクリックします。

開くもの： ［Monitoring Tests］タブ

Getting Started： 「*CA Application Test の使用*」の［How to monitor a test］

Quick Links： Monitor Tests

Vservices

リダイレクト： DevTest コンソール

開くもの： ［VSE］をクリックできるサーバ コンソールおよび DevTest ネットワーク

トピック： 「*CA Service Virtualization の使用*」の「VSEasy を使用した仮想サービスの作成および展開」

Application Insight

CA Continuous Application Insight で共通のタスクを開始します。

Analyze Transactions

開くもの： ［Analyze Transactions］タブ

Getting Started：「*CA Continuous Application Insight の使用*」の［How to work with the shelf］

Quick Links： Analyze Business Transactions

Document Transactions

開くもの： ［Document Transactions］タブ

Getting Started：「*CA Continuous Application Insight の使用*」の［How to document transactions for a manual test］

Manage Tickets

開くもの： ［Manage Tickets］タブ

Getting Started： 「*CA Continuous Application Insight の使用*」の[How to manage a ticket]

Quick Links： Manage Tickets

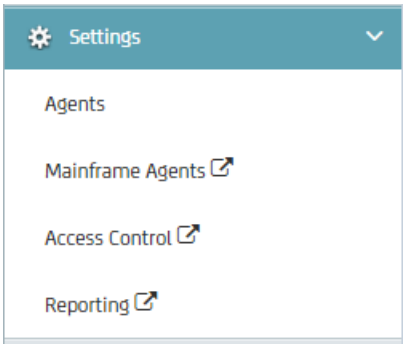
Explore Defects

開くもの： ［Explore Defects］タブ

トピック： 「*CA Continuous Application Insight の使用*」の「障害の処理」

設定

［設定］タブには［Quick Links］エントリがなく、［Getting Started］へのリンクもありません。



Agents

開くもの： ［Agenats］タブ

トピック： 「エージェント」を参照してください。

メインフレーム エージェント

リダイレクト： DevTest コンソール、サーバ コンソール、DevTest ネットワーク

トピック： 「エージェント」の「メインフレーム ブリッジ」。

Access Control

リダイレクト： DevTest コンソール、サーバ コンソール。［セキュリティ］オプションを表示するには、［管理］タブを展開します。

トピック： 「*管理*」の「アクセス制御（ACL）」。

レポート

リダイレクト： DevTest コンソール、レポート

トピック： 「*CA Application Test の使用*」の「レポート」。

DevTest ポータル ホーム ページ

DevTest ポータルのホーム ページには、アプリケーションのアクティビティをモニタできる、一連のポートレットが含まれています。 それぞれのポートレットには、手動でリフレッシュできる、DevTest コンポーネントの情報が表示されます。

ホーム ページには、以下のポートレットが含まれています。

はじめに

ポータルの主要な機能について説明するドキュメント トピックへのリンクが表示されます。

Quick Links

DevTest Solutions の操作へのリンクが表示されます。

ヘルプ

DevTest コミュニティに質問し、問題をレポートするための、DevTest ドキュメントへのリンクが表示されます。

仮想サービス: Current Status

現在のトランザクション数の合計、展開されている仮想サービスの数、および記録されている仮想サービスの数が表示されます。

テスト: Current Status

利用可能なテスト結果が表示されます。 ［詳細表示...］をクリックすると、［Monitoring Test］ウィンドウに移動します。

エージェント: Current Status

エージェントのステータスが表示されます。 エージェント名または［More...］を選択すると、［Agenats］ウィンドウ内のエージェント管理操作に移動します。 現在のエージェント情報を表示するには、手動でポートレットをリフレッシュします。

New Ticket Alerts

新しいチケットのリストと、新しいチケット、識別済みチケット、およびクローズされたチケットの数が表示されます。 チケットの名前、日付、および時間が、リストに表示されます。［新規］、［識別済み］、または［クローズ］のチケットを選択すると、［Manage Tickets］ウィンドウに移動します。［Manage Tickets］ウィンドウでは、チケットを表示し、既存のチケットを更新し、チケットのリストを表示し、CA Continuous Application Insight データベース内のチケットを検索することができます。

Path Alerts

例外フラグが設定されたパスのリストが表示されます。パスを選択すると、［Explore Defects］ウィンドウに移動します。

Points of Interest

Explore Defects で固定された、対象のポイントすべてのリストが表示されます。［Points of Interest］のリストにある名前を選択すると、［Explore Defects］ウィンドウに移動します。

第 2 章: CA Application Test チュートリアル

このセクションには、CA Application Test のさまざまな側面について説明する一連のチュートリアルが含まれます。 チュートリアルはシーケンシャルです。 これらは提示された順番で実行してください。

最初のいくつかのチュートリアルでは、DevTest ワークステーション を使用して簡単なテスト ケースを作成する方法について説明します。 プロジェクト、プロパティ、データ セット、フィルタ、およびアサーションなどの基本概念を理解できます。

その後のチュートリアルでは、テスト ステップを設定し、いくつかの一般的なテクノロジを使用した作業や、それらをテストする方法について詳しく説明します。 これらのテクノロジには、Java オブジェクト、Web ページ、Enterprise JavaBeans （EJB）、Web サービス、およびデータベースが含まれます。 また、クイック テストをステージングする方法についても学習します。

チュートリアルを実行するには、DevTest ワークステーション のインストールおよびレジストリへのアクセスが必要です。

いくつかのチュートリアルでは、テスト用のシステムとしてデモ サーバを使用します。 デモ サーバのインストールの詳細については、「*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。

チュートリアル 1 - プロジェクト、テスト ケース、およびプロパティ

チュートリアルのタスク

このチュートリアルでは、以下のことを行います。

- プロジェクトの作成
- テスト ケースの作成
- プロパティの追加
- 簡単なテスト ステップの追加
- 対話型テスト ラン（ITR）ユーティリティの使用

前提条件

- DevTest ワークステーション がインストールされていて、DevTest のライセンス認証情報が入力されている。
- 「用語集 (P. 159)」を確認済みである。

手順 1 - DevTest ワークステーション の起動

次の手順に従ってください:

1. レジストリを起動します。
   - 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
   a. ［スタート］-［すべてのプログラム］-［DevTest Solutions ］-［エンタープライズ ダッシュボード］をクリックしてエンタープライズ ダッシュボードを起動します。 「Enterprise Dashboard started」というメッセージが表示されるまで待ちます。
   b. ［スタート］-［すべてのプログラム］-［DevTest Solutions ］-［レジストリ］をクリックしてレジストリを起動します。
   - 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、別の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。
2. ［スタート］-［すべてのプログラム］-［DevTest Solutions ］-［ワークステーション］をクリックします。
3. ［DevTest レジストリの設定］ダイアログ ボックスが表示されたら、レジストリを選択して［OK］をクリックします。
4. ［ログイン］ダイアログ ボックスが開きます。有効なユーザ名とパスワードを入力し、［ログイン］をクリックします。

手順 2 - プロジェクトの作成

作成するプロジェクトには、チュートリアルに必要なテスト ケース サンプル ファイルがすべて保持されます。

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. ［プロジェクト名］フィールドで、デフォルト値を削除し、「**My Tutorials**」と入力します。
3. ［Create］をクリックします。

   My Tutorials プロジェクトが作成されます。

手順 3 - 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。

次の手順に従ってください:

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］を選択します。
2. ファイル名を「**tutorial1**」に設定します。
3. ［保存］をクリックします。

   DevTest ワークステーション によって tutorial1 というラベルが付けられた新規タブが開かれます。 モデル エディタの緑の矢印は、テスト ケースの開始を表します。

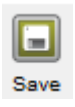
手順 4 - プロジェクト設定へのプロパティの追加

この手順では、プロジェクト設定にグローバル プロパティを設定します。このプロパティは、後ほどチュートリアルで使用します。

デフォルト設定には project.config という名前が付けられ、新しいプロジェクトに対して自動的に作成されます。 project.config ファイルは、プロジェクト パネルの Configs フォルダにあります。 ファイルの拡張子は表示されません。 project.config ファイルにはプロパティを追加できます。また、必要に応じて設定ファイルを作成することもできます

次の手順に従ってください:

1. プロジェクト パネルで、My Tutorials の Configs フォルダにある **project** 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］をクリックして行を追加します。
3. ［キー］フィールドに「**config_prop**」と入力します。
4. ［値］フィールドに「**42**」と入力します。
5. メイン ツールバーの 

［保存］をクリックします。

手順 5 - テスト ステップの追加

テスト ケースには 1 つ以上のテスト ステップを含めます。この手順では、ログ メッセージの出力テスト ステップを追加して、ログ ファイルにテキストを書き込みます。

次の手順に従ってください:

1. ［tutorial1］タブをクリックします。
2.  ［ステップの追加］をクリックし、［ユーティリティ］を選択して、［ログ メッセージの出力］を選択します。

   「ログ メッセージの出力」という名前のステップがモデル エディタに追加されます。
3. ログ メッセージの出力ステップを右クリックし、［名前の変更］を選択します。
4. 名前を「My Output Log Message」に変更します。
5. My Output Log Message がまだ選択されていることを確認します。右ペインで、［ログ メッセージの出力］の隣にある矢印をクリックします。

   ［ログ メッセージの出力］トレイが開きます。

手順 6 - ログ メッセージの追加

ログ エディタが開いている状態で、さまざまなプロパティが含まれるログ メッセージを追加します。

ログ メッセージ内のプロパティは、いくつかのソースが元になっています。

- LISA_HOME プロパティは自動的に設定されます。
- java.version プロパティはシステム プロパティです。
- 手順 4 では、プロジェクト設定に config_prop プロパティを追加しました。
- ログ メッセージ自体に、**MyOutputLogMessage_step_prop** という名前のプロパティを作成します。

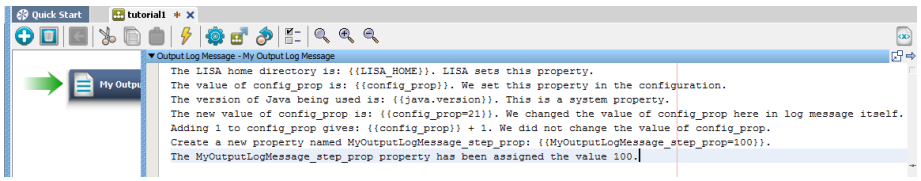
プロパティの構文は **{{プロパティ名}}** です。

次の手順に従ってください:

1. ログ エディタで、プレースホルダのテキストを削除します。
2. 以下のテキストをコピーして、ログ エディタに貼り付けます。

```
The LISA home directory is: {{LISA_HOME}}. LISA sets this property.
The value of config_prop is: {{config_prop}}. We set this property
in the configuration.
The version of Java being used is: {{java.version}}. This is a
system property.
The new value of config_prop is: {{config_prop=21}}. We changed the
value of config_prop here in log message itself.
Adding 1 to config_prop gives: {{config_prop}} + 1. We did not
change the value of config_prop.
Create a new property named MyOutputLogMessage_step_prop:
{{MyOutputLogMessage_step_prop=100}}.
The MyOutputLogMessage_step_prop property has been assigned the
value 100.
```

ログ エディタは以下の図のようになります。



手順 7 - 2 番目のログ メッセージの追加

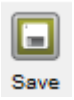
テスト ケースの 2 番目のテスト ステップでは、別のメッセージをログ ファイルに書き込みます。

次の手順に従ってください:

1. ウィンドウの左上隅にある矢印をクリックして、1 番目のログ メッセージ ステップを閉じます。
2.  ［ステップの追加］をクリックし、［ユーティリティ］を選択して、［ログ メッセージの出力］を選択します。

   「ログ メッセージの出力」という名前のステップがモデル エディタに追加され、［ログ メッセージの出力］トレイが開きます。
3. ログ エディタで、プレースホルダのテキストを削除します。
4. 以下のテキストをコピーして、ログ エディタに貼り付けます。

   ```
   The current value of config_prop is: {{config_prop}}.
   The current value of MyOutputLogMessage_step_prop:
   {{MyOutputLogMessage_step_prop}}.
   ```

   注: このログ メッセージでは、**config_prop** および **MyOutputLogMessage_step_prop** の値を変更しません。
5. ウィンドウの左上隅にある矢印をクリックして、2 番目のログ メッセージ ステップを閉じます。
6. アプリケーションのメイン ツールバーから、［保存］ 

をクリックするか、［ファイル］-［保存］-［tutorial1］を選択します。

手順 8 - My Output Log Message ステップの実行

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 ケース全体を確認することができます。

次の手順に従ってください:

1. ツールバーの  ［対話型テスト ラン（ITR）の開始］をクリックします。

   ITR が開きます。 ITR には、左側に［実行履歴］ペイン、右側にタブが表示されます。

2. ［実行履歴］ペインで、  ［次のステップの実行］をクリックします。

   My Output Log Message ステップが実行されます。 ［応答］タブには、My Output Log Message ステップからの応答が表示されます。実際の値によってプロパティが置き換えられます。

手順 9 - プロパティ値の確認

ITR では、プロパティがどのように作成および変更されるかを確認できます。

次の手順に従ってください:

1. ITR の［プロパティ］タブをクリックします。

   ［プロパティ］タブには、My Output LogMessage ステップの実行前後の各プロパティの値が表示されます。ステップで作成された値は緑で強調表示されます。ステップで変更された値は黄色で強調表示されます。config_prop の値が 42 から 21 に変更されたことに注目してください。

2. これらの値を手順 8 の応答と比較します。

手順 10 - ログ メッセージの出力ステップの実行

この手順では、ITR を使用してテスト ケースの 2 番目のステップを実行します。

次の手順に従ってください:

1. ITR で、  ［次のステップの実行］をクリックして、ログ メッセージの出力ステップを実行します。
2. ［応答］タブをクリックして、応答を表示します。 project.config ファイルで config_prop を 42 に設定しましたが、My Output Log Message ステップでこの値を 21 に変更しました。また、ログ メッセージの出力ステップではこの値は変更されませんでした。
   MyOutputLogMessage_step_prop プロパティの値も、My Output Log Message ステップからログ メッセージの出力ステップに引き継がれています。
3. 現在および以前のプロパティ値を表示するには、［プロパティ］タブをクリックします。
4. 操作が終了したら、［tutorial1］および［プロジェクト］タブを閉じます。

レビュー

このチュートリアルでは、まずプロパティについて確認しました。 そして、特別な構文 {{プロパティ名}} を使用するとプロパティが表示されることを確認しました。この構文を変化させて {{プロパティ名=値}} とすると、プロパティを設定できます。 プロパティを設定した後、テスト ケースの後続のステップでそれを使用または変更します。

このチュートリアルでは、以下のことを行いました。

- テスト ケースを作成し、保存する方法を学習しました。
- 簡単なテスト ステップ（ログ メッセージの出力）を追加する方法を学習しました。
- プロパティを保存するために設定を使用しました。
- 対話型テスト ラン（ITR）ユーティリティを簡単に確認しました。

チュートリアル 2 - データ セット

このチュートリアルでは、簡単なデータ セットを作成および使用する方法を学習します。 また、テスト ケースにデータ セットのデータを提供する方法を学習します。

チュートリアルのタスク

このチュートリアルでは、以下のことを行います。

- 簡単なデータ セットの作成
- さまざまな方法でのデータ セットの使用
- データ セットを使用して、一連のテスト ステップを繰り返し実行

前提条件

- 「チュートリアル 1 - プロパティ」を完了している。
- DevTest ワークステーション が開いている。

手順 1 - データ セットの作成

このチュートリアルでは、データ セットとしてカンマ区切りテキスト ファイルを使用します。 このオプションは、データ セットの作成に使用できるオプションのうちの 1 つです。 テキスト ファイルを作成した後、これを My Tutorials プロジェクトにインポートします。

次の手順に従ってください:

1. メモ帳などのテキスト エディタで、テキスト ファイルを作成します。
2. 以下のプロパティと値をコピーして、テキスト エディタに貼り付けます。 テキスト ファイルではスペースを使用しないでください。

```
month,day,year
3,2,1956
4,7,2007
1,3,2010
5,8,{{yearglobal}}
8,10,2004
12,11,{{yearglobal}}
10,12,2007
3,5,2011
```

   最初の行は、このデータが割り当てられるプロパティの名前を指定します（month、day、year）。 残りの行は、テスト ケースで読み取って使用するデータを指定します。2 つの行には、**yearglobal** という名前のプロパティが含まれています。

3. dates.txt としてファイルを保存します。
4. プロジェクト パネルで、My Tutorials プロジェクトの Data 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。
5. dates.txt ファイルを保存したフォルダに移動して、そのファイル名を選択します。
6. ［開く］をクリックします。 Data フォルダに dates.txt ファイルが表示されるようになります。

手順 2 - テスト ケースの作成

My Tutorials プロジェクトにテスト ケースを追加します。

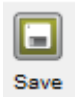
次の手順に従ってください:

1. プロジェクト パネルで Tests フォルダを右クリックし、［新規テスト ケースの作成］を選択します。
2. ファイル名を「**tutorial2**」とします。
3. ［保存］をクリックします。

手順 3 - プロジェクト設定へのプロパティの追加

dates.txt ファイルには、yearglobal という名前のプロパティが含まれています。 この手順では、プロジェクト設定に yearglobal プロパティを追加します。

次の手順に従ってください:

1. プロジェクト パネルで、project.config をダブルクリックします。
2.  ［追加］をクリックして、行を追加します。
3. ［キー］フィールドに「yearglobal」と入力します。
4. ［値］フィールドに「1999」と入力します。
5. ［保存］ 

をクリックします。

手順 4 - ログ メッセージの出力テスト ステップの追加

ログにテキストを書き込むには、テスト ステップのログ メッセージの出力ステップを使用します。

次の手順に従ってください:

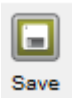
1. ［tutorial2］タブをクリックします。
2.  ［追加］をクリックします。

   ［ステップの追加］メニューが表示されます。
3. ［ユーティリティ］を選択し、［ログ メッセージの出力］を選択します。

   「**ログ メッセージの出力**」という名前のステップがモデル エディタに追加されます。
4. ［ログ メッセージの出力］を右クリックし、［名前の変更］を選択します。 名前を「DSstep1」に変更します。
5. 右ペインで、［ログ メッセージの出力］の隣にある矢印をクリックします。

   ［ログ メッセージの出力］トレイが開きます。
6. プレースホルダ テキストを削除します。
7. 以下のログ メッセージを入力します。

   ```
   Date is: {{month}}/{{day}}/{{year}}
   ```

   注: 中かっこが重要です。 中かっこが含まれている場合にのみ、テスト ケースは正しく実行されます。
8. ［ログ メッセージの出力］トレイを閉じるには、モデル エディタ内の任意の場所をクリックします。
9. 

［保存］をクリックします。

手順 5 - 別のログ メッセージの出力ステップの作成

DSstep1 テスト ステップと同様の別のテスト ステップを作成します。

次の手順に従ってください:

1.  ［追加］をクリックします。

   ［ステップの追加］メニューが表示されます。

2. ［ユーティリティ］を選択し、［ログ メッセージの出力］を選択します。

   「ログ メッセージの出力」という名前のステップがモデル エディタに追加されます。

3. ［ログ メッセージの出力］を右クリックし、［名前の変更］を選択します。 名前を「DSstep2」に変更します。

4. 右ペインで、［ログ メッセージの出力］の隣にある矢印をクリックします。

   ［ログ メッセージの出力］トレイが開きます。

5. プレースホルダ テキストを削除します。

6. 以下のログ メッセージを入力します。

   ```
   Date is: {{month}}/{{day}}/{{year}}
   ```

7. ［ログ メッセージの出力］トレイを閉じるには、モデル エディタ内の任意の場所をクリックします。

8. 

［保存］をクリックします。

手順 6 - テストの実行

テストを実行し、何が起きるかを確認するには、対話型テスト ラン（ITR）を使用します。

次の手順に従ってください:

1. ツールバーの  ［対話型テスト ラン（ITR）の開始］をクリックします。

   ITR が開きます。

2. ［実行履歴］ペインで、  ［自動的にテストを実行］をクリックします。
3. 処理が完了したら、［OK］をクリックします。
4. ［実行履歴］ペインで、［DSstep1］および［DSstep2］をクリックします。

   month、day、year プロパティが実際の値と置換されていないことに注目します。 テスト ケースにデータ セットを追加していないので、この結果は予想されたものです。

手順 7 - データ セットの追加

ここでは、DSstep1 テスト ステップに dates.txt データ セットを追加します。

次の手順に従ってください:

1. モデル エディタで、［DSstep1］を選択します。
2. 右ペインで、データ セット ステップのタブをダブルクリックします。
3. データ セット エレメントの下にある  ［追加］をクリックします。
4. ［共通データセット］リストから、［区切りデータ ファイルからの読み取り］を選択します。

   データ セットがテスト ステップに追加されます。

   右ペインでデータ セット エディタが開きます。
5. データ セット エディタで、名前を「DatesDS」に設定します。
6.  ［ファイルの場所］をクリックし、LISA_HOME¥Projects¥My Tutorials¥Data ディレクトリに移動して dates.txt ファイルを選択します。
7. ［テスト アンド キープ］ボタンをクリックします。

   テストが成功すると、［データ セット エディタ］ウィンドウに「テストは成功しました」というメッセージが返されます。
8. ［OK］をクリックします。
9. ツールバーの  ［対話型テスト ラン（ITR）の開始］をクリックし、［新しい ITR を開始］を選択します。
10. ［実行履歴］ペインで、［自動的にテストを実行］  をクリックします。
11. 処理が完了したら、［OK］をクリックします。
12. ［実行履歴］ペインで、［DSstep1］および［DSstep2］をクリックします。

    データ セットの最初の行のデータが［応答］タブに表示されます。両方のステップの応答に同じ日付が表示されます。これは DSstep1 のデータ セットからのみ読み取りを行っているためです。

手順 8 - データ セットの動作の変更

データ セット内の行がすべて読み取られるまでテスト ステップがループされるように、データ セットを変更します。

次の手順に従ってください:

1. モデル エディタで、DSstep1 テスト ステップを選択します。
2. DSstep1 のステップ エレメント パネルで、データ セット エレメントの下の［DatesDS］の隣にある矢印をクリックします。

   データ セット エディタが開きます。
3. ［データが終了した場合］フィールドで、［実行］オプションを選択します。
4. ［実行］フィールドのドロップダウン矢印をクリックし、表示されるリストから［テストを終了する］を選択します。

   この設定により、データ行がすべて読み取られた時点でテストが終了します。
5. ［テスト アンド キープ］ボタンをクリックします。
6. ［OK］をクリックしてテストの成功メッセージを閉じます。
7. モデル エディタで、DSstep2 テスト ステップを選択します。
8. ［ステップ情報］タブで、［次のステップ］ドロップダウン リストを［DSstep1］に設定します。

   この設定により、2 つのテスト ステップがループします。 モデル エディタ内の矢印は、実行の順番を表しています。最初に DSstep1、次に DSstep2、DatesDS と続きます。
9. ツールバーの  ［対話型テスト ラン（ITR）の開始］をクリックし、［新しい ITR を開始］を選択します。
10. ITR で［自動的にテストを実行］  をクリックします。

    データ セットのすべてのデータ行に対して、テスト ケースがループで実行されます。
11. 処理が完了したら、［OK］をクリックします。
12. 

［保存］をクリックします。

チュートリアル 2 - レビュー

このチュートリアルでは、以下のことを行いました。

- カンマ区切り形式でデータ セットを作成しました。
- データ セットを使用して簡単なテスト ケースを実行しました。
- テスト ステップからデータ セットのデータにどのようにアクセスするかを学習しました。

チュートリアル 3 - フィルタおよびアサーション

このチュートリアルでは、チュートリアル 2 で作成したテスト ケースを変更して、フィルタおよびアサーションを含めます。

フィルタおよびアサーションの説明については、「*CA Application Test の使用*」の「フィルタ」および「アサーション」を参照してください。

チュートリアルのタスク

このチュートリアルでは、以下のことを行います。

- 既存のテスト ケースを新しい名前で保存
- テスト ステップへのアサーションの追加
- テスト ステップへのフィルタの追加

前提条件

- 「チュートリアル 2 - データ セット」を完了している。
- DevTest ワークステーション が開いている。

手順 1 - 既存のテスト ケースからのテスト ケースの作成

この手順では、tutorial2.tst を開き、tutorial3.tst として保存します。

次の手順に従ってください:

1. My Tutorials プロジェクトの tutorial2.tst テスト ケースを開きます。
2. メニュー バーから、[ファイル]-[名前を付けて保存] を選択します。
3. [ファイル名] フィールドに「**tutorial3**」と入力します。
4. [保存] をクリックします。

   My Tutorials プロジェクトに tutorial3 テスト ケースが作成および保存されます。

手順 2 - テスト ステップのアクションの変更

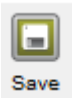
DSstep1 が次のステップになるように、両方のテスト ステップの [次のステップ] アクションを変更します。 最初のステップでのみ、データ セットから読み取りを行います。

次の手順に従ってください:

1. モデル エディタで、[DSstep1] を選択します。
2. [ステップ情報] タブで、[次のステップ] を「**DSstep1**」に変更します。

   このアクションにより、出力は同じステップである DSStep1 に戻ります。一時的に、アラート アイコンが [DSStep2] の隣に表示されます。
3. モデル エディタで [DSstep2] をダブルクリックし、[ログ メッセージの出力] を以下のように変更します。

   ```
   Date contains 1999. It is: {{month}}/{{day}}/{{year}}.
   ```

   注: 中かっこが重要です。 中かっこが含まれている場合にのみ、テスト ケースは正しく実行されます。
4. 

   [保存] をクリックします。

手順 3 - アサーションの追加

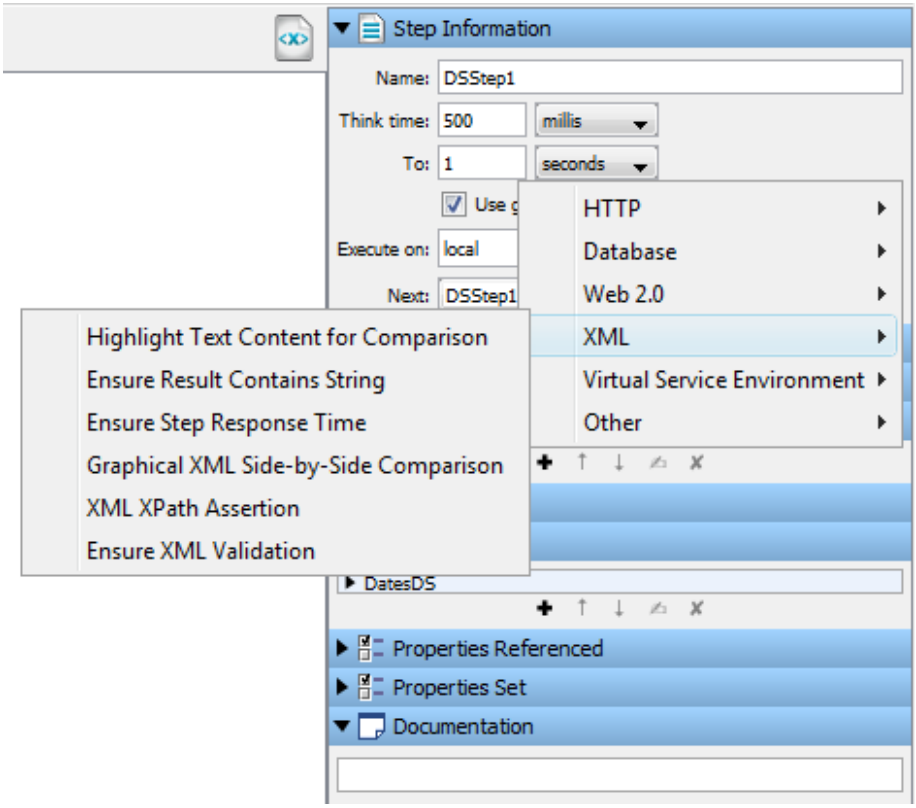
さまざまなタイプのアサーションをテスト ケースに追加できます。 この手順では、「**結果が文字列を含むことを確認**」という名前の XML アサーションを追加します。

アサーションのロジックは以下のとおりです。

- 応答に 1999 という文字列が含まれている場合は、次に DSstep2 ステップが実行されます。
- 応答に 1999 という文字列が含まれていない場合は、次に DSstep1 ステップが実行されます。

次の手順に従ってください:

1. モデル エディタで、［DSstep1］を選択します。
2. ［アサーション］タブを開きます。
3. ［追加］をクリックします。
4. ［XML］サブメニューから、［結果が文字列を含むことを確認］を選択します。



5. DSstep1 に適用される新しいアサーションが［アサーション］タブに追加されます。

   アサーション エディタが開きます。

6. アサーション エディタで、以下の手順を実行します。

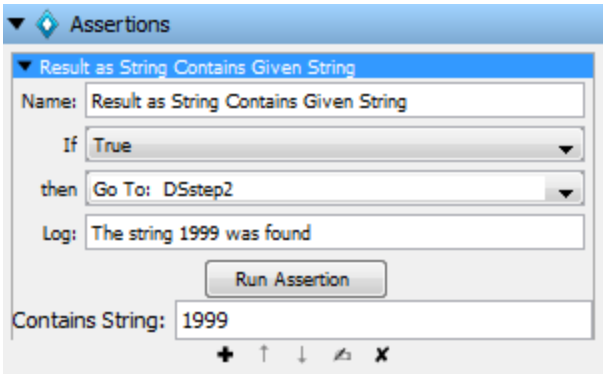
   a. ［条件］リストで、［True］を選択します。

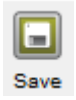
   b. ［次のステップ］リストで、［DSstep2 へ移動する］を選択します。

   c. ［ログ］フィールドに「The string 1999 was found」と入力します。

   d. ［含む文字列］フィールドに「1999」と入力します。

図 1: ［文字列としての結果に指定された文字列を含める］アサーション ダイアログ ボックスのスクリーンショット



7. 

