

SilkCentral Test Manager 12.0



API ヘルプ

Micro Focus
575 Anton Blvd., Suite 510
Costa Mesa, CA 92626

Copyright © 2012 Micro Focus. All rights reserved. SilkCentral Test Manager は Borland Software Corporation に由来する成果物を含んでいます, Copyright © 2012 Borland Software Corporation (a Micro Focus company).

MICRO FOCUS, Micro Focus ロゴ、及びその他は Micro Focus IP Development Limited またはその米国、英国、その他の国に存在する子会社・関連会社の商標または登録商標です。

その他、記載の各名称は、各所有社の知的所有財産です。

2012-03-08

目次

はじめに	5
プラグインを作成する	6
コードカバレッジとの統合	7
独自のコードカバレッジプラグインを作成する	7
コードカバレッジプラグインの Test Manager コード分析フロー	8
サンプルプロファイルクラス	9
コードカバレッジ XSD	11
サンプル XML データ	12
コード分析のサンプル インターフェイス クラス	14
Linux AUT 環境にコード分析フレームワークをインストールする	15
ソース管理との統合	16
ソース管理との統合のインターフェイス	16
ソース管理との統合の慣習	17
問題追跡システムとの統合	18
Web サービス インターフェイス	18
WSDL 仕様	21
Web サービス プロファイルを設定する	27
Java API インターフェイス	28
要件管理システムとの統合	29
Java API インターフェイス	29
サードパーティ テスト タイプとの統合	30
プラグイン実装	30
パッケージング	30
プラグインへパラメータを渡す	31
API の構造	31
サンプルコード	32
設定 XML ファイル	35
プラグインのメタ情報	35
一般プロパティのメタ情報	35
文字列プロパティのメタ情報	36
ファイルプロパティのメタ情報	36
カスタム アイコン	36
デプロイメント	37
ビデオ キャプチャの開始と終了を示す	37
SilkCentral の Web サービス	39
Web サービス クイック スタート	39
前提条件	39
Web サービス 入門	40
Web サービス クライアントの概要	40
サンプル事例：要件を追加する	43
セッション処理	44
利用可能な Web サービス	45
Services Exchange	45
reportData インターフェイス	46
TMAttach インターフェイス	46
createTestPlan インターフェイス	48
exportTestPlan インターフェイス	51
updateTestPlan インターフェイス	51
createRequirements インターフェイス	52

exportRequirements インターフェイス	53
updateRequirements インターフェイス	54
updateRequirementsByExternalID インターフェイス	55
createExecutionDefinitions インターフェイス	57
exportExecutionDefinitions インターフェイス	60
updateExecutionDefinitions インターフェイス	61
createLibraries インターフェイス	62
exportLibraryStructure インターフェイス	63
exportLibraryStructureWithoutSteps インターフェイス	64
getLibraryInfoByName インターフェイス	65
Web サービス デモ クライアント	66

はじめに

本書では、サードパーティ ツールを SilkCentral® Test Manager™ (Test Manager) に統合するプラグインを、作成/運用していくために必要となる情報を提供していきます。主に、Web サービスの仕様、API の説明、そしてプラグインを Test Manager に統合する方法についての説明などが含まれます。



注: 本書では、読者が Web サービスの実装と利用に精通していると想定しています。

概要

Test Manager では、サードパーティ アプリケーションとの統合のための API を提供しています。SilkCentral の API を使うことにより、既存のソース管理、問題追跡、要件管理などの各ツールを、Test Manager プラグインを設定することで統合することができます。Test Manager には、数多くのサンプルプラグインが同梱されています。

利用可能な Java クラスやメソッドに関する詳細については、[Javadoc](#) を参照してください。このリンクが動作しなかった場合には、Test Manager メニューの **ヘルプ > ドキュメント > Test Manager API 仕様** を選択して、Javadoc を開いてください。

Test Manager 統合プラグイン

Test Manager プラグインは、「そのまま」提供されているものであり、すべての不具合も含み、なんら保証されるものではありません。Borland は、市販性、特定目的との適合性、ウィルスがないこと、正確性または完全性、権限、平穩享有権、平穩所有権、権利侵害のないことなどを含む（ただし必ずしもこれらに限定されない）あらゆるものに関して、明示的、黙示的、または平和的にかかわらず、すべての保証および条件を、放棄します。

プラグインのご利用は、ご自身の責任において行ってください。Borland は、プラグインを使用することによって生じる利益損失を含む（ただし必ずしもこれらに限定されない）、直接的、間接的、および特別のあらゆる種の偶発的または付随的な損害に対して、責任を負いません。

プラグインを作成する

概要

このセクションでは、Test Manager に対してプラグインを作成する方法について説明します。すべてのタイプのプラグインに共通するタスクについてのみ、ここでは説明します。

プラグインの種

Test Manager は、複数のプラグイン API を提供しています。各 API は、種 (*species*) として捉えられます。

コンパイル


プラグインの開発やコンパイルにおいて、適切な Java バージョンについては、Test Manager のリリースノートを参照してください。このことは、Test Manager の Java ランタイム環境と互換性を確保するために重要となります。

デプロイメント

プラグイン クラスを作成し、API の種を実装したら、プラグイン パッケージ (jar や zip) を作成することができます。

- 作成するプラグインが他に依存していない場合（もしくは、既に SilkCentral 内にあるライブラリとのみ依存関係がある場合）は、単に、作成したクラスが含まれた JAR ファイルを作成します。
- 作成したプラグインが追加ライブラリに依存する場合、それらのライブラリを、サブディレクトリ lib に置き、すべてのライブラリを 1 つの ZIP アーカイブにパッケージします。

作成したファイルを、<アプリケーション サーバーのインストール ディレクトリ>%plugins% にあるプラグイン ディレクトリに置きます。

 **注:** 新しくデプロイしたプラグインを Test Manager で利用可能にするには、アプリケーション サーバーとフロントエンド サーバーを再起動する必要があります。サーバーの再起動に関する情報については、『SilkCentral 管理モジュール ヘルプ』を参照してください。

配信


プラグインの種のタイプが SilkCentral によって認識されているのと同様、どのサーバー（実行サーバー、アプリケーション サーバー、およびフロントエンド サーバー）でどの種が必要であるかも認識されています。各プラグインはアプリケーション サーバーにインストールすることができます。そして、SilkCentral が各サーバーへ適切なプラグインを自動配信します。

コード カバレッジとの統合


このセクションで説明する Java API インターフェイスは、サードパーティ製（外部）コード カバレッジ ツールの統合を実現する、Test Manager 用プラグインを作成する場合に必要となります。

コード カバレッジ ツールを使用すると、テストでどのようなコードがカバーされるかについての情報を取得できます。Test Manager には、次のコード カバレッジ ツールがあらかじめ組み込まれています。

- SilkCentral Java コード分析 (Java コード分析エージェント)
- DevPartner Studio .NET コード分析 (Windows コード分析フレームワーク)


 **注:** テスト対象アプリケーションが Linux 上で動作する場合は、トピック「Linux AUT 環境にコード分析フレームワークをインストールする」を参照してください。


前出の 2 つのツールで十分でない場合は、独自のコード カバレッジとの統合を作成してデプロイできます。「独自のコード カバレッジ プラグインを作成する」を参照してください。

 **注:** アプリケーション サーバーにカスタム アプリケーションをデプロイする（「プラグインの作成」トピックを参照）以外に、コード分析フレームワーク サーバーにもカスタム アプリケーションをデプロイする必要があります。コード分析フレームワーク サーバーには、AUT およびコード カバレッジ ツールが配置されます。このパスは ¥Silk¥SC Test Manager¥Plugins となります。

独自のコード カバレッジプラグインを作成する

このトピックでは、コード カバレッジ プラグインを作成する方法について説明します。SilkCentral のベースラインの概念をよく理解する必要があります。SilkCentral では、各実行の前にベースラインが必要です。ベースラインには、テスト アプリケーションのすべての名前空間/パッケージ/クラス/メソッドが含まれています。

 **注:** コード カバレッジを実行したら、SilkCentral API には XML ファイルを返す必要があります。したがって、コード カバレッジ情報をデータベースに格納するコード カバレッジ ツールの場合は、データを取り出す追加手順を実行する必要があります。

 **注:** 同じコード分析フレームワークに対して複数のテストを同時実行する、複数実行サーバーはサポートされていません。


1. ライブラリ scc.jar には拡張が必要なインターフェイス群が含まれているため、このライブラリをクラスパスに追加します。この JAR ファイルは、SilkCentral のインストールディレクトリ以下の lib ディレクトリ内にあります。

2. 次の 2 つのインポート文を追加します。

```
import com.segue.scc.published.api.codeanalysis.CodeAnalysisProfile;  
import com.segue.scc.published.api.codeanalysis.CodeAnalysisProfileException;
```

3. CodeAnalysisProfile を実装するクラスを作成します。

4. 以降の手順で記述されているコード カバレッジ インターフェイスから、必要なメソッドをすべて追加します。「サンプル インターフェイス クラス」を参照して定義を確認し、メソッドを手作業で実装するか、または、インポートおよびメソッドが定義されているトピック「サンプル プロファイル クラス」からコピーして貼り付けます。

 **注:** 以降の手順でコーディングするメソッドは、実際には、必要に応じて SilkCentral によって呼び出されます。つまり、ユーザーはこれらのメソッドを直接呼び出しません。

5. getBaseline をコーディングします。このメソッドは、アプリケーションの名前空間/パッケージ/クラス/メソッドをすべて含む XML ファイルを返す必要があります。ファイルの形式については、「サンプル XML データ」トピック ファイルを参照してください。XML は、サンプル XSD ファイルを使用し

て検証する必要があります。XSD については、「コード カバレッジ XSD」トピックを参照してください。

この関数は、カバレッジを開始する前に呼び出されます。テスト実行を開始する SilkCentral 実行サーバーまたは SilkCentral **Manual Test Client** の **開始** ボタンのいずれかによってトリガされるとコード分析が開始され、カバレッジされるすべてのオブジェクトが返されます。この出力は、CA-Framework インストール フォルダに格納されている XML スキーマで指定された形式を使用して、XML に変換する必要があります。

6. `startCoverage` をコーディングします。このメソッドを呼び出すことで、コード カバレッジ ツールにデータの収集開始を指示します。開始すると `true` が返されます。

これは、`getBaseLine()` メソッドが完了した後、SilkCentral コード カバレッジ フレームワークによって呼び出されます。コード カバレッジ ツールは、ここでコード カバレッジ データの収集を開始する必要があります。

7. `stopCoverage` をコーディングします。このメソッドを呼び出すことで、コード カバレッジ ツールにデータの収集停止を指示します。成功すると `true` が返されます。

このメソッドは `startCoverage` の後に呼び出されます。テスト実行を終了する SilkCentral 実行サーバーまたは SilkCentral **Manual Test Client** の **停止** ボタンのいずれかによってトリガされ、コード分析を停止します。

8. `getCoverage` をコーディングします。このメソッドは、`startCoverage` メソッドと `stopCoverage` メソッドの間に収集されたデータを含む XML ファイルを返します。ファイルの形式については、「サンプル XML データ」トピック ファイルを参照してください。XML は、サンプル XSD ファイルを使用して検証する必要があります。XSD については、「コード カバレッジ XSD」トピックを参照してください。

この関数は `stopCoverage()` の後に呼び出され、収集されたすべてのカバレッジ データを返します。この出力は、XML スキーマで指定された形式を使用して、XML に変換する必要があります。

9. `GetName` をコーディングします。このメソッドによって、コード カバレッジ ツールの参照に使用される名前が提供されます。たとえば、この値は **コード分析設定の編集** ダイアログ ボックスの、**コード分析プロファイル** リスト ボックス内の値の 1 つとして使用されます。

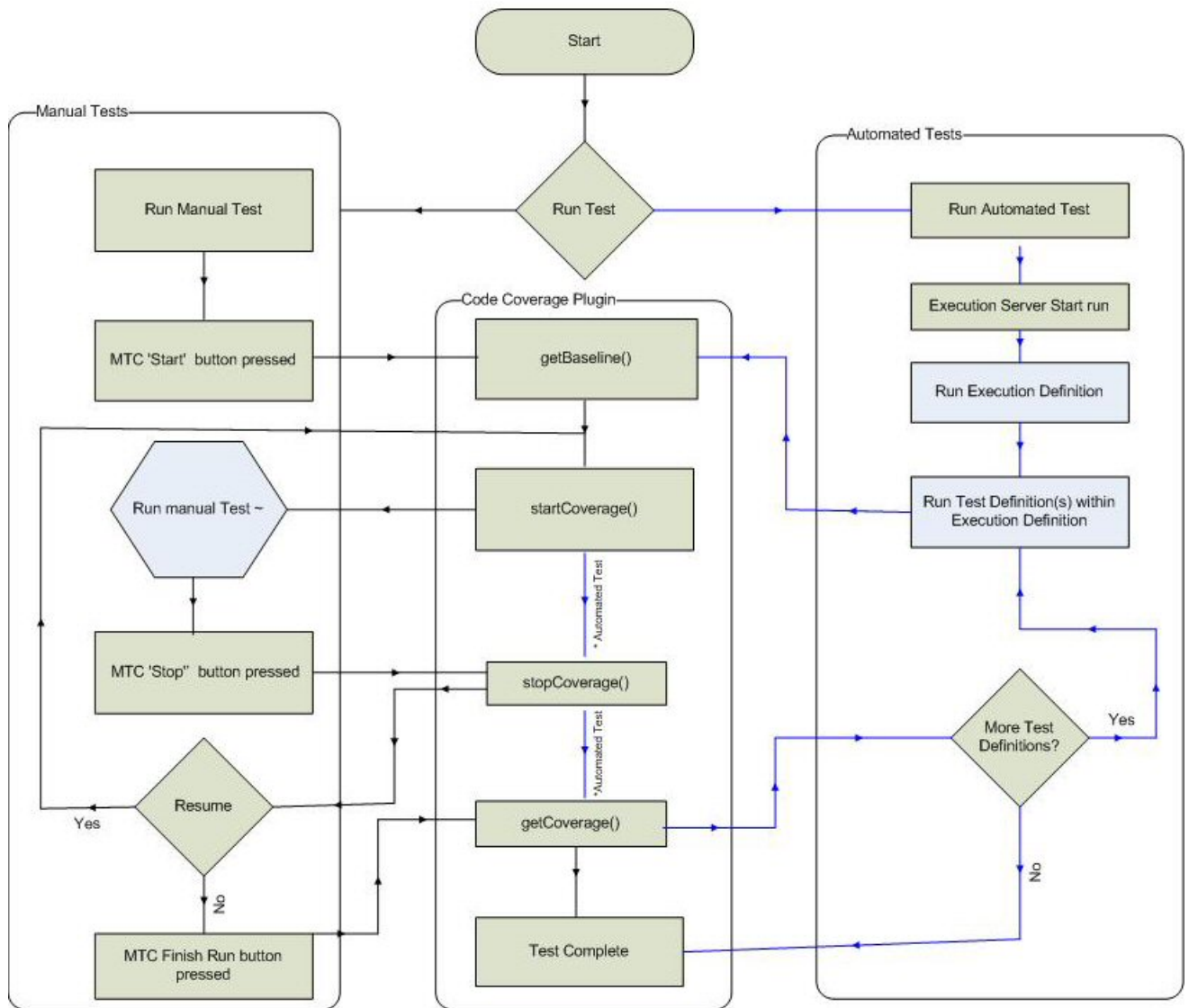
このメソッドは、最初は SilkCentral コード カバレッジ フレームワークによって呼び出されます。プラグインの名前は、SCTM のコード カバレッジ ドロップ ダウンに表示されます。