［保存］をクリックします。

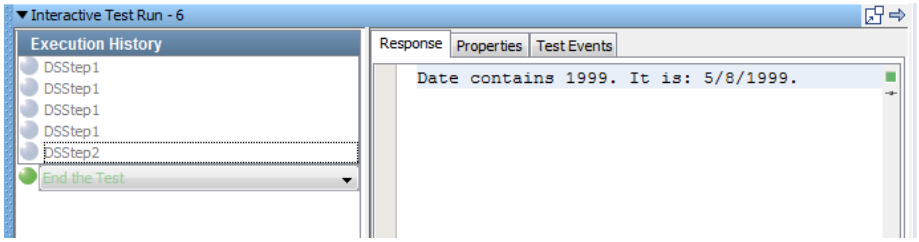
手順 4 - アサーションのテスト

アサーションが想定したように動作するかどうかを判断するには、対話型テスト ラン（ITR）ユーティリティを使用します。

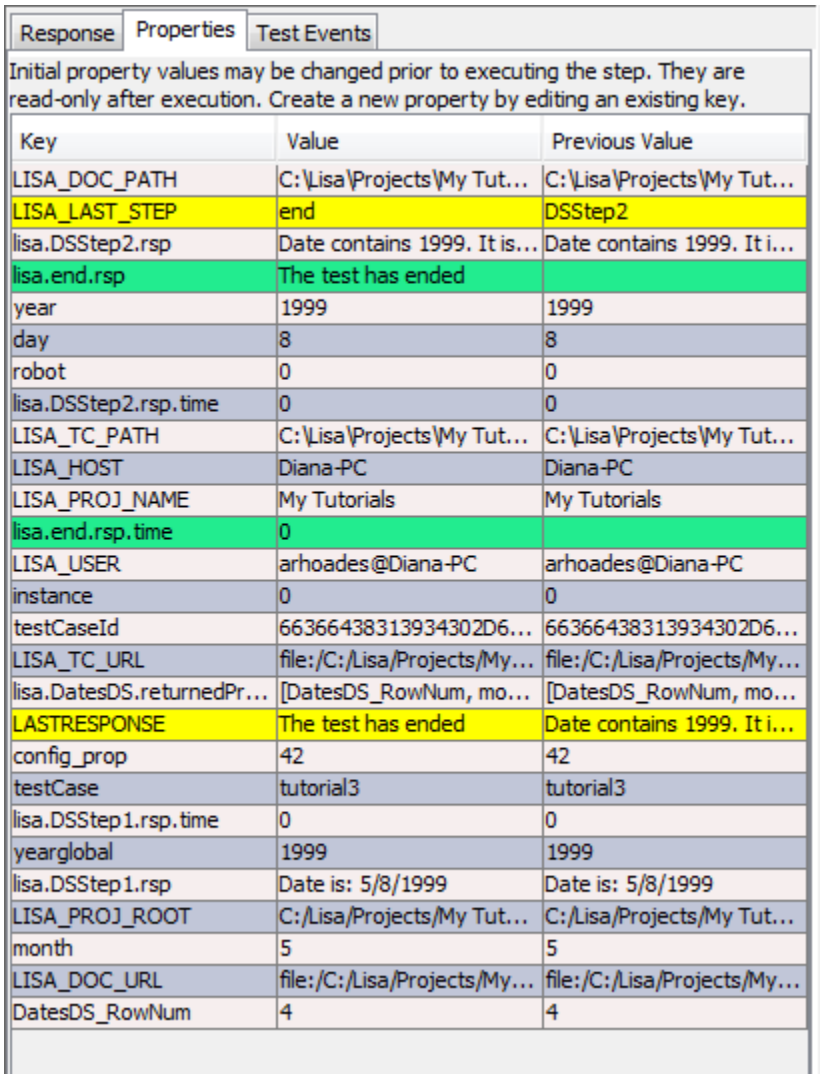
次の手順に従ってください:

1. 新しい ITR セッションを開始します。
2. ［実行履歴］ペインで、［自動的にテストを実行］  をクリックします。
3. 処理が完了したら、［OK］をクリックします。
4. ［応答］タブを確認します。

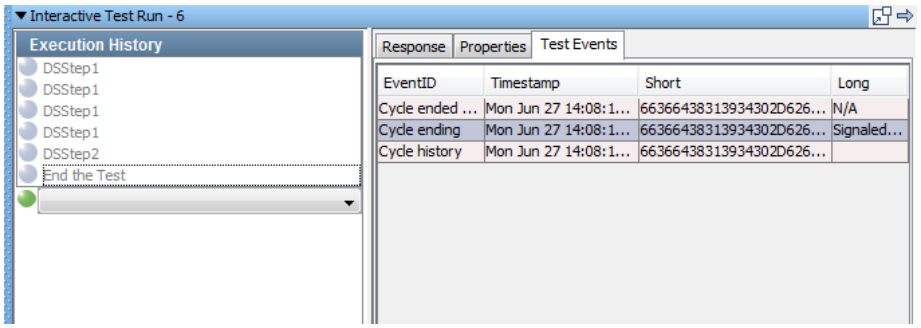
   注: DSstep1 で year が 1999 である日付が見つかると、次に DSstep2 ステップが実行されます。

5. ［プロパティ］タブをクリックし、プロパティの動作を確認します。



6. ［テストイベント］タブをクリックし、生成されたイベントを確認します。



手順 5 - フィルタの追加

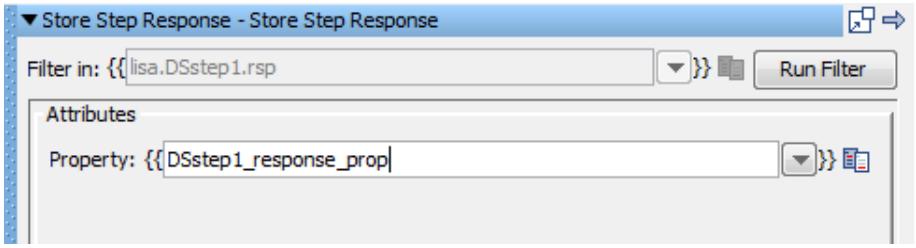
さまざまなタイプのフィルタをテスト ケースに追加できます。 この手順では、［ステップ応答の格納］という名前のユーティリティ フィルタを追加します。 このタイプのフィルタによって、ステップの応答をプロパティとして保存できます。

次の手順に従ってください:

1. モデル エディタで、［DSstep1］を選択します。
2. ［フィルタ］タブを開きます。
3. ［追加］をクリックします。
4. ［ユーティリティ］メニューのサブメニューから、［ステップ応答の格納］を選択します。

   フィルタ エディタが開きます。
5. フィルタ エディタで、プロパティ名を「**DSstep1_response_prop**」に設定します。

   このプロパティには、ステップの応答が格納されます。



6. モデル エディタで［DSstep2］をダブルクリックし、出力ログ メッセージの末尾に以下のテキストを追加します。

   ```
   The value of DSstep1_response_prop is: {{DSstep1_response_prop}}.
   ```
7. Save ［保存］をクリックします。

手順 6 - フィルタのテスト

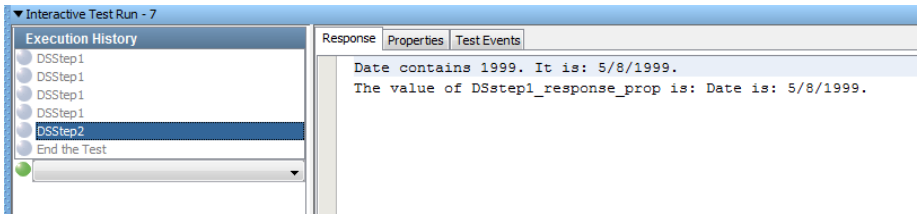
フィルタが想定したように動作するかどうかを判断するには、対話型テスト ラン（ITR）ユーティリティを使用します。

次の手順に従ってください:

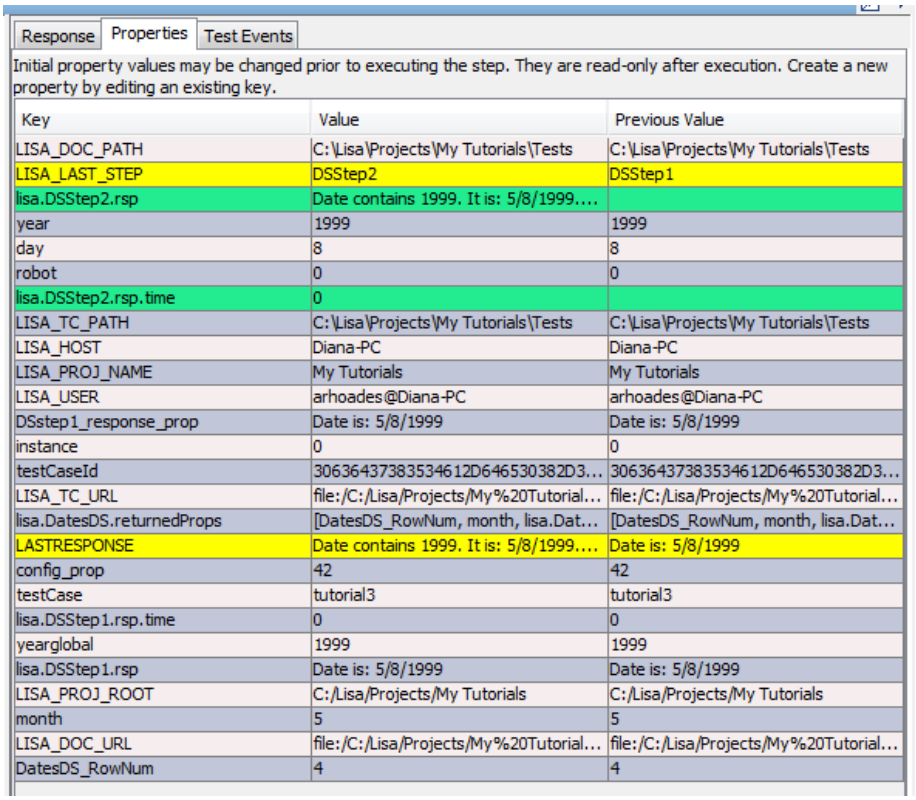
1. 新しい ITR セッションを開始します。
2. ［実行履歴］ペインで、  ［自動的にテストを実行］をクリックします。
3. 処理が完了したら、［OK］をクリックします。
4. ［応答］タブを確認します。

   DSstep2 テスト ステップにより、出力ログ メッセージに追加したテキストが表示されます。

図 2: チュートリアル 3 の ITR の［応答］タブのスクリーンショット

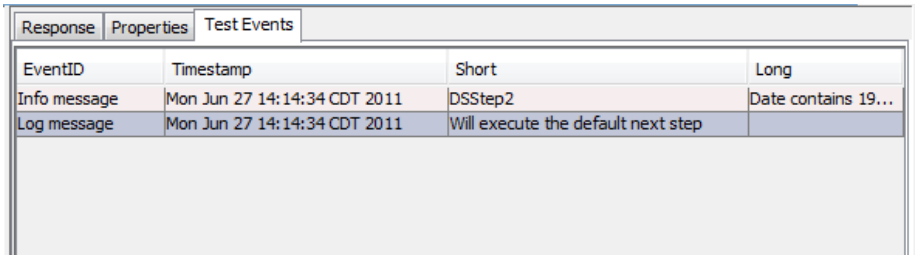


5. ［プロパティ］タブをクリックし、DSstep1_response_prop プロパティがどこで作成され、変更されるかを確認します。



Key	Value	Previous Value
LISA_DOC_PATH	C:\Lisa\Projects\My Tutorials\Tests	C:\Lisa\Projects\My Tutorials\Tests
LISA_LAST_STEP	DSStep2	DSStep1
lisa.DSStep2.rsp	Date contains 1999. It is: 5/8/1999....	
year	1999	1999
day	8	8
robot	0	0
lisa.DSStep2.rsp.time	0	
LISA_TC_PATH	C:\Lisa\Projects\My Tutorials\Tests	C:\Lisa\Projects\My Tutorials\Tests
LISA_HOST	Diana-PC	Diana-PC
LISA_PROJ_NAME	My Tutorials	My Tutorials
LISA_USER	arhoades@Diana-PC	arhoades@Diana-PC
DSstep1_response_prop	Date is: 5/8/1999	Date is: 5/8/1999
instance	0	0
testCaseId	30636437383534612D646530382D3...	30636437383534612D646530382D3...
LISA_TC_URL	file:/C:/Lisa/Projects/My%20Tutorial...	file:/C:/Lisa/Projects/My%20Tutorial...
lisa.DatesDS.returnedProps	[DatesDS_RowNum, month, lisa.Dat...	[DatesDS_RowNum, month, lisa.Dat...
LASTRESPONSE	Date contains 1999. It is: 5/8/1999....	Date is: 5/8/1999
config_prop	42	42
testCase	tutorial3	tutorial3
lisa.DSStep1.rsp.time	0	0
yearglobal	1999	1999
lisa.DSStep1.rsp	Date is: 5/8/1999	Date is: 5/8/1999
LISA_PROJ_ROOT	C:/Lisa/Projects/My Tutorials	C:/Lisa/Projects/My Tutorials
month	5	5
LISA_DOC_URL	file:/C:/Lisa/Projects/My%20Tutorial...	file:/C:/Lisa/Projects/My%20Tutorial...
DatesDS_RowNum	4	4

6. ［テストイベント］タブをクリックし、生成されたイベントを確認します。



EventID	Timestamp	Short	Long
Info message	Mon Jun 27 14:14:34 CDT 2011	DSStep2	Date contains 19...
Log message	Mon Jun 27 14:14:34 CDT 2011	Will execute the default next step	

チュートリアル 3 - レビュー

このチュートリアルでは、以下のことを行います。

- フィルタおよびアサーションを初めて使用しました。
- 既存のテスト ケースを開いて変更しました。
- 簡単なアサーションを追加する方法を学習しました。
- 簡単なフィルタを追加する方法を学習しました。
- アサーションおよびフィルタが想定したように動作したことを検証するために、対話型テスト ラン（ITR）ユーティリティを使用しました。

詳細

DevTest にはテスト ケース開発で発生するほとんどの状況を網羅するフィルタおよびアサーションが用意されています。 適切なフィルタがない場合は、SDK （Software Developers Kit）を使用してカスタム フィルタおよびアサーションを開発できます。 詳細については、「*SDK の使用*」を参照してください。

チュートリアル 4 - Java オブジェクト（POJO）の操作

このチュートリアルでは、簡単な Java オブジェクトを作成して操作し、**java.util.Date** クラスを使用して日付オブジェクトを作成します。

まず、オブジェクトを作成して、オブジェクトに対してメソッドをコールする方法を確認します。 その後、単純な DevTest モデル エディタにオブジェクトを埋め込みます。

チュートリアルのタスク

このチュートリアルでは、以下のことを行います。

- 動的 Java 実行テスト ステップの使用
- 簡単なオブジェクトの複合オブジェクト エディタでの使用
- インライン フィルタの使用とプロパティへの結果の保存

前提条件

- 「チュートリアル 3 - フィルタおよびアサーション」を完了している。
- DevTest ワークステーション が開いている。

手順 1 - テスト ケースの作成

テスト ケースを作成する方法

- My Tutorials プロジェクトに「tutorial4」という名前のテスト ケースを作成します。

手順 2 - 動的 Java 実行テスト ステップの作成

動的 Java 実行テスト ステップでは、DevTest クラスパスのクラスから Java オブジェクトを作成できます。以下の手順では、**java.util.Date** クラスを使用します。

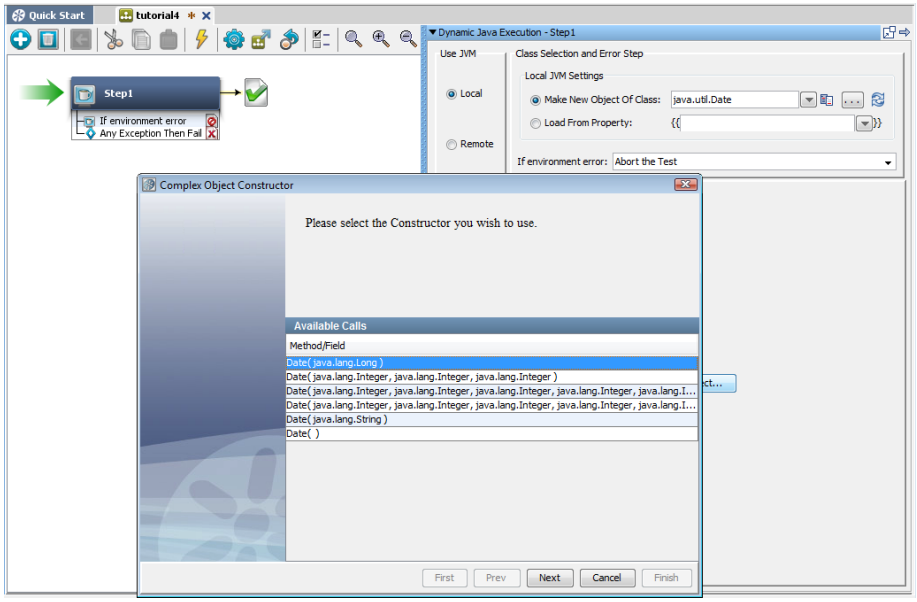
次の手順に従ってください:

1.  ［ステップの追加］をクリックします。

   ［ステップの追加］メニューが表示されます。

2. ［Java/J2EE］を選択し、次に［動的 Java 実行］を選択します。

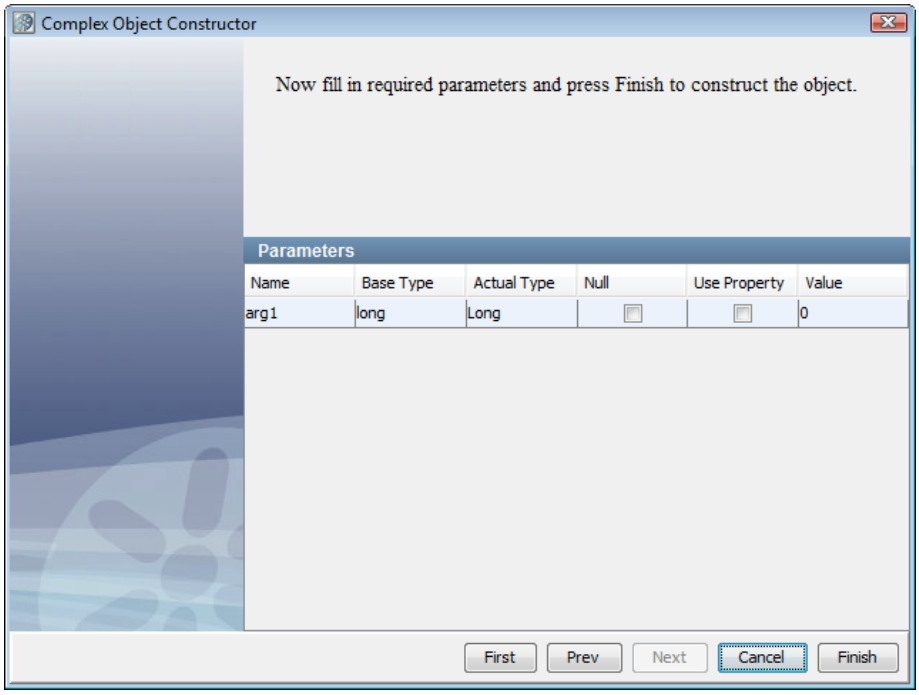
   動的 Java 実行エディタが開きます。



3. ［ローカル JVM 設定］領域で、［クラスの新規オブジェクトを作成］が選択されていることを確認します。
4. ［クラスの新規オブジェクトを作成］の右側にあるフィールドに、「**java.util.Date**」と入力します。
5. ［オブジェクトの構築/ロード］をクリックします。

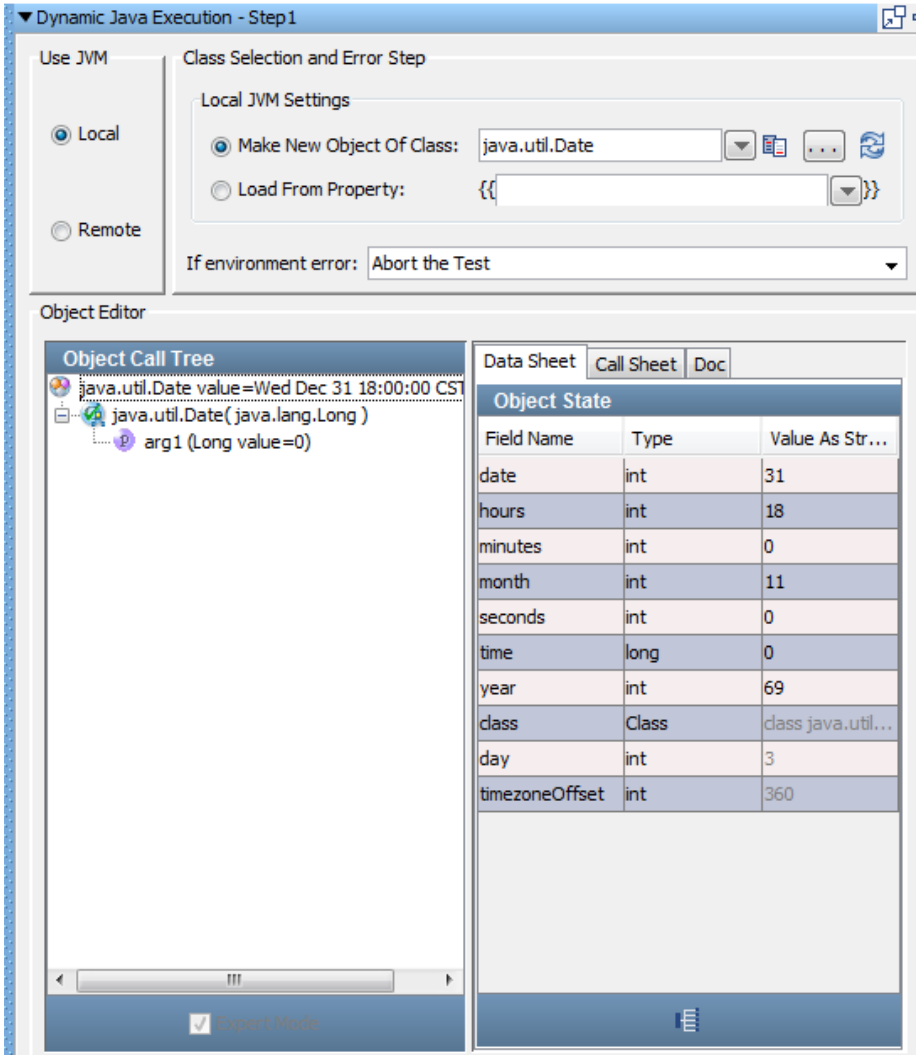
   複合オブジェクト コンストラクタ ウィザードが表示されます。 最初のステップには、使用可能なコンストラクタが表示されています。

6. ［Date(java.lang.Long)］コンストラクタを選択します。



7. ［次へ］をクリックします。
8. ［終了］をクリックします。

   複合オブジェクト エディタが開きます。



複合オブジェクト エディタに操作する Java オブジェクトが表示されます。

手順 3 - Java オブジェクトのコール

複合オブジェクト エディタは 2 つのパネルで構成されています。 左側のパネルには、オブジェクト コール ツリーが表示されます。ここでは、メソッド呼び出しやそれらの入力パラメータおよび戻り値を追跡します。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。


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。 このアイコンは、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）。

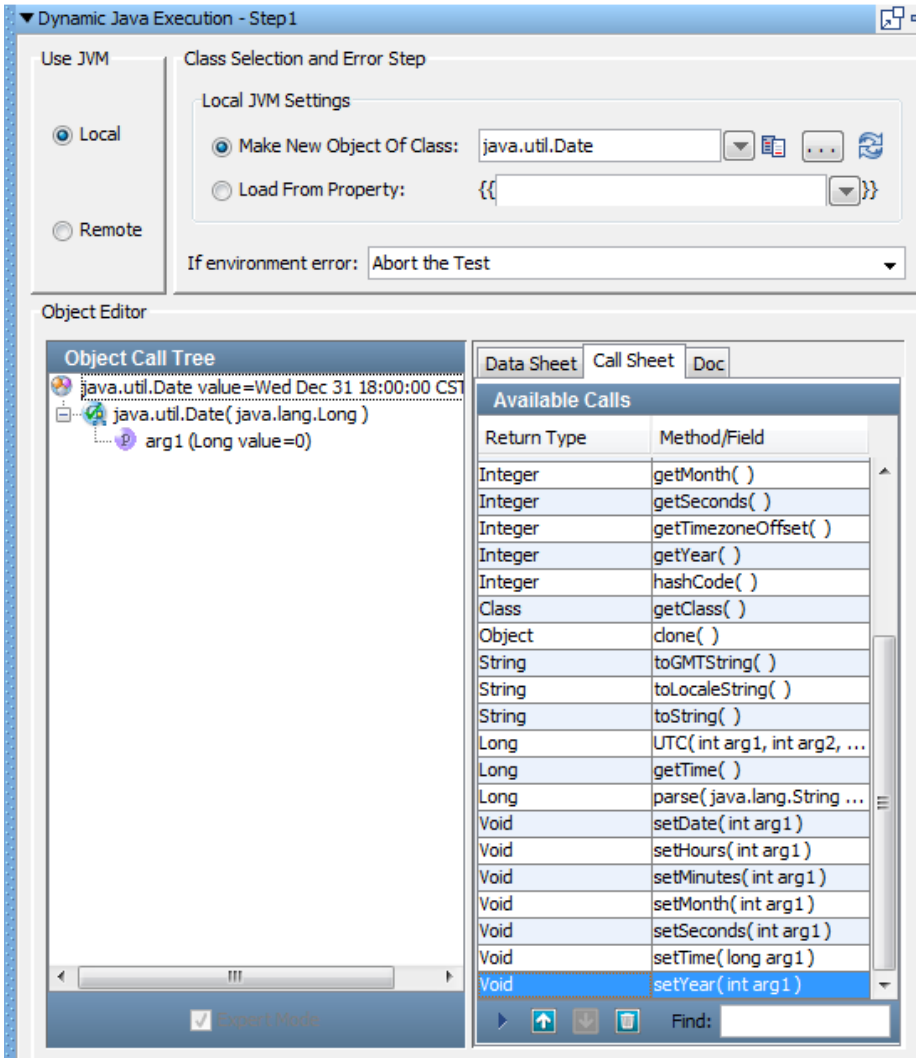
右側のパネルの内容は、左側のパネルで選択された項目によって変化します。

Java オブジェクトをコールする方法

1. 複合オブジェクト エディタの右側のパネルで、［コール シート］タブをクリックします。

   ［コール シート］タブには、コールできる使用可能なメソッドが表示されます。

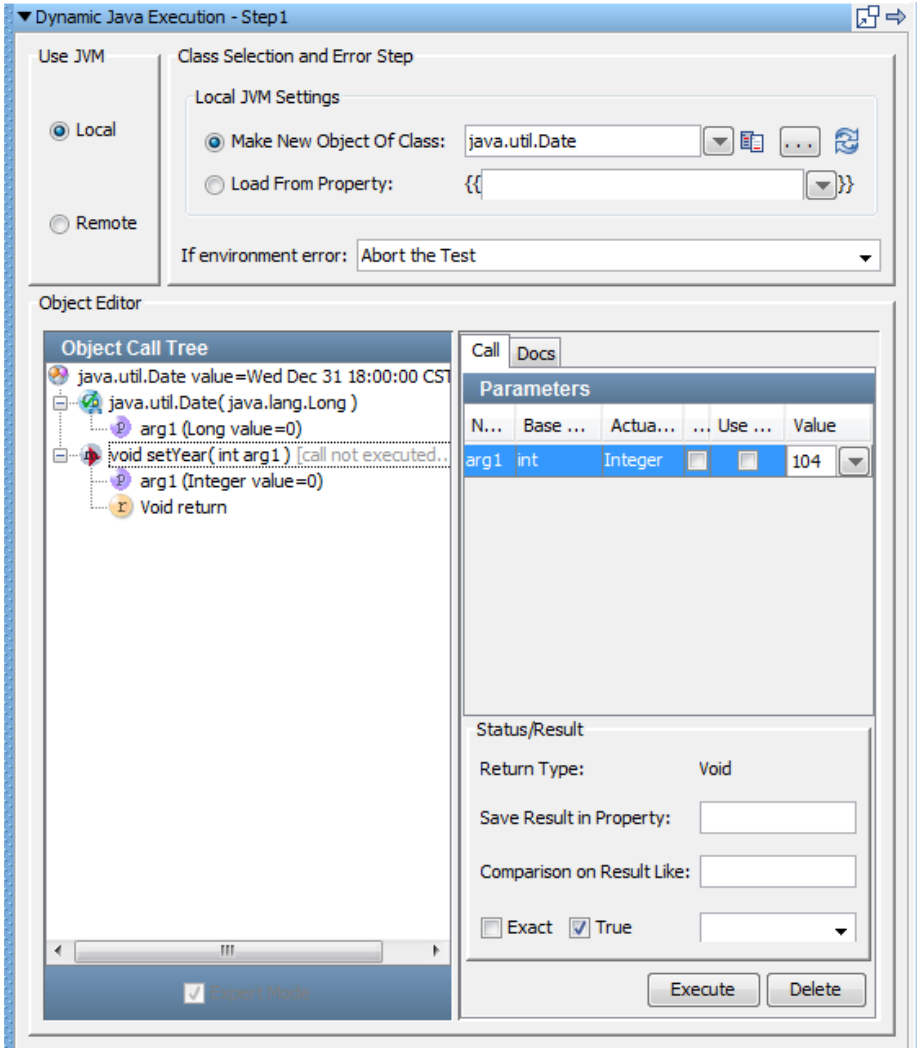
2. setYear() メソッドをダブルクリックします。 または setYear() メソッドを選択し、  ［選択されたメソッドをオブジェクト コール ツリーに追加］をクリックすることもできます。



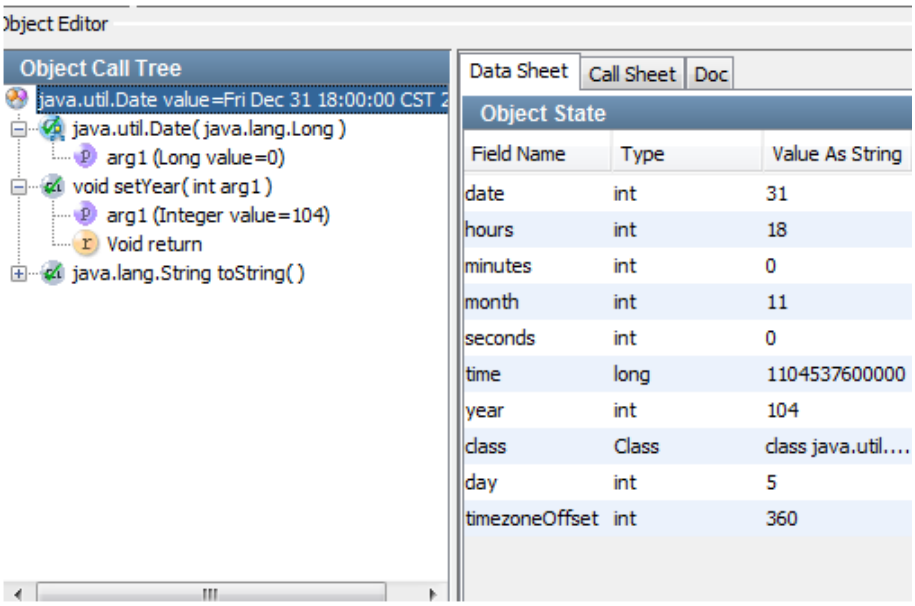
setYear() メソッドがオブジェクト コール ツリーに追加されます。右側のパネルに［コール］タブと［ドキュメント］タブが表示されます。

［コール］タブには引数情報が表示されます。

3. arg1 の［値］フィールドに「**104**」と入力します。
4. ［実行］をクリックします。



5. オブジェクト コール ツリーで、**java.util.Date** オブジェクトを選択します。
6. ［データ シート］タブで、year フィールドが 104 に設定されていることを確認します。


Object Editor
Object Call Tree
java.util.Date value=Fri Dec 31 18:00:00 CST 2
java.util.Date(java.lang.Long)
arg1 (Long value=0)
void setYear(int arg1)
arg1 (Integer value=104)
Void return
java.lang.String toString( )
Data Sheet
Call Sheet
Doc
Object State
Field Name
Type
Value As String
date int 31
hours int 18
minutes int 0
month int 11
seconds int 0
time long 1104537600000
year int 104
class Class class java.util....
day int 5
timezoneOffset int 360

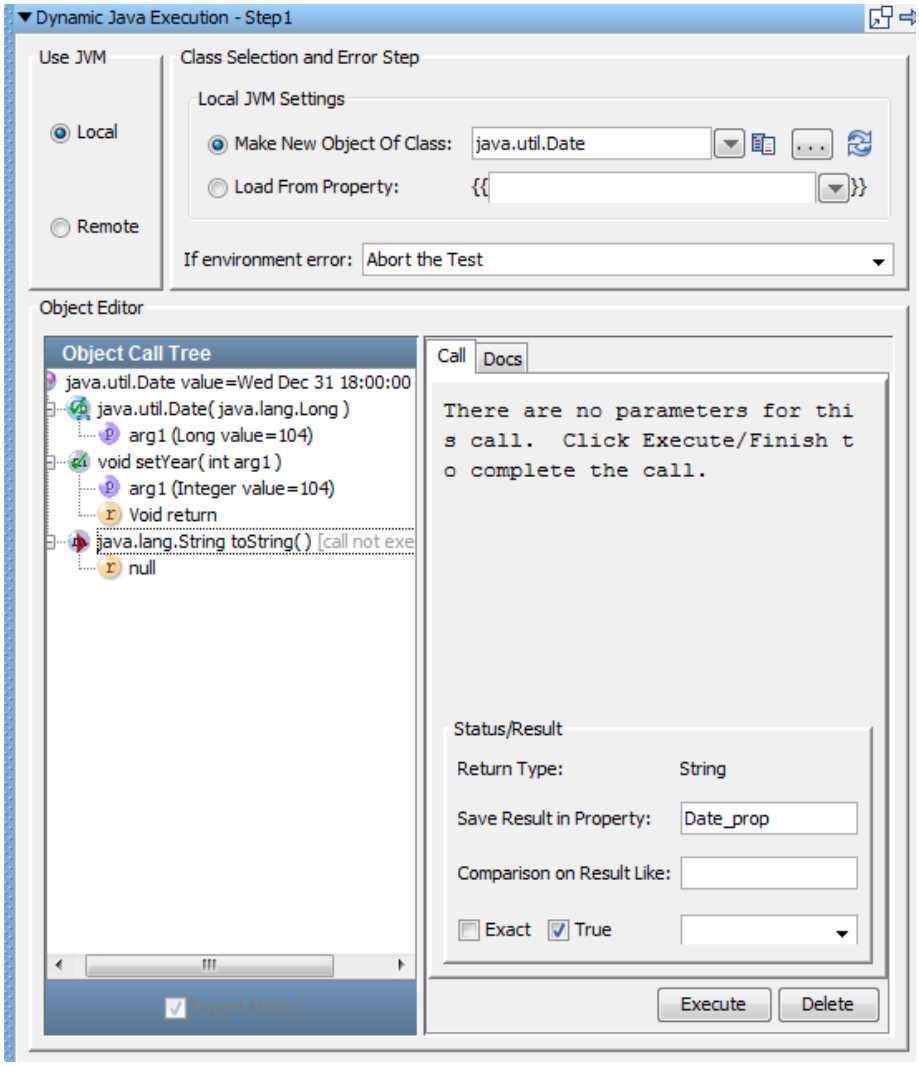
手順 4 - インライン フィルタの追加

複合オブジェクト エディタからインライン フィルタを追加できます。 インライン フィルタ（およびアサーション）は、エレメント パネルのテスト 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。

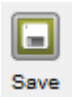
次の手順に従ってください：

1. **java.util.Date** オブジェクトを選択した状態で［コール シート］タブをクリックします。
2. プロパティに設定する日付を取得するために、toString() メソッドを呼び出します。

   右側のパネルに［コール］タブと［ドキュメント］タブが表示されます。
3. ［コール］タブの［ステータス/結果］領域で、［結果を保存するプロパティ］フィールドにプロパティ名として「**Date_prop**」と入力してインライン フィルタを追加します。



4. ［実行］をクリックします。



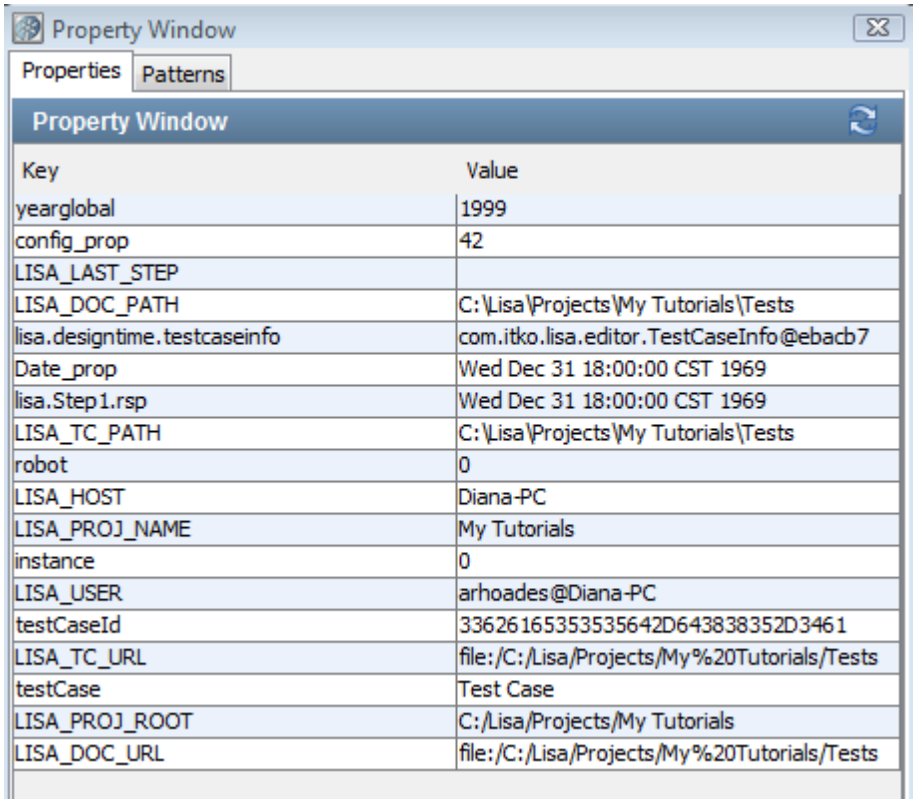
5. ［保存］をクリックします。

手順 5 - 作成したプロパティの確認

Date_prop プロパティが作成されたことを確認するには、プロパティ ウィンドウを表示します。

次の手順に従ってください：

1. 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。 または、メイン メニューから［ヘルプ］-［プロパティの表示］を選択します。
2. ［リフレッシュ］  をクリックします。
3. **Date_prop** プロパティを見つけます。



4. ［閉じる］をクリックします。

チュートリアル 4 - レビュー

このチュートリアルでは、以下のことを行いました。

- **java.util.Date** タイプの Java オブジェクトを操作するためにテスト ステップを作成しました。
- 複合オブジェクト エディタを使用して Java オブジェクトを操作しました。
- オブジェクトにインライン フィルタを追加し、結果をプロパティに保存する方法を学習しました。

チュートリアル 5 - デモ サーバの Web アプリケーションの実行

このチュートリアルでは、DevTest に付属している簡単な Web アプリケーションを使用します。

LISA Bank アプリケーションは、金融口座の情報が含まれるデータベース テーブルに接続する簡単なフロント エンドです。 このアプリケーションのビジネス ロジックは、Enterprise JavaBeans と Web サービスで構成されています。 この Web アプリケーションでは、ユーザのプロファイルの表示、口座の作成、住所の追加などの操作を行えます。

このチュートリアルの目的は、このアプリケーションについて理解することです。このアプリケーションは、テスト中のシステムとして後のチュートリアルで使用します。

前提条件

- デモ サーバが実行されている。

手順 1 - Web アプリケーションの起動

次の手順に従ってください:

1. 新しいブラウザ ウィンドウを開きます。
2. 以下の URL を入力します。パス内の **localhost** は、使用するコンピュータの IP アドレスに置き換えてください。

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

   ログイン ページが表示されます。

手順 2 - Web アプリケーションへのログイン

事前に定義されているユーザ名 lisa_simpson を使用します。

次の手順に従ってください:

1. ［名前］フィールドに「**lisa_simpson**」と入力します。
2. ［パスワード］フィールドに「**golisa**」と入力します。
3. ［ログイン］をクリックします。

初期画面が表示されます。 左側には、［プロファイルの表示］、［口座の開設］、［口座の解約］、［住所の追加］、［住所の削除］、［ログアウト］といった実行可能なさまざまなアクションのボタンが表示されます。

手順 3 - 口座の開設

現在のユーザには口座がありません。

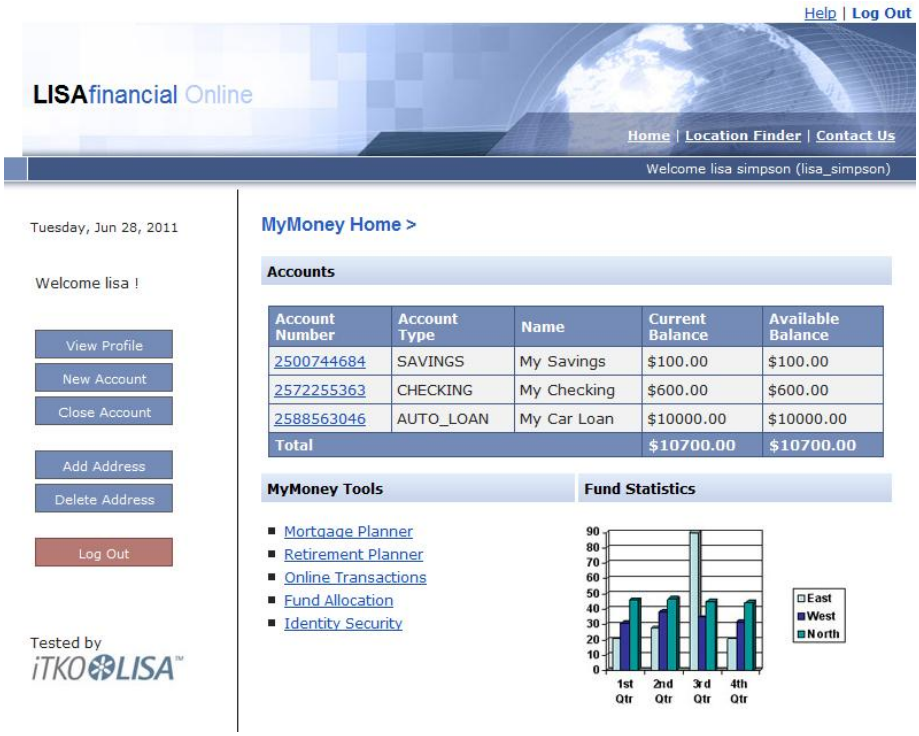
次の手順に従ってください:

1. ［口座の開設］をクリックします。
2. ［預金種別］リストから［普通預金］を選択します。
3. ［口座名］フィールドに「**My Savings**」と入力します。
4. ［初期残高］フィールドに「**100.00**」と入力します。
5. ［口座の追加］をクリックします。

新しい普通預金口座が［口座］セクションに追加されます。

6. さらにこの手順を繰り返して、当座預金と自動車ローンの口座を作成します。当座預金の初期残高は 600.00 ドルに設定します。自動車ローンの初期残高は 10000.00 ドルに設定します。

   ［初期残高］フィールドでは、カンマを使用しないでください。


Help | Log Out
LISAfinancial Online
Home | Location Finder | Contact Us
Welcome lisa simpson (lisa_simpson)
Tuesday, Jun 28, 2011
Welcome lisa !
View Profile
New Account
Close Account
Add Address
Delete Address
Log Out
Tested by
iTKO LISA
MyMoney Home >
Accounts
Account Number
Account Type
Name
Current Balance
Available Balance
2500744684
SAVINGS
My Savings
$100.00
$100.00
2572255363
CHECKING
My Checking
$600.00
$600.00
2588563046
AUTO_LOAN
My Car Loan
$10000.00
$10000.00
Total
$10700.00
$10700.00
MyMoney Tools
Fund Statistics
Mortgage Planner
Retirement Planner
Online Transactions
Fund Allocation
Identity Security
90
80
70
60
50
40
30
20
10
0
1st Qtr
2nd Qtr
3rd Qtr
4th Qtr
East
West
North

手順 4 - 口座の解約

このアプリケーションでは口座を解約することができます。

次の手順に従ってください:

1. ［口座の解約］をクリックします。
2. リストから **My Savings** を選択して、［口座の選択］をクリックします。

3. ［削除の確認］をクリックします。

My Savings 口座が［口座］セクションから削除されます。

手順 5 - Web アプリケーションからのログアウト

Web アプリケーションからログアウトする方法

1. ［ログアウト］をクリックします。

チュートリアル 5 - レビュー

このチュートリアルでは、以下のことを行いました。

- LISA Bank アプリケーションにログインしました。
- 新しい口座を開設しました。
- 口座を解約しました。

チュートリアル 6 - Web サイトのテスト

このチュートリアルでは、Web レコーダを使用して、Web サイトを介してパスを記録します。次に、HTTP 要求/応答の各ペアに対して HTTP/HTML 要求テスト ステップを作成します。

HTTP/HTML 要求テスト ステップでは、テスト ケースで Web サーバの要求を行い、その結果を受信できます。 ページが想定したように動作することを確認するために、簡単な Web サイトをテストします。

チュートリアルのタスク

このチュートリアルでは、以下のことを行います。

- Web レコーダを使用した、HTTP/HTML 要求ステップが含まれるテスト ケースの作成
- レコーダが作成するテスト ケースの編集および実行
- テスト ケースへのデータ セットの追加

前提条件

- 「チュートリアル 5 - デモ サーバの Web アプリケーションの実行」を完了している。
- DevTest ワークステーション が開いている。
- デモ サーバへのアクセス権がある。

チュートリアルのパート

- チュートリアル 6 - パート A - テスト ケースの記録 (P. 65)
- チュートリアル 6 - パート B - テスト ケースの実行 (P. 72)
- チュートリアル 6 - パート C - 要求テスト ステップの変更 (P. 73)

チュートリアル 6 - パート A - テスト ケースの記録

Web レコーダを使用して、HTTP/HTML 要求ステップが含まれるテスト ケースを作成します。

手順 1 - テスト ケースの作成

次の手順に従ってください:

1. My Tutorials プロジェクトで、Tests サブフォルダに「tutorial6a」という名前のテスト ケースを作成します。

   モデル エディタが開きます。

手順 2 - Web レコーダの開始

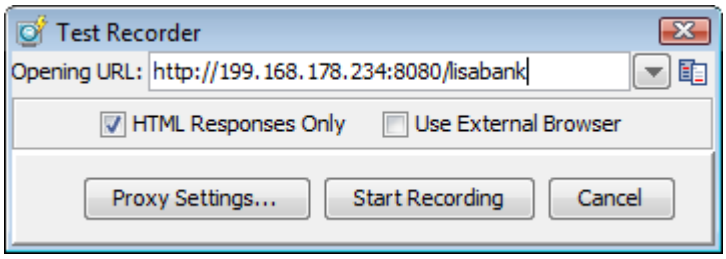
このチュートリアルでは、HTTP プロキシを使用して Web サイトを記録します。

次の手順に従ってください:

1. メイン メニューから［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）］を選択します。

   ［テスト レコーダ］ダイアログ ボックスが表示されます。

2. ［URL を開いています］フィールドに以下の URL を入力します。 パス内の IP アドレスは、使用しているコンピュータの IP アドレスに置き換えてください。



3. ［レコーディングを開始］をクリックします。

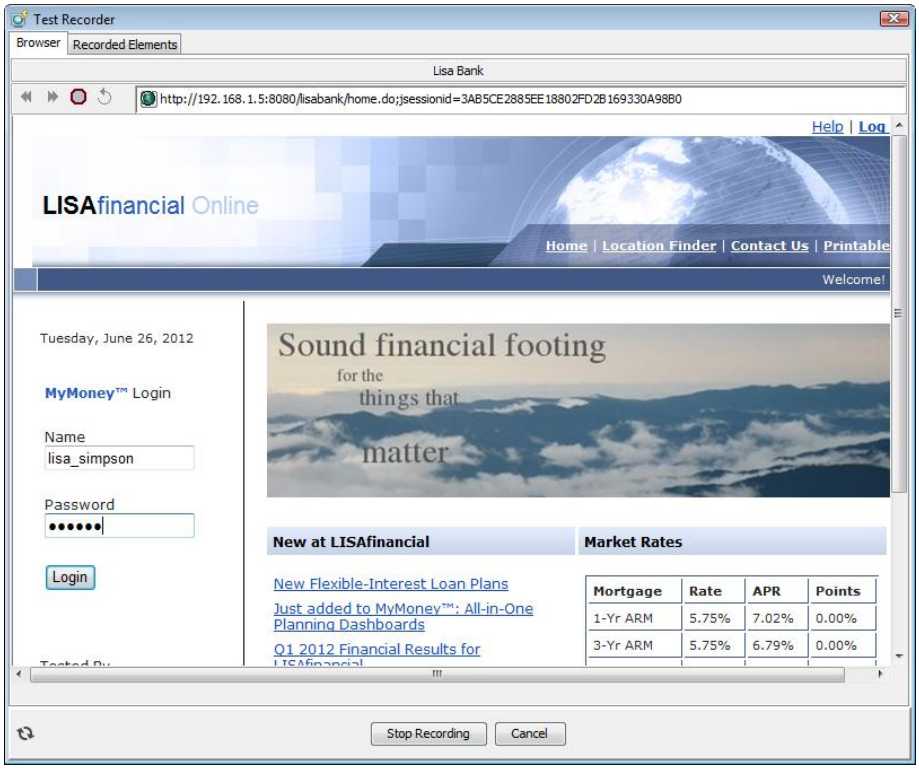
   ［テスト レコーダ］ウィンドウが表示されます。 ［テスト レコーダ］ウィンドウには［ブラウザ］と［記録されたエレメント］という 2 つのタブがあります。 LISA Bank アプリケーションのログイン ページが［ブラウザ］タブに表示されます。

手順 3 - LISA Bank アプリケーションの記録

LISA Bank アプリケーションのアクションを実行すると、参照した各ページに対する要求と応答の情報が記録されます。

次の手順に従ってください:

1. ［名前］フィールドに「**lisa_simpson**」と入力します。 ［パスワード］フィールドに「**golisa**」と入力します。



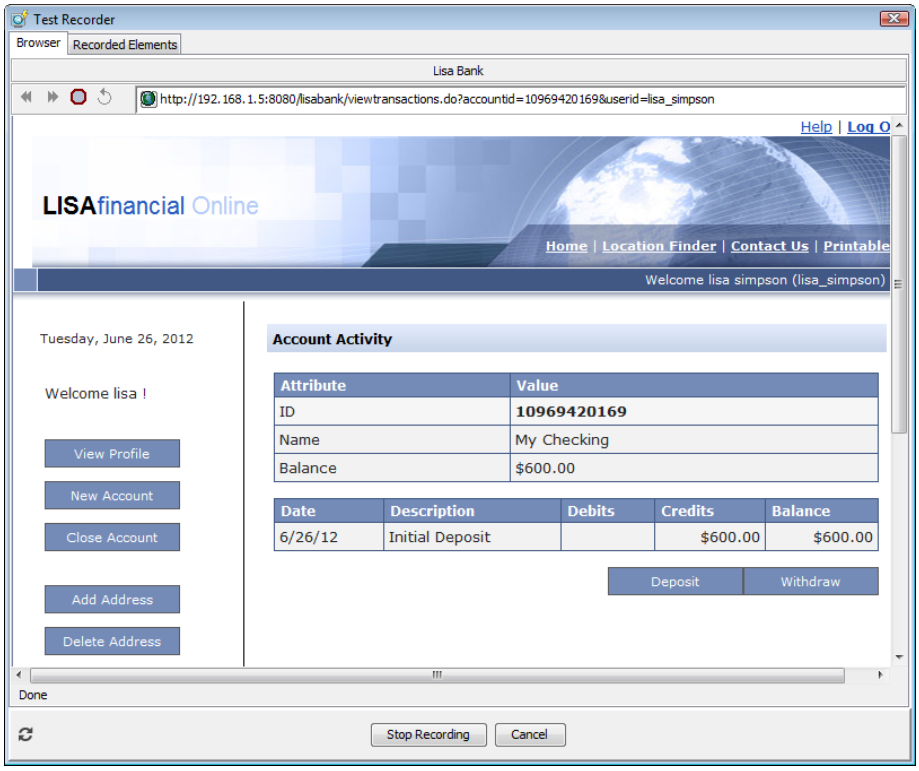
2. ［ログイン］をクリックします。

   初期画面が表示されます。

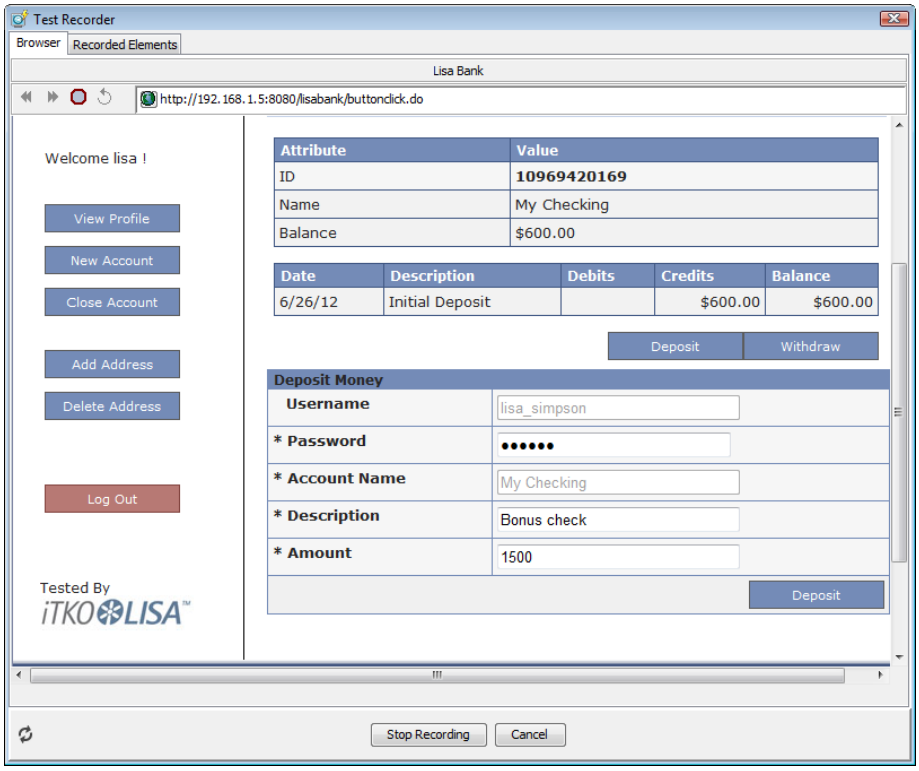
3. ［口座］セクションで、当座預金の口座番号をクリックします。

   ［口座のアクティビティ］ウィンドウが表示されます。

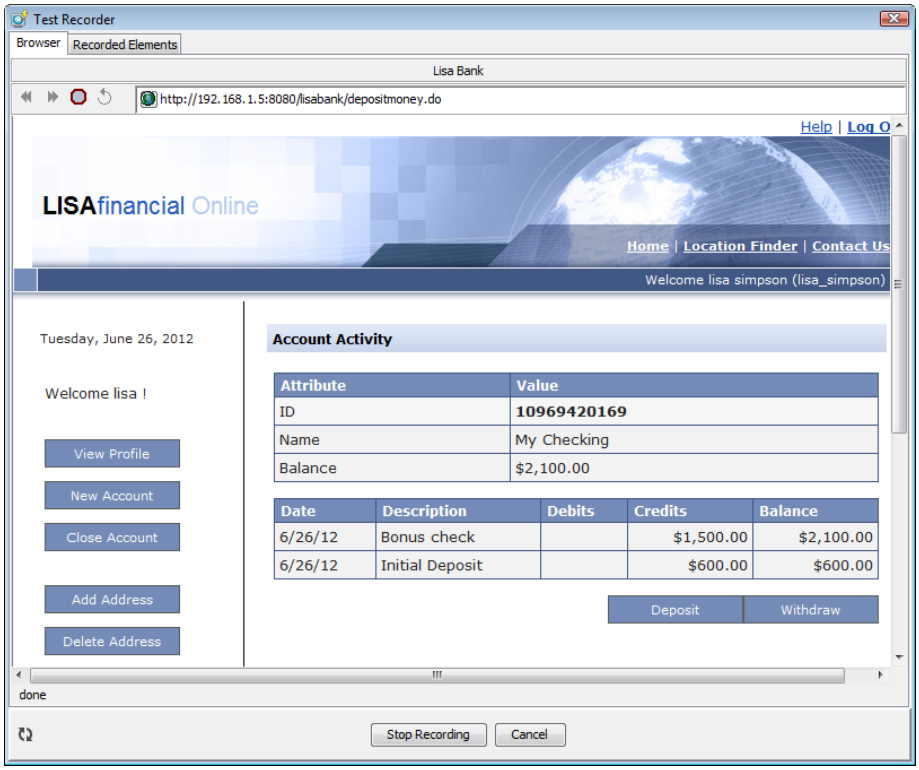
4. ［お預け入れ］をクリックします。



5. ［お預け入れ］領域で、パスワードの「**golisa**」、トランザクションの説明、および預け入れ金額を入力します。
6. ［お預け入れ］をクリックします。



［口座のアクティビティ］ウィンドウに、更新された残高および預け入れの記録が表示されます。



7. 左側のナビゲーション ペインで、［ログアウト］をクリックします。

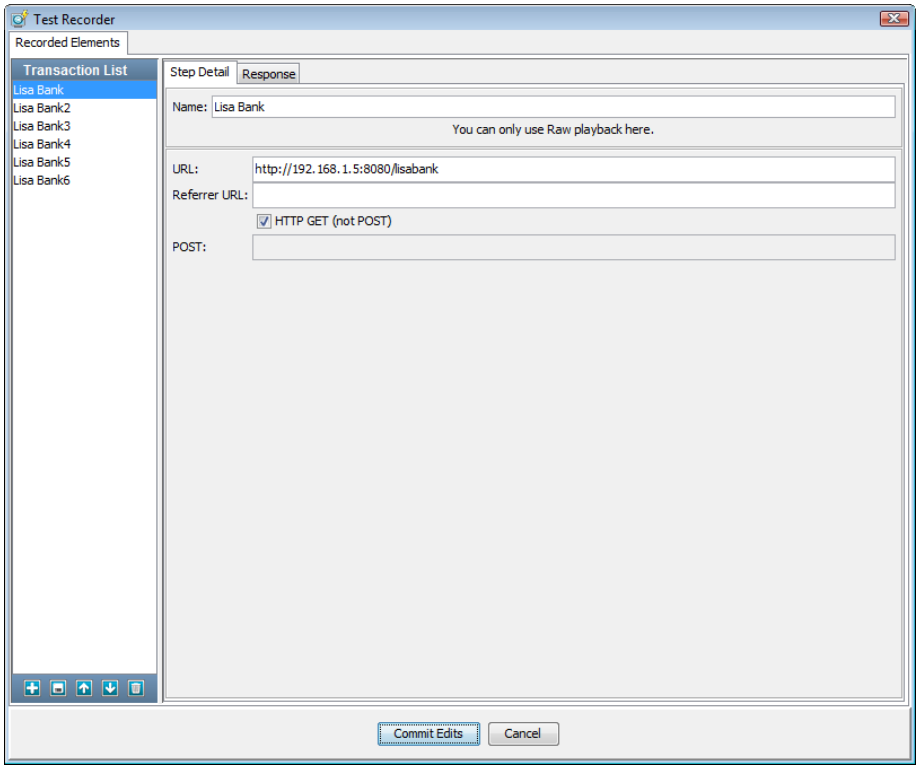
手順 4 - Web レコーダの停止

記録を停止した後、トランザクションに関する詳細を表示できます。

次の手順に従ってください:

1. ［テスト レコーダ］ウィンドウの下部にある［レコーディングの停止］をクリックします。

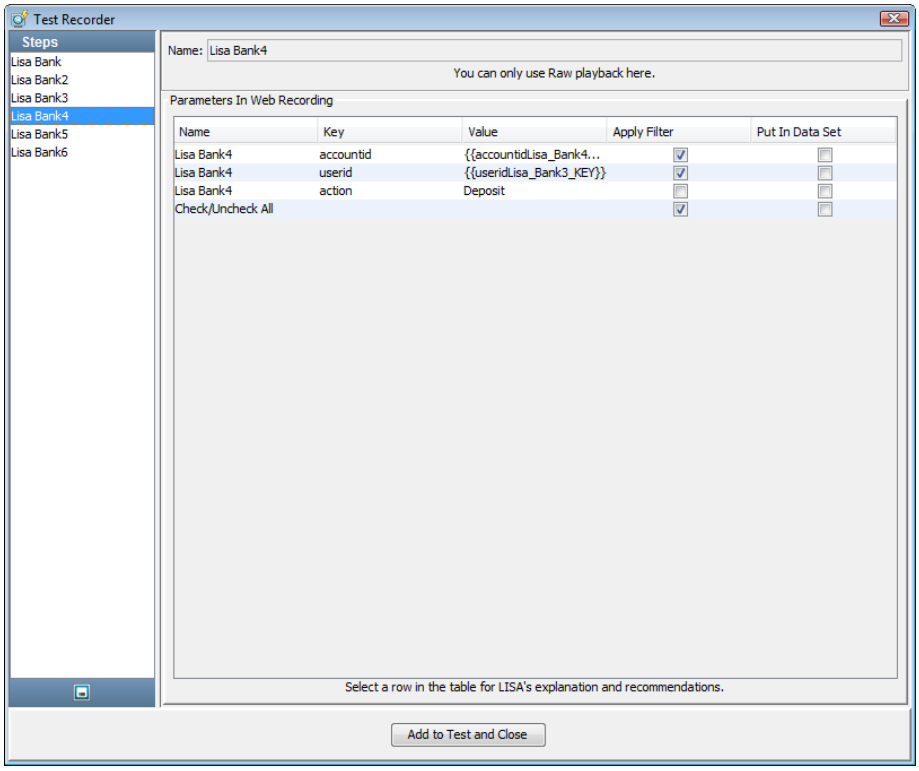
   フィルタおよびプロパティが Web ページの参照に対して自動的に作成されます。 預け入れのためのフォーム フィールドも表示されます。左ペインには、トランザクション（ステップ）のリストが表示されます。 右ペインには、選択したトランザクションのステップの詳細と応答が表示されます。



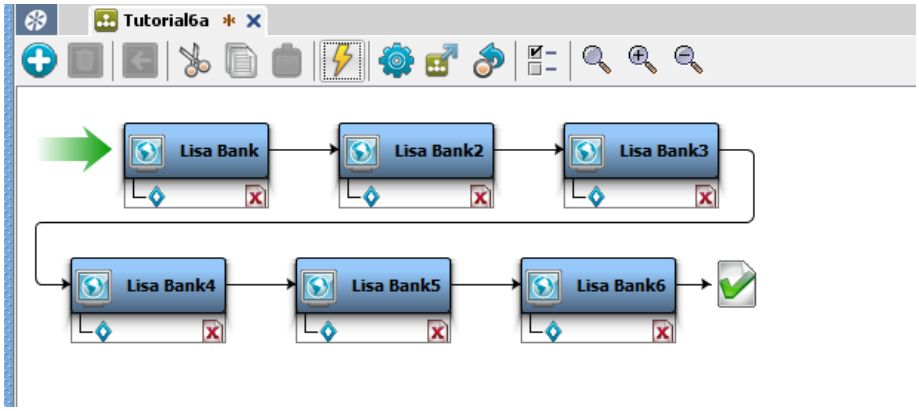
2. ［編集内容をコミット］をクリックします。

   パラメータのページが表示されます。

3. ［テストに追加して閉じる］をクリックします。



モデル エディタに新しいテスト ケースが入力されます。トランザクション情報はすべて保存された記録から取得されます。テスト ケースの各テスト ステップは、HTTP 要求を表しています。



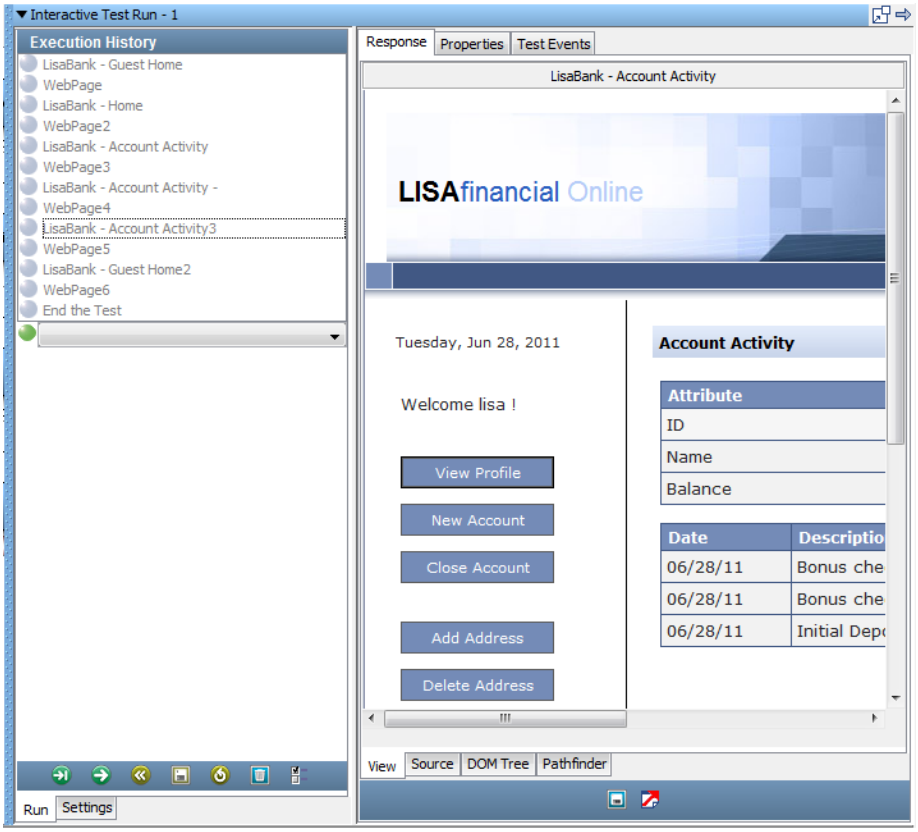
4. テスト ケースを保存します。

チュートリアル 6 - パート B - テスト ケースの実行

このセクションでは、パート A に続いて、対話型テスト ラン（ITR）ユーティリティで保存されたテスト ケースを実行し、結果を表示します。

次の手順に従ってください:

1. 新しい ITR セッションを開始します。
2. ［実行履歴］ペインで、  ［自動的にテストを実行］をクリックします。
3. 処理が完了したら、［OK］をクリックします。



ITR で当座預金への預け入れが再生されると、［表示］タブに表示されたページが表示されます。

［ソース］タブには、ステップでキャプチャされたページの HTML コードが表示されます。

DevTest はブラウザとして動作し、Web サーバに同じ HTTP 要求を送信します。

4. ITR ユーティリティを閉じます。

チュートリアル 6 - パート C - 要求テスト ステップの変更

Web レコーダは、各要求に対して HTTP/HTML 要求テスト ステップを作成します。

これらのテスト ステップは、DevTest のその他のテスト ステップと同じように編集および変更できます。 レコーダは、記録中に入力したパラメータを、要求の Post パラメータおよび Get パラメータの値として使用します。

LISA Bank テストを一般化するために、これらのハードコードされた説明や預け入れ金額の値（「cash」や「1000.00」など）をデータ セットから取得したプロパティに置き換えます。 データ セットについては、「チュートリアル 2 - データ セット」で操作しました。

レコーディング結果の LISA Bank5 テスト ステップでは、［ホスト名］および［ポート］パラメータがパラメータ化され、設定に追加されます。 説明および金額の値はハードコードされています。

チュートリアルのこのパートでは、数値カウント データ セットを使用して、さまざまな金額を預け入れられるようにテスト ケースをパラメータ化します。 その後、テスト ケースを実行するときには、記録されたものとは別の預け入れ金額を使用します。

手順 1 - テスト ケースのコピー

次の手順に従ってください:

1. モデル エディタで tutorial6a テスト ケースが開いていることを確認します。
2. メニュー バーから［ファイル］-［名前を付けて保存］を選択します。
3. ［ファイル名］フィールドに「tutorial6c」と入力します。
4. ［保存］をクリックします。

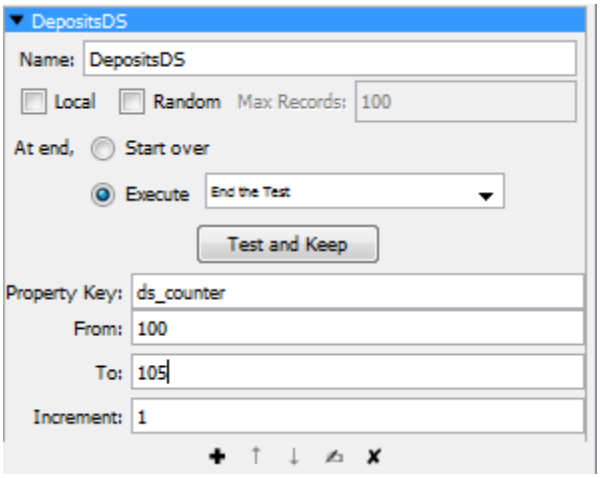
手順 2 - データ セットの追加

以下の手順では、数値カウント データ セットを追加します。 このタイプのデータ セットを使用すると、プロパティに数値を割り当てることができます。 データ セットを使用するたびに、固定値によって数値を変更できます。

次の手順に従ってください:

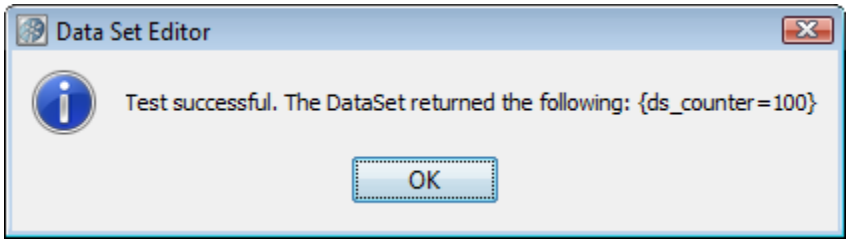
1. モデル エディタで、最初のテスト ステップを選択します。
2. 右ペインで、データ セット ステップのタブをダブルクリックします。
3. データ セット エレメントの下にある ✚ ［追加］をクリックします。
4. ［共通データセット］リストから、［数値カウント データ セットの作成］を選択します。

   右ペインでデータ セット エディタが開きます。
5. 以下の値を入力します。
   a. ［名前］フィールドに「DepositsDS」と入力します。
   b. ［終了時の処理］フィールドで、［実行］オプションを選択し、リストから［テストを終了する］を選択します。
   c. ［プロパティ キー］フィールドに「ds_counter」と入力します。
   d. ［開始］フィールドに「100」と入力します。
   e. ［終了］フィールドに「105」と入力します。
   f. ［増分］フィールドに「1」と入力します。



6. ［テスト アンド キープ］ボタンをクリックして、データ セットをテストします。

   成功メッセージが表示され、データ セットの最初の行のデータが表示されます。



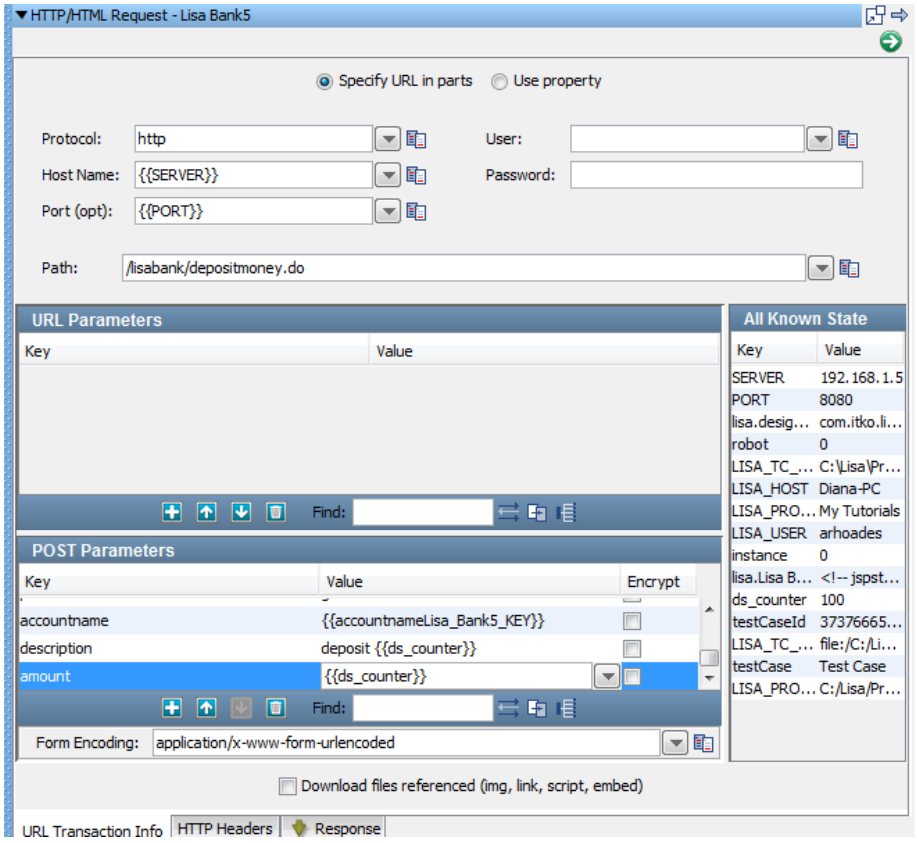
7. ［OK］をクリックします。

手順 3 - 記録された預け入れの POST パラメータの変更

ここでは、ds_counter プロパティ（データ セットで作成済み）を使用して、さまざまな預け入れ金額の値を指定します。

次の手順に従ってください：

1. モデル エディタで、LISA Bank5 ステップをダブルクリックします。
2. ［POST パラメータ］領域で、description キーの値を **deposit {{ds_counter}}** に変更します。
3. amount キーの値を **{{ds_counter}}** に変更します。

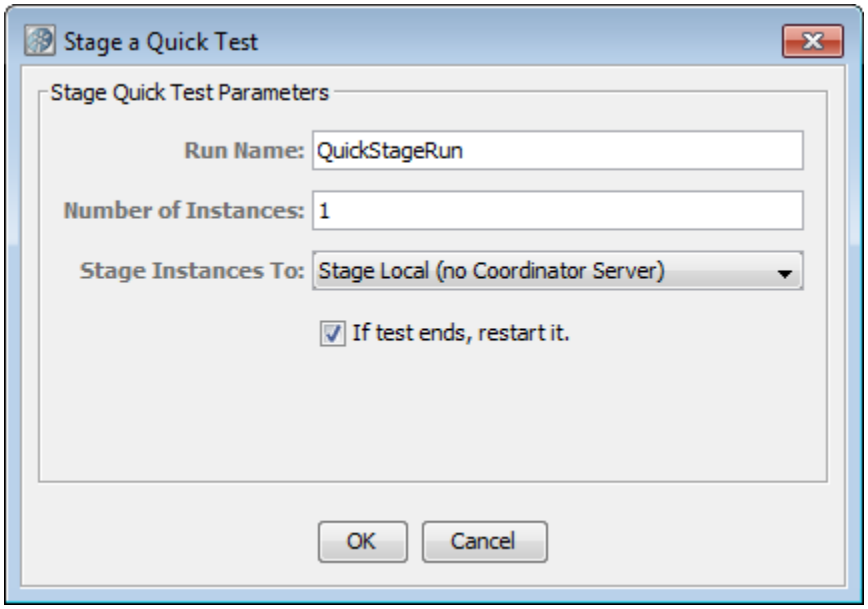


4. テスト ケースを保存します。

手順 4 - テスト ケースのステージング

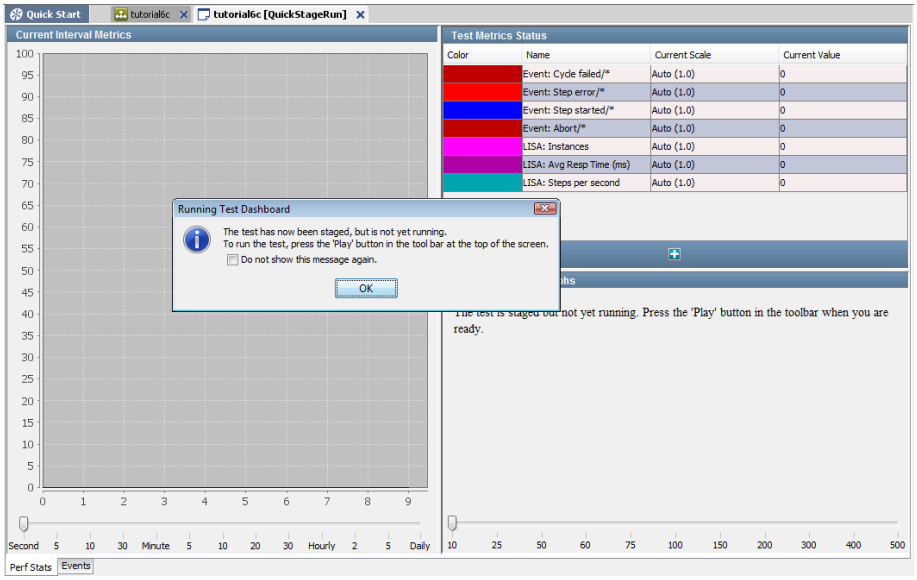
クイック テストをステージング(実行)する方法

1. ツールバーの  ［クイック テストのステージング］をクリックします。
2. ［クイック テストのステージング］ウィンドウで、［テストが終了したら再起動する］が選択されていることを確認します。



3. ［OK］をクリックします。

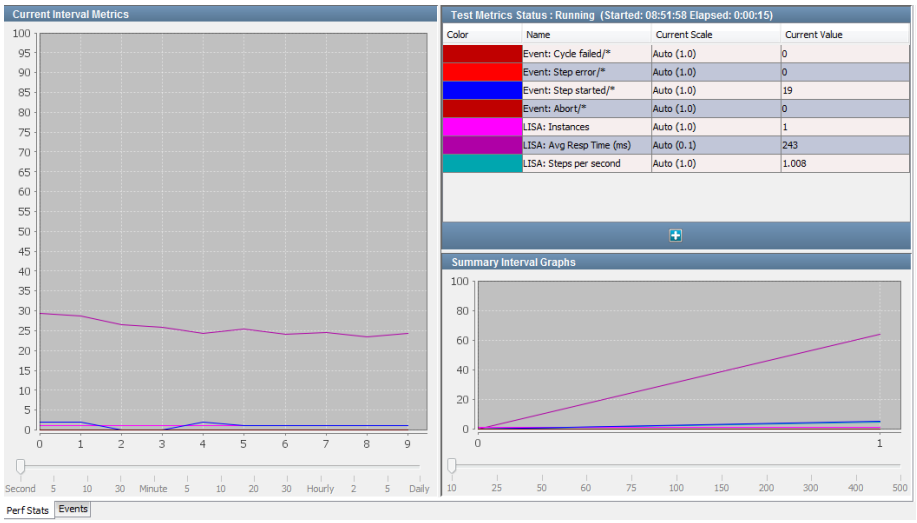
テスト モニタが開きますが、テストはまだ開始されていません。



4. ［OK］をクリックします。

5. ツールバーの  ［再生］をクリックします。

折れ線グラフでテストの進捗状況が表示されます。



手順 5 - LISA Bank での新しい預け入れの表示

次の手順に従ってください：

1. ユーザ「lisa_simpson」およびパスワード「*golisa*」を使用して、LISA Bank アプリケーションに再度ログインします。
2. 預け入れ金額を表示するには、当座預金の口座番号のリンクをクリックします。

   金額が 100 から始まって 105 に達するまで 1 ずつ増加していることに注目します。

チュートリアル 6 - レビュー

このチュートリアルでは、数値カウント データ セットを使用して、記録されたテストに入力を提供しました。

以下のことを行いました。

- テスト ケースをコピーし、数値カウント データ セットを追加しました。
- 記録された預け入れの POST パラメータを変更しました。
- クイック テストをステージングしました。

チュートリアル 7 - Enterprise JavaBean（EJB）のテスト

LISA Bank アプリケーションには、EJB の完全なセットが用意されており、Java インターフェースから口座の操作やユーザおよび口座情報の取得などを実行できます。

このチュートリアルでは、Enterprise JavaBean 実行テスト ステップを使用して、テスト ケースから EJB メソッドをコールし、アサーションで応答をテストします。 また、簡単な EJB をテストして、addUser および deleteUser メソッドが想定したように動作することを確認します。

チュートリアルのタスク

このチュートリアルでは、以下のことを行います。

- Enterprise JavaBean 実行ステップを使用します。
- 複合オブジェクト エディタで EJB オブジェクトを使用します。

前提条件

- 「チュートリアル 6 - Web サイトのテスト」を完了している。
- DevTest ワークステーション が開いている。
- デモ サーバへのアクセス権がある。

手順 1 - テスト ケースの作成

次の手順に従ってください:

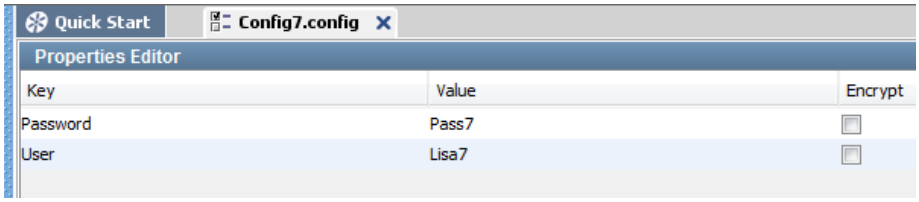
1. プロジェクト ペインで Tests フォルダを右クリックし、［新規テスト ケースの作成］を選択します。
2. ファイル名を「tutorial7」に設定します。
3. ［保存］をクリックします。

手順 2 - 設定の作成

設定については、「チュートリアル 2 - データ セット」で操作しました。

次の手順に従ってください:

1. project.config ファイルを開きます。
2. 設定に［ユーザ］および［パスワード］プロパティがない場合は、これらのプロパティを追加します。 値を設定する必要はありません。
3. config7 という名前の設定を作成します。
4. 設定 config7 に User プロパティを追加し、値を「**Lisa7**」に設定します。
5. 設定 config7 に Password プロパティを追加し、値を「Pass7」に設定します。



6. ［保存］をクリックします。
7. プロジェクト ペインで、設定 config7 を右クリックし、［アクティブ化］を選択します。

   設定が紫で表示されるようになります。

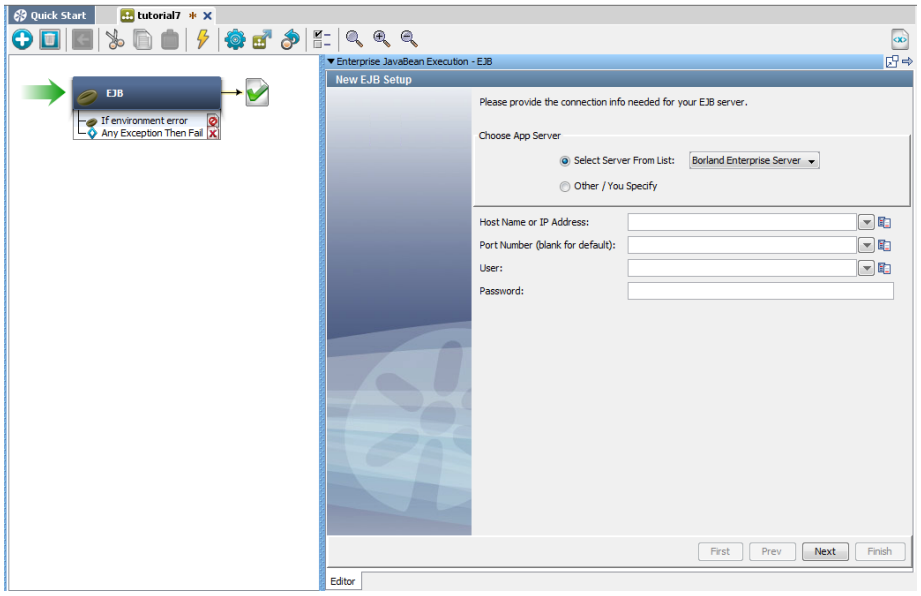
手順 3 - EJB テスト ステップの追加

Enterprise JavaBean 実行テスト ステップでは、実行中の EJB にコールを実行することができます。

次の手順に従ってください:

1. ［tutorial7］タブをクリックします。
2.  ［ステップの追加］をクリックします。
3. ［Java/J2EE］を選択し、［Enterprise JavaBean 実行］を選択します。

   新規 EJB セットアップ ウィザードが表示されます。



手順 4 - サーバへの接続

新規 EJB セットアップ ウィザードから、EJB サーバへの接続情報を指定するように求められます。

次の手順に従ってください:

1. ［リストからサーバを選択］ドロップダウン リストから、［JBoss 3.2/4.0］を選択します。
2. ［ホスト名または IP アドレス］フィールドに「**localhost**」と入力します。
3. ［次へ］をクリックします。

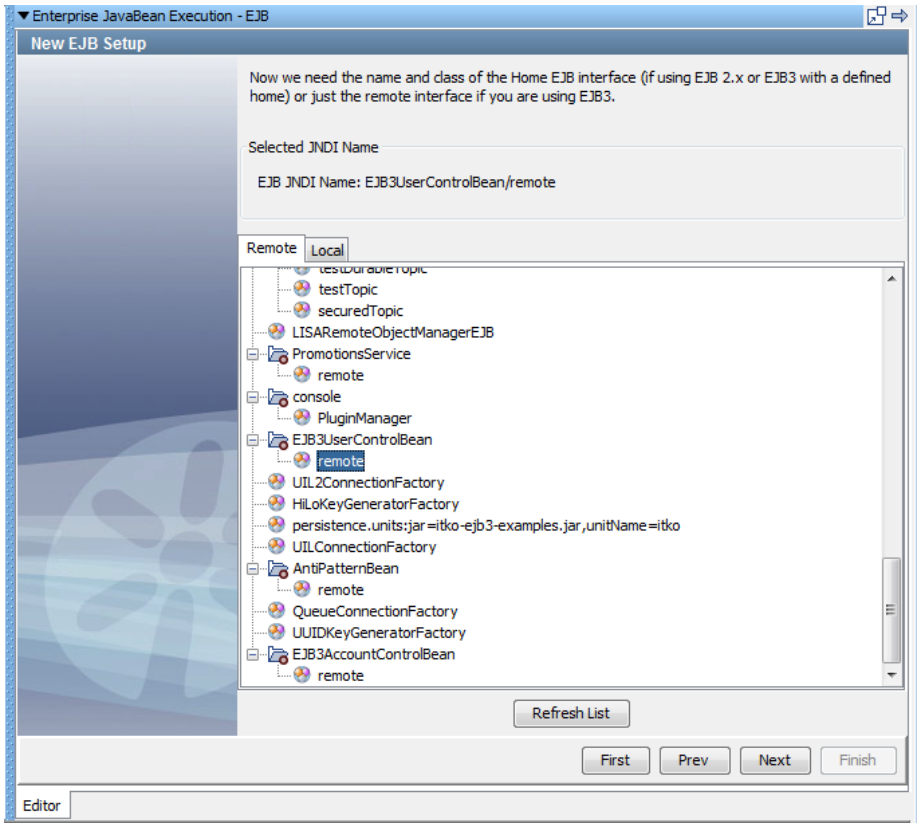
   JNDI 名のリストは EJB サーバから取得されます。

手順 5 - EJB インターフェースの指定

新規 EJB セットアップ ウィザードから、EJB インターフェースの名前を指定するように求められます。

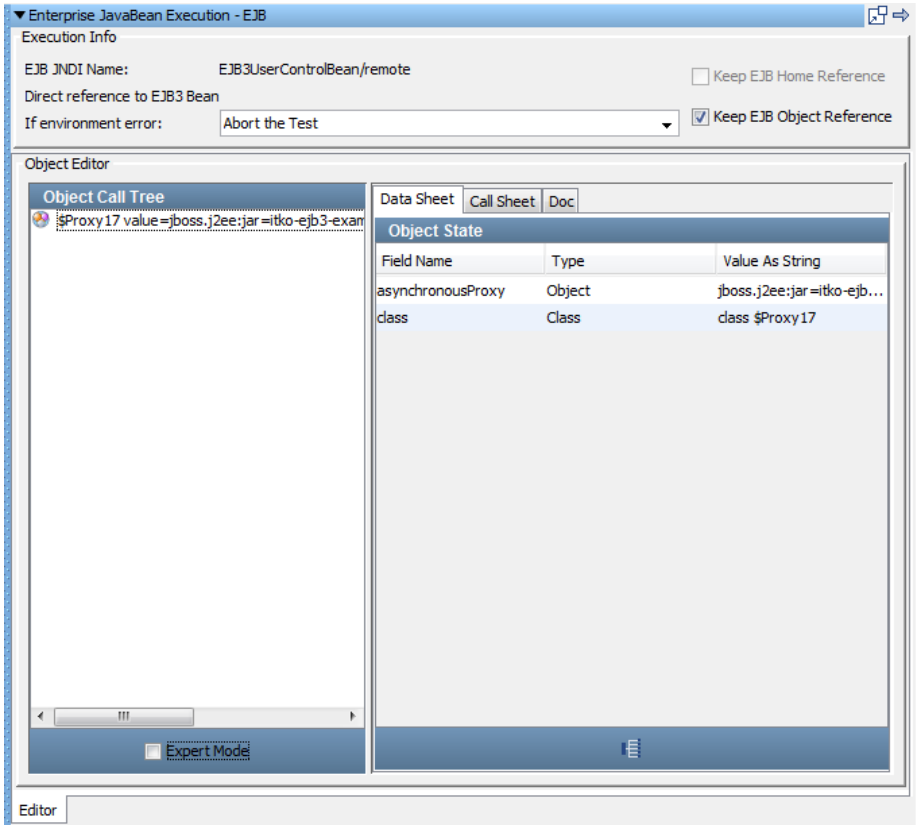
次の手順に従ってください:

1. ［リモート］タブで、［EJB3UserControlBean/remote］を選択します。



2. ［次へ］をクリックします。

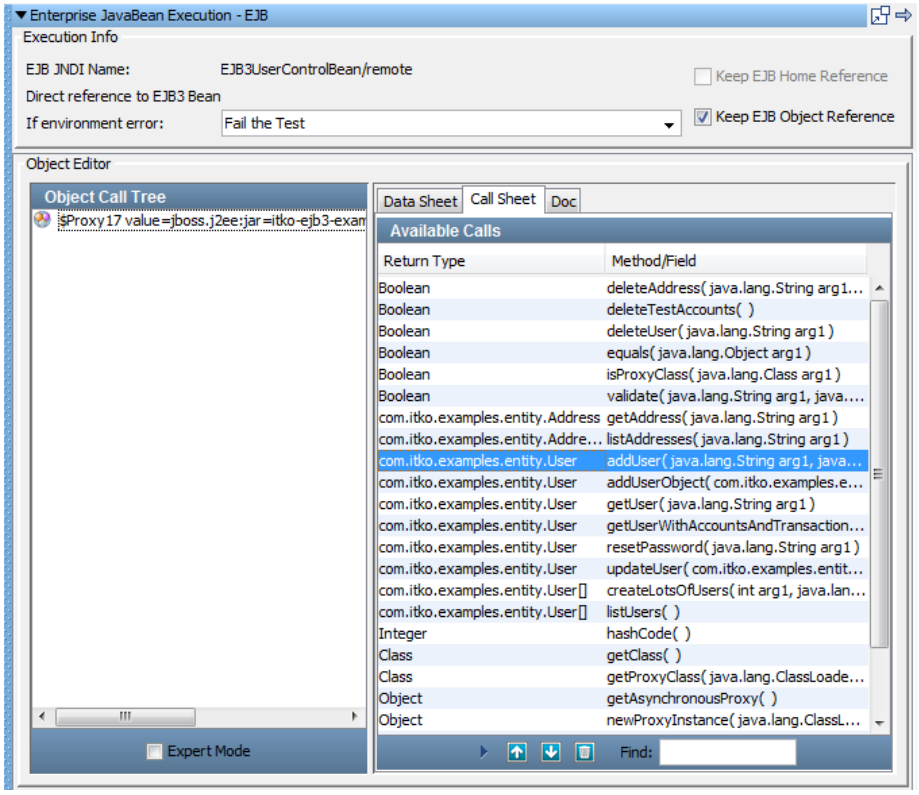
   複合オブジェクト エディタが開きます。


Enterprise JavaBean Execution - EJB
Execution Info
EJB JNDI Name:
EJB3UserControlBean/remote
Keep EJB Home Reference
Direct reference to EJB3 Bean
If environment error:
Abort the Test
Keep EJB Object Reference
Object Editor
Object Call Tree
$Proxy17 value=jboss.j2ee:jar=itko-ejb3-exam
Expert Mode
Data Sheet
Call Sheet
Doc
Object State
Field Name
Type
Value As String
asynchronousProxy
Object
jboss.j2ee:jar=itko-ejb...
class
Class
class $Proxy17
Editor

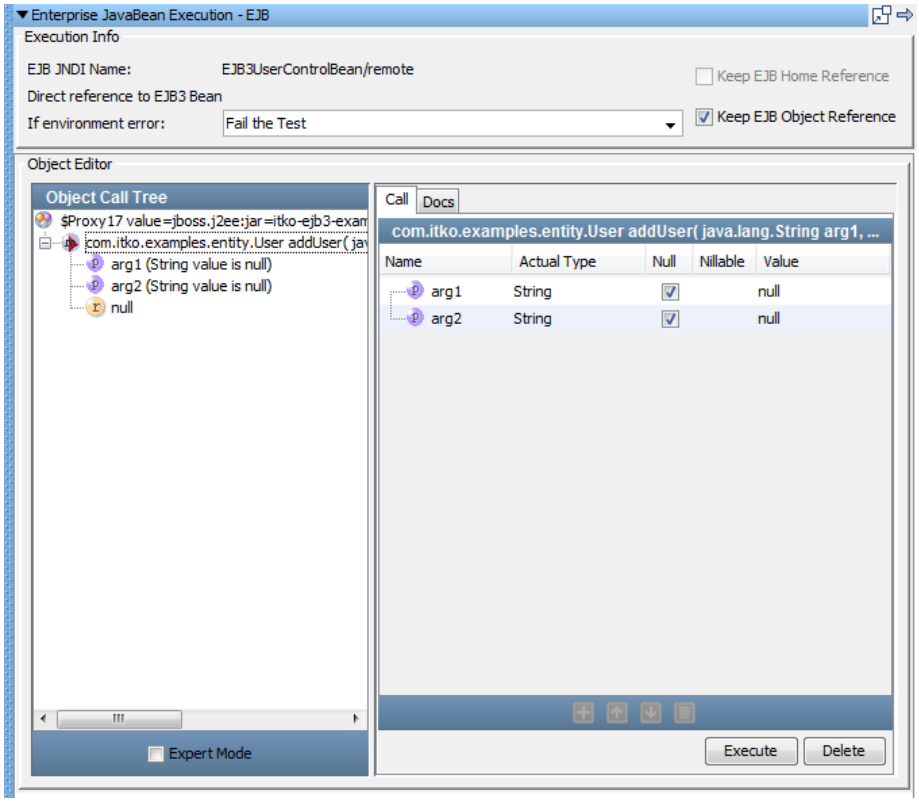
手順 6 - EJB の設定

次の手順に従ってください：

1. 同じ EJB オブジェクトを繰り返し使用する場合、［EJB オブジェクト参照の保持］チェック ボックスがオンになっていないときは、オンにします。
2. ［環境エラーの場合］フィールドに、この EJB ステップの実行中に例外が発生した場合に実行するステップを設定します。 リストから［テストを失敗させる］を選択します。
3. ［オブジェクト エディタ］領域で、［コール シート］タブを選択し、addUser メソッドを選択します。
4.  ［選択されたメソッドをオブジェクト コール ツリーに追加］をクリックします。



オブジェクト コール ツリーに addUser メソッドが表示されます。

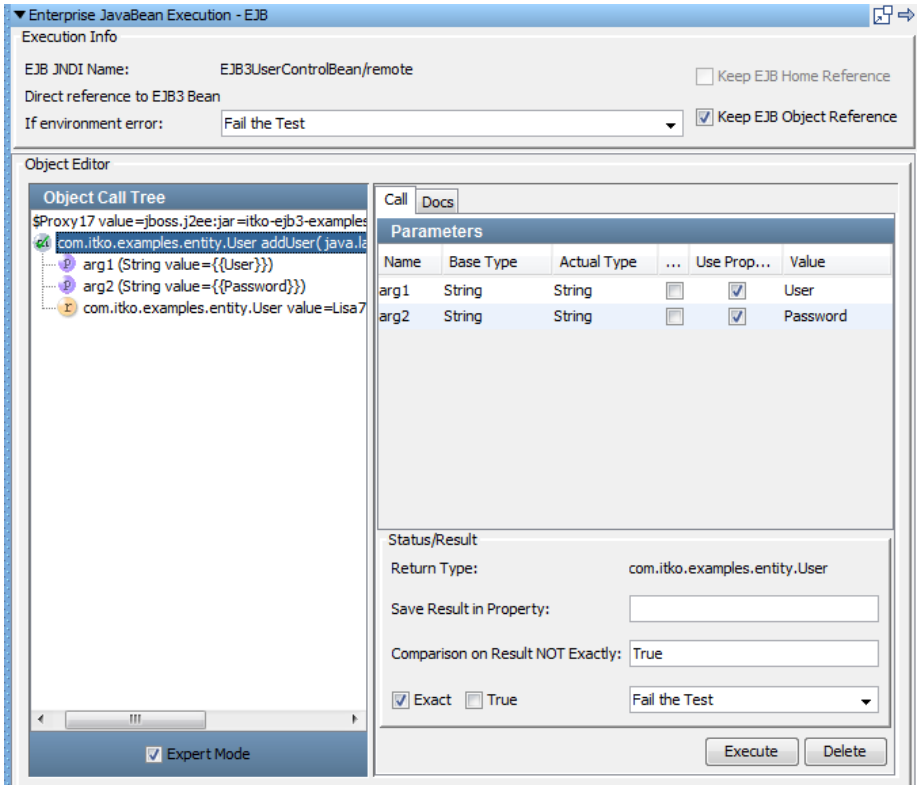


重要：User および Password は一度しか追加できません。 このチュートリアルを 2 回以上実行するには、User および Password に設定されている値を変更してください。

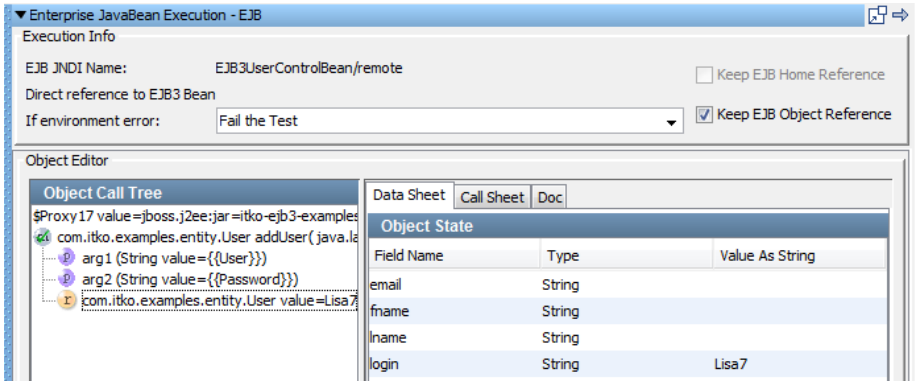
メソッド パラメータを入力し、インライン アサーションを追加する方法

1. ［オブジェクト コール ツリー］ペインで、［エキスパート モード］が有効になっていない場合は選択します。
2. 各引数の［プロパティの使用］チェック ボックスを選択します。

   ［プロパティの使用］は［パラメータ］領域の列見出しです。
3. arg1 の［値］列で、プロパティの［デフォルト］リストから［**ユーザ**］を選択します。
4. arg2 の［値］列で、プロパティの［デフォルト］リストから［**パスワード**］を選択します。
5. ［ステータス/結果］領域で、［完全］を選択して［True］チェック ボックスをオフにして、インライン アサーションを追加します。
6. ［結果の比較 - NOT Exactly］フィールドに「**True**」と入力します。
7. ［完全］ドロップダウンから、［テストを失敗させる］を選択します。
8. ［実行］をクリックします。

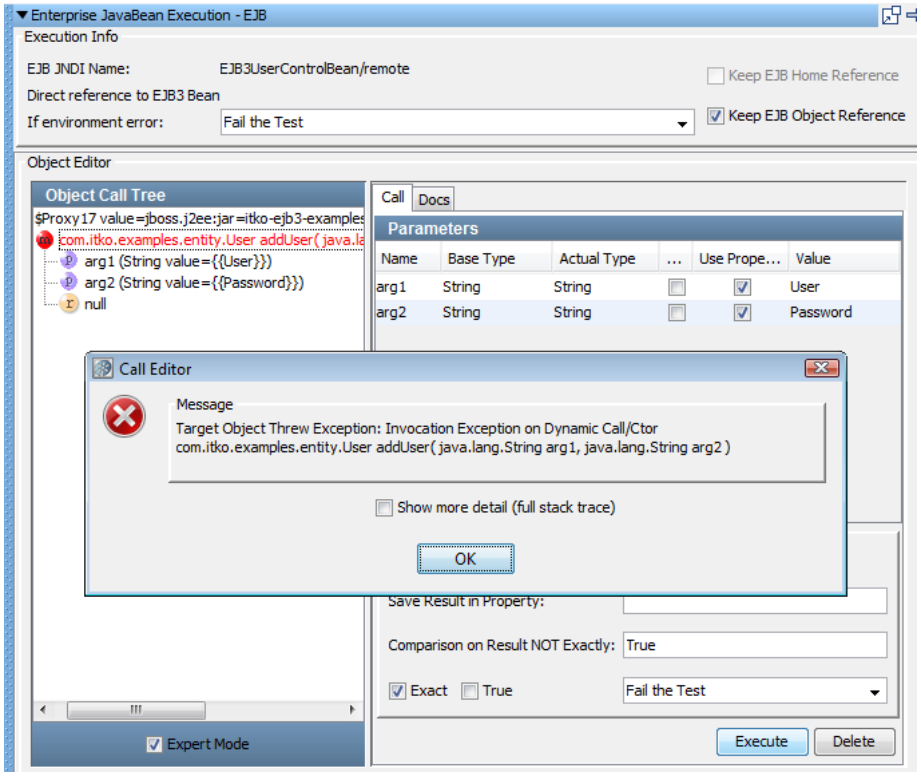


メソッドのパラメータがオブジェクト コール ツリーの［入力パラメータ］ アイコンの隣に表示されます。 このメソッドの戻り値は、オブジェクト コール ツリーの User フィールドの値（Lisa7）です。



9. ［実行］をクリックして addUser メソッドを再度テストします。

10. 戻り値 が null に変わり、すでにユーザが追加されているためエラーが表示されます。

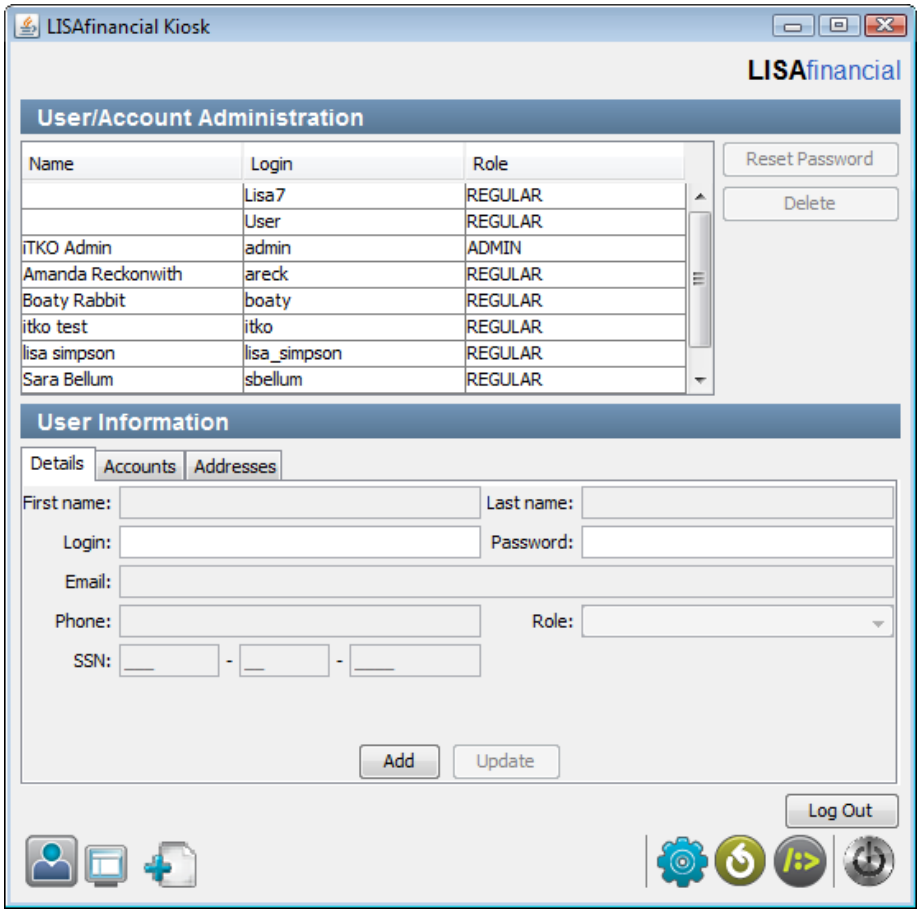


手順 8 - メソッドの実行の確認

LISA Bank アプリケーションから、ユーザが追加されたことを確認できます。

次の手順に従ってください：

1. LISA Bank アプリケーションに移動します。
2. ユーザ「**admin**」およびパスワード「**admin**」を使用してログインします。
3. Lisa7 が追加されたことを確認するには、ユーザのリストを表示します。

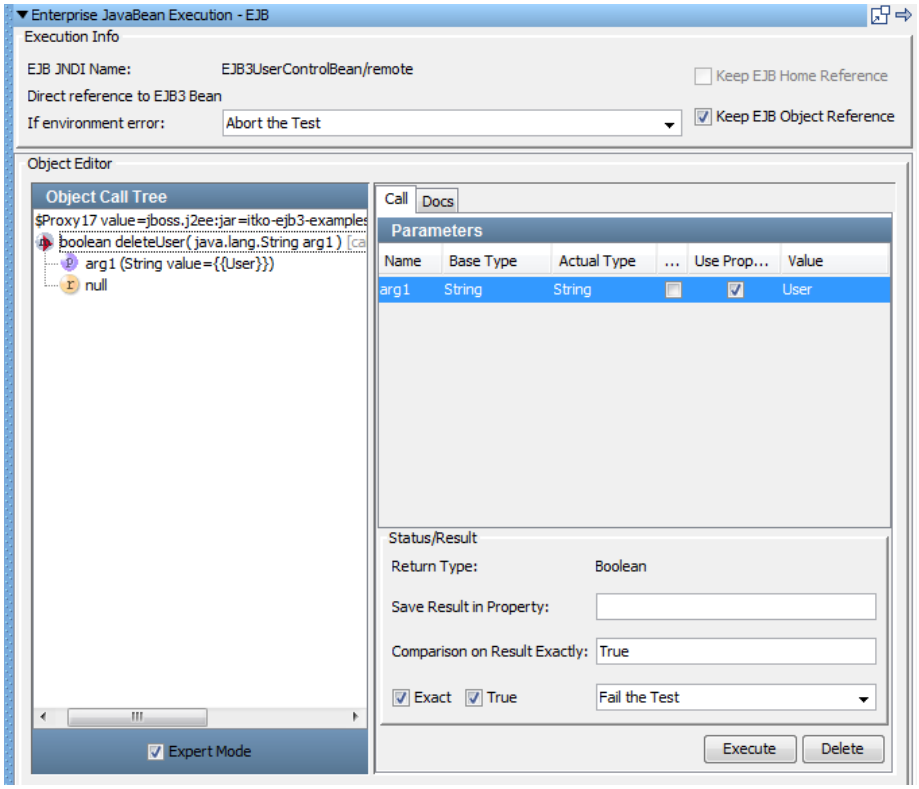


手順 9 - 別の EJB テスト ステップの追加

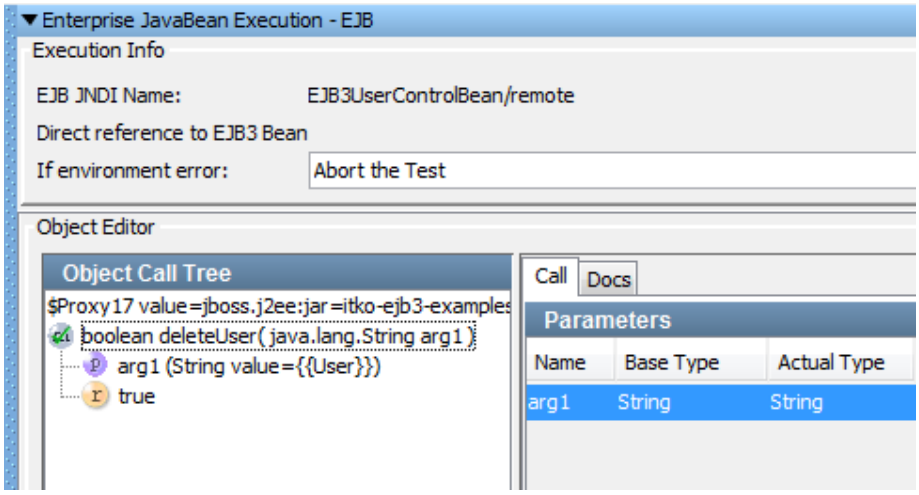
ここでは、上記の手順を再度実行して deleteUser メソッドを呼び出します。

次の手順に従ってください:

1. 手順 3 からチュートリアルを繰り返して、DeleteUser という名前の EJB ステップを追加します。
2. メソッド パラメータ プロパティ User を使用します。



3. ［実行］をクリックしてこのメソッドを実行し、結果を取得します。



戻り値  は true で、ユーザが削除されたことを意味します。

4. ［保存］をクリックします。

チュートリアル 7 - レビュー

このチュートリアルでは、以下のことを行いました。

- 2 つの EJB テスト ステップで構成されるテスト ケースを作成しました。

  デモ サーバのサンプル アプリケーションから EJB オブジェクトをロードしました。
- EJB テスト ステップを作成し、EJB をロードしました。
- 複合オブジェクト エディタを使用して EJB オブジェクトを操作しました。

チュートリアル 8 - Web サービスのテスト

このチュートリアルでは、Web サービス実行（XML）テスト ステップを使用して、テスト ケースで Web サービス操作をコールします。 その後、要求と応答をテストします。 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。

チュートリアルのタスク

このチュートリアルでは、以下のことを行います。

- Web サービス実行（XML）テスト ステップの追加
- Web サービス操作の実行

前提条件

- チュートリアル 5 を完了している。
- DevTest ワークステーション が開いている。
- デモ サーバへのアクセス権がある。

手順 1 - テスト ケースの作成

次の手順に従ってください:

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］を選択します。
2. ファイル名を「**tutorial8**」に設定します。
3. ［保存］をクリックします。
4. プロジェクト パネルで、project.config を右クリックし、［アクティブ化］を選択します。

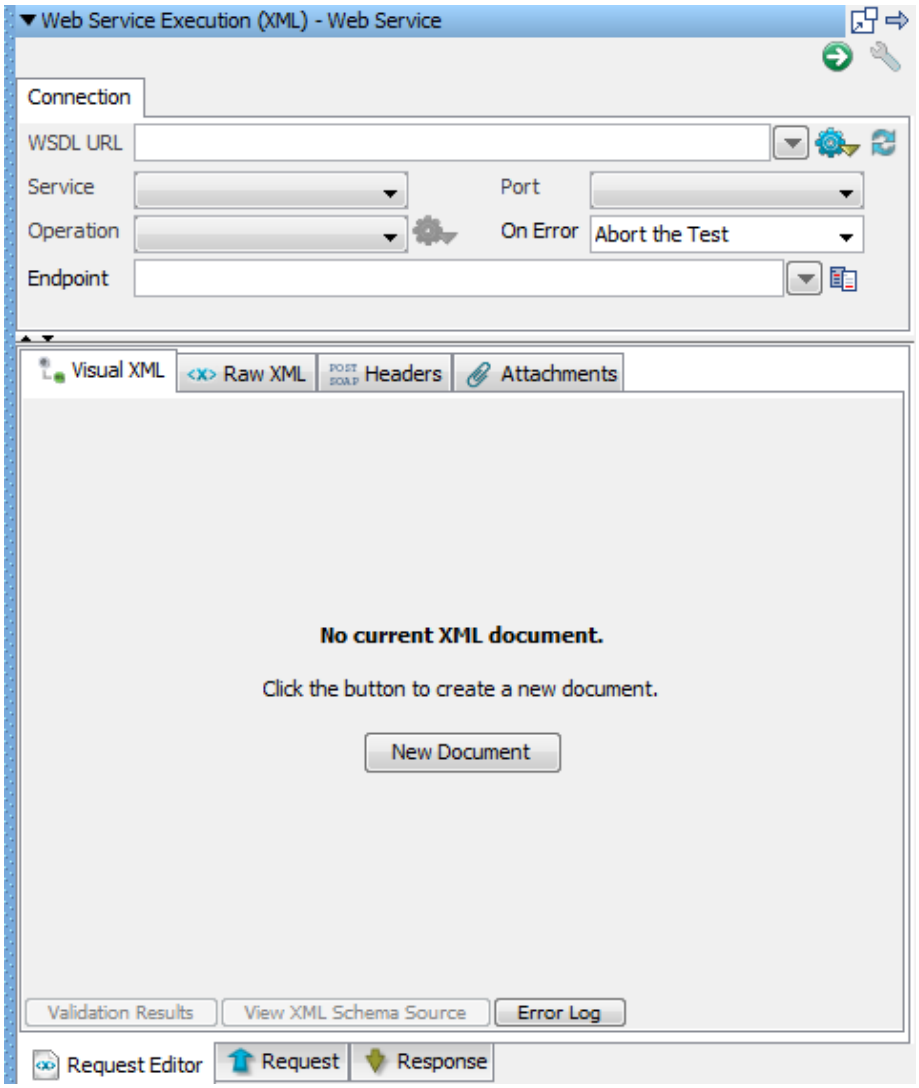
手順 2 - Web サービス実行（XML）テスト ステップの追加

Web サービス実行（XML）テスト ステップでは、SOAP ベースの Web サービスで処理を実行することができます。

次の手順に従ってください：

1. ［tutorial8a］タブをクリックします。
2.  ［ステップの追加］をクリックします。
3. ［Web/Web サービス］を選択し、［Web サービス実行（XML）］を選択します。

   Web サービス ステップがモデル エディタに追加されます。
4. Web サービス実行（XML）エディタを開くには、Web サービス ステップをダブルクリックします。



5. ［新規ドキュメント］をクリックします。

手順 3 - Web サービス クライアントの作成

ここでは、コールされる操作を指定し、操作に送られる SOAP メッセージを作成します。

次の手順に従ってください:

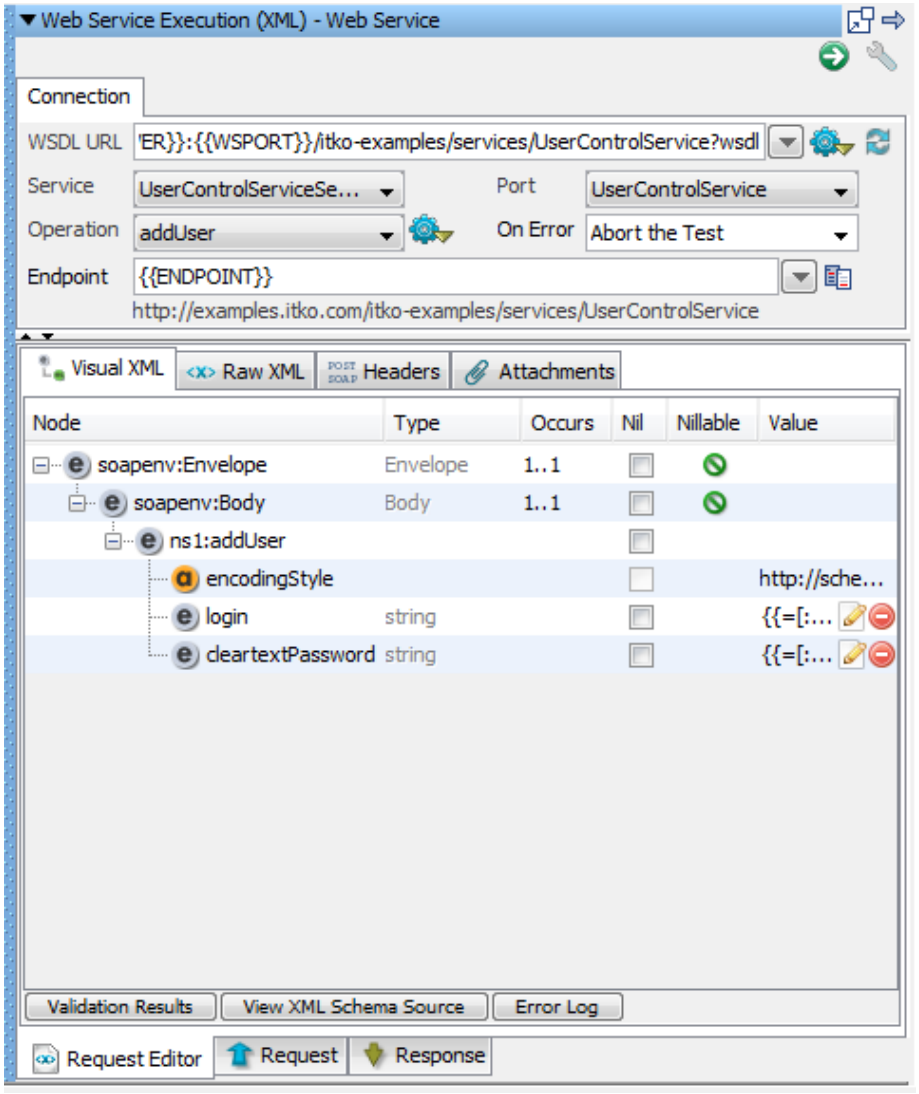
1. ［WSDL URL］フィールドに以下の場所を入力します。

   注: WSSERVER および WSPORT プロパティは、サーバおよびポートを表します。

   ```
   http://localhost:8080/itko-examples/services/UserControlService
   ?wsdl
   ```

2. ［サービス］フィールドで、［UserControlServiceService］を選択します。
3. ［ポート］フィールドで、［UserControlService］を選択します。
4. ［操作］フィールドで、addUser 操作を選択します。
5. ［エラー時］フィールドで、［テストを中止する］を選択します。

   DevTest は、この条件を使用して、Web サービス クライアントを作成します。 ビジュアル XML エディタには、SOAP メッセージのグラフィカル ビューが表示されます。



6. テスト ケースを保存します。

手順 4 - Web サービス要求の実行

次の手順に従ってください:

1.  ［WS 要求の実行］をクリックします。

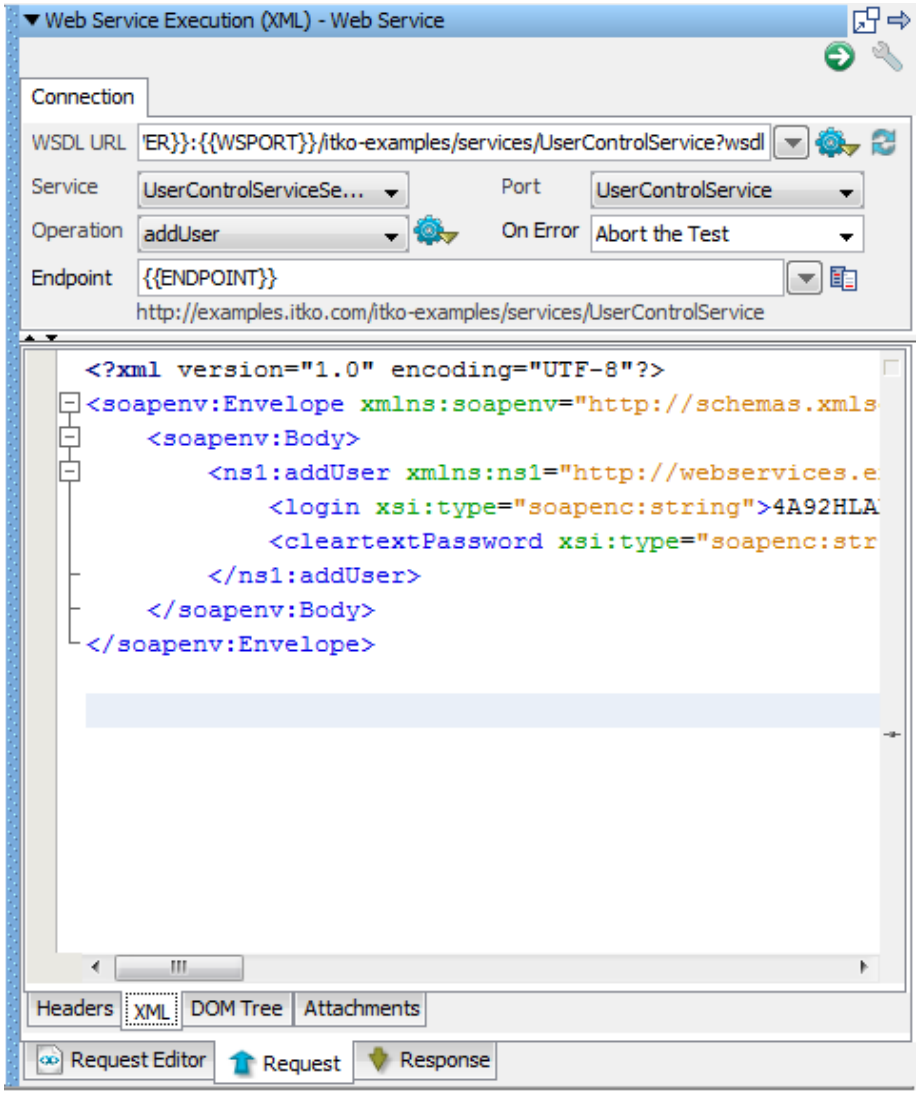
   テストが実行されます。

手順 5 - 要求および応答の表示

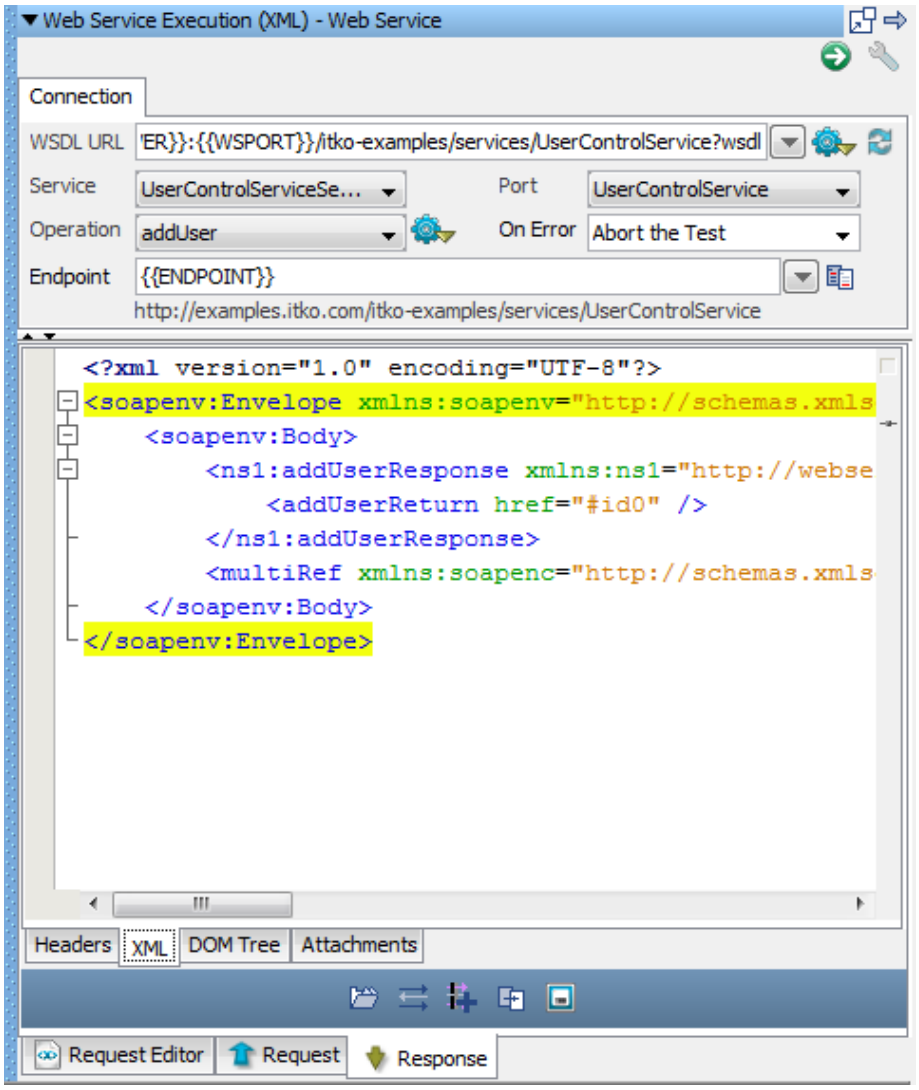
［要求］タブには、何らかの後処理（DevTest プロパティの置換など）の後に送信された結果としての要求データが表示されます。 ［応答］タブには、結果として受信した応答データが表示されます。

次の手順に従ってください:

1. 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。



2. 実行時の応答を表示するには、［応答］タブを選択します。



チュートリアル 8 - レビュー

このチュートリアルでは、以下のことを行いました。

- Web サービス実行（XML）テスト ステップでテスト ケースを作成しました。
- addUser 操作を実行しました。
- この操作の要求および応答を表示しました。

チュートリアル 9 - データベースの検査およびテスト

このチュートリアルでは、チュートリアル 5 で使用した Web アプリケーションの一部であるデータベース テーブルを検査およびテストします。

SQL データベース実行（JDBC）ステップを使用して、テスト ケースでデータベースと対話し、アサーションを使用して応答をテストします。 アプリケーションの一部である Derby データベースの Users テーブルを検査します。

チュートリアルのタスク

このチュートリアルでは、以下のことを行います。

- SQL データベース実行（JDBC）ステップの使用
- 設定へのアプリケーション プロパティの保存
- アサーションの追加および変更
- フィルタの追加

前提条件

- チュートリアル 5 を完了している。
- DevTest ワークステーション が開いている。
- デモ サーバへのアクセス権がある。

手順 1 - テスト ケースの作成

次の手順に従ってください:

1. プロジェクト ペインで Tests フォルダを右クリックし、［新規テスト ケースの作成］を選択します。
2. ファイル名を「tutorial9」に設定します。
3. ［保存］をクリックします。

手順 2 - 設定へのデータベース プロパティの追加

データベースへの接続に必要なプロパティを設定に保存します。 この方法は、DevTest の標準プラクティスであり、テスト ケースの移植性が高まります。

次の手順に従ってください:

1. project.config がアクティブな設定でない場合は、プロジェクト ペインで project.config を右クリックし、［アクティブ化］を選択します。
2. project.config 設定を開きます。
3. 以下のプロパティを追加します。

   DBDriver

   org.apache.derby.jdbc.ClientDriver

   DBConnect

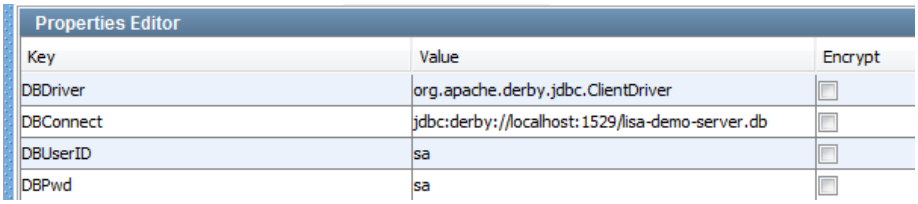
   jdbc:derby://localhost:1529/lisa-demo-server.db

   DBUserID

   sa

   DBPwd

   sa



4. ［保存］をクリックします。

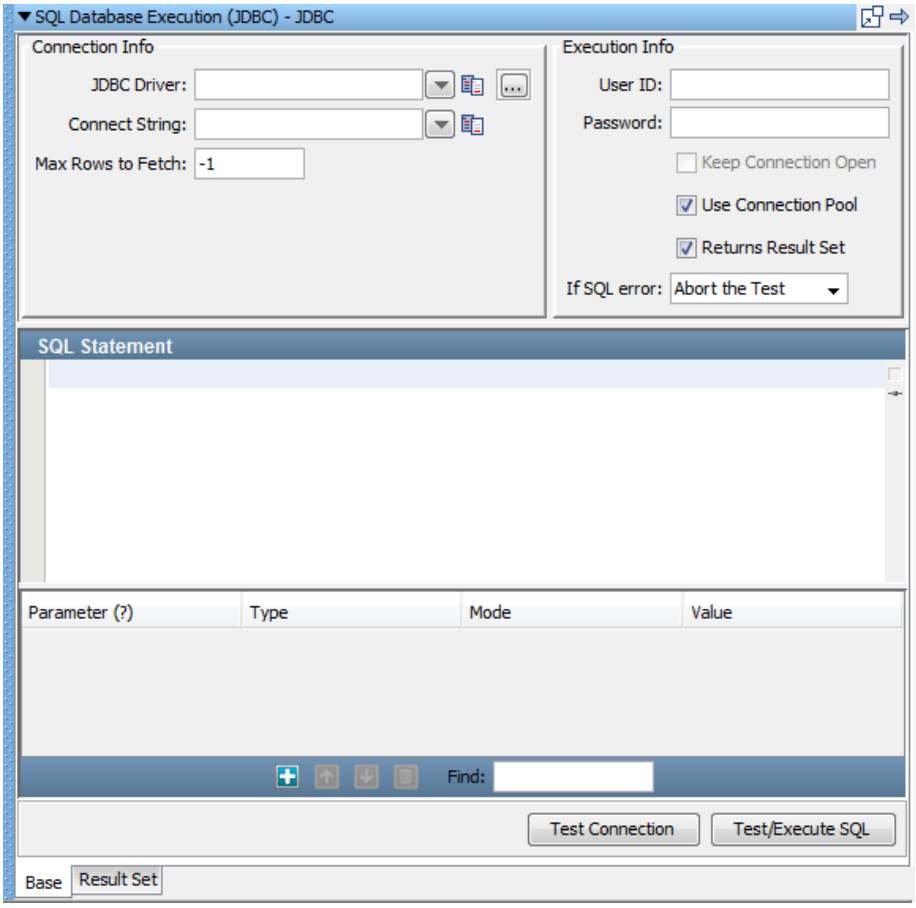
手順 3 - SQL データベース実行（JDBC）テスト ステップの追加

SQL データベース実行（JDBC）テスト ステップでは、JDBC を使用してデータベースに接続し、データベースに対して SQL クエリを実行することができます。

次の手順に従ってください:

1. ［tutorial9］タブをクリックします。
2.  ［ステップの追加］をクリックします。
3. ［他のトランザクション］を選択し、［SQL データベース実行（JDBC）］を選択します。

   JDBC がモデル エディタに追加されます。
4. ステップ エディタを開くには、JDBC ステップをダブルクリックします。



手順 4 - データベースへの接続

接続情報を指定するには、project.config 設定に追加したプロパティを使用します。

次の手順に従ってください:

1. ステップ エディタの［接続情報］領域および［実行情報］領域に以下の値を入力します。 パスワードを入力する際には、値がマスクされます。

   JDBC ドライバ

   {{DBDriver}}

   接続文字列

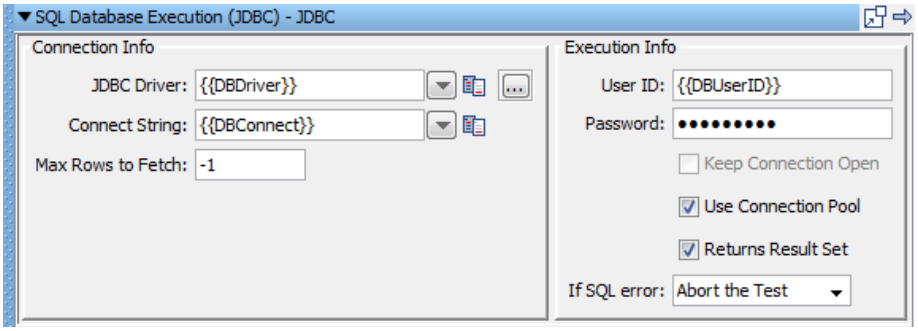
   {{DBConnect}}

   ユーザ ID

   {{DBUserID}}

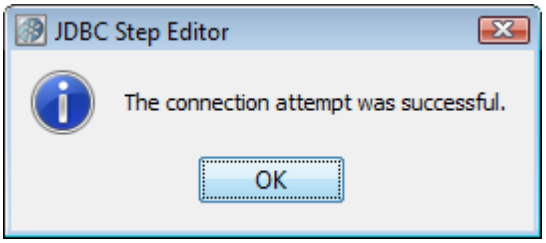
   パスワード

   {{DBPwd}}



2. ステップ エディタの下部にある［テスト接続］ボタンをクリックします。

   接続が有効であることを示すメッセージが表示されます。



3. ［OK］をクリックします。

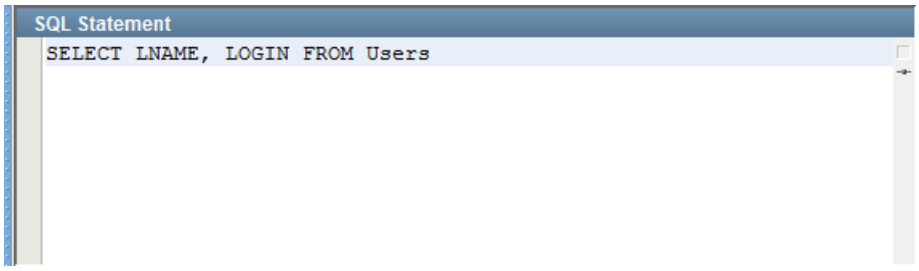
手順 5 - SQL クエリの実行

ここでは、Users テーブルからデータを取得する SQL ステートメントを指定して実行します。

次の手順に従ってください:

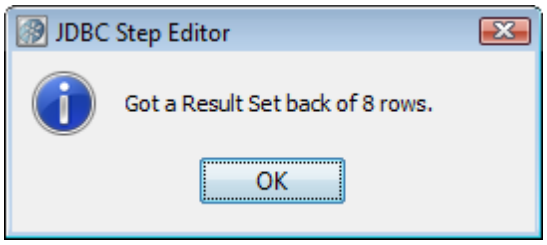
1. ［SQL ステートメント］ペインに以下のステートメントを入力します。

```
SELECT LNAME, LOGIN FROM Users
```



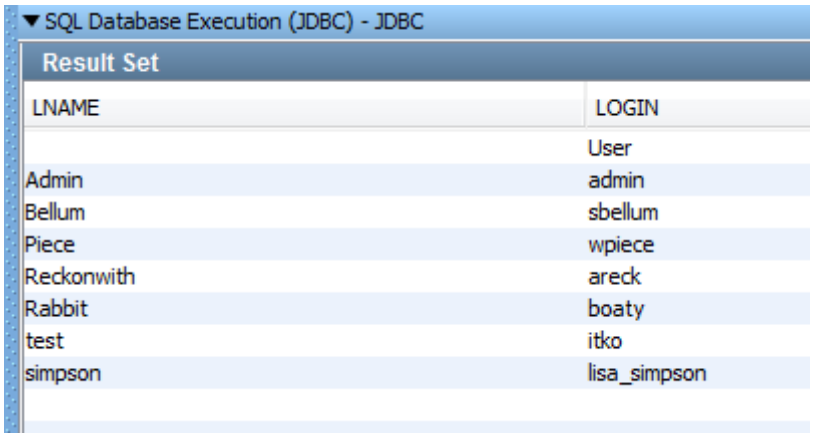
2. ステップ エディタの下部にある［SQL をテスト/実行］ボタンをクリックします。

   メッセージで有効なクエリであることが確認され、返された行の数が表示されます。



3. ［OK］をクリックします。

   ［結果セット］タブが表示されます。



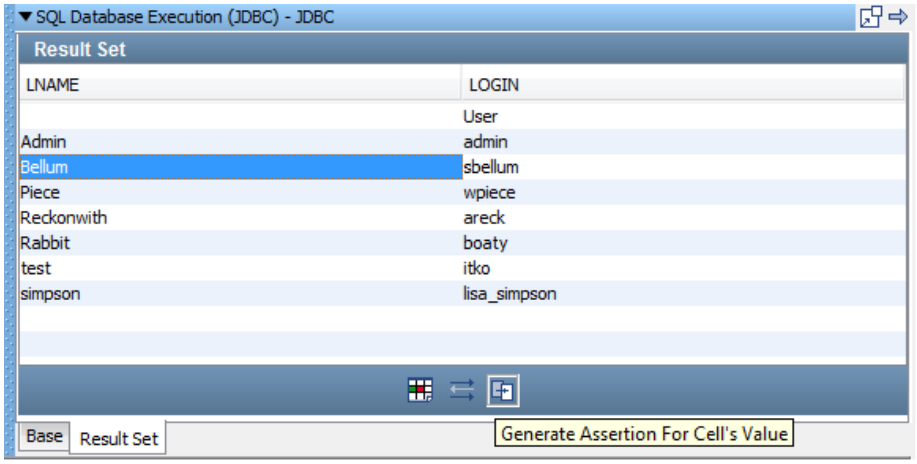
手順 6 - アサーションの追加

結果セットに特定の姓が存在することをテストするアサーションを追加します。

次の手順に従ってください:

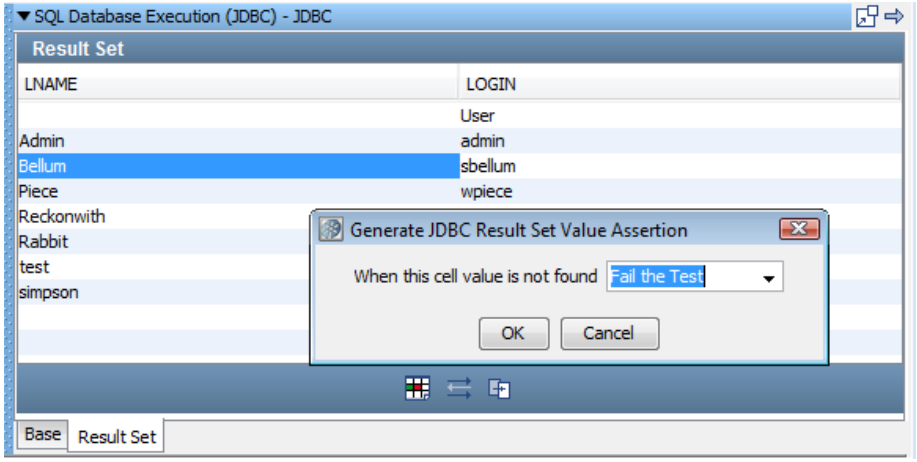
1. ［結果セット］タブで、［LNAME］列のセルを選択します。
2.  ［セルの値に対するアサーションの生成］をクリックします。

   ［JDBC 結果セット値アサーションの生成］ダイアログ ボックスが表示されます。



3. ドロップダウン リストから［テストを失敗させる］オプションを選択します。

   選択した姓が見つからない場合、テストは失敗します。



4. ［OK］をクリックします。

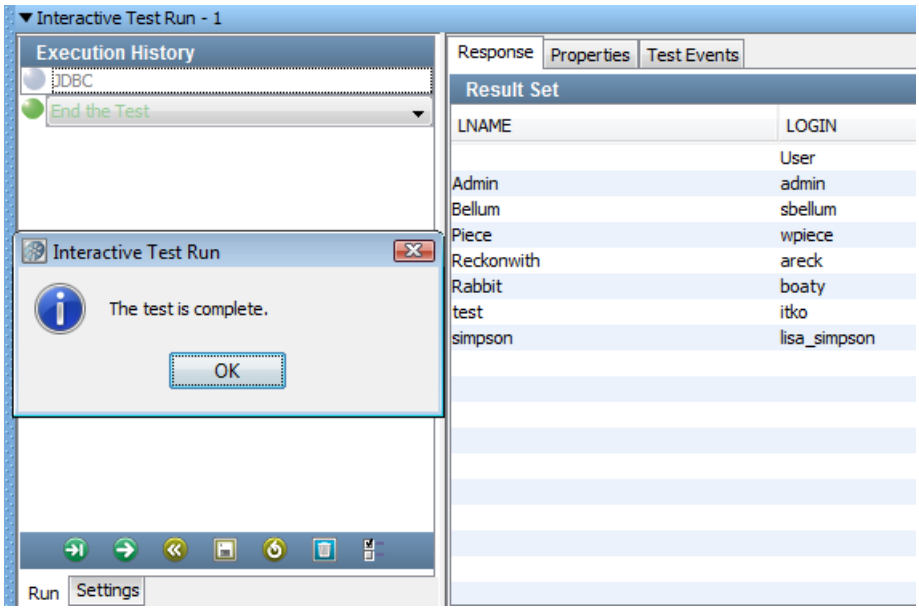
5. ［保存］をクリックします。

手順 7 - テスト ケースの実行

次の手順に従ってください:

1. ツールバーの  ［対話型テスト ラン（ITR）の開始］をクリックします。
2.  ［次のステップの実行］をクリックします。

   テストが正常に実行されます。 結果セットが［応答］タブに表示されます。



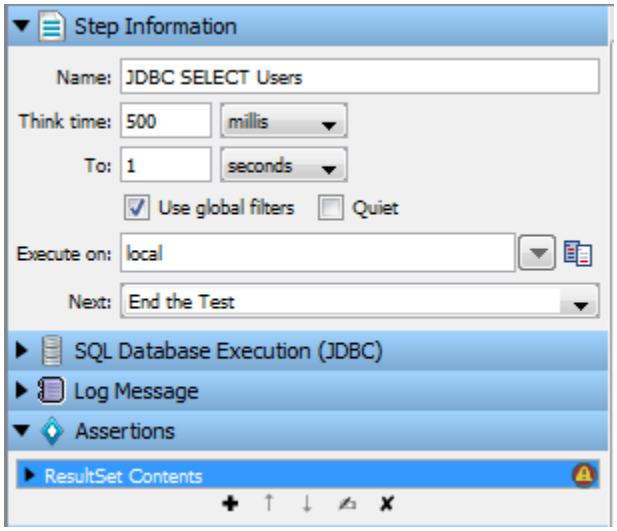
3. ITR トレイを隠します。

手順 8 - アサーションの変更

ここでは、テストが失敗するようアサーションを変更します。

次の手順に従ってください:

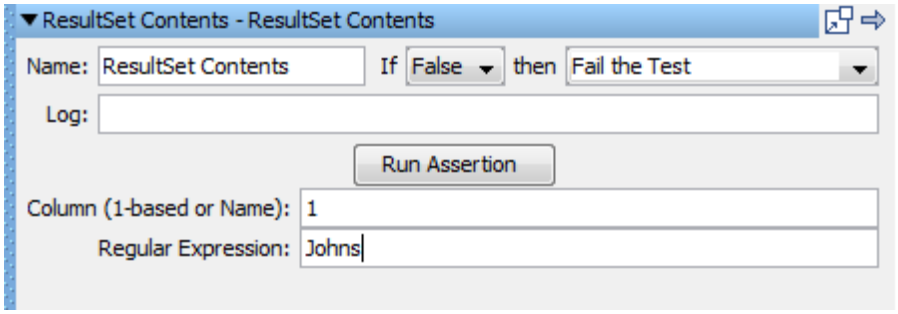
1. モデル エディタで、JDBC SELECT Users テスト ステップをクリックします。
2. エレメント ツリーの［アサーション］タブを開きます。



3. 先に作成した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。下の部分に、このアサーションでは、指定した値が結果セットの最初の列にあることを確認することが示されています。

4. ［正規表現］フィールドの値を「Johns」に変更します。



5. 新しい ITR を開始して、テスト ケースを再度実行します。

   このテストは失敗します。

6. ITR トレイを隠します。

手順 9 - フィルタの追加

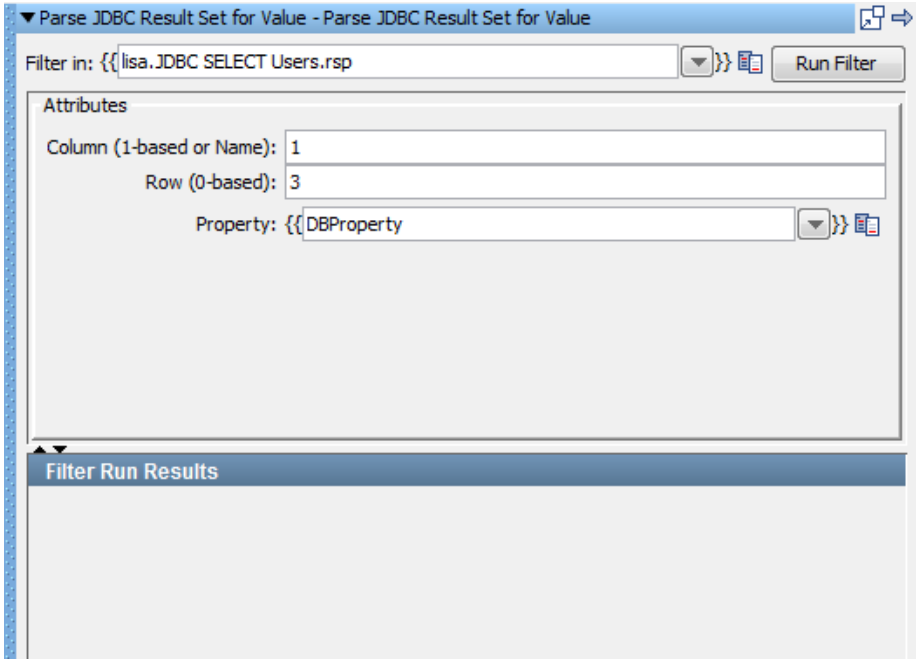
データベース フィルタを追加して、結果セットの最初の列および 4 番目の行の値を取得します。 値はプロパティに格納されます。

次の手順に従ってください:

1. モデル エディタで、JDBC SELECT Users テスト ステップを選択します。
2. エレメント ツリーの［フィルタ］タブを開きます。
3. ✚ ［追加］をクリックします。
4. ［データベース］フィルタ サブメニューから、［JDBC 結果セットから値を抽出］を選択します。

   フィルタ エディタが開きます。
5. ［列］フィールドに「*1*」と入力します。 または、実際の列名として「*LNAME*」と入力します。
6. ［行］フィールドに「*3*」と入力します。

   このフィールドはゼロ ベースです。 そのため、値 3 は 4 番目の行を表わします。
7. ［プロパティ］フィールドに「*DBProperty*」と入力します。



8. ［保存］をクリックします。

手順 10 - フィルタおよびアサーションのテスト

次の手順に従ってください:

1. 新しい ITR を開始して、テスト ケースを再度実行します。

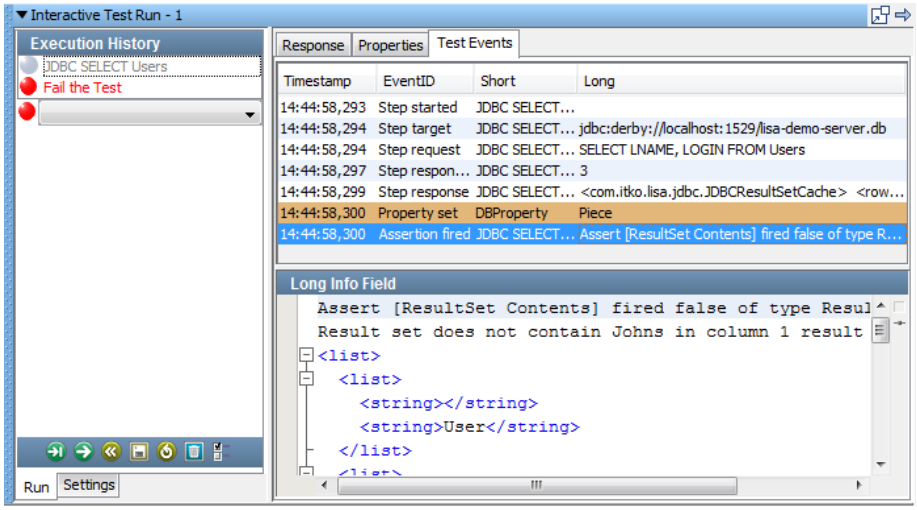
   結果セットで「Johns」が見つからないため、このテストは失敗します。

2. ［テスト イベント］タブをクリックします。
3. ［プロパティの設定］イベントをクリックします。

   DBProperty にフィルタによって指定された値が設定されていることに注目します。

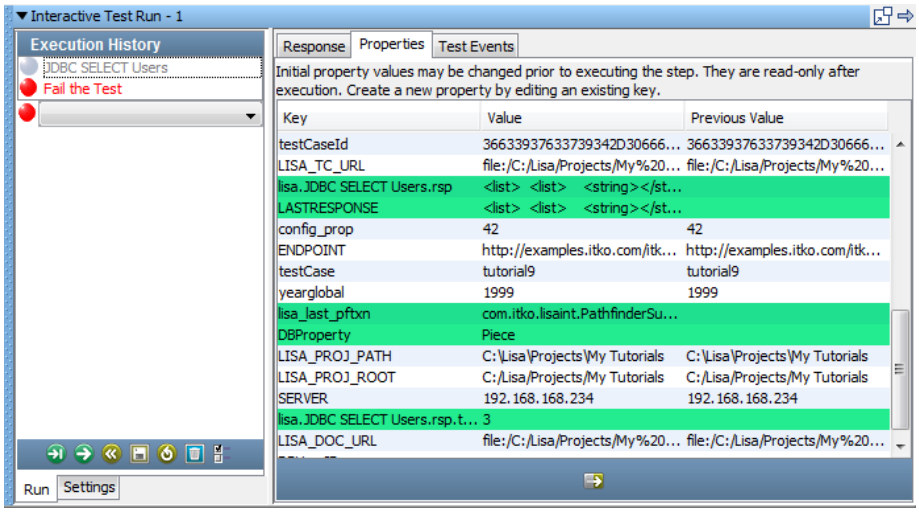
4. ［アサーションの起動］イベントをクリックします。

   ［詳細フィールド］領域に、結果セットの最初の列に「Johns」という値が含まれていなかったため、アサーションが起動したことが示されています。



5. ［プロパティ］タブをクリックします。
6. DBProperty の行を見つけて確認します。

図 3: チュートリアル 9 の ITR 結果のスクリーンショット



チュートリアル 9 - レビュー

このチュートリアルでは、テスト ケースを作成し、データベースに対してクエリを実行しました。 デモ サーバのアプリケーションに付属している Apache Derby データベースの Users テーブルを使用しました。また以下の方法を学習しました。

- データベースへの接続。
- データベースに対する SQL クエリの実行。
- アサーションおよびフィルタの追加。

チュートリアル 10 - クイック テストのステージング

このチュートリアルでは、クイック ステージング オプション（クイック テスト）を使用して、テストのステージング方法と結果のレポートを確認する方法について学習します。DevTest に付属している multi-tier-combo のサンプルのクイック テストを実行します。 クイック テストの実行は、テストをステージングするための最も簡単な方法です。

チュートリアルのタスク

このチュートリアルでは、以下のことを行います。

- multi-tier-combo テスト ケースの使用
- クイック テスト機能の使用
- レポートの選択と形式の設定

前提条件

- チュートリアル 5 ～ 9 を完了している。
- DevTest ワークステーション が開いている。
- デモ サーバへのアクセス権がある。

手順 1 - テスト ケースを開く

examples プロジェクトからテスト ケースを開きます。

次の手順に従ってください:

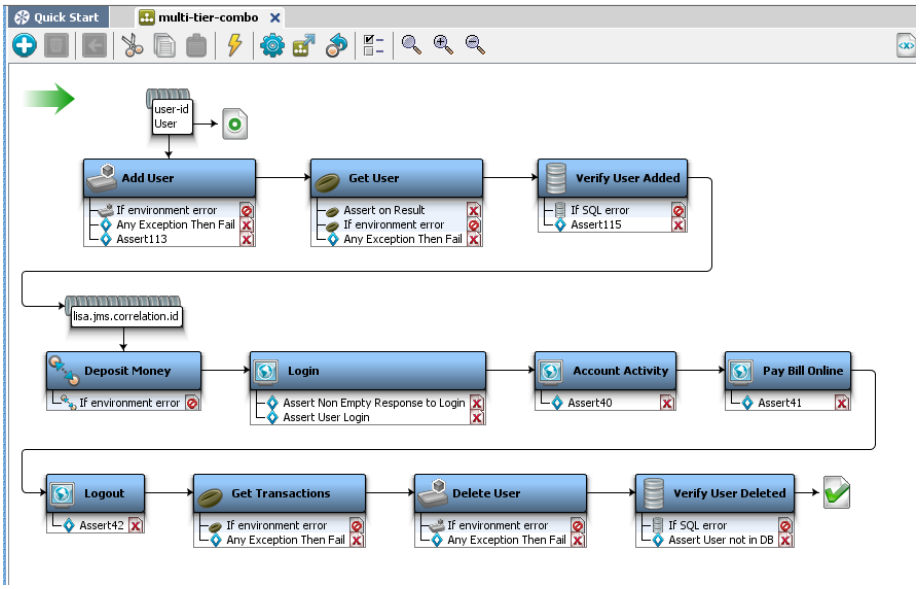
1. メイン メニューから［ファイル］-［開く］-［テスト ケース］-［ファイル システム］を選択します。
2. LISA_HOME¥examples¥Tests フォルダに移動します。
3. multi-tier-combo を選択し、［開く］をクリックします。

   multi-tier-combo テスト ケースがモデル エディタで開かれます。

手順 2 - テスト ケースの確認

このテスト ケースのさまざまなテスト ステップを確認します。 以下に例を示します。

- 「ユーザの追加」は Web サービス実行（レガシー）ステップです。
- 「Get User」（ユーザの取得）は Enterprise JavaBean 実行ステップです。
- 「Verify User Added」（追加されたユーザの検証）は SQL データベース実行（JDBC）ステップです。
- 「お預け入れ」は JMS メッセージング（JNDI）ステップです。



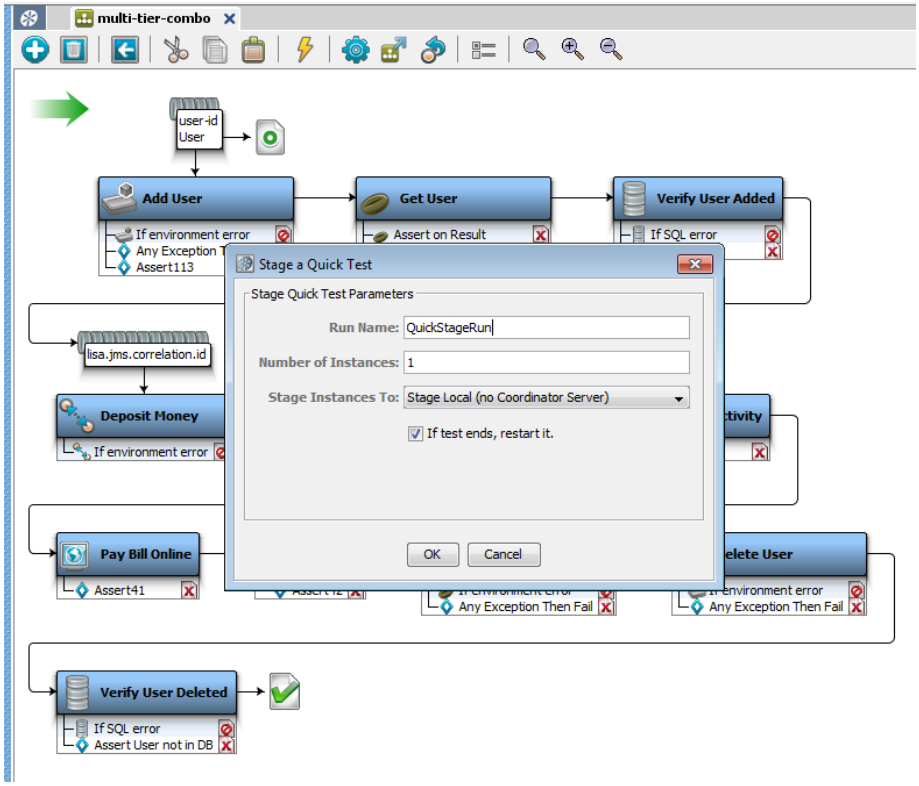
これらのステップの多くはチュートリアル 6 ～ 9 で使用しました。 このチュートリアルでは、これらのテスト ステップをすべて使用して、アプリケーションの複数のレイヤが関係するより現実的なテスト ケースを作成します。

手順 2 - パート A - クイック テストの実行

次の手順に従ってください:

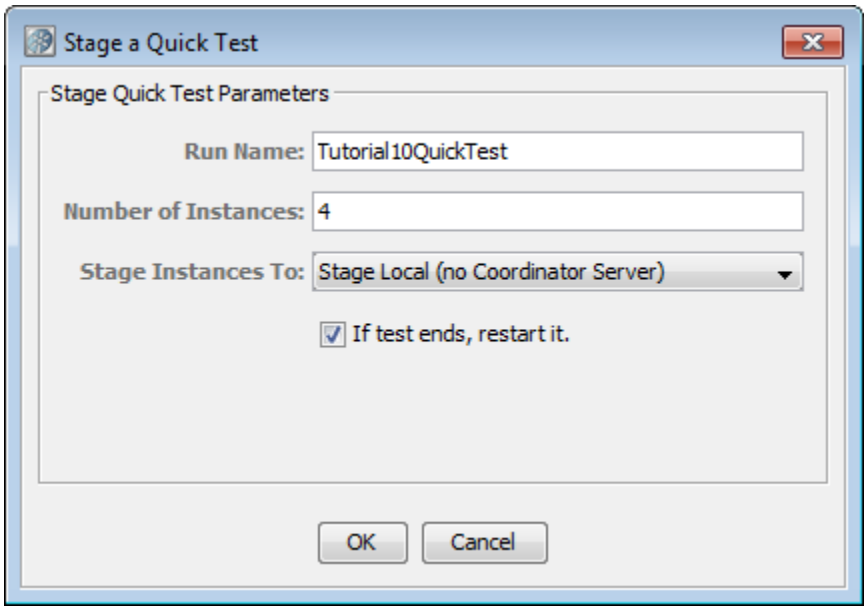
1. メニュー バーから、テスト ケース ツールバーにある  ［クイック テストのステージング］をクリックします。

   クイック テストを行うために、サンプル テスト ケースをモデル エディタで開くことができます。または、プロジェクト パネルでテストを右クリックし、そこからクイック テストをステージングするためのパラメータを入力できます。



2. ［クイック テストのステージング］ダイアログ ボックスに、以下のような必要な情報を入力します。
   - ［ラン名］：一意の名前（Tutorial10QuickTest）を入力します。
   - ［インスタンス数］：テストを同時に実行するユーザの数（4）を入力します。
   - ［インスタンスのステージング先］：コーディネータ サーバの名前を選択するか、ローカルにステージングします。

- テストを再起動するには、［テストが終了したら再起動する］を選択します。

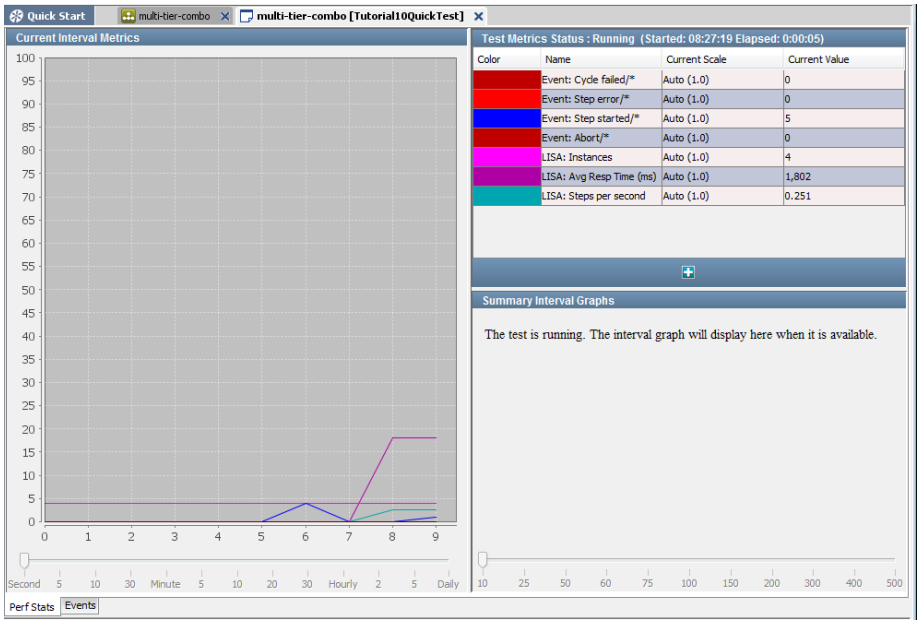


3. ［OK］をクリックします。

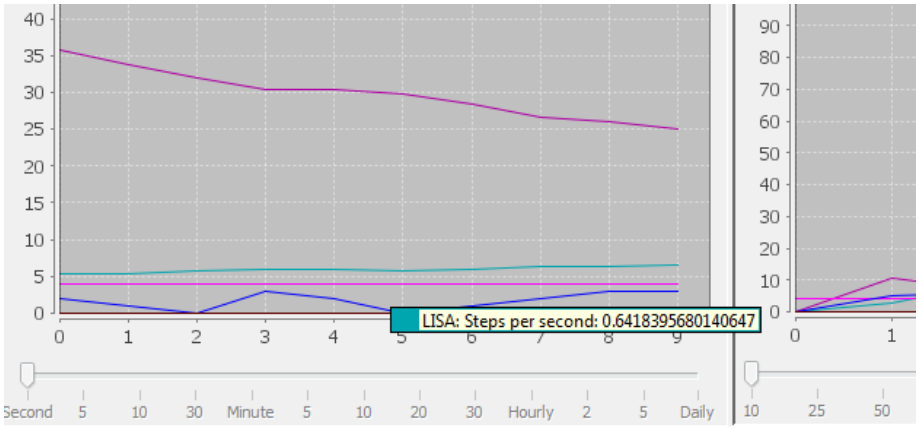
   ［テスト ラン］ウィンドウが開きますが、テストはまだ開始されていません。

4. テストの実行を開始するには、メイン ツールバーの  をクリックします。

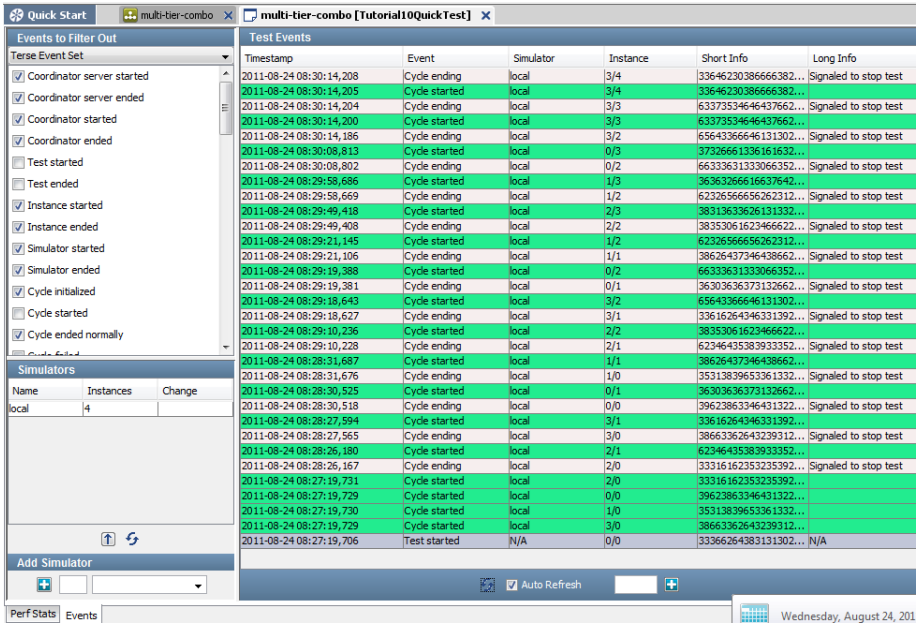
   テストが開始され、ただちに結果がグラフに表示されます。



グラフの行をロールオーバすると、説明を表示することができます。



5. 表示するイベントを選択するには、［イベント］タブを選択します。


Quick Start
multi-tier-combo
multi-tier-combo [Tutorial10QuickTest]
Events to Filter Out
Terse Event Set
Coordinator server started
Coordinator server ended
Coordinator started
Coordinator ended
Test started
Test ended
Instance started
Instance ended
Simulator started
Simulator ended
Cycle initialized
Cycle started
Cycle ended normally
Simulators
Name
Instances
Change
local
4
Add Simulator
Test Events
Timestamp
Event
Simulator
Instance
Short Info
Long Info
2011-08-24 08:30:14,208 Cycle ending local 3/4 33646230386666382... Signaled to stop test
2011-08-24 08:30:14,205 Cycle started local 3/4 33646230386666382...
2011-08-24 08:30:14,204 Cycle ending local 3/3 63373534646437662... Signaled to stop test
2011-08-24 08:30:14,200 Cycle started local 3/3 63373534646437662...
2011-08-24 08:30:14,186 Cycle ending local 3/2 65643366646131302... Signaled to stop test
2011-08-24 08:30:08,813 Cycle started local 0/3 37326661336161632...
2011-08-24 08:30:08,802 Cycle ending local 0/2 66333631333066352... Signaled to stop test
2011-08-24 08:29:58,686 Cycle started local 1/3 36363266616637642...
2011-08-24 08:29:58,669 Cycle ending local 1/2 62326566656262312... Signaled to stop test
2011-08-24 08:29:49,418 Cycle started local 2/3 38313633626131332...
2011-08-24 08:29:49,408 Cycle ending local 2/2 38353061623466622... Signaled to stop test
2011-08-24 08:29:21,145 Cycle started local 1/2 62326566656262312...
2011-08-24 08:29:21,106 Cycle ending local 1/1 38626437346438662... Signaled to stop test
2011-08-24 08:29:19,388 Cycle started local 0/2 66333631333066352...
2011-08-24 08:29:19,381 Cycle ending local 0/1 36303636373132662... Signaled to stop test
2011-08-24 08:29:18,643 Cycle started local 3/2 65643366646131302...
2011-08-24 08:29:18,627 Cycle ending local 3/1 33616264346331392... Signaled to stop test
2011-08-24 08:29:10,236 Cycle started local 2/2 38353061623466622...
2011-08-24 08:29:10,228 Cycle ending local 2/1 62346435383933352... Signaled to stop test
2011-08-24 08:28:31,687 Cycle started local 1/1 38626437346438662...
2011-08-24 08:28:31,676 Cycle ending local 1/0 35313839653361332... Signaled to stop test
2011-08-24 08:28:30,525 Cycle started local 0/1 36303636373132662...
2011-08-24 08:28:30,518 Cycle ending local 0/0 39623863346431322... Signaled to stop test
2011-08-24 08:28:27,594 Cycle started local 3/1 33616264346331392...
2011-08-24 08:28:27,565 Cycle ending local 3/0 38663362643239312... Signaled to stop test
2011-08-24 08:28:26,180 Cycle started local 2/1 62346435383933352...
2011-08-24 08:28:26,167 Cycle ending local 2/0 33316162353235392... Signaled to stop test
2011-08-24 08:27:19,731 Cycle started local 2/0 33316162353235392...
2011-08-24 08:27:19,729 Cycle started local 0/0 39623863346431322...
2011-08-24 08:27:19,730 Cycle started local 1/0 35313839653361332...
2011-08-24 08:27:19,729 Cycle started local 3/0 38663362643239312...
2011-08-24 08:27:19,706 Test started N/A 0/0 33366264383131302... N/A
Auto Refresh
Perf Stats
Events
Wednesday, August 24, 201

手順 2 - パート B - 生成されたレポートの表示

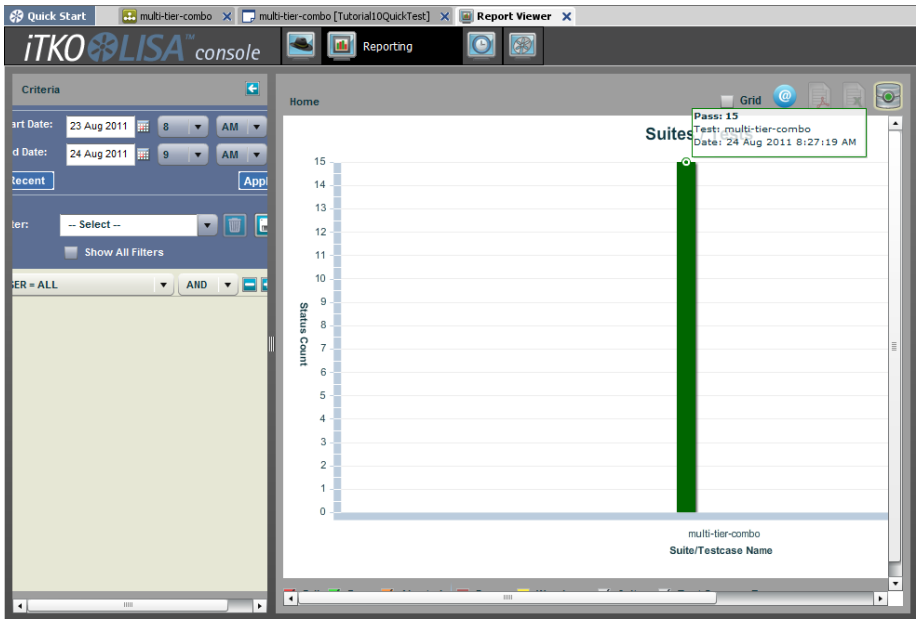
DevTest には、レポートを表示するためのレポート ビューアがあります。

次の手順に従ってください:

1. メイン メニューから［表示］-［レポート コンソール］をクリックするか、ツールバーの［レポート］ 

をクリックします。

   レポート ビューアが開きます。



2. 最近のアクティビティだけを参照するには、［最近使用したもの］ボタンをクリックします。

   上記の図のグラフは、このテスト ケースのテストがすべて正常に終了したことを示しています。

グラフを右クリックすると別のメニューが表示されます。 レポートの詳細については、「*CA Application Test の使用*」の「レポート」セクションを参照してください。

チュートリアル 10 - レビュー

このチュートリアルでは、クイック テストを使用して、サンプルの multi-tier-combo をテストしました。 また以下の方法を学習しました。

- 複数のタイプのテスト ステップが含まれるテスト ケースの確認。
- クイック テスト機能のテストの設定および実行。
- テストの実行で生成されるレポートの確認。

第 3 章: モバイル テストのチュートリアル

このセクションには、モバイル テストのさまざまな側面について説明するチュートリアルが含まれます。

各チュートリアルの前提条件

- DevTest ワークステーション がインストールされていて、DevTest Solutions のライセンス認証情報が入力されている。
- 「モバイル テスト環境の設定」で説明されているタスクをすべて完了している。
- 「*チュートリアル 1 - iOS テスト ケースの記録*」については、UICatalog.app.zip チュートリアル アプリケーションをローカル コンピュータにダウンロードして解凍する。
- 「*チュートリアル 2 - Android テスト ケースの記録*」については、ApiDemos.apk チュートリアル アプリケーションをローカル コンピュータにダウンロードする。

このセクションには、以下のトピックが含まれています。

チュートリアル 1 - iOS テスト ケースの記録

このチュートリアルでは、iOS シミュレータを使用してテスト ケースを記録するために必要な各手順について説明します。

このチュートリアルでは、以下のタスクを実行します。

- アセットの作成
- モバイル テスト ケースの記録
- ITR でのテスト ケースの実行

手順 1 - DevTest ワークステーション の起動

次の手順に従ってください:

1. レジストリを起動します。
   - DevTest サーバ の場合
     a. ［スタート］-［すべてのプログラム］-［DevTest Solutions］-［レジストリ］をクリックします。
     b. 「DevTest レジストリは使用可能です」というメッセージが表示されるまで待機します。
   - DevTest ワークステーション の場合、別の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。
2. ［スタート］-［すべてのプログラム］-［DevTest Solutions］-［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。
3. レジストリを選択し、［OK］をクリックします。

手順 2 - プロジェクトの作成

作成するプロジェクトには、モバイル チュートリアルに必要なテスト ケース サンプル ファイルがすべて保持されます。

注: 前のチュートリアルで MyMobileTutorials プロジェクトを作成した場合は、手順 3 「Android シミュレータの設定ファイルの作成 (P. 130)」に進みます。

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. ［プロジェクト名］フィールドに「**MyMobileTutorials**」と入力します。
3. デフォルト設定を使用して、LISA_HOME ディレクトリにプロジェクトを作成します。
4. ［Create］をクリックします。

   MyMobileTutorials プロジェクトが作成されます。

手順 3 - iOS シミュレータの設定ファイルの作成

デフォルト設定には project.config という名前が付けられ、新しいプロジェクトに対して自動的に作成されます。 project.config ファイルは、プロジェクト パネルの Configs フォルダにあります。 iOS シミュレータの使用に固有のアセット情報が含まれる設定ファイルを作成できます。

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. 新しい設定の名前として、「**MobileiOSSimulator**」と入力します。
3. ［OK］をクリックします。

   プロパティ エディタに MobileiOSSimulator 設定ファイルが表示されます。
4. プロジェクト パネルで MobileiOSSimulator 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。

手順 4 - iOS シミュレータ アセットの作成

特定の設定と関連付けられたシミュレータ セッション アセットでは、テストに使用されるモバイル シミュレータを指定できます。

次の手順に従ってください:

1. 「手順 3 - iOS シミュレータの設定ファイルの作成 (P. 122)」で作成した MobileiOSSimulator 設定ファイルを開きます（まだ開いていない場合）。
2. アセット ブラウザで、ペインの下部にある［追加］ をクリックします。
3. ［Mobile Session］をクリックします。

   ［Mobile Session］ダイアログ ボックスが表示されます。

   注: これらのフィールドのいずれかに値をすでに定義している場合、そのフィールドのリストから値を選択できます。リストの値を追加または削除するには、［リストのメンテナンス］を選択します。
4. 以下のフィールドに入力します。

   name

   「**iOSSimulator**」と入力します。

   説明

   モバイル チュートリアル 1 で使用する iOS シミュレータを入力します。

   Platform

   ドロップダウン メニューから［**iOS**］を選択します。

   アプリケーション

   ダウンロードした **UICatalog.app** ファイルに移動して選択します。

   ファミリ

   ドロップダウン メニューから［**iPhone のみ**］を選択します。

   Target

   ドロップダウン メニューから［**シミュレータ**］を選択します。

   シミュレータ

   マシン上の iOS シミュレータに移動して選択します。

   iOS のバージョン

6.1 を選択します。

5. （オプション）保存する前にアセットを検証するには、  をクリックします。
6. ［OK］をクリックします。

   新しい iOS シミュレータ アセットがアセット ブラウザに表示されます。
7. ［保存］をクリックします。

手順 5 - テスト ケースの記録

次の手順に従ってください:

1. MobileiOSSimulator 設定ファイルがアクティブであることを確認します。
2. **Tutorial_iOS_Simulator** という名前のテスト ケースを作成します。
   a. 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。
   b. 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ます。
   c. 新しいテスト ケースの名前を入力します。
   d. [保存] をクリックします。
   e. [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]-[モバイル レコーダ] をクリックします。

   モバイル テスト レコーダおよびモバイル シミュレータのウィンドウが開きます。
4. 複数のモバイル アセットを定義した場合は、[Choose Mobile asset（モバイル アセットの選択）] ドロップダウン リストを使用して接続するアセットを選択し、[OK] をクリックします。
5. 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。

   これで記録されます。
6. 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。

   注: 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。レコーダ ウィンドウで実行されたアクションはすべて、シミュレータに自動的に送信されます。
7. 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。 各テスト ステップは、アプリケーションの特定の画面上で実行したアクションを表します。

8. 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。

      ［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。

      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 ケース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」を参照してください。

手順 6 - ITR でのテスト ケースの実行

次の手順に従ってください:

1. 「手順 5 - テスト ケースの記録 (P. 125)」で作成したテスト ケースを開きます（まだ開いていない場合）。
2. 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。
3. 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。

   テストが完了すると、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。
4. ［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］を参照してください。

チュートリアル 2 - Android テスト ケースの記録

このチュートリアルでは、Android シミュレータを使用してテスト ケースを記録するために必要な各手順について説明します。

このチュートリアルでは、以下のタスクを実行します。

- アセットの作成
- モバイル テスト ケースの記録
- ITR でのテスト ケースの実行

手順 1 - DevTest ワークステーション の起動

次の手順に従ってください:

1. レジストリを起動します。
   - DevTest サーバ の場合
     a. ［スタート］-［すべてのプログラム］-［DevTest Solutions］-［レジストリ］をクリックします。
     b. 「DevTest レジストリは使用可能です」というメッセージが表示されるまで待機します。
   - DevTest ワークステーション の場合、別の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。
2. ［スタート］-［すべてのプログラム］-［DevTest Solutions］-［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。
3. レジストリを選択し、［OK］をクリックします。

手順 2 - プロジェクトの作成

作成するプロジェクトには、モバイル チュートリアルに必要なテスト ケース サンプル ファイルがすべて保持されます。

注: 前のチュートリアルで MyMobileTutorials プロジェクトを作成した場合は、手順 3 「Android シミュレータの設定ファイルの作成 (P. 130)」に進みます。

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. ［プロジェクト名］フィールドに「**MyMobileTutorials**」と入力します。
3. デフォルト設定を使用して、LISA_HOME ディレクトリにプロジェクトを作成します。
4. ［Create］をクリックします。

   MyMobileTutorials プロジェクトが作成されます。

手順 3 - Android シミュレータの設定ファイルの作成

デフォルト設定には project.config という名前が付けられ、新しいプロジェクトに対して自動的に作成されます。 project.config ファイルは、プロジェクト パネルの Configs フォルダにあります。 iOS シミュレータの使用に固有のアセット情報が含まれる設定ファイルを作成できます。

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. 新しい設定の名前として、「**MobileAndroidSimulator**」と入力します。
3. ［OK］をクリックします。

   プロパティ エディタに MobileAndroidSimulator 設定ファイルが表示されます。
4. プロジェクト パネルで MobileAndroidSimulator 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。

手順 4 - Android エミュレータ アセットの作成

特定の設定と関連付けられたシミュレータ セッション アセットでは、テストに使用されるモバイル シミュレータを指定できます。

次の手順に従ってください:

1. 「手順 3 - Android シミュレータの設定ファイルの作成 (P. 130)」で作成した MobileAndroidSimulator 設定ファイルを開きます(まだ開いていない場合)。
2. アセット ブラウザで、ペインの下部にある［追加］  をクリックします。
3. ［Mobile Session］をクリックします。

   ［Mobile Session］ダイアログ ボックスが表示されます。
4. 以下のフィールドを定義します。

   name

   「**AndroidSimulator**」と入力します。

   説明

   モバイル チュートリアル 2 で使用する Android シミュレータを入力します。

   Platform

   ドロップダウン メニューから［**Android**］を選択します。

   アプリケーション

   ダウンロードした **ApiDemos.apk** ファイルに移動して選択します。

   Target

   ドロップダウン メニューから［**エミュレータ**］を選択します。

   AVD

   モバイル テストを設定するときに定義した Android AVD の名前を入力します。

   SDK **バージョン**

   4.2.2 を選択します。
5. アセットを検証するには、  をクリックします。
6. ［OK］をクリックします。

新しい Android シミュレータ アセットがアセット ブラウザに表示されます。

7. ［保存］をクリックします。

手順 5 - テスト ケースの記録

次の手順に従ってください:

1. MobileAndroidSimulator 設定ファイルがアクティブであることを確認します。
2. **Tutorial_Android_Simulator** という名前のテスト ケースを作成します。
   a. 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。
   b. 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ます。
   c. 新しいテスト ケースの名前を入力します。
   d. [保存] をクリックします。
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。

   モバイル テスト レコーダおよびモバイル シミュレータのウィンドウが開きます。
5. 複数のモバイル アセットを定義した場合は、[Choose Mobile asset（モバイル アセットの選択）] ドロップダウン リストを使用して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。

   これで記録されます。
7. 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。

   注: 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。レコーダ ウィンドウで実行されたアクションはすべて、シミュレータに自動的に送信されます。
8. 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。 各テスト ステップは、アプリケーションの特定の画面上で実行したアクションを表します。

9. 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。

      ［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。

      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 ケース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」を参照してください。

手順 6 - ITR でのテスト ケースの実行

次の手順に従ってください:

1. 「手順 5 - テスト ケースの記録 (P. 125)」で作成したテスト ケースを開きます（まだ開いていない場合）。
2. 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。
3. 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。

   テストが完了すると、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。
4. ［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］を参照してください。

第 4 章: CA Service Virtualization チュートリアル

このチュートリアルでは、以下のことを行います。

- 設定ファイルの作成およびアクティブ化
- VSE レコーダの設定
- テスト ケースの記録
- 仮想サービス モデルの展開
- 仮想サービス モデルのテスト

このセクションには、以下のトピックが含まれています。

前提条件

以下の手順が、このチュートリアルを完了するための前提条件となります。

- DevTest ワークステーション および VSE がインストールされている。
- 以下のトピックの確認が終わっている。
  - 用語集 (P. 159)
  - 仮想化の概要
  - CA Service Virtualization について

VSI を作成およびテストするためのプロセス

前提条件が満たされたら、以下の手順に従います。

手順 1 - DevTest ワークステーション の起動

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。
   - 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
   a. ［スタート］-［すべてのプログラム］-［DevTest Solutions ］-［エンタープライズ ダッシュボード］をクリックしてエンタープライズ ダッシュボードを起動します。 「Enterprise Dashboard started」というメッセージが表示されるまで待ちます。
   b. ［スタート］-［すべてのプログラム］-［DevTest Solutions ］-［レジストリ］をクリックしてレジストリを起動します。
   - 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、別の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。
2. ［スタート］-［すべてのプログラム］-［DevTest Solutions ］-［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。
3. レジストリを選択し、［OK］をクリックします。
4. ［ログイン］ダイアログ ボックスが開きます。有効なユーザ名とパスワードを入力し、［ログイン］をクリックします。
5. 「手順 2 - VSE の起動 (P. 139)」に進みます。