- 10 プラグインをビルドして jar ファイルを作成し、その jar ファイルを zip ファイルに格納します。

- 11 次のロケーションにプラグインをデプロイします。

- SilkCentral インストール フォルダの Plugins ディレクトリ
- CA-Framework インストール フォルダの Plugins ディレクトリ

コード カバレッジ プラグインの Test Manager コード分析フロー



サンプルプロファイルクラス

次のサンプルファイルは、コードカバレッジプラグインに必要なすべてのメソッド、インポート、および実装の概要を示しています。

```
//Add the library scc.jar to your classpath as it contains the interfaces that
//must be extended. The JAR file can be found in the lib directory of the Test
//Manager installation directory.
//
```

```
//make sure to include these imports after adding the scc.jar external reference
import com.seguscc.published.api.codeanalysis.CodeAnalysisProfile;
import com.seguscc.published.api.codeanalysis.CodeAnalysisProfileException;
import com.seguscc.published.api.codeanalysis.CodeAnalysisResult;
```

```
public class myProfileClass implements CodeAnalysisProfile{
```

```
    // This function is called first by the SCTM Code Coverage framework
    // The name of the plug-in is displayed in the code coverage drop down in SCTM
```

```

@Override
public String getName() {
    // The name of the plugin cannot be an empty string
    return "my plugin name";
}

// This function is called before starting coverage,
// this should return all of the objects to be covered and needs to be
// converted into xml using the format specified in the XML schema
// CodeCoverage.xsd included in the CA-Framework installation folder.
// This is triggered by either the SCTM Execution Server starting a test run
// or the SCTM MTC 'Start' button to start code analysis
@Override
public CodeAnalysisResult getBaseline() throws CodeAnalysisProfileException {
    CodeAnalysisResult result = new CodeAnalysisResult();
    try{
        String baselineData = MyCodeCoverageTool.getAllCoveredObjectsData();
        String xmlString = xmltransformXML(baselineData);
        result.Xml(xmlString);
        String myCustomLogMessage = "Code Coverage baseline successfully retrieved.";
        result.AddLogMsg(myCustomLogMessage);
    }catch(Exception e){
        throw new CodeAnalysisProfileException(e);
    }
    return result;
}

//This function is called by the SCTM Code Coverage Framework after the
getBaseLine() method is complete
// this is where you should start my code coverage tool
// collecting code coverage data

@Override
public boolean startCoverage() throws CodeAnalysisProfileException {
    try{
        MyCodeCoverageTool.StartCollectingCoverageData();
    }catch(Exception e){
        throw new CodeAnalysisProfileException(e);
    }
}

//This function is called after startCoverage,
//This is triggered by either the SCTM Execution Server finishing a test run
// or the SCTM MTC 'Stop' button to stop code analysis
//Call to my code coverage tool to stop collecting data here.
@Override
public boolean stopCoverage() throws CodeAnalysisProfileException {
    try{
        MyCodeCoverageTool.StopCollectingCoverageData();
    }catch(Exception e){
        throw new CodeAnalysisProfileException(e);
    }
}

// This function is called after stopCoverage(),
// this should return all the coverage data collected and needs to be
// converted into xml using the format specified in the XML schema
// CCoverage.xsd included in the CA-Framework installation folder

```

```

@Override
public CodeAnalysisResult getCoverage() throws CodeAnalysisProfileException {
    CodeAnalysisResult result = new CodeAnalysisResult();
    try{
        String coverageData = MyCodeCoverageTool.getActualCoverageData();
        String xmlString = xmltransformXML(coverageData);
        result.Xml(xmlString);
        String myCustomLogMessage = "Code Coverage successfully retrieved.";
        result.AddLogMsg(myCustomLogMessage);
    }catch(Exception e){
        throw new CodeAnalysisProfileException(e);
    }

    return result;
}

private String transformXML(String myData){
    //code to transform from my data to the SCTM needed xml
    ...
    return xmlString;
}
}

```

コード カバレッジ XSD

次に、コード カバレッジ ツールによって生成された XML の検証に使用するコード カバレッジ XSD を示します。このドキュメントがある場所は、<CA Framework installation>%CodeAnalysis %CodeCoverage.xsd です。

```

<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="data" xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="coverage">
    <xs:complexType>
      <!--type will be method when defined as a child to class or line when defined as
a child to method-->
      <xs:attribute name="type" type="xs:string" />
      <!--hits for the definition file will be 0, the update file will define the hits count-->
    >
      <xs:attribute name="hits" type="xs:string" />
      <!--the total count will be sent with both the definition and update file, both
counts will match-->
      <xs:attribute name="total" type="xs:string" />
      <!--this will be an empty string for the definition file, the line numbers will be
sent in the update file delimited by a colon-->
      <xs:attribute name="lines" type="xs:string" />
    </xs:complexType>
  </xs:element>
  <xs:element name="data">
    <xs:complexType>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element ref="coverage" />
        <xs:element name="class">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="sourcefile" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                  <!--full path to the code file-->

```

```

        <xs:attribute name="name" type="xs:string" />
    </xs:complexType>
</xs:element>
<xs:element ref="coverage" minOccurs="0" maxOccurs="unbounded" />
<xs:element name="method" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="coverage" minOccurs="0"
maxOccurs="unbounded" />
        </xs:sequence>
    <!--
        <field_signature> ::= <field_type>
        <field_type>      ::= <base_type>|<object_type>|<array_type>
        <base_type>       ::= B|C|D|F|I|J|S|Z
        <object_type>    ::= L<fullclassname>;
        <array_type>     ::= [<field_type>

The meaning of the base types is as follows:
B byte signed byte
C char character
D double double precision IEEE float
F float single precision IEEE float
I int integer
J long long integer
L<fullclassname>; ... an object of the given class
S short signed short
Z boolean true or false
[<field sig> ... array

        example signature for a java method 'doctypeDecl' with 3
string params and a return type void

                                doctypeDecl : (Ljava/lang/String;Ljava/lang/String;Ljava/lang/
String;)V

                                refer to org.apache.bcel.classfile.Utility for more information on
signatureToString
-->
    <xs:attribute name="name" type="xs:string" />
    <!--method invocation count, this will be 0 for the definition file-->
    <xs:attribute name="inv" type="xs:string" />
</xs:complexType>
</xs:element>
</xs:sequence>
    <xs:attribute name="name" type="xs:string" />
</xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>
</xs:element>
</xs:schema>

```

サンプル XML データ

コード カバレッジ API では、次の形式の XML を想定しています。

XML の検証に提供されているサンプル XSD ドキュメントも使用できます。

```
<?xml version="1.0" encoding="UTF-8"?><!-- Generated by 'MyPluginTool'at  
'2010-11-05T16:11:09'  
-->  
<data>  
  <class name="ProjectA.ClassA1">  
    <sourcefile name="C:¥Documents and Settings¥TestApp¥ProjectA¥ClassA1.cs"/>  
    <coverage hits="8" total="8" type="method"/>  
    <coverage hits="30" total="30" type="line"/>  
    <method inv="2" name="ClassA1 : ()V">  
      <coverage hits="3" lines="11:2,12:2,14:2" total="3" type="line"/>  
    </method>  
    <method inv="2" name="BoolByteMethod : (LSystem/Boolean;LSystem/SByte;)V">  
      <coverage hits="3" lines="17:2,18:2,19:2" total="3" type="line"/>  
    </method>  
    <method inv="1" name="CharStringMethod : (LSystem/Char;LSystem/String;)V">  
      <coverage hits="3" lines="38:1,39:1,40:1" total="3" type="line"/>  
    </method>  
    <method inv="2" name="DateMethod : (LSystem/DateTime;)V">  
      <coverage hits="3" lines="22:2,23:2,24:2" total="3" type="line"/>  
    </method>  
    <method inv="1" name="DecimalMethod : (LSystem/Decimal;LSystem/  
Single;LSystem/Double;)V">  
      <coverage hits="4" lines="27:1,28:1,29:1,30:1" total="4" type="line"/>  
    </method>  
    <method inv="1" name="IntMethod : (LSystem/Int32;LSystem/Int64;LSystem/  
Int16;)V">  
      <coverage hits="3" lines="33:1,34:1,35:1" total="3" type="line"/>  
    </method>  
    <method inv="1" name="passMeArrays : (LSystem/Int32[];LSystem/Decimal[];)V">  
      <coverage hits="6" lines="51:1,52:1,53:1,55:1,56:1,58:1" total="6" type="line"/>  
    </method>  
    <method inv="1" name="passMeObjects : (LSystem/Object;LSystem/Object/  
ClassA1;)V">  
      <coverage hits="5" lines="43:1,44:1,45:1,46:1,48:1" total="5" type="line"/>  
    </method>  
  </class>  
  <class name="TestApp.Form1">  
    <sourcefile name="C:¥Documents and Settings¥TestApp¥Form1.Designer.cs"/>  
    <coverage hits="2" total="10" type="method"/>  
    <coverage hits="24" total="110" type="line"/>  
    <method inv="1" name="btnClassA_Click : (LSystem/Object;LSystem/Object/  
EventArgs;)V">  
      <coverage hits="3" lines="25:1,26:1,27:1" total="3" type="line"/>  
    </method>  
    <method inv="1" name="CallAllClassAMethods : ()V">  
      <coverage hits="21"  
lines="35:1,36:1,37:1,38:1,39:1,40:1,41:1,42:1,43:1,44:1,45:1,46:1,48:1,49:1,50:1,51:1,52  
:1,53:1,54:1,55:1,56:1" total="21" type="line"/>  
    </method>  
  </class>  
</data>
```

コード分析のサンプル インターフェイス クラス

次のサンプルコードでは、コード分析のサンプル インターフェイスを定義しています。

```
package com.seguel.scc.published.api.codeanalysis;

public interface CodeAnalysisProfile
{
    /**
     * This function is called by the SCTM Code Coverage Framework
     * after the getBaseline() method is complete
     * this is where you should start your code coverage tool
     * collecting code coverage data
     * @return true if the operation is successful
     * @throws CodeAnalysisProfileException if an error occurs in this operation
     */
    boolean startCoverage() throws CodeAnalysisProfileException;

    /**
     * This function is called after startCoverage,
     * This is triggered by either the SCTM Execution Server finishing a test run
     * or the SCTM MTC 'Stop' button to stop code analysis
     * Call to your code coverage tool to stop collecting data here.
     * @return true if the operation is successful
     * @throws CodeAnalysisProfileException if an error occurs in this operation
     */
    boolean stopCoverage() throws CodeAnalysisProfileException;

    /**
     * This function is called before starting coverage.
     * This is triggered by either the SCTM Execution Server starting a test run
     * or the SCTM MTC 'Start' button to start code analysis
     * @return all of the objects to be covered. The output needs to be
     * converted into xml using the format specified in the XML schema
     * CCoverage.xsd included in the CA-Framework installation folder
     * @throws CodeAnalysisProfileException if an error occurs in this operation
     */
    String getBaseline() throws CodeAnalysisProfileException;

    /**
     * This function is called after stopCoverage().
     * @return all the coverage data collected. The output needs to be
     * converted into xml using the specified in the XML schema
     * CCoverage.xsd included in the CA-Framework installation folder
     * @throws CodeAnalysisProfileException if an error occurs in this operation
     */
    String getCoverage() throws CodeAnalysisProfileException;

    /**
     * This function is called first by the SCTM Code Coverage framework
     * The name of the plug-in is displayed in the code coverage drop down in SCTM
     * @return the name of the plug-in tool
     */
    String getName();
}
```

Linux AUT 環境にコード分析フレームワークをインストールする

作成するコード カバレッジ プラグインが、Linux オペレーティング システム上の .NET テスト対象アプリケーションと情報をやり取りする場合は、次の手順を実行する必要があります。

1. Linux 用のコード分析フレームワークは、**ヘルプ > ツール > Linux コード分析フレームワーク** から入手できます。これをダウンロードして、Linux マシン上のルート フォルダまたは他の任意のフォルダにコピーします。
2. テスト対象アプリケーションのあるマシンに Java 1.6 以降がインストールされていることを確認します。
3. CA-Framework.tar.gz を抽出します。
4. コード分析プラグインを <install dir>/SC Test Manager 12.0/Plugins フォルダに配置します。
5. ディレクトリを <install dir>/SC Test Manager 12.0/Code Analysis に変更します。
6. CA-Framework プロセスを実行する startCodeAnalysisFramework.sh シェル スクリプトを見つけます。
7. コマンド `dos2unix startCodeAnalysisFramework.sh` を実行して、ファイルを UNIX 形式に変換します。
8. コマンド `chmod 775 startCodeAnalysisFramework.sh` を実行して、シェルスクリプトの実行に必要なアクセス許可を設定します。
9. シェルスクリプト `./startCodeAnalysisFramework.sh` を実行して、CAFramework プロセスを実行します。

Test Manager からコード分析フレームワークを使用する準備が整います。

ソース管理との統合

ソース管理プロファイルにより、Test Manager を外部のソース管理システムと統合できます。Test Manager には、次のソース管理システムのためのプラグインが同梱されています：

- StarTeam®
- Serena® Version Manager™ (PVCS®)
- Concurrent Version System (CVS)
- Microsoft® Visual SourceSafe® (MSVSS)
- Universal Naming Convention (UNC) (ファイル-システム アクセス)
- Subversion
- Apache Commons Virtual File System (VFS)
- Microsoft Team Foundation Server 2010 (TFS)

追加プラグインの詳細については、カスタマー サポートにお問い合わせください。

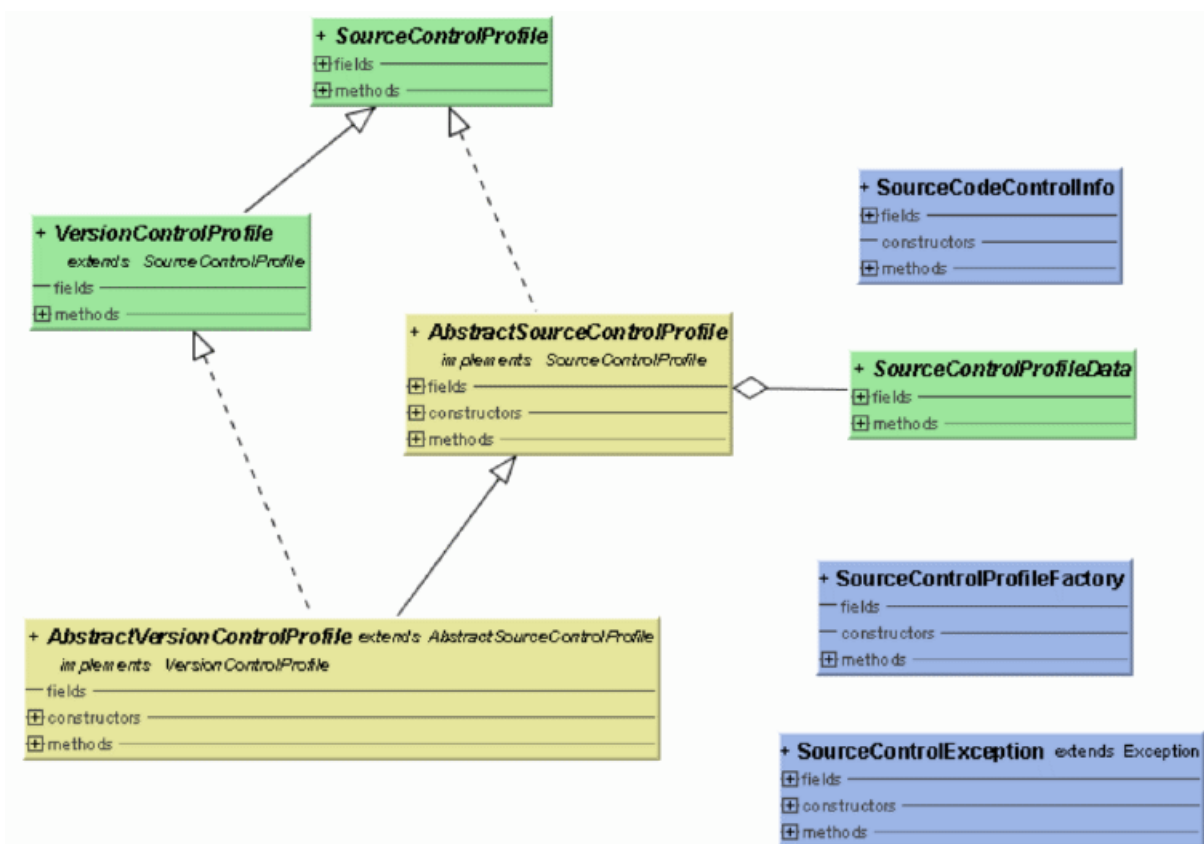
カスタム ソース管理プラグインは、一度デプロイされると、Test Manager 上で構成できるようになります。これにより、Test Manager 実行サーバーが、テストを実行するためにどこからプログラム ソースを取得するかを定義することができます。その他の情報については、『*SilkCentral Test Manager 12.0*』ヘルプを参照してください。

ソース管理との統合のインターフェイス

Test Manager では SourceControlProfile と VersionControlProfile が区別されます。SourceControlProfile はバージョン付けがなく、VersionControlProfile はバージョン付けがある点が異なります。

次は、ソース管理インテグレーションで使用される Test Manager インターフェイスです：

- SourceControlProfile
- VersionControlProfile
- SourceControlProfileData
- SourceControlException
- SourceControlInfo



利用可能な Java クラスやメソッドに関する詳細については、[Javadoc](#) を参照してください。このリンクが動作しなかった場合には、Test Manager メニューの **ヘルプ > ドキュメント > Test Manager API 仕様** を選択して、Javadoc を開いてください。

ソース管理との統合の慣習

各実装は、デフォルト コンストラクタと、SourceControlProfileData をパラメータに持つコンストラクタを提供する必要があります。パラメータ付きのコンストラクタは任意です。このコンストラクタを提供しない場合には、代わりに SourceControlProfileData のための Bean 設定メソッドを用意する必要があります。

各インターフェイスメソッドでは、例外として SourceControlException が指定されているため、インターフェイスで使用されているメソッドでは、いずれも RuntimeException を送出することは許されていません。

問題追跡システムとの統合

本セクションで言及されるインターフェイスは、サードパーティ製（外部）問題追跡システム（ITS : Issue Tracking System）の統合を実現する、Test Manager 用プラグインを作成する際に必要となります。

2つのタイプのインターフェイスが提供されています：

- Web サービス インターフェイス
- Java API インターフェイス

問題追跡プロファイルにより、Test Manager は外部の問題追跡システムを統合することができます。Test Manager で現在サポートされていて特別な設定をせずにそのまま使用できる問題追跡ソフトウェアパッケージは、次のとおりです。

- SilkCentral Issue Manager
- IBM® Rational® ClearQuest®
- StarTeam®
- Issue Tracking Web Service
- Bugzilla
- Atlassian JIRA

追加プラグインの詳細については、カスタマー サポートにお問い合わせください。

問題追跡プロファイルを定義することにより、**テスト**領域内のテストを、サードパーティの問題追跡システム内にある問題へ、リンク付けすることができます。リンク付けされた問題の状態は、サードパーティの問題追跡システムから定期的に更新されます。

その他の情報については、『*Test Manager* ヘルプ』を参照してください。

Web サービス インターフェイス

WSDL がベースとしているインターフェイスを以下に記載してあります。このインターフェイスには、メソッドやタイプに関するドキュメントも記載されています。以下では Java コードでインターフェイスを説明していますが、だからと言って、Web サービスを Java で記述しなければならない、ということではありません。



注: Web サービスでは、どんなプログラミング言語で記述する場合でも、「WSDL 仕様」で定義されているのと同じ SOAP API を提供する必要があります。



注: Web サービス インターフェイスでは、新しい問題の作成のみ行うことができます。既存の問題をテストに割り当てる場合には、Java API を使用する必要があります。

Web サービス インターフェイス

```
/**
 * This interface is used to generate the WSDL of the
 * Web Service a customer can implement
 * to integrate any, in the first place not supported,
 * Issue Tracking System.
 *
 * The WSDL document will be generated out of this
 * interface using Apache Axis Java2WSDL tool
 *
 */
public interface IssueTrackingSOAPInterface {
```

```

/**
 * This method returns an ID that is used for all
 * actions during a session to identify the user. The
 * returned value will be passed to the other methods as
 * parameter.
 * @param user
 * @param password
 * @return a sessionID which is used for all other
 * actions to determine the user
 */
public long logonUser(String user, String password)
    throws IssueTrackingServiceException;

/**
 * Will be called when starting to enter a new issue
 * @param sessionID
 * @return a (temporary) id for the new issue
 */
public String newIssue(long sessionID)
    throws IssueTrackingServiceException;

/**
 * Returns all IssueField objects of the (temporary)
 * issue with the provided issueID.
 * The IssueField objects contain the current values
 * of the fields.
 * @param sessionID
 * @param issueID
 * @return
 */
public IssueField[] getFields(long sessionID,
    String issueID)
    throws IssueTrackingServiceException;

/**
 * This method is used to synchronize with the current
 * states of the external tracking tool.
 * @param sessionID
 * @param issueIDs array containing the ids of the
 * requested issues
 * @return state info of requested issues [0] id, [1]
 * state, [2] longvalue (changedAt)
 */
public String[][] getStates(long sessionID,
    String[] issueIDs)
    throws IssueTrackingServiceException;

/**
 * This method is used to get all possible states an
 * issue may have to do a mapping between internal and
 * external states.
 * @param sessionID
 * @return all states an issue may have in this
 * profile
 */
public String[] getPossibleStates(long sessionID)
    throws IssueTrackingServiceException;

```

```

/**
 * This method is called if a field with type SELECT_
 * FIELD has dependent fields and was changed.
 * e.g.: The field "Product" has a depending field
 * "Release". If "Product" changes, the IssueField
 * for "Release" has to be returned containing the
 * proper options for the selected product.
 * @param sessionID
 * @param issueID (temporary) id of the issue
 * @param field that has changed
 * @return all fields of the issue that have changed
 * (eg. other options) due to the change of the submitted
 * field
 */
public IssueField[] fieldChanged(long sessionID,
    String issueID, IssueField field)
    throws IssueTrackingServiceException;

/**
 * this method stores the issue with the values which
 * are submitted in the fields array.
 *
 * @param sessionID
 * @param id the (temporary) id of the issue
 * @param fields array of fields containing the values
 * to store
 * @return new id of the issue after storing into the
 * tracking system
 */
public String store(long sessionID, String id,
    IssueField[] fields)
    throws IssueTrackingServiceException;

/**
 * This method returns the Http link to the issue with
 * the submitted id if available.
 * @param sessionID
 * @param id
 * @return http link to the issue, null if either no
 * link is available or the id is invalid
 */
public String getLink(long sessionID, String id)
    throws IssueTrackingServiceException;

/**
 * Not used in this version. May be used in a future
 * version to speed up the update process.
 * @param product only info of issues linked to this
 * product will be returned
 * @param changedSince timeMillis, only info of
 * issues changed after will be returned.
 * Can be ignored by
 * implementing services if not supported,
 * but then all issues of this
 * product will be returned (may slow down updates).
 * @return array with actual info about the changed
 * issues
 */

```

```

public IssueInfo[] getChanges(String product,
    long changedSince)
    throws IssueTrackingServiceException;
}

/**
 * The ComplexType IssueField corresponds to the following
 * definition.
 */

/**
 * An IssueField represents one input field in the
 * issue tracking dialog.
 * @param name A unique name for the field (ID)
 * @param type The GUI type of the field. SelectBox =
 * 1, Checkbox = 2, TextField = 3, TextArea = 4 (handled as TextField in this version)
 * @param value The current value of the field.
 * @param options The options for a field of type
 * SelectBox (type = 1)
 * @param label The label for the field shown in the
 * dialog
 * @param required Defines whether the user has to
 * enter a value for this field or if it can be left blank
 * @param length The maximum number of characters that
 * can be entered in a TextField or TextArea
 * @param dependent Array of names of other fields
 * whose content depend on the selection of this field.
 * Used for SelectBoxes (type = 1)
 */

/**
 * The ComplexType IssueInfo is not used in this version as
 * the method getChanges is not yet called.
 */

```

WSDL 仕様

WSDL ファイル（以下に添付）では、統合を実現するために実装しなければならない SOAP-API が記述されています。この WSDL を実装した Web サービスをホストする必要があります。

Web サービス インターフェイス

以下が、Web サービス インターフェイスの仕様です。このファイルは、Apache のツール、*Java2WSDL* を使用して生成したものです。

```

<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
targetNamespace="http://scc.seguate.com/issuetracking"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://scc.seguate.com/issuetracking"
xmlns:intf="http://scc.seguate.com/issuetracking"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <wsdl:types>
    <schema targetNamespace="http://scc.seguate.com/issuetracking"
xmlns="http://www.w3.org/2001/XMLSchema">

```

```

<import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
<complexType name="IssueField">
  <sequence>
    <element name="Name" nillable="true" type="xsd:string"/>

    <element name="Type" type="xsd:int"/>
    <element name="Value" nillable="true" type="xsd:string"/>
    <element maxOccurs="unbounded" name="Options"
nillable="true" type="xsd:string"/>
    <element name="Label" nillable="true" type="xsd:string"/>
    <element name="IsRequired" type="xsd:boolean"/>
    <element name="MaxLength" type="xsd:int"/>
    <element maxOccurs="unbounded" name="Dependent"
nillable="true" type="xsd:string"/>
  </sequence>
</complexType>
<complexType name="ArrayOfIssueField">
  <complexContent>
    <restriction base="soapenc:Array">
      <attribute ref="soapenc:arrayType"
wsdl:arrayType="impl:IssueField[]"/>
    </restriction>
  </complexContent>
</complexType>
<complexType name="IssueTrackingServiceException">
  <sequence/>
</complexType>
<complexType name="ArrayOf_xsd_string">
  <complexContent>
    <restriction base="soapenc:Array">
      <attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:string[]"/>
    </restriction>
  </complexContent>
</complexType>
<complexType name="ArrayOfArrayOf_xsd_string">
  <complexContent>
    <restriction base="soapenc:Array">
      <attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:string[][]"/>
    </restriction>
  </complexContent>
</complexType>
<complexType name="IssueInfo">
  <sequence>
    <element name="Id" nillable="true" type="xsd:string"/>
    <element name="Status" nillable="true" type="xsd:string"/>
    <element name="Synopsis" nillable="true" type="xsd:string"/>
    <element name="Product" nillable="true" type="xsd:string"/>
    <element name="Component" nillable="true" type="xsd:string"/>
    <element name="Platform" nillable="true" type="xsd:string"/>
    <element name="Url" nillable="true" type="xsd:string"/>
    <element name="ChangedOn" type="xsd:long"/>
    <element name="ChangedBy" nillable="true" type="xsd:string"/>
  </sequence>
</complexType>
<complexType name="ArrayOfIssueInfo">
  <complexContent>
    <restriction base="soapenc:Array">
      <attribute ref="soapenc:arrayType" wsdl:arrayType="impl:IssueInfo[]"/>
    </restriction>
  </complexContent>
</complexType>

```

```

    </restriction>
  </complexContent>
</complexType>
</schema>
</wsdl:types>
<wsdl:message name="getChangesRequest">
  <wsdl:part name="product" type="xsd:string"/>
  <wsdl:part name="changedSince" type="xsd:long"/>
</wsdl:message>
<wsdl:message name="fieldChangedResponse">
  <wsdl:part name="fieldChangedReturn" type="impl:ArrayOfIssueField"/>
</wsdl:message>
<wsdl:message name="getFieldsResponse">
  <wsdl:part name="getFieldsReturn" type="impl:ArrayOfIssueField"/>
</wsdl:message>
<wsdl:message name="getPossibleStatesResponse">
  <wsdl:part name="getPossibleStatesReturn" type="impl:ArrayOf_xsd_string"/>
</wsdl:message>
<wsdl:message name="logonUserResponse">
  <wsdl:part name="logonUserReturn" type="xsd:long"/>
</wsdl:message>
<wsdl:message name="getStatesResponse">
  <wsdl:part name="getStatesReturn" type="impl:ArrayOfArrayOf_xsd_string"/>
</wsdl:message>
<wsdl:message name="getPossibleStatesRequest">
  <wsdl:part name="sessionID" type="xsd:long"/>
</wsdl:message>
<wsdl:message name="newIssueResponse">
  <wsdl:part name="newIssueReturn" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="getChangesResponse">
  <wsdl:part name="getChangesReturn" type="impl:ArrayOfIssueInfo"/>
</wsdl:message>
<wsdl:message name="logonUserRequest">
  <wsdl:part name="user" type="xsd:string"/>
  <wsdl:part name="password" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="getFieldsRequest">
  <wsdl:part name="sessionID" type="xsd:long"/>
  <wsdl:part name="issueID" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="storeResponse">
  <wsdl:part name="storeReturn" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="getStatesRequest">
  <wsdl:part name="sessionID" type="xsd:long"/>
  <wsdl:part name="issueIDs" type="impl:ArrayOf_xsd_string"/>
</wsdl:message>
<wsdl:message name="getLinkRequest">
  <wsdl:part name="sessionID" type="xsd:long"/>
  <wsdl:part name="id" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="newIssueRequest">
  <wsdl:part name="sessionID" type="xsd:long"/>
</wsdl:message>
<wsdl:message name="IssueTrackingServiceException">
  <wsdl:part name="fault" type="impl:IssueTrackingServiceException"/>
</wsdl:message>

```

```

<wsdl:message name="fieldChangedRequest">
  <wsdl:part name="sessionID" type="xsd:long"/>
  <wsdl:part name="issueID" type="xsd:string"/>
  <wsdl:part name="field" type="impl:IssueField"/>
</wsdl:message>
<wsdl:message name="getLinkResponse">
  <wsdl:part name="getLinkReturn" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="storeRequest">
  <wsdl:part name="sessionID" type="xsd:long"/>
  <wsdl:part name="id" type="xsd:string"/>
  <wsdl:part name="fields" type="impl:ArrayOfIssueField"/>
</wsdl:message>
<wsdl:portType name="IssueTrackingPortType">
  <wsdl:operation name="getFields" parameterOrder="sessionID issueID">
    <wsdl:input message="impl:getFieldsRequest" name="getFieldsRequest"/>
    <wsdl:output message="impl:getFieldsResponse" name="getFieldsResponse"/>
    <wsdl:fault message="impl:IssueTrackingServiceException"
name="IssueTrackingServiceException"/>
  </wsdl:operation>
  <wsdl:operation name="store" parameterOrder="sessionID id fields">
    <wsdl:input message="impl:storeRequest" name="storeRequest"/>
    <wsdl:output message="impl:storeResponse" name="storeResponse"/>
    <wsdl:fault message="impl:IssueTrackingServiceException"
name="IssueTrackingServiceException"/>
  </wsdl:operation>
  <wsdl:operation name="logonUser" parameterOrder="user password">
    <wsdl:input message="impl:logonUserRequest" name="logonUserRequest"/>
    <wsdl:output message="impl:logonUserResponse" name="logonUserResponse"/>
    <wsdl:fault message="impl:IssueTrackingServiceException"
name="IssueTrackingServiceException"/>
  </wsdl:operation>
  <wsdl:operation name="newIssue" parameterOrder="sessionID">
    <wsdl:input message="impl:newIssueRequest" name="newIssueRequest"/>
    <wsdl:output message="impl:newIssueResponse" name="newIssueResponse"/>
    <wsdl:fault message="impl:IssueTrackingServiceException"
name="IssueTrackingServiceException"/>
  </wsdl:operation>
  <wsdl:operation name="getStates" parameterOrder="sessionID issueIDs">
    <wsdl:input message="impl:getStatesRequest" name="getStatesRequest"/>
    <wsdl:output message="impl:getStatesResponse" name="getStatesResponse"/>
    <wsdl:fault message="impl:IssueTrackingServiceException"
name="IssueTrackingServiceException"/>
  </wsdl:operation>
  <wsdl:operation name="getPossibleStates" parameterOrder="sessionID">
    <wsdl:input message="impl:getPossibleStatesRequest"
name="getPossibleStatesRequest"/>
    <wsdl:output message="impl:getPossibleStatesResponse"
name="getPossibleStatesResponse"/>
    <wsdl:fault message="impl:IssueTrackingServiceException"
name="IssueTrackingServiceException"/>
  </wsdl:operation>
  <wsdl:operation name="fieldChanged" parameterOrder="sessionID issueID field">
    <wsdl:input message="impl:fieldChangedRequest" name="fieldChangedRequest"/>
    <wsdl:output message="impl:fieldChangedResponse" name="fieldChangedResponse"/>
    <wsdl:fault message="impl:IssueTrackingServiceException"
name="IssueTrackingServiceException"/>
  </wsdl:operation>

```

```

    <wsdl:operation name="getLink" parameterOrder="sessionID id">
      <wsdl:input message="impl:getLinkRequest" name="getLinkRequest"/>
      <wsdl:output message="impl:getLinkResponse" name="getLinkResponse"/>
      <wsdl:fault message="impl:IssueTrackingServiceException"
name="IssueTrackingServiceException"/>
    </wsdl:operation>
    <wsdl:operation name="getChanges" parameterOrder="product changedSince">
      <wsdl:input message="impl:getChangesRequest" name="getChangesRequest"/>
      <wsdl:output message="impl:getChangesResponse" name="getChangesResponse"/>
      <wsdl:fault message="impl:IssueTrackingServiceException"
name="IssueTrackingServiceException"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="IssueTrackingSOAPBinding" type="impl:IssueTrackingPortType">
    <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="getFields">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="getFieldsRequest">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://scc.seguate.com/issuetracking" use="encoded"/>
      </wsdl:input>
      <wsdl:output name="getFieldsResponse">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://scc.seguate.com/issuetracking" use="encoded"/>
      </wsdl:output>
      <wsdl:fault name="IssueTrackingServiceException">
        <wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://scc.seguate.com/issuetracking" use="encoded"/>
      </wsdl:fault>
    </wsdl:operation>
    <wsdl:operation name="store">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="storeRequest">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://scc.seguate.com/issuetracking" use="encoded"/>
      </wsdl:input>
      <wsdl:output name="storeResponse">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://scc.seguate.com/issuetracking" use="encoded"/>
      </wsdl:output>
      <wsdl:fault name="IssueTrackingServiceException">
        <wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://scc.seguate.com/issuetracking" use="encoded"/>
      </wsdl:fault>
    </wsdl:operation>
    <wsdl:operation name="logonUser">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="logonUserRequest">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://scc.seguate.com/issuetracking" use="encoded"/>
      </wsdl:input>
      <wsdl:output name="logonUserResponse">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://scc.seguate.com/issuetracking" use="encoded"/>
      </wsdl:output>
      <wsdl:fault name="IssueTrackingServiceException">
        <wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://scc.seguate.com/issuetracking" use="encoded"/>

```

```

</wsdl:fault>
</wsdl:operation>
<wsdl:operation name="newIssue">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="newIssueRequest">
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://scc.seguate.com/issuetracking" use="encoded"/>
  </wsdl:input>
  <wsdl:output name="newIssueResponse">
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://scc.seguate.com/issuetracking" use="encoded"/>
  </wsdl:output>
  <wsdl:fault name="IssueTrackingServiceException">
    <wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://scc.seguate.com/issuetracking" use="encoded"/>
  </wsdl:fault>
</wsdl:operation>
<wsdl:operation name="getStates">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="getStatesRequest">
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://scc.seguate.com/issuetracking" use="encoded"/>
  </wsdl:input>
  <wsdl:output name="getStatesResponse">
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://scc.seguate.com/issuetracking" use="encoded"/>
  </wsdl:output>
  <wsdl:fault name="IssueTrackingServiceException">
    <wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://scc.seguate.com/issuetracking" use="encoded"/>
  </wsdl:fault>
</wsdl:operation>
<wsdl:operation name="getPossibleStates">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="getPossibleStatesRequest">
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://scc.seguate.com/issuetracking" use="encoded"/>
  </wsdl:input>
  <wsdl:output name="getPossibleStatesResponse">
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://scc.seguate.com/issuetracking" use="encoded"/>
  </wsdl:output>
  <wsdl:fault name="IssueTrackingServiceException">
    <wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://scc.seguate.com/issuetracking" use="encoded"/>
  </wsdl:fault>
</wsdl:operation>
<wsdl:operation name="fieldChanged">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="fieldChangedRequest">
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://scc.seguate.com/issuetracking" use="encoded"/>
  </wsdl:input>
  <wsdl:output name="fieldChangedResponse">
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://scc.seguate.com/issuetracking" use="encoded"/>
  </wsdl:output>
  <wsdl:fault name="IssueTrackingServiceException">

```

```

        <wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://scc.segue.com/issuetracking" use="encoded"/>
    </wsdl:operation>
    <wsdl:operation name="getLink">
        <wsdlsoap:operation soapAction=""/>
        <wsdl:input name="getLinkRequest">
            <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://scc.segue.com/issuetracking" use="encoded"/>
        </wsdl:input>
        <wsdl:output name="getLinkResponse">
            <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://scc.segue.com/issuetracking" use="encoded"/>
        </wsdl:output>
        <wsdl:fault name="IssueTrackingServiceException">
            <wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://scc.segue.com/issuetracking" use="encoded"/>
        </wsdl:fault>
    </wsdl:operation>
    <wsdl:operation name="getChanges">
        <wsdlsoap:operation soapAction=""/>
        <wsdl:input name="getChangesRequest">
            <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://scc.segue.com/issuetracking" use="encoded"/>
        </wsdl:input>
        <wsdl:output name="getChangesResponse">
            <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://scc.segue.com/issuetracking" use="encoded"/>
        </wsdl:output>
        <wsdl:fault name="IssueTrackingServiceException">
            <wsdlsoap:fault encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://scc.segue.com/issuetracking" use="encoded"/>
        </wsdl:fault>
    </wsdl:operation>
</wsdl:binding>
<wsdl:service name="IssueTrackingService">
    <wsdl:port binding="impl:IssueTrackingSOAPBinding" name="IssueTrackingPort">
        <wsdlsoap:address location="http://issuetracking.endpoint"/>
    </wsdl:port>
</wsdl:service>
</wsdl:definitions>

```

Web サービスプロファイルを設定する

一旦、作成したカスタム問題追跡 Web サービスをホストした後は、問題追跡プロファイルを設定することができます。

1. メニューで **プロジェクト設定 > 問題追跡** をクリックします。システムに対して作成されている問題追跡プロファイルがすべて一覧されている、**問題追跡** ページが開きます。
2. 新しい問題追跡プロファイルの **名前** と **説明** を入力します。
3. **問題追跡 Web サービス** を **タイプ** リストボックスから選択します。
4. **エンドポイント** フィールドに、ホストされている Web サービスの URL を入力します。
5. 有効な **ユーザー名** と **パスワード** を、それぞれのフィールドに入力します。
6. **OK** をクリックします。

Java API インターフェイス

利用可能な Java クラスやメソッドに関する詳細については、[Javadoc](#) を参照してください。このリンクが動作しなかった場合には、Test Manager メニューの **ヘルプ > ドキュメント > Test Manager API 仕様** を選択して、Javadoc を開いてください。

ビルド環境

ライブラリ scc.jar に拡張する必要があるインターフェイス群が含まれているため、このライブラリをクラスパスに追加します。この JAR ファイルは、Test Manager のインストールディレクトリ以下の lib ディレクトリ内にあります。

2 つの インターフェイス/クラスを拡張する必要があります :

- com.segue.scc.published.api.issuetracking82.IssueTrackingProfile
- com.segue.scc.published.api.issuetracking.Issue

クラス/インターフェイス



- IssueTrackingProfile
- IssueTrackingData
- Issue
- IssueTrackingField
- IssueTrackingProfileException

要件管理システムとの統合

Test Manager は、サードパーティの要件管理システム（RMS）ツールと統合し、要件間のリンク付けや同期化を行うことができます。

このセクションでは、Test Manager 上の要件を、サードパーティの要件管理システム上の要件と同期化するための、サードパーティ製プラグインの実装に使用する、Java アプリケーション プログラミング インターフェイスについて説明します。このセクションでは、要件プラグインとそのデプロイメントを識別するインターフェイスについて説明します。

JAR または ZIP ファイルは、Test Manager の標準プラグインの概念に適合する形で提供されます。つまり、すべてのフロントエンド サーバーに対して自動的にデプロイされ、設定や同期化のためのサードパーティ ツールにアクセスできるようになります。このプラグインは特定のインターフェイスを実装しており、これにより Test Manager は、そのプラグインが要件プラグインであると識別でき、サードパーティ ツールへログインするために必要な、ログイン プロパティを提供します。

利用可能な Java クラスやメソッドに関する詳細については、[Javadoc](#) を参照してください。このリンクが動作しなかった場合には、Test Manager メニューの **ヘルプ > ドキュメント > Test Manager API 仕様** を選択して、Javadoc を開いてください。

追加プラグインの詳細については、カスタマー サポートにお問い合わせください。

その他の情報については、『*SilkCentral* 管理モジュール』ヘルプを参照してください。

Java API インターフェイス

利用可能な Java クラスやメソッドに関する詳細については、[Javadoc](#) を参照してください。このリンクが動作しなかった場合には、Test Manager メニューの **ヘルプ > ドキュメント > Test Manager API 仕様** を選択して、Javadoc を開いてください。

最初に取り扱う基本インターフェイスは `RMPluginProfile` (`com.seguate.tm.published.api.requirements.javaplugin`) です。RMPluginProfile で、このプラグインが要件プラグインとして指定されています。

要件 Java プラグイン API には、次の追加インターフェイスが含まれます。

- `RMAction`
- `RMAttachment`
- `RMDataProvider`
- 省略可能 : `RMIconProvider`
- `RMNode`
- `RMNodeType`
- `RMPluginProfile`
- `RMProject`
- `RMTest`
- `RMTestParameter`
- 省略可能 : `RMTestProvider`
- `RMTestStep`

サードパーティ テスト タイプとの統合

Test Manager では、最初から利用可能なテスト タイプの標準セット以外のテスト タイプ --- SilkPerformer、SilkTest Classic、手動テスト、JUnit、JUnit、Windows Scripting Host (WSH) など --- のためのカスタム プラグインを作成することができます。新しいテスト タイプ プラグインを作成すると、そのカスタム テスト タイプが、Test Manager における新しいテストの作成に使用できる標準テスト タイプと一緒に、**テストの新規作成** ダイアログ ボックスの **タイプ** リスト ボックスから利用できるようになります。

プラグインは、テストを定義したり、テスト実行を実装したりするために必須のプロパティを指定します。プロパティに関するメタ情報は、「XML ファイルの設定」によって定義されます。

このプラグイン アプローチの目的は、共通テスト フレームワーク (JUnit、JUnit、スクリプト言語 (WSH) など) をベースとするテストをサポートし、ユーザー固有のテスト環境に合わせた容易な Test Manager のカスタマイゼーションを可能にすることです。Test Manager の洗練されたパブリック API により、ユーザー固有の自動化テストのニーズに見合ったソリューションを実装することができます。Test Manager は、オープンであり、Java 実装またはコマンドライン呼び出しから呼び出すことのできるサードパーティ ツールに対して拡張することができます。

利用可能な Java クラスやメソッドに関する詳細については、[Javadoc](#) を参照してください。このリンクが動作しなかった場合には、Test Manager メニューの **ヘルプ** > **ドキュメント** > **Test Manager API 仕様** を選択して、Javadoc を開いてください。

Javadoc に記述されたクラスは、tm-testlaunchapi.jar ファイルに含まれています。

追加プラグインの詳細については、カスタマー サポートにお問い合わせください。

このセクションには、process executor テスト タイプを実装するコード サンプルが含まれています。process executor は、あらゆる実行可能ファイルの起動に使用することができ、処理テストランチャ クラスを拡張します。その他の情報については、**ヘルプ** > **ツール** から テスト起動プラグイン サンプル をダウンロードし、Readme.txt をお読みください。

プラグイン実装

この API の原則は、よく知られている Java Beans の概念を元にしてしています。これにより、開発者が容易にテスト起動プラグインを実装できるようになっています。テキスト形式の情報を Java コードに入れなくてもすむように、プロパティに関するメタ情報は、XML ファイル内で定義することができます。

プラグインの実装では、統合されるためのコールバック インターフェイスを実装し、ZIP アーカイブ内にパッケージングします。他のインターフェイスは、代わりにプラグイン フレームワークで提供されており、プラグイン実装は情報にアクセスしたり、結果を取得したりすることができます。

パッケージング

プラグインは、Java コードベースと XML 設定ファイルを持つ ZIP アーカイブにパッケージングします。このアーカイブには、テスト起動プラグイン実装も一部含まれます。このコードベースは、Java アーカイブ ファイル (.jar) に収めるか、Java のパッケージ構造をそのまま表すフォルダ階層内のクラス ファイル (.class) に直接入れます。

TestLaunchBean プラグイン クラスは、Bean の標準どおり、TestLaunchBean インターフェイスを実装しています。zip アーカイブ内の XML 設定ファイルには、このクラスと同じ名前を付けます。これにより、複数のプラグインと XML ファイルを、1 つのアーカイブ内にパッケージングすることができます。

プラグインへパラメータを渡す

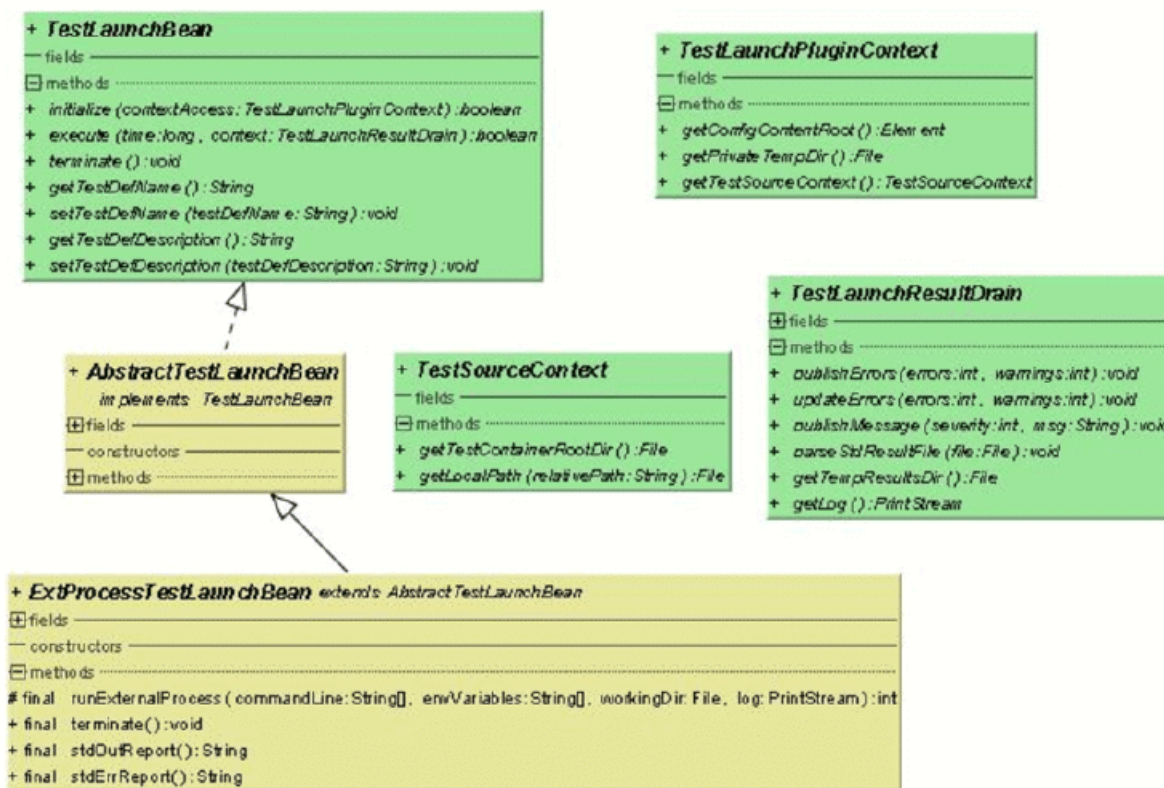
プラグイン インターフェイスは、Test Manager テスト 領域で定義されている、すべてのカスタム パラメータへのフルアクセスを提供します。カスタム パラメータは、サードパーティのテスト タイプに対してのみサポートされています。プラグインは、事前定義パラメータを指定することができないので、特定のテストに対して、パラメータが定義されているか、どのように定義されるかは、プラグイン実装によって決まります。

TestLaunchPluginContext インターフェイスにある getParameterValueStrings() メソッドを使用して、パラメータ名 (キー) からそれに対応する値 (String で表されます) へのマッピングを保持するコンテナを取得します。

JUnit テスト タイプでは、任意の JUnit テスト クラスから、Java のシステム プロパティとして、元のテストのカスタム パラメータにアクセスすることができます。ランチャは、-D という VM 引数を使って、実行する仮想マシンにこのパラメータを渡します。

API の構造

次の画像は、API for サードパーティ テスト タイプ インテグレーションのための API の構造を表したものです。



利用可能な Java クラスやメソッドに関する詳細については、[Javadoc](#) を参照してください。このリンクが動作しなかった場合には、Test Manager メニューの **ヘルプ > ドキュメント > Test Manager API 仕様** を選択して、Javadoc を開いてください。

利用可能なインターフェイス

- TestLaunchBean

- ExtProcessTestLaunchBean
- TestLaunchPluginContext
- TestSourceContext
- TestLaunchResultDrain

サンプルコード

このサンプルコードブロックは、あらゆる実行可能ファイルの起動に使用することができる process executor テストタイプを実装しています。これは、公開 ExtProcessTestLaunchBean クラスを拡張します。

```
package com.borland.sctm.testlauncher;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;

import com.segure.tm.published.api.testlaunch.ExtProcessTestLaunchBean;
import com.segure.tm.published.api.testlaunch.TestLaunchResultDrain;

/**
 * Implements an SCTM test type that can be used to launch
 * any executables, for example from a command line.
 * Extends SCTM published process test launcher class,
 * see Test Manager API Specification (in Help -> Documentation) for
 * further details.
 */
public class ProcessExecutor extends ExtProcessTestLaunchBean {

    // test properties that will be set by SCTM using appropriate setter
    // methods (bean convention),
    // property names must conform to property tags used in the XML file

    /**
     * Represents property <command> defined in ProcessExecutor.xml.
     */
    private String command;

    /**
     * Represents property <arguments> defined in ProcessExecutor.xml.
     */
    private String arguments;

    /**
     * Represents property <workingfolder> defined in ProcessExecutor.xml.
     */
    private String workingfolder;

    /**
     * Java Bean compliant setter used by SCTM to forward the command to be
     * executed.
     * Conforms to property <command> defined in ProcessExecutor.xml.
     */
    public void setCommand(String command) {
        this.command = command;
    }
}

/**
```

```

* Java Bean compliant setter used by SCTM to forward the arguments
* that will be passed to the command.
* Conforms to property <arguments> defined in ProcessExecutor.xml.
*/
public void setArguments(String arguments) {
    this.arguments = arguments;
}

/**
* Java Bean compliant setter used by SCTM to forward the working
* folder where the command will be executed.
* Conforms to property <workingfolder> defined in
* ProcessExecutor.xml.
*/
public void setWorkingfolder(String workingfolder) {
    this.workingfolder = workingfolder;
}

/**
* Main plug-in method. See Test Manager API Specification
* (in Help &gt; Documentation) for further details.
*/
@Override
public boolean execute(long time, TestLaunchResultDrain context)
    throws InterruptedException {
    try {
        String[] cmd = getCommandArgs(context);
        File workingDir = getWorkingFolderFile(context);
        String[] envVariables = getEnvironmentVariables(context);

        int processExitCode = runExternalProcess(cmd, envVariables, workingDir,
            context.getLog());

        boolean outputXmlFound = handleOutputXmlIfExists(context);

        if (! outputXmlFound && processExitCode != 0) {
            // if no output.xml file was produced, the exit code indicates
            // success or failure
            context.publishMessage(TestLaunchResultDrain.SEVERITY_ERROR,
                "Process exited with return code "
                + String.valueOf(processExitCode));
            context.updateErrors(1, 0);
            // set error, test will get status 'failed'
        }
    } catch (IOException e) {
        // prints exception message to Messages tab in Test Run
        // Results
        context.publishMessage(TestLaunchResultDrain.SEVERITY_FATAL,
            e.getMessage());
        // prints exception stack trace to 'log.txt' that can be viewed in Files
        // tab
        e.printStackTrace(context.getLog());
        context.publishErrors(1, 0);
        return false; // set test status to 'not executed'
    }
    return true;
}

```

```

/**
 * Initializes environment variables to be set additionally to those
 * inherited from the system environment of the Execution Server.
 * @param context the test execution context
 * @return String array containing the set environment variables
 * @throws IOException
 */
private String[] getEnvironmentVariables(TestLaunchResultDrain context)
throws IOException {
    String[] envVariables = {
        "SCTM_EXEC_RESULTS_FOLDER="
        + context.getTempResultsDir().getAbsolutePath(),
        "SCTM_EXEC_SOURCES_FOLDER="
        + sourceAccess().getTestContainerRootDir().getAbsolutePath(),
    };
    return envVariables;
}

/**
 * Let SCTM parse the standard report xml file (output.xml) if exists.
 * See also Test Manager Web Help - Creating a Test Package. A XSD file
 * can be found in SCTM Help > Tools > Test Package XML Schema
 * Definition File
 * @param context the test execution context
 * @return true if output.xml exists
 * @throws IOException
 */
private boolean handleOutputXmlIfExists(TestLaunchResultDrain context)
throws IOException {
    String outputFileName = context.getTempResultsDir().getAbsolutePath()
    + File.separator + TestLaunchResultDrain.OUTPUT_XML_RESULT_FILE;
    File outputfile = new File(outputFileName);
    boolean outputXmlExists = outputfile.exists();
    if (outputXmlExists) {
        context.parseStdResultFile(outputfile);
        context.publishMessage(TestLaunchResultDrain.SEVERITY_INFO,
            String.format("output.xml parsed from '%s'", outputFileName));
    }
    return outputXmlExists;
}

/**
 * Retrieves the working folder on the Execution Server. If not configured
 * the results directory is used as working folder.
 * @param context the test execution context
 * @return the working folder file object
 * @throws IOException
 */
private File getWorkingFolderFile(TestLaunchResultDrain context)
throws IOException {
    final File workingFolderFile;
    if (workingfolder != null) {
        workingFolderFile = new File(workingfolder);
    } else {
        workingFolderFile = context.getTempResultsDir();
    }
    context.publishMessage(TestLaunchResultDrain.SEVERITY_INFO,
        String.format("process is executed in working folder '%s'",

```

```

        workingFolderFile.getAbsolutePath());
    return workingFolderFile;
}

/**
 * Retrieves the command line arguments specified.
 * @param context the test execution context
 * @return an array of command line arguments
 */
private String[] getCommandArgs(TestLaunchResultDrain context) {
    final ArrayList<String> cmdList = new ArrayList<String>();
    final StringBuilder cmd = new StringBuilder();
    cmdList.add(command);
    cmd.append(command);
    if (arguments != null) {
        String[] lines = arguments.split("[¥¥r¥¥n]+");
        for (String line : lines) {
            cmdList.add(line);
            cmd.append(" ").append(line);
        }
    }
    context.publishMessage(TestLaunchResultDrain.SEVERITY_INFO,
        String.format("executed command '%s'", cmd.toString()));
    context.getLog().printf("start '%s'%n", cmd.toString());
    return (String[]) cmdList.toArray(new String[cmdList.size()]);
}
}
}

```

設定 XML ファイル

サードパーティ テスト タイプ プラグインについてのメタ情報が含まれている、設定 XML ファイルについて説明します。

プラグインのメタ情報

プラグイン メタ情報は、概して、そのプラグインとテストのタイプに関する情報を提供します。利用可能な情報タイプは、次のとおりです。

タイプ	説明
id	インストールされているすべてのプラグイン内で一意の内部文字列で、タイプを識別します。
label	選択リストや編集ダイアログに表示される、このタイプ用 GUI テキスト。
description	このテスト タイプを説明する補足テキスト情報。
version	タイプの個々のバージョンを区別するためのもの。

一般プロパティのメタ情報

編集可能プロパティには、各プロパティ タイプで同じ一般情報があります（このほかに、タイプ固有の情報があります）。プロパティの名前は、コード内で `get<<propertyname>>` メソッドによって定義されている名前と一致しなければなりません。このときその名前の最初の文字は小文字でなければなりません。

タイプ	説明
label	GUI に表示されるテキスト。
description	プロパティを説明する補足テキスト情報。
isOptional	値が true の場合、ユーザー入力は必須ではない、ということを示します。
default	新しいテストを作成する際に使用される、このプロパティのデフォルト値。この値は、プロパティタイプに合っている必要があります。

文字列プロパティのメタ情報

次は、プラグインで利用できる文字列プロパティのメタ情報のタイプの一覧です。

タイプ	説明
maxLength	ユーザーが入力できる最大長を示します。
editMultiLine	編集フィールドが複数行持てるのかどうかを示します。
isPassword	値が true の場合、ユーザー入力が *** で隠蔽されます。

ファイルプロパティのメタ情報

次は、プラグインで利用できるファイルプロパティのメタ情報のタイプの一覧です。

特定のファイルの値では、実行サーバー上の絶対パスを持つ非バージョン管理下のファイルと、バージョン管理下のファイルは区別されます。ソース管理下にあるファイルは、常にテストコンテナのルートディレクトリに対して相対的であり、大抵は、実行されるテストソースを指定する際に使用されます。絶対パスも実行サーバー上に存在する必要があり、大抵、実行サーバー上でテストを起動するために使用される、ツールやリソースを指定するのに使用されます。

タイプ	説明
openBrowser	値が true の場合、テストコンテナ内のファイル群からファイルを選択するためのブラウザが開かれることを示します。
isSourceControl	値が true の場合、ファイルはソース管理システム上にあります。
fileNameSpec	標準 Windows ファイル参照ダイアログで分かる、許容ファイル名に対する制限を示します。

カスタムアイコン

カスタムテストタイプに対してカスタムアイコンをデザインし、作成したテストタイプが識別しやすいものにすることができます。構成 XML ファイルで識別子 PluginId を定義してあるプラグインに対してアイコンを定義するには、そのプラグイン ZIP コンテナのルートディレクトリに、次の 4 つのアイコンを置く必要があります。

名前	説明
<PluginId>.gif	作成したテストタイプのデフォルトアイコン。たとえば、ProcessExecutor.gif のようになります。


名前	説明
<PluginId>_package.gif	テストパッケージルートおよびスイート ノードに使用されるアイコン (特定のテスト タイプのテストを、テストパッケージに変換した場合) たとえば、ProcessExecutor_package.gif のようになります。
<PluginId>_linked.gif	テストの親フォルダが、リンク付けされたテスト コンテナだった場合に使用されるアイコン。たとえば、ProcessExecutor_linked.gif のようになります。
<PluginId>_incomplete.gif	製品またはテストの親テスト コンテナのソース管理プロファイルが未定義の場合に使用されるアイコン。

テスト タイプ用に新しいアイコンをデザインする場合は、次のルールを適用します。


- GIF タイプのアイコンのみを使用します。ファイル拡張子は大文字/小文字を区別し、必ず小文字の .gif である必要があります。
- <SCTM デプロイ フォルダ>%wwwroot%¥silkroot¥img¥PluginIcons から古いアイコンや無効なアイコンを削除します。そうしないと、プラグイン ZIP コンテナのルートディレクトリにある新しいアイコンでアイコンが更新されません。
- アイコンのサイズは 16 x 16 ピクセルです。
- アイコンに許容される最大色数は 256 色です。
- アイコンは 1 ビットの透過性を持ちます。

デプロイメント

プラグインの ZIP アーカイブは、Test Manager インストール ディレクトリ下にある、Plugins サブディレクトリの中になければなりません。このディレクトリに置いたプラグインを統合するには、アプリケーション サーバーとフロントエンド サーバーを、SilkCentral Service Manager を使用して再起動します。

 **注:** ホット デプロイメントはサポートされていません。

アーカイブを変更するたびに、これら 2 つのサーバーは再起動する必要があります。このアーカイブは、実行サーバーへ自動的にアップロードされます。

 **注:** 一旦プラグインをベースとしたテストを作成したら、そのプラグイン アーカイブを削除してはいけません。存在しなくなってしまったプラグイン アーカイブをベースとしたテストは、変更もしくは実行された際に、不明なエラーを送出します。

ビデオ キャプチャの開始と終了を示す

利用可能な Java クラスやメソッドに関する詳細については、[Javadoc](#) を参照してください。このリンクが動作しなかった場合には、Test Manager メニューの **ヘルプ > ドキュメント > Test Manager API 仕様** を選択して、Javadoc を開いてください。

Test Manager で、新しいサードパーティ プラグインを作成し、単一のテスト実行で複数のテスト ケースの処理をサポートするサードパーティ テスト タイプの場合で、キャプチャされるビデオを特定のテスト ケースに関連付ける場合、次の 2 つの方法で処理できます。

プラグインで実行中のサードパーティ テスト

これらのテストでは、TestLaunchResultDrain クラスの indicateTestStart メソッドおよび indicateTestStop メソッドの使用を推奨します。

外部処理で実行中のサードパーティ テスト

これらのテストでは、TCP/IP ベースのサービスを使用して START メッセージおよび FINISH メッセージを Test Manager 実行サーバーのポート 19138 に送信できます。これらのメッセージタイプによって、テストのテスト ケースが開始されたか、またはそれぞれ終了したことが、実行サーバーに通知されます。メッセージは、Unicode (UTF8) 形式または ASCII 形式でエンコードする必要があります。

メッセージ タイプ 書式

START START <Test Name>, <Test ID> <LF>。LF は ASCII コード 10 です。

FINISH FINISH <Test Name>, <Test ID>, <Passed> LF。LF は ASCII コード 10 です。Passed は True または False です。ビデオ キャプチャが、エラー時のみに実行されるよう設定されている場合、Passed が False に設定されている場合に、ビデオは結果にのみ保存されます。

要求が認識された場合、実行サーバーはメッセージ OK で応答します。これ以外の場合は、実行サーバーはエラー メッセージで応答します。常に、実行サーバーの応答を待ってから、次のテスト ケースを実行します。これを行わなかった場合、記録されたビデオは、実際のテスト ケースに一致しない場合があります。

テストが実行されている外部処理が、Java 環境に基づいている場合、TestCaseStartFinishSocketClient クラスの indicateTestStart メソッドおよび indicateTestStop メソッドを使用することを推奨します。これは、tm-testlaunchapi.jar ファイルに含まれています。


SilkCentral の Web サービス

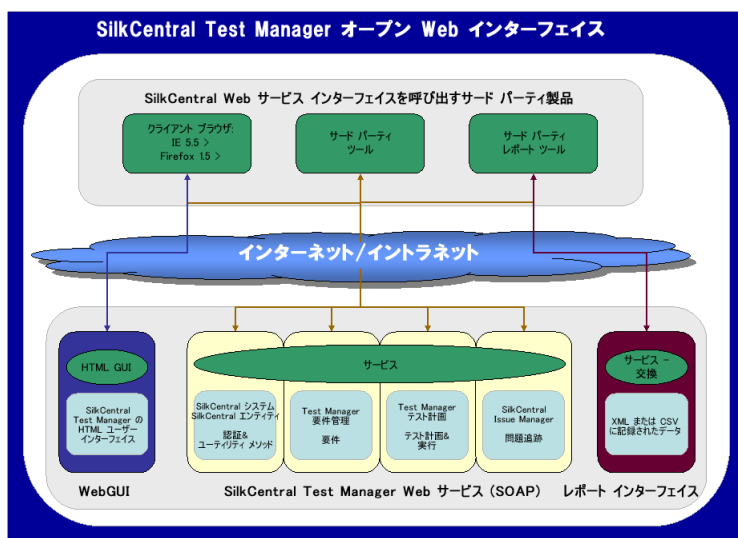
Web サービスにセットアップは必要なく、デフォルトで各フロントエンド サーバー上で有効になっています。たとえば、`http://www.yourFrontend.com/login` が SilkCentral にアクセスするために使用する URL の場合、`http://www.yourFrontend.com/Services1.0/services` と `http://www.yourFrontend.com/AlmServices1.0/services` は利用可能な Web サービスにアクセスするために使用する基本 URL です。

ブラウザでこのベース URL へアクセスすると、利用可能なすべての Web サービスの簡単な HTML リストが表示されます。このリストは、Apache Axis によって提供されています。SilkCentral が使用する SOAP スタックは `http://ws.apache.org/axis/` です。

この基本 URL では WSDL (Web サービス定義言語: Web Service Definition Language) 標準 XML ファイルへのリンクが掲載されており、各ファイルでは、それぞれ 1 つの Web サービスのインターフェイスが記述されています。これらのファイルは、可読形式ではありません。このため、SOAP が有効化されたクライアント (たとえば、SilkPerformer Java Explorer) が、WSDL ファイルを読み取り、そこから、そのファイルに対応する Web サービス上にあるメソッドを呼び出すために必要な情報を取得します。

利用可能な Java クラスやメソッドに関する詳細については、[Javadoc](#) を参照してください。このリンクが動作しなかった場合には、Test Manager メニューの **ヘルプ > ドキュメント > Test Manager API 仕様** を選択して、Javadoc を開いてください。

 **注:** バージョン 2008 より前の Test Manager に実装されたオリジナルの Web サービスも、引き続き `http://hostname:port/services` から利用できます。



Web サービス クイック スタート

このセクションでは、前提条件、サンプル例、その他 Web サービスのインテグレーションに関連するトピックが提供されます。

前提条件

次の前提条件は、Web サービス クライアントを開発する前に検討される必要があります。

- オブジェクト指向プログラミング (OOP) の基本知識。サンプル例は Java 言語で記述されているため、Java の経験は理解の助けとなるでしょう。Java の経験はなくても、C++、C#、Python、または Perl に関する経験がある開発者は、より簡単にこのサンプル例についていくことができるはずです。HashMaps や Lists といったコレクションの操作に関する経験があると、より望ましいです。
- JUnit テストについては簡単に言及されています。この Java テスト フレームワークについては必須ではありませんが、知識があると理解の助けとなるでしょう。
- Web サービス技術への入門的な経験。このヘルプ ドキュメントは、Web サービスや SOAP などの教育コースとなることを意図したものではありません。最低限、「Hello World!」 Web サービス クライアントくらいは、コーディングし、実行に成功している必要があります。
- Test Manager Web サービス アーキテクチャに関する知識。

Web サービス入門

適切なバージョンの SDK がインストールされ、それが PATH にあるかどうかを確認するには、『Test Manager リリース ノート』を参照してください。

JUnit jar がクラスパスにあると便利でしょう。JUnit.jar ファイルは、<http://www.junit.org/index.htm> からダウンロードすることができます。

1. Axis Web サービス ライブラリを、<http://ws.apache.org/axis/> よりダウンロードします。
2. すべての jar を、クラスパス上にある lib ディレクトリに展開します。
これには、環境変数 AXISCLASSPATH を使用すると便利です。
3. 目的の Web サービスの WSDL を指定しながら、以下のコマンドを実行します。

```
java -cp %AXISCLASSPATH%
      org.apache.axis.wsdl.WSDL2Java
      -o . -d Session -p
package.structure.you.want.to.use.for.your.client.yourwebservice
      -t http://url.to.your.service/yourWebService?wsdl
```

4. 自動生成された Java クラス YourWebServiceTestCase を探します。
このクラスは、YourWebService が提供するすべてのメソッドを使用するよう、既にできています。単にデフォルトの空の変数を置き換えてください。
たとえば、呼び出し login(String username, String password) における空の username、password の文字列は、使用される実際のユーザー名、パスワードの値に置き換える必要があります。このクラスは JUnit テストですが、クライアントをビルドするために、単にコードを、Java ドライバクラスの単純な main メソッドにコピー & 貼り付けすることができます。

Web サービス クライアントの概要

Web サービスは大抵の場合、SOAP over HTTP プロトコルを使用します。このシナリオで、SOAP エンベロープは送信されます。コレクションやその他複雑なオブジェクトが SOAP エンベロープにバンドルされている場合、ASCII データ構造を読み書きしづらくなる可能性があります。開発者は、SOAP エンベロープを直接操作して Web サービス クライアントをビルドしようとはなりません。経験豊富な開発者は、大抵 Web サービス クライアントのビルドを、SOAP エンベロープのレベルでは行いません。これを行うことは、面倒であり間違いの元でもあります。このため、すべての主なプログラミング言語で、Web サービス開発キットが提供されています。Axis は最も広く使用されている Java キットです。SOAP は、最も一般的な C++ や C のためのツールです。他にも、Perl、Python などのためのツールがあります。

実装言語 (Java、C++、C#、Perl、Python) にかかわらず、Web サービス クライアントの構築の場合、大抵の場合、同じパターンに従います。

1. 開発キット ツールに Web サービス WSDL を指定
2. ステップ 1 で返されたクライアント スタブを取得
3. ステップ 2 で生成されたクライアント スタブを編集して、完全なクライアントを取得

Axis は、このパターンに従います。ここで、WSDL からクライアントスタブを構築するのに、WSDL2Java ツールを使用します。WSDL2Java の使用方法の詳細情報については、『[Axis Reference Guide](#)』を参照してください。先ほどの概要部分で使用されていたスイッチの簡単な説明は以下のとおりです。

- -o : クライアントスタブの出力ディレクトリ
- -d : Web サービス接続の範囲
- -p : 名前空間にパッケージ構造を使用、このパッケージ構造にデプロイする
- -t : この Web サービス用の JUnit テストケース クラスを出力

JUnit テストケース クラスを生成します。このクラスは、Web サービスへバインドするコードを提供してくれるだけでなく、サービスの各メソッドを実行するテストコードも出力してくれます。

WSDL2Java ツールは、Web サービスへのクライアントをサポートするために、複数のクラスを生成します。サービス名が YourWebService である場合、次のクラス群が出力されることとなります。

- YourWebService : YourWebService を表すインターフェイス。
- YourWebServiceService : YourWebService ロケータを表すインターフェイス。
- YourWebServiceServiceLocator : YourWebService ロケータ、YourWebServiceService を実装。
- YourWebServiceSoapBindingStub : YourWebService POJO のシリアライゼーションをハンドリングするクライアントスタブ。
- YourWebServiceServiceTestCase : YourWebService のすべてのメソッドを実行する JUnit テストケース
- Serializeable Objects : YourWebService が使用するオブジェクトに対応するクライアント側のオブジェクト

インターフェイスと実装クラスの名前を、より人に分かりやすいものに変更します。

- YourWebServiceService => IYourWebServiceLocator へ（「I」は「インターフェイス」を表しています）。
- YourWebServiceServiceLocator => YourWebServiceLocator へ、IYourWebServiceLocator を実装。
- YourWebService => IYourWebService へ。

Axis が生成する JUnit テストケースは意義のあるものです。これは、そのまま即座に実行できるクライアントとなります。ユーザーが行う必要があるのは、デフォルトの初期化値に、その Web サービスの有効な値を反映させるだけです。また、ロケータを使用するバインディングコードも提供されており、「検索-バインド-実行」の SOA サイクルを開始することができます。ここで、tmrequirementsmanagement Web サービスで提供されているテストケースからの抜粋を見てみましょう：

```
/* default test automatically generated by WSDL2Java */
public void test2tmrequirementsmanagementLogin() throws Exception {
    com.borland.tm.webservices.tmrequirementsmanagement
        .TmrequirementsmanagementSoapBindingStub binding;
    try {
        binding = (com.borland.tm.webservices.tmrequirementsmanagement
            .TmrequirementsmanagementSoapBindingStub)
            new com.borland.tm.webservices.tmrequirementsmanagement
                .RequirementsServiceLocator().gettmrequirementsmanagement();
    }
    catch (javax.xml.rpc.ServiceException jre) {
        if(jre.getLinkedCause()!=null)
            jre.getLinkedCause().printStackTrace();
        throw new junit.framework.AssertionFailedError("JAX-RPC
            ServiceException caught: " + jre);
    }
    assertNotNull("binding is null", binding);

    // Time out after a minute
    binding.setTimeout(60000);
}
```

```

// Test operation
try {
    java.lang.String value = null;
    value = binding.login(new java.util.HashMap());
}
catch (com.borland.tm.webservices.tmrequirementsmanagement.RMServiceException e1) {
    throw new junit.framework.AssertionFailedError("RMServiceException
        Exception caught: " + e1);
}
// TBD - validate results
}

```

JUnit テストにはバインドおよびログイン用のコードが含まれています。ログインのコードはデフォルトの HashMap を設定するだけで作成できます。POJO Web サービス クライアントを作成する場合は、このコードをコピーして貼り付けおよび編集するだけで、バインドとログイン メソッドをすぐに利用できます。

```

public TmrequirementsmanagementSoapBindingStub getBinding()
    throws Exception {
    TmrequirementsmanagementSoapBindingStub binding;
    try {
        binding = (TmrequirementsmanagementSoapBindingStub)
            new RequirementsServiceLocator().gettmrequirementsmanagement();
    }
    catch (ServiceException jre) {
        // Handle error
    }

    if(binding == null)
    {
        // handle error here
        System.err.println("binding is null.");
        System.exit(0);
    }

    // Time out after a minute
    binding.setTimeout(60000);

    return binding;
}

public String login(String username, String password)
{
    // mBinding = getBinding() already obtained
    // Now get mSessionID member variable
    try {
        HashMap map = new HashMap();
        map.put("username", username);
        map.put("password", password);
        mSessionID = mBinding.login(map);
    }
    catch (RMServiceException e1) {
        // Handle error here
    }
    return mSessionID;
}

```

サンプル事例：要件を追加する

前のステップで行ったことの詳細をこのセクションでフォローアップし、Test Manager へ要件を追加するサンプル事例を完了させます。

先に進む前に、次の事前要件を満たしている必要があります。

- *tmrequirementsmanagement* Web サービスについて説明している各ステップを完了している。
 - バインド、ログイン、その他 *tmrequirementsmanagement* メソッドを持った、作業 POJO や JUnit クラスが作成されている。
 - 他の Test Manager API ヘルプ トピックを読んでいる。
1. ログイン メソッドとユーザー認証情報を持つプロパティ マップを使用してサービスへバインドします。
 2. 以前のステップで取得したセッション ID を保存します。
 3. 希望するデータを保持する要件オブジェクトを構築します。
 4. セッション ID と生成した要件オブジェクトを使用して、`updateRequirement` メソッドを呼び出します。
 5. `updateRequirement` メソッドによって返された要件 ID を保存します。
 6. 要件プロパティの `PropertyValue` 配列を作成します。
 7. 前で作成された配列を使用して、`updateProperties` メソッドを呼び出します。

WSDL2Java は、上述の Web サービス オブジェクトを作成します。

- Requirement
- PropertyValue

上記のオブジェクトの OOP メソッドを利用するだけで、Web サービスを使用していくことができます。SOAP エンベロープの構築は必要ありません。以下は、このユース ケースを完了させるために必要なコードの抜粋です。

```
/** propertyID for requirement risk */
public static final String PROPERTY_RISK = "Risk";

/** propertyID for requirement reviewed */
public static final String PROPERTY_REVIEWED = "Reviewed";

/** propertyID for requirement priority */
public static final String PROPERTY_PRIORITY = "Priority";

/** propertyID for requirement obsolete property */
public static final String PROPERTY_OBSOLETE = "Obsolete";

// Construct Top Level Requirement
Requirement topLevelRequirement = new Requirement();
topLevelRequirement.setName("tmReqMgt TopLevelReq");
topLevelRequirement.setDescription("tmReqMgt TopLevel Desc");
PropertyValue propRisk = new PropertyValue("1", PROPERTY_RISK, null);
PropertyValue propPriority = new PropertyValue("1", PROPERTY_PRIORITY, null);
PropertyValue[] properties = new PropertyValue[]{propRisk, propPriority};

/* First add requirement skeleton, get its ID
 * mWebService is a binding stub, see above getBinding() snippet
 * mSessionID is the stored session ID, see above login() snippet
 */
String requirementID = mWebService.updateRequirement(mSessionID,
topLevelRequirement, null);
```

```

// Now loop through and set properties
for(PropertyValue propValue: properties)
{
    propValue.setRequirementId(requirementID);
    mWebService.updateProperty(mSessionID, requirementID, propValue);
}

// Add Child Requirement
Requirement childRequirement = new Requirement();
childRequirement.setName("tmReqMgt ChildReq");
childRequirement.setDescription("tmReqMgt ChildLevel Desc");
childRequirement.setParentId(requirementID);
propRisk = new PropertyValue("2", PROPERTY_RISK, null);
propPriority = new PropertyValue("2", PROPERTY_PRIORITY, null);
properties = new PropertyValue[]{propRisk, propPriority};
String childReqID = mWebService.updateRequirement(mSessionID,
    childRequirement, null);
// Now loop through and set properties
for(PropertyValue propValue: properties)
{
    propValue.setRequirementId(requirementID);
    mWebService.updateProperty(mSessionID, childReqID, propValue);
}

// Print Results
System.out.println("Login Successful with mSessionID: "
    + mSessionID);
System.out.println("Top Level Requirement ID: " + requirementID);
System.out.println("Child Requirement ID: " + childReqID);

```

セッション処理

SilkCentral のデータは、未許可のアクセスから保護されています。データ アクセスの許可の前に、まずログイン認証が提供されなければなりません。これは、HTML フロントエンドで作業するときだけでなく、SOAP 呼び出しを介して Test Manager と通信するときにも当てはまります。

データに対してクエリをかけたり、Test Manager に対して設定変更を適用する際にも、まず最初の手順は認証です。認証が成功したら、ユーザー セッションが作成され、そのユーザー ログインのコンテキスト内で、以降のオペレーションを実行できるようになります。

Web ブラウザを介して Test Manager へアクセスしている場合には、セッション情報はユーザーに対して表示されません。ブラウザがクッキーを使用して、セッション情報をハンドリングします。HTML を介して SilkCentral を使用している場合に比べ、SOAP 呼び出しはセッション情報を手動で処理する必要があります。

サービスを介した認証は、sccsystem Web サービスの logonUser() SOAP 呼び出しによって行われます。このメソッド呼び出しはセッション識別子を返します。このセッション識別子は、サーバー上に作成されたセッションを参照していると共に、そのセッションのコンテキスト内で SilkCentral へアクセスするためのキーとしても使用されます。

実行にセッションが必要なそれ以降の各 SOAP 呼び出しは、このような返り値の識別子をパラメータの 1 つに取り、その妥当性をチェックし、そのセッションのコンテキスト内で実行します。

以下の Java コード サンプルは、Web サービスを介した Test Manager へのシンプルなアクセスと、セッション識別子の利用方法を示しています。

```

long sessionID = sccsystem.logonUser("admin", "admin");
Project[] projects = sccentities.getProjects(sessionID);


```

Web サービスを介して作成された SilkCentral セッションは、明示的に終了させることはできません。代わりにセッションは、一定時間使用されないと、自動的に終了されます。サーバー上でセッションがタイムアウトするとすぐに、そのセッションを使用しようとするそれ以降の SOAP 呼び出しは例外を投げるようになります。


デモ用クライアントは、<http://sso.borland.com> の Test Manager セクションからダウンロードすることができます。このデモプロジェクトは Test Manager の実行 Web サービスを利用しており、ユーザーが Web サービス インターフェイスに慣れるのに役立ちます。

利用可能な Web サービス

以下の表には、利用可能な SilkCentral Web サービスが一覧されています。

 **注:** また、WSDL URL には、Web サービス クライアントの作成には使用しない、システム内部 Web サービスも一覧されています。本ドキュメントでは、公開 Web サービスのみを記載していきます。


利用可能な Java クラスやメソッドに関する詳細については、[Javadoc](#) を参照してください。このリンクが動作しなかった場合には、Test Manager メニューの **ヘルプ > ドキュメント > Test Manager API 仕様** を選択して、Javadoc を開いてください。

 **注:** Web サービスで作業する場合、システムから返される時間はすべて協定世界時 (UTC) です。Web サービスでの作業では、すべての時間参照に UTC を使用してください。

WS 名 (インターフェイス)	WSDL URL	説明
sccsystem (SystemServiceI)	/Services1.0/services/sccsystem?wsdl	これはルート サービスであり、認証機能と、基本的なユーティリティ メソッドを提供します。
sccentities (MainEntities)	/axislegacy/sccentities?wsdl	このサービスは、SilkCentral プラットフォーム上にある プロジェクト エンティティと 製品 エンティティへのアクセスを提供します。
tmrequirementsmanagement (RMservice)	/Services1.0/services/tmrequirementsmanagement?wsdl	このサービスは、SilkCentral Test Manager の 要件 領域へのアクセスを提供します。
tmplanning (IPlanningService)	/Services1.0/services/tmplanning?wsdl	このサービスは、SilkCentral Test Manager の テスト 領域へのアクセスを提供します。また、実行を開始したり、実行結果をフェッチすることができます。
tmexecution (IExecutionWebservice)	/Services1.0/services/tmexecution?wsdl	このサービスは、SilkCentral Test Manager の 実行 領域へのアクセスを提供します。
tmfilters (FilterService)	/Services1.0/services/tmfilters?wsdl	このサービスを使用すると、フィルタの作成、読み取り、更新、および削除を行うことができます。
scim (RadarServiceIF)	/axislegacy/scim?wsdl	このサービスは、Issue Manager へのアクセスを提供します。

Services Exchange

このセクションでは、Services Exchange における HTTP ベースのインターフェイスについて説明します。これらは、レポート、添付ファイル、テスト計画、実行、およびライブラリを処理する際に使用されます。

 **注:** Web サービスで作業する場合、システムから返される時間はすべて協定世界時 (UTC) です。Web サービスでの作業では、すべての時間参照に UTC を使用してください。

reportData インターフェイス

reportData インターフェイスは、レポートのデータを要求する際に使用されます。次の表は、reportData インターフェイスのパラメータを表しています。

インターフェイス URL	パラメータ	説明
/servicesExchange? hid=reportData	sid	セッション ID : ユーザー認証
	reportFilterID	レポートフィルタ ID
	type	本体形式を返す: (csv または xml)
	includeHeaders	レポートヘッダのインクルードの有無 (true または false)
	userName	(任意) ユーザー名 : sid の代わりに使用
	passWord	(任意) ユーザー パスワード : sid の代わりに使用

reportData インターフェイスの例

```
String sessionID;  
String reportID;  
  
URL report = new URL("http", host, port,  
    "/servicesExchange?hid=reportData" +  
    "&type=csv" +  
    "&sid=" + sessionID +  
    "&reportFilterID=" + reportID +  
    "&includeHeaders=true" +  
    "&execNode_Id_0=1");  
  
BufferedReader in =  
    new BufferedReader(  
        new InputStreamReader(  
            report.openStream(), "UTF-8"));
```

レポートにパラメータが必要な場合は、各パラメータについて、レポート URL に次のコードを追加する必要があります。

```
"&parametername=parametervalue"
```

この例で、パラメータ execNode_Id_0 の値は 1 に設定されています。

TMAttach インターフェイス

TMAttach インターフェイスは、テストや要件に対する添付ファイルのアップロードに使用されます。次の表は、TMAttach インターフェイスのパラメータを表しています。

インターフェイス URL	パラメータ	説明
/servicesExchange? hid=TMAttach	sid	セッション ID : ユーザー 認証
	entityType	対象エンティティ タイプ : (テスト、要件、または TestStepParent)
	entityID	対象エンティティ ID : (テスト ID、要件 ID、または手動テスト ID)
	description	添付情報の説明。 URL エンコード テキストで、添付ファイルの記述に使用されます。
	isURL	true の場合、添付ファイルは URL です。false の場合、添付ファイルはファイルです。
	URL	(任意) 添付される URL。
	stepPosition	(任意) テスト ステップの順。手動テストのステップを識別します (たとえば、最初のステップの順は 1 です)。entityType が TestStepParent だった場合、順は必須となります。
	userName	(任意) ユーザー名 : sid の代わりに使用
	passWord	(任意) ユーザー パスワード : sid の代わりに使用

TMAttach Web サービスの例

次のコードは、バイナリの添付ファイルをアップロードするために、便利な HTTP-POST API を取得するのに Apache HttpClient を使用しています。リクエスト毎に 1 つの添付ファイルのみアップロードすることができます。

リクエスト毎に 1 つの添付ファイルのみアップロードすることができます。Apache HttpComponents をダウンロードするには、<http://hc.apache.org/downloads.cgi> を参照してください。必要なライブラリについては、コンポーネントのドキュメントを参照してください。

```
import org.apache.commons.httpclient.*; // Apache HttpClient

String sessionID = null;
String testNodeID = null; // receiving test
File fileToUpload = null; // attachment
String AttachmentDescription = ""; // descriptive text

HttpClient client = new HttpClient();
String formURL = "http://localhost:19120/
servicesExchange?hid=TMAttach" +
"&sid=" + sessionID +
"&entityID=" + testNodeID +
"&entityType=Test" +
```

```

"&isURL=false";
PostMethod filePost = new PostMethod(formURL);
Part[] parts = {
    new StringPart("description", attachmentDescription),
    new FilePart(fileToUpload.getName(), fileToUpload)
};
filePost.setRequestEntity(new MultipartRequestEntity(parts,
    filePost.getParams()));
client.getHttpConnectionManager().
    getParams().setConnectionTimeout(60000);
// Execute and check for success
int status = client.executeMethod(filePost);
// verify http return code...
// if(status == HttpStatus.SC_OK) ...

```

createTestPlan インターフェイス

createTestPlan インターフェイスは、新しいテストを作成するために使用されます。すべての呼び出しの HTTP の応答には、変更されたテストの XML 構造が含まれます。新しいコードの識別子は、更新された XML テストの構造から取得できます。

次の表は、createTestPlan インターフェイスのパラメータを表しています。

インターフェイス URL	パラメータ	説明
/servicesExchange? hid=createTestPlan	sid	セッション ID - ユーザー認証
	parentNodeID	テスト ツリーにおいて新しいテストが追加されるコンテナの ID
	userName	省略可能 : ユーザー名 - sid の代わりに使用
	passWord	省略可能 : ユーザー パスワード - sid の代わりに使用

テスト計画を検証するために使用される XML スキーマ定義ファイルは、フロントエンド サーバー URL `http://<ホスト>:<ポート>/silkroot/xsl/testplan.xsd` を使用してダウンロードするか、フロントエンドサーバーのインストール フォルダ `<Test Manager インストール フォルダ>/wwwroot/silkroot/xsl/testplan.xsd` からコピーできます。

createTestPlan Web サービスの例

以下のコードでは、Apache HttpClient を使用して、テストが作成されます。

```

import org.apache.commons.httpclient.*; // Apache HttpClient

long sessionID = mWebServiceHelper.getSessionId();

URL service = new URL("http", mWebServiceHelper.getHost(),
    mWebServiceHelper.getPort(),
    String.format("/servicesExchange?hid=%s&sid=%s&parentNodeID=%d",
        "createTestPlan", sessionID,
        PARENT_NODE_ID));

HttpClient client = new HttpClient();
PostMethod filePost = new PostMethod(service.toExternalForm());
String xmlFile = loadTestPlanUtf8("testPlan.xml");
StringPart xmlFileItem = new StringPart("testPlan", xmlFile,

```

```

"UTF-8");
xmlFormItem.setContentType("text/xml");
Part[] parts = {xmlFormItem};

filePost.setRequestEntity(new MultipartRequestEntity(parts,
    filePost.getParams()));
client.getHttpConnectionManager().getParams().setConnectionTimeout(60000)
;
int status = client.executeMethod(filePost);
System.out.println(filePost.getStatusLine());

```

リクエスト毎に 1 つの添付ファイルのみアップロードすることができます。Apache HttpComponents をダウンロードするには、<http://hc.apache.org/downloads.cgi> を参照してください。必要なライブラリについては、コンポーネントのドキュメントを参照してください。

テストの例

次のコードは、createTestPlan サービスを使用して Test Manager へアップロードすることができるテストの例を示しています。

```

<?xml version="1.0" encoding="UTF-8"?>
<TestPlan xmlns="http://www.borland.com/TestPlanSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.borland.com/TestPlanSchema
../..../..../TM/wwwroot/silkroot/xsl/testplan.xsd">

<Folder name="Folder1" id="5438">
<Description>Description of the folder</Description>
<Property name="property1">
<propertyValue>value1</propertyValue>
</Property>
<Test name="TestDef1" type="plugin.SilkTest">
<Description>Description of the test</Description>
<Property name="property2">
<propertyValue>value2</propertyValue>
</Property>
<Property name="property3">
<propertyValue>value3</propertyValue>
<propertyValue>value4</propertyValue>
</Property>
<Parameter name="param1" type="string">string1</Parameter>
<Parameter name="param2" type="boolean">true</Parameter>
<Parameter name="paramDate"
type="date">01.01.2001</Parameter>
<Parameter name="paramInherited" type="string"
inherited="true">
inheritedValue1
</Parameter>
<Step id="1" name="StepA">
<ActionDescription>do it</ActionDescription>
<ExpectedResult>everything</ExpectedResult>
</Step>
<Step id="2" name="StepB">
<ActionDescription>and</ActionDescription>
<ExpectedResult>everything should come</ExpectedResult>
</Step>
</Test>
<Test name="ManualTest1" id="5441" type="_ManualTestType"

```

```

plannedTime="03:45">
<Description>Description of the manual test</Description>
<Step id="1" name="StepA">
  <ActionDescription>do it</ActionDescription>
  <ExpectedResult>everything</ExpectedResult>
</Step>
<Step id="2" name="StepB">
  <ActionDescription>and</ActionDescription>
  <ExpectedResult>everything should come</ExpectedResult>
</Step>
<Step id="3" name="StepC">
  <ActionDescription>do it now</ActionDescription>
  <ExpectedResult>
    everything should come as you wish
  </ExpectedResult>
</Step>
</Test>
<Folder name="Folder2" id="5439">
  <Description>Description of the folder</Description>
  <Property name="property4">
    <propertyValue>value5</propertyValue>
  </Property>
  <Parameter name="param3" type="number">123</Parameter>
  <Folder name="Folder2_1" id="5442">
    <Description>Description of the folder</Description>
    <Test name="TestDef2" type="plugin.SilkPerformer">
      <Description>Description of the test</Description>
      <Property name="_sp_Project File">
        <propertyValue>ApplicationAccess.ltp</propertyValue>
      </Property>
      <Property name="_sp_Workload">
        <propertyValue>Workload1</propertyValue>
      </Property>
    </Test>
    <Test name="TestDef3" type="JUnitTestType"
      externalId="com.borland.MyTest">
      <Description>Description of the test</Description>
      <Property name="_junit_ClassFile">
        <propertyValue>com.borland.MyTest</propertyValue>
      </Property>
      <Property name="_junit_TestMethod">
        <propertyValue>testMethod</propertyValue>
      </Property>
      <Step id="1" name="StepA">
        <ActionDescription>do it</ActionDescription>
        <ExpectedResult>everything</ExpectedResult>
      </Step>
      <Step id="2" name="StepB">
        <ActionDescription>and</ActionDescription>
        <ExpectedResult>everything should come</ExpectedResult>
      </Step>
    </Test>
  </Folder>
</Folder>
</Folder>
</TestPlan>

```

exportTestPlan インターフェイス

exportTestPlan インターフェイスは、テスト計画を XML ファイルにエクスポートするために使用されます。次の表は、exportTestPlan インターフェイスのパラメータを表しています。

インターフェイス URL	パラメータ	説明
/servicesExchange? hid=exportTestPlan	sid	セッション ID - ユーザー認証
	nodeID	この ID のノードと、このノードの子ノードすべてが、再帰的にエクスポートされます。
	userName	省略可能 : ユーザー名 - sid の代わりに使用
	passWord	省略可能 : ユーザー パスワード - sid の代わりに使用

exportTestPlan Web サービスの例

以下のコードでは、Apache HttpClient を使用して、テストがエクスポートされます。

```
import org.apache.commons.httpclient.*; // Apache HttpClient

long sessionID = mWebServiceHelper.getSessionId();

URL service = new URL("http", mWebServiceHelper.getHost(),
    mWebServiceHelper.getPort(),
    String.format("/servicesExchange?hid=%s&sid=%s&nodeID=%d",
        "exportTestPlan", sessionID,
        PARENT_NODE_ID));

HttpClient client = new HttpClient();
client.getHttpConnectionManager().getParams().setConnectionTimeout(60000);
;
GetMethod fileGet = new GetMethod(service.toExternalForm());
int status = client.executeMethod(fileGet);
System.out.println(fileGet.getStatusLine());
String exportedTestPlanResponse = fileGet.getResponseBodyAsString();
System.out.println(exportedTestPlanResponse);
```

Apache HttpComponents をダウンロードするには、<http://hc.apache.org/downloads.cgi> を参照してください。必要なライブラリについては、コンポーネントのドキュメントを参照してください。

updateTestPlan インターフェイス

updateTestPlan インターフェイスは、XML ファイルから既存のルート ノードに対してテストを更新します。すべての呼び出しの HTTP の応答には、変更されたテストの XML 構造が含まれます。新しいコードの識別子は、更新された XML テストの構造から取得できます。

次の表は、updateTestPlan インターフェイスのパラメータを表しています。

インターフェイス URL	パラメータ	説明
/servicesExchange? hid=updateTestPlan	sid	セッション ID - ユーザー認証
	userName	省略可能 : ユーザー名 - sid の代わりに使用
	passWord	省略可能 : ユーザー パスワード - sid の代わりに使用

テスト計画を検証するために使用される XML スキーマ定義ファイルは、フロントエンド サーバー URL `http://<ホスト>:<ポート>/silkroot/xsl/testplan.xsd` を使用してダウンロードするか、フロントエンド サーバーのインストール フォルダ <Test Manager インストール フォルダ>/wwwroot/silkroot/xsl/testplan.xsd からコピーできます。

updateTestPlan Web サービスの例

以下のコードでは、Apache HttpClient を使用して、テストが更新されます。

```
import org.apache.commons.httpclient.*; // Apache HttpClient

long sessionID = mWebServiceHelper.getSessionId();
String xml = loadTestPlanUtf8(DEMO_TEST_PLAN_XML);
HttpClient client = new HttpClient();

URL webServiceUrl = new URL("http", mWebServiceHelper.getHost(),
    mWebServiceHelper.getPort(),
    String.format("/servicesExchange?hid=%s&sid=%s",
        "updateTestPlan",
        sessionID));
StringPart testPlanXml = new StringPart(DEMO_TEST_PLAN_XML, xml,
    "UTF-8");
testPlanXml.setContentType("text/xml");
Part[] parts = {testPlanXml};
PostMethod filePost = new PostMethod(webServiceUrl.toExternalForm());
filePost.setRequestEntity(new MultipartRequestEntity(parts,
    filePost.getParams()));
client.getHttpConnectionManager().getParams().setConnectionTimeout(60000);
int status = client.executeMethod(filePost);
System.out.println(filePost.getStatusLine());

String responseXml = filePost.getResponseBodyAsString();
```

リクエスト毎に 1 つの添付ファイルのみアップロードすることができます。Apache HttpComponents をダウンロードするには、<http://hc.apache.org/downloads.cgi> を参照してください。必要なライブラリについては、コンポーネントのドキュメントを参照してください。

createRequirements インターフェイス

createRequirements インターフェイスは、新しい要件を作成するために使用されます。すべての呼び出しの HTTP の応答には、変更された要件の XML 構造が含まれます。新しいコードの識別子は、更新された XML 要件の構造から取得できます。

次の表は、createRequirements インターフェイスのパラメータを表しています。

インターフェイス URL	パラメータ	説明
/servicesExchange? hid=createRequirements	sid	セッション ID - ユーザー認証
	parentNodeID	要件ツリーにおいて新しい要件が追加されるコンテナの ID
	userName	省略可能 : ユーザー名 - sid の代わりに使用
	passWord	省略可能 : ユーザー パスワード - sid の代わりに使用

要件を検証するために使用される XML スキーマ定義ファイルは、フロントエンド サーバー URL `http://<ホスト>:<ポート>/silkroot/xsl/requirements.xsd` を使用してダウンロードするか、フロントエンド サーバーのインストール フォルダ `<Test Manager インストール フォルダ>/wwwroot/silkroot/xsl/requirements.xsd` からコピーできます。

createRequirements Web サービスの例

以下のコードでは、Apache HttpClient を使用して、要件が作成されます。

```
import org.apache.commons.httpclient.*; // Apache HttpClient

long sessionID = mWebServiceHelper.getSessionId();

URL service = new URL("http", mWebServiceHelper.getHost(),
    mWebServiceHelper.getPort(),
    String.format("/servicesExchange?hid=%s&sid=%s&parentNodeID=%d",
        "createRequirements", sessionID,
        PARENT_NODE_ID));

HttpClient client = new HttpClient();
PostMethod filePost = new PostMethod(service.toExternalForm());
String xmlFile = loadRequirementsUtf8("requirements.xml");
StringPart xmlFileItem = new StringPart("requirements", xmlFile,
    "UTF-8");
xmlFileItem.setContentType("text/xml");
Part[] parts = {xmlFileItem};

filePost.setRequestEntity(new MultipartRequestEntity(parts,
    filePost.getParams()));
client.getHttpConnectionManager().getParams().setConnectionTimeout(60000);
int status = client.executeMethod(filePost);
System.out.println(filePost.getStatusLine());
```

リクエスト毎に 1 つの添付ファイルのみアップロードすることができます。Apache HttpComponents をダウンロードするには、<http://hc.apache.org/downloads.cgi> を参照してください。必要なライブラリについては、コンポーネントのドキュメントを参照してください。

exportRequirements インターフェイス

exportRequirements インターフェイスは、要件を XML ファイルにエクスポートするために使用されません。次の表は、exportRequirements インターフェイスのパラメータを表しています。

インターフェイス URL	パラメータ	説明
/servicesExchange? hid=exportRequirements	sid	セッション ID - ユーザー認証
	nodeID	この ID のノードと、このノードの子ノードすべてが、再帰的にエクスポートされます。
	userName	省略可能 : ユーザー名 - sid の代わりに使用
	passWord	省略可能 : ユーザー パスワード - sid の代わりに使用

exportRequirements Web サービスの例

以下のコードでは、Apache HttpClient を使用して、要件がエクスポートされます。

```
import org.apache.commons.httpclient.*; // Apache HttpClient

long sessionID = mWebServiceHelper.getSessionId();

URL service = new URL("http", mWebServiceHelper.getHost(),
    mWebServiceHelper.getPort(),
    String.format("/servicesExchange?hid=%s&sid=%s&nodeID=%d",
        "exportRequirements", sessionID,
        PARENT_NODE_ID));

HttpClient client = new HttpClient();
client.getHttpConnectionManager().getParams().setConnectionTimeout(60000);
HttpMethod fileGet = new GetMethod(service.toExternalForm());
int status = client.executeMethod(fileGet);
System.out.println(fileGet.getStatusLine());
String exportedRequirementResponse = fileGet.getResponseBodyAsString();
System.out.println(exportedRequirementResponse);
```

Apache HttpComponents をダウンロードするには、<http://hc.apache.org/downloads.cgi> を参照してください。必要なライブラリについては、コンポーネントのドキュメントを参照してください。

updateRequirements インターフェイス

updateRequirements インターフェイスは、XML ファイルから既存のルート ノードに対して要件を更新します。要件は、要件ツリーにおける内部 Test Manager ノード ID で識別されます。要件ツリー ノードと、そのノードのすべての子が更新されます。Test Manager において、新しいノードは追加され、紛失しているノードは obsolete (廃止) となり、移動されたノードは単純に移動されます。すべての呼び出しの HTTP の応答には、変更された要件の XML 構造が含まれます。新しいコードの識別子は、更新された XML 要件の構造から取得できます。

次の表は、updateRequirements インターフェイスのパラメータを表しています。

インターフェイス URL	パラメータ	説明
/servicesExchange? hid=updateRequirements	sid	セッション ID - ユーザー認証

インターフェイス URL	パラメータ	説明
	userName	省略可能：ユーザー名 - sid の代わりに使用
	passWord	省略可能：ユーザー パスワード - sid の代わりに使用
	NodeID	この ID のノードと、このノードの子ノードすべてが、再帰的に更新されます。

要件を検証するために使用される XML スキーマ定義ファイルは、フロントエンド サーバー URL `http://<ホスト>:<ポート>/silkroot/xsl/requirements.xsd` を使用してダウンロードするか、フロントエンド サーバーのインストール フォルダ `<Test Manager インストール フォルダ>/wwwroot/silkroot/xsl/requirements.xsd` からコピーできます。

updateRequirements Web サービスの例

以下のコードでは、Apache HttpClient を使用して、要件が更新されます