手順 2 - VSE の起動

次の手順に従ってください:

1. ［スタート］-［すべてのプログラム］-［DevTest Solutions ］-［Virtual Service Environment］をクリックします。

   「仮想サービス環境の準備ができました」というメッセージが表示されたら、VSE が初期化されています。
2. 「手順 3 - デモ サーバの起動 (P. 140)」に進みます。

手順 3 - デモ サーバの起動

DevTest ワークステーション と VSE の両方が実行されたら、このチュートリアルを完了するために使用するデモ サーバを起動できます。

次の手順に従ってください:

1. ［スタート］-［すべてのプログラム］-［DevTest Solutions ］-［DevTest Demo Server］をクリックします。

   「Started in XXs:YYms」というメッセージが表示されたら、デモ サーバが起動しています。

2. 「手順 4 - テスト ケースの実行 (P. 140)」に進みます。

手順 4 - テスト ケースの実行

LISA Bank が使用可能であることを確認するには、対話型テスト ラン（ITR）でテスト ケースを実行します。

次の手順に従ってください:

1. プロジェクト パネルで、［**webservices.tst**］をダブルクリックします。

   webservices テスト ケースがタブで開きます。

2. ［ITR］  をクリックして ITR を開きます。

3. デモ サーバに対してテスト ケースを実行するには、［自動的にテストを実行］  をクリックします。

   テストが正常に完了した場合、デモ サーバが使用可能であり、テスト ケースが正しく実行されたことがわかります。

4. ITR ウィンドウを閉じます。

5. エディタ タブで［X］をクリックして、テスト ケースを閉じます。

6. 「手順 5 - 設定ファイルの作成 (P. 141)」に進みます。

手順 5 - 設定ファイルの作成

webservices テスト ケースは、ソース アプリケーション ドライバです。 DevTest ワークステーション で VSE レコーダを使用して、このテスト ケースのトランザクションをリスンおよび記録します。 ソース クライアント アプリケーションとライブ システムの間に VSE レコーダを挿入します。 ライブ LISA Bank システム Web サービスは、ローカルホストのポート 8080 で実行されます。 エンドポイントが VSE レコーダのリスン ポート（8001）を使用するように、エンドポイントを設定します。 レコーダは、ポート 8080 上のライブ Web サービス アプリケーションに対する要求/応答の送受信をインターセプトします。

テスト ケースの各ステップには、そのステップのエンドポイントを指定できるフィールドが含まれています。 デフォルトでは、DevTest は、{{ENDPOINT1}} プロパティを使用してこのエンドポイントを定義します。 エンドポイントをハード コードせずにプロパティを使用することにより、プロジェクト設定の {{ENDPOINT1}} 値を変更することで、各ステップのエンドポイントを迅速に変更できます。

デフォルト設定には project.config という名前が付けられ、新しいプロジェクトに対して自動的に作成されます。 project.config ファイルは、プロジェクト パネルの Configs フォルダにあります。 また、設定ファイルを作成することもできます。

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. 新しい設定の名前として、「**VSRecorder**」と入力します。
3. ［OK］をクリックします。

   プロパティ エディタで VSRecorder の設定ファイルが開きます。

設定ファイルにプロパティ追加する方法

1. 行を追加するには、プロパティ エディタの下部にある［追加］  をクリックします。
2. ［キー］フィールドから「ENDPOINT1」を選択します。
3. ［値］フィールドに以下の値を入力します。

   ```
   http://localhost:8001/itkoExamples/EJB3UserControlBean?wsdl
   ```

注: テスト ケースのポート（8080）は、VSE レコーダのリスン ポート（8001）に変更されます。 レコーダは、ポート 8080 上のライブ Web サービス アプリケーションに対する要求/応答の送受信をインターセプトできます。

4. メイン ツールバーの［保存］をクリックします。
5. 設定ファイルを閉じるには、エディタ タブの［X］をクリックします。
6. 「手順 6 - 設定ファイルのアクティブ化 (P. 143)」に進みます。

次の手順に進みます (P. 143)。

手順 6 - 設定ファイルのアクティブ化

DevTest で、プライマリまたはアクティブな設定ファイルとして新しく作成された設定ファイルを使用するには、設定ファイルをアクティブにします。

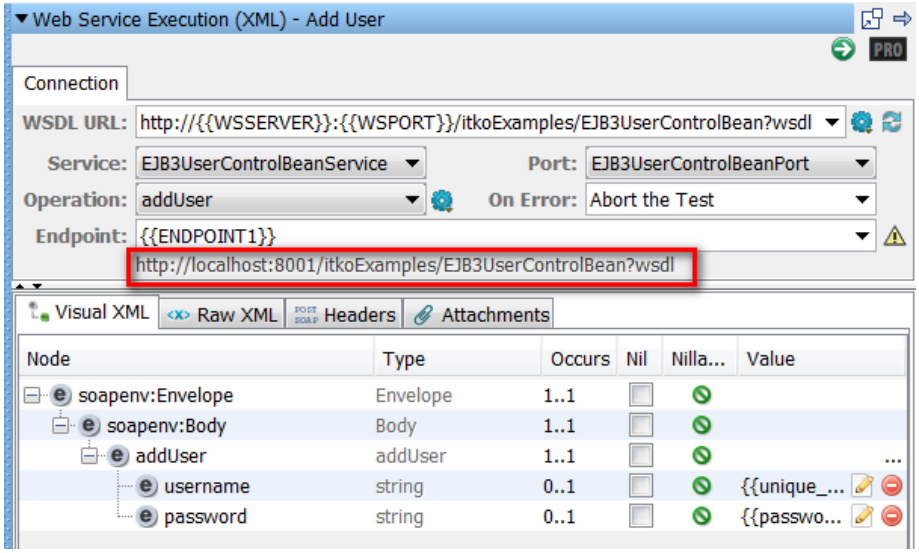
設定ファイルをアクティブにする方法

1. プロジェクト パネルで、［VSRecorder］設定ファイルを右クリックします。
2. ［アクティブ化］を選択します。

VSRecorder 設定ファイルの ENDPOINT1 値が使用されていることを確認する方法

1. プロジェクト パネルで、［**webservices**］テスト ケースをダブルクリックします。

   エディタが開きます。
2. ［**Add User**］テスト ステップをダブルクリックし、エンドポイントがポート 8001 に設定されていることを確認します。

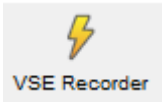


3. 「手順 7 - VSE レコーダの設定 (P. 144)」に進みます。

手順 7 - VSE レコーダの設定

エンドポイントを設定したら、VSE レコーダを設定することができます。

次の手順に従ってください:

1. メイン ツールバーの［VSE レコーダ］ 

をクリックします。

   ［基本］タブで、データを格納する仮想サービス イメージの設定を行います。 トランザクション ワークフロー モデルを作成する仮想サービス モデルの設定を行います。

2. ［イメージの書き込み先］フィールドのデフォルトのサービス イメージ名（*newimage.vsi*）を、*VSETutorial.vsi* に置き換えます。

3. ［インポート トラフィック］フィールドを空白のままにします。

   このチュートリアルでは、既存のトラフィック ファイルをインポートしません。

4. トランスポート プロトコルに［HTTP/S］を選択します。

   このチュートリアルでは、機密データ（社会保障番号や銀行口座番号など）を渡さないため、データをディセンシタイズしません。 ［エクスポート先］フィールド（レコーディングの RAW トラフィック ファイルをエクスポートするために使用）に移動します。 エクスポートされたファイルは、レコーディング セッション全体の RAW バックアップの役割を果たします。

5. ［参照］をクリックして、ファイルに「**rt_VSETutorial**」という名前を付け、VServices フォルダに保存します。

   「rt_」というプレフィックスは、このファイルが RAW トラフィック ファイルであることを示しています。

6. ［参照］をクリックして、仮想サービス モデル ファイルを作成します。

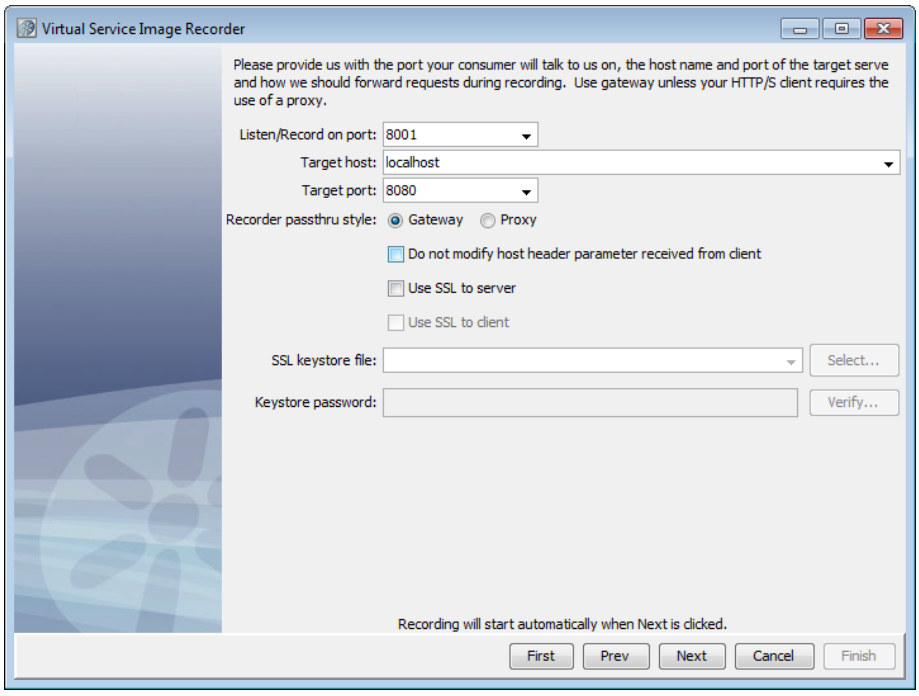
7. 「**VSETutorial**」という名前を入力し、［**保存**］をクリックします。

8. ［VS モデル スタイル］の選択内容をそのままにして、［次へ］をクリックします。

   次のウィンドウで、レコーダのポートを指定します。

9. デフォルトでは、レコーダは、VSRecorder 設定ファイルでエンドポイントとして設定したポート 8001 でリスンおよび記録します。 ［ターゲット ホスト］フィールドに［**localhost**］を追加します。

10. ［ターゲット ポート］（ライブ LISA Bank システム トランザクションが実行されるポート）を「**8080**」に更新します。



11. ［レコーダ転送方式］で［ゲートウェイ］オプションを選択します。

    これらの設定が完了したら、レコーディングを開始することができます。

12. ［次へ］をクリックします。

13. 「手順 8 - テスト ケースの記録 (P. 146)」に進みます。

手順 8 - テスト ケースの記録

次の手順に従ってください:

1. ［次へ］をクリックして、レコーディングを開始します。

   ウィンドウに表示されているように、レコーダは、ポート 8001 でターゲット ポート 8080 をリスンします。 ウィンドウには、セッションおよびトランザクション数が表示されます。テスト ケースはまだ実行されていないため、数は 0 です。テスト ケースの実行が完了すると、このウィンドウには 3 つトランザクションがカウントされます。

2. DevTest ワークステーション で、テスト ケースのタブをクリックして ITR を開き、ステップを自動で実行します。

3. レコーダに移動し、3 つのトランザクションが正常に記録されたことを確認します。

   テスト ケースが記録されたら、［次へ］をクリックしてレコーディング セッションを終了します。DevTest はレコーディングを処理し、トランザクションのリストを示します。より高度な仮想サービス トランザクションでは、次のウィンドウでポストレコーディング モデルとデータを変更します。このテスト ケースでは、続行する準備ができているため、［次へ］をクリックします。

   次のウィンドウでは、ベース パス、バインディング、およびロジックは正しく設定されています。

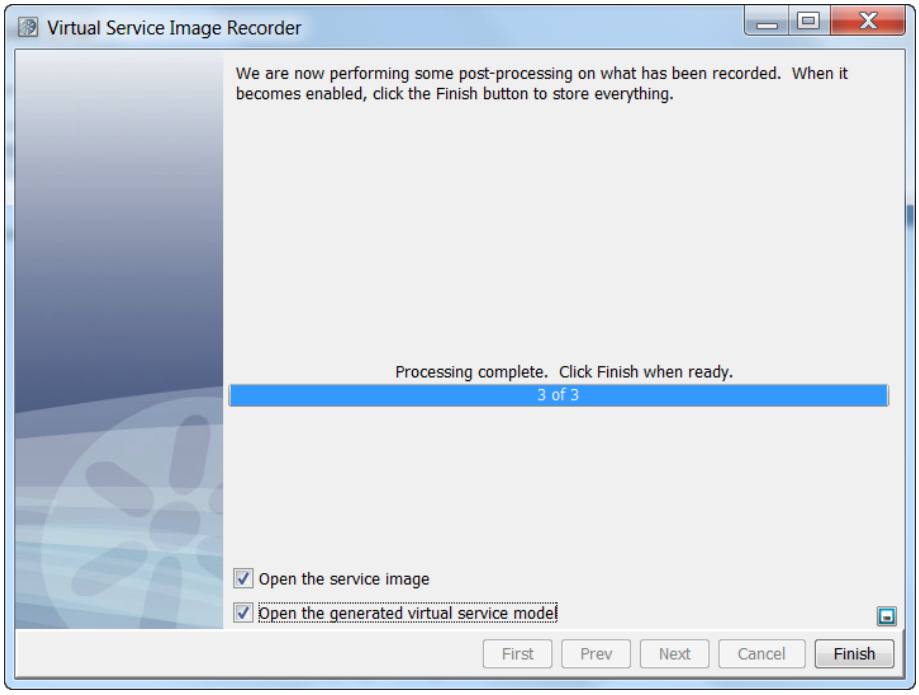
4. ［次へ］をクリックします。

   次のウィンドウは、Web サービス（SOAP）データ プロトコル ハンドラが要求側に選択されたことを示しています。

5. この選択内容を変更せずに、［次へ］をクリックします。

6. 次のウィンドウでは変更を行わず、Enter を押します。

   DevTest は、最終時点で 1 回レコーディングを処理し、仮想サービス モデルと仮想サービス イメージ ファイルを作成します。



これらの各ファイルは、DevTest ワークステーション で開いて編集できます。

7. 仮想サービス モデルおよび仮想サービス イメージ ファイルを参照するには、［サービス イメージを開く］および［生成された仮想サービス モデルを開く］チェック ボックスをオンにします。
8. ［終了］をクリックします。

   DevTest はレコーダを閉じて、DevTest ワークステーション で選択されたファイルを開きます。
9. 「手順 9 - 仮想サービス モデルの展開 (P. 148)」に進みます。

手順 9 - 仮想サービス モデルの展開

仮想サービス モデル（VSM）を展開する方法

1. デモ サーバ ウィンドウを閉じるには、右上隅の **X** をクリックします。

   このウィンドウを閉じると、LISA Bank をホストしているデモ サーバがシャットダウンされます。 このシャットダウンでは、テスト中のシステムのないテスト ケースが残され、作成した仮想サービスが使用可能になります。

2. DevTest ワークステーション を開き、仮想サービスを管理する VSE UI を提供するサーバ コンソールを開きます。

3. 左側のナビゲーション ペインで VSE をクリックします。

4. DevTest ワークステーション で、プロジェクト パネルの［VSETutorial］仮想サービス モデルを右クリックし、［VSE に展開/再展開］を選択します。

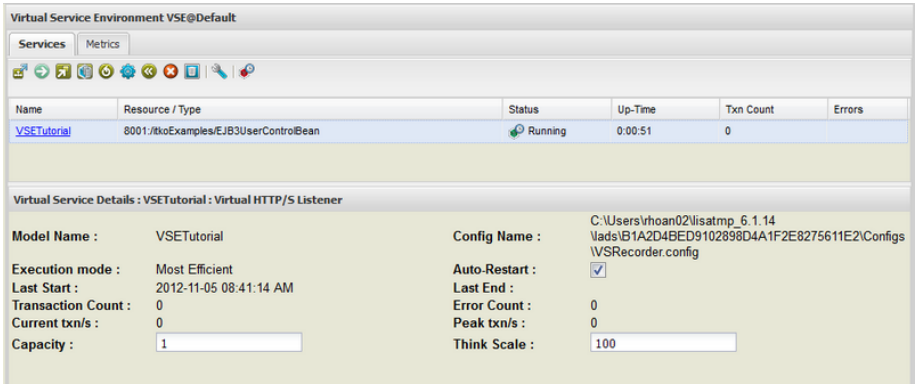
   ［仮想サービスの展開］ダイアログ ボックスでは、仮想サービス モデルおよび VSRecorder 設定ファイルが選択されています。［グループ タグ］は空白です。 ［同時実行数］が 1 に設定され、［反応時間スケール］が 100% であり、［サービスが終了した場合、自動的に再起動する］チェック ボックスがオンになっています。［展開時にサービスを開始する］チェック ボックスがオンになっているため、仮想サービスを展開すると、その仮想サービスが開始されます。

   このモデルおよび設定を共有するか、またはリモートで展開するには、必要に応じて［MAR として保存］ボタンをクリックして MAR ファイルを作成し、モデルを配布します。 あらかじめ入力されている設定はすべて正しいため、［展開］をクリックすると仮想サービスが展開され、VSE ダッシュボードに表示されます。

   仮想サービスが VSE に展開されます。

5. VSE コンソールに戻るには、ブラウザの［DevTest コンソール］タブをクリックします。

   ［サービス］タブに、VSETutorial 仮想サービス モデルが表示されます。



仮想サービスが開始され、トランザクションを処理する準備が整います。トランザクションは、［トランザクション数］列で追跡されます。サービスは、最初、処理したトランザクション数を 0 と表示します。

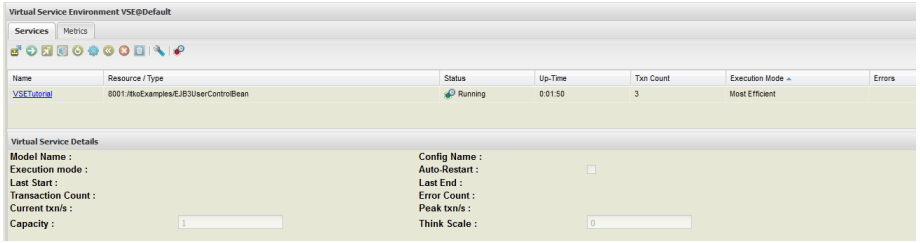
6. テスト ケースを実行し、このウィンドウに戻ります。
7. 「手順 10 - 仮想サービス モデルのテスト (P. 150)」に進みます。

手順 10 - 仮想サービス モデルのテスト

仮想サービス モデル(VSM)をテストする方法

1. **webservices** テスト ケースを開き、ITR でそれを実行します。
2. ITR が完了したら、仮想サービスに戻ります。

   現在、トランザクション数は 3 であり、仮想サービスのテストを行っていることを確認できます。



   注: また、仮想サービスに対するテスト ケースまたはスイートをステージングし、実行できます。

3. プロジェクト ペインで［**webservices**］テスト ケースを右クリックし、［クイック テストのステージング］を選択します。
4. ［インスタンス数］に「**10**」と入力します。
5. ［テストが終了したら再起動する］チェック ボックスをオフにし、［OK］をクリックします。
6. テスト モニタ ウィンドウで［再生］をクリックし、テストを実行します。

   10 のインスタンスでテスト ケースを実行した後に VSE ダッシュボードに戻ると、トランザクション数が 33 と表示されます。

7. 「チュートリアルの復習 (P. 151)」に進みます。

チュートリアルの復習

このチュートリアルでは、VSE の基本的な機能を確認するために仮想サービスを作成およびテストしました。

このチュートリアルでは、以下のことを行いました。

- 設定ファイルの作成およびアクティブ化
- VSE レコーダの設定
- テスト ケースの記録
- 仮想サービス モデルの展開
- ITR の使用、仮想サービス モデルに対するテストのステージング

第 5 章: CA Continuous Application Insight チュートリアル

このチュートリアルでは、LISA Bank デモ アプリケーションを使用してトランザクションを生成します。 その後、DevTest ポータルの［Analyze Transactions］ウィンドウにトランザクションを表示します。

前提条件

- 以下のコンポーネントが実行中である： エンタープライズ ダッシュボード、レジストリ、デモ サーバ、ブローカ、およびポータル

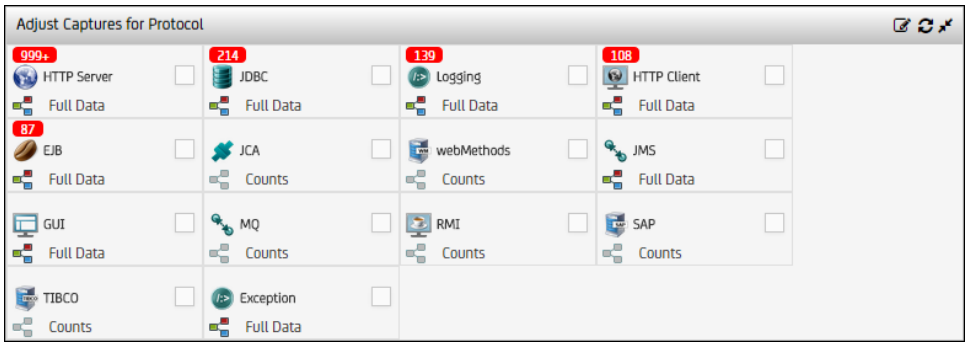
このセクションには、以下のトピックが含まれています。

手順 1 - デフォルトのキャプチャ レベルを上げる

DevTest Java エージェントがキャプチャできるプロトコルにはそれぞれキャプチャ レベルがあります。

デフォルトのキャプチャ レベルは、［カウント］です。 このチュートリアルの後半でトランザクションを表示するには、関連するプロトコルのキャプチャ レベルを［全データ］に設定する必要があります。

以下の図は、［エージェント］ウィンドウの［Adjust Captures for Protocol］を示しています。



次の手順に従ってください:

1. DevTest ポータルを開きます。 ポータルがローカル コンピュータ上で実行されている場合、URL は **http://localhost:1507/devtest** です。
2. 左側のナビゲーション メニューから［設定］-［エージェント］を選択します。

   ［エージェント］ウィンドウが表示されます。
3. ［エージェント］ペインで、**JBoss_LISABank** エージェントを選択します。
4. ［Adjust Captures for Protocol］ペインが折りたたまれている場合は、ペインを展開します。
5. プロトコルの一部が表示されない場合は、［編集］をクリックして、［すべてのプロトコルを表示します］を選択します。
6. 以下のプロトコルのキャプチャ レベルが［全データ］に設定されていることを確認します。
   - HTTP サーバ
   - JDBC

- EJB
- ログ

キャプチャ レベルを変更するには、プロトコルのチェック ボックスをオンにして、キャプチャ レベルのドロップダウン リストから［全データ］を選択します。

手順 2 - デモ アプリケーションからのトランザクションの生成

DevTest Solutions には、LISA Bank という名前のデモ アプリケーションが含まれています。

この手順では、デモ アプリケーションにログインし、口座を作成し、ログアウトします。

次の手順に従ってください:

1. Web ブラウザに「**http://localhost:8080/lisabank**」と入力します。 デモ サーバが別のコンピュータで実行されている場合は、**localhost** をリモート コンピュータのホスト名または IP アドレスに置き換えます。

   ログイン ページが表示されます。
2. ［名前］フィールドに「**lisa_simpson**」と入力します。
3. ［パスワード］フィールドに「**golisa**」と入力します。
4. ［ログイン］をクリックします。

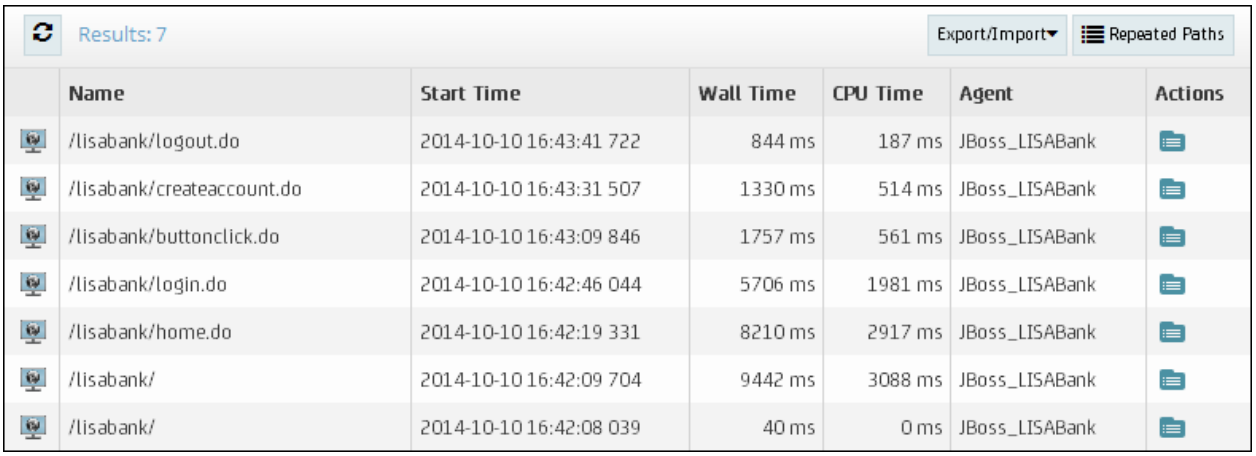
   初期画面ページが表示されます。 左側には、実行できるさまざまなアクションのボタンがあります。
5. 以下の手順を実行して口座を作成します。
   a. ［口座の開設］をクリックします。
   b. ［口座名］フィールドに「**My Checking**」と入力します。
   c. ［初期残高］フィールドで、デフォルト値を **500** に置き換えます。
   d. ［口座の追加］をクリックします。
6. ［ログアウト］をクリックします。

手順 3 - [Analyze Transactions]ウィンドウでのトランザクションの表示

［Analyze Transactions］ウィンドウでは、以下のタスクを実行できます。

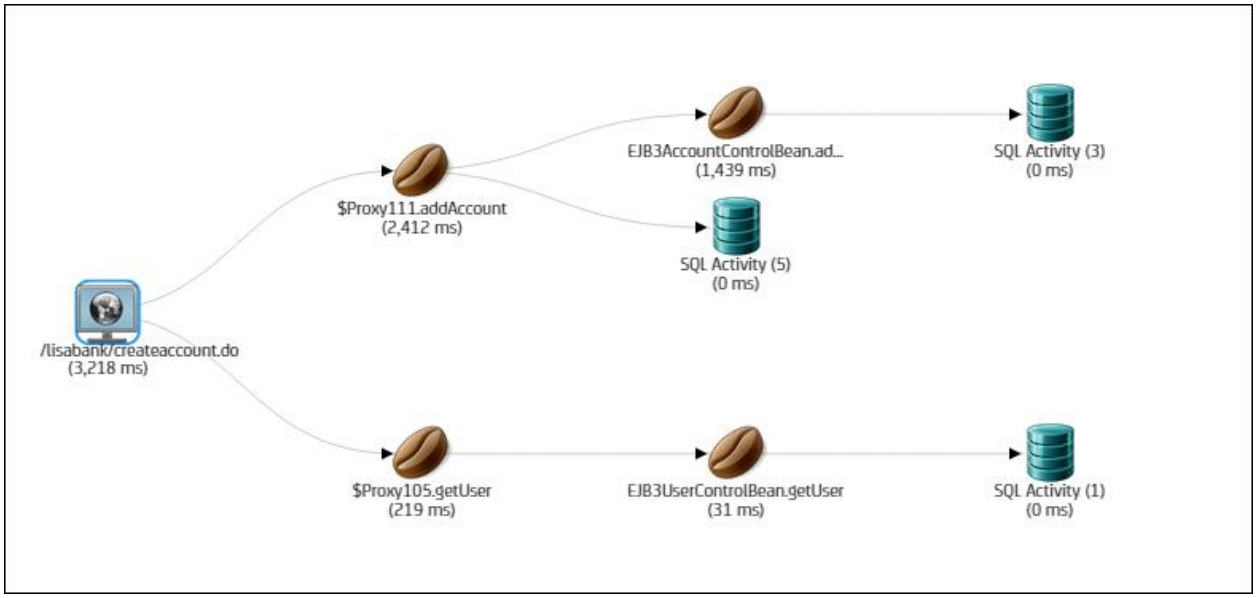
- トランザクションの表示
- トランザクションの検索およびフィルタ
- コンポーネントの詳細の表示

以下の図は、［Analyze Transactions］ウィンドウのトランザクションのセットを示しています。 トランザクションは、［リスト］ビューに表示されます。 トランザクションは、逆の年代順に表示されます。



Name	Start Time	Wall Time	CPU Time	Agent	Actions
/lisabank/logout.do	2014-10-10 16:43:41 722	844 ms	187 ms	JBoss_LISABank	
/lisabank/createaccount.do	2014-10-10 16:43:31 507	1330 ms	514 ms	JBoss_LISABank	
/lisabank/buttonclick.do	2014-10-10 16:43:09 846	1757 ms	561 ms	JBoss_LISABank	
/lisabank/login.do	2014-10-10 16:42:46 044	5706 ms	1981 ms	JBoss_LISABank	
/lisabank/home.do	2014-10-10 16:42:19 331	8210 ms	2917 ms	JBoss_LISABank	
/lisabank/	2014-10-10 16:42:09 704	9442 ms	3088 ms	JBoss_LISABank	
/lisabank/	2014-10-10 16:42:08 039	40 ms	0 ms	JBoss_LISABank	

以下の図は、トランザクションの 1 つのパス グラフを示しています。



次の手順に従ってください:

1. DevTest ポータルに戻ります。
2. 左側のナビゲーション メニューから［Application Insight］-［Analyze Transactions］を選択します。

   ［Analyze Transactions］ウィンドウが表示されます。
3. **/lisabank/createaccount.do** という名前のトランザクションを見つけます。
4. ［Actions］列の［Open transaction details］ボタンをクリックします。

   トランザクション詳細ダイアログ ボックスが表示されます。上部の領域にパス グラフが表示されます。それより下の領域に、選択したノードに関する情報が表示されます。
5. パス グラフの EJB ノードをクリックします。
6. 各タブに表示される情報を確認します。
7. ダイアログ ボックスを閉じます。

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」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」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」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」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」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）」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」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」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」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」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」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」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」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」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」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」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」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」および「ステージング ドキュメント エディタ - ［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」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」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」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」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」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」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」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」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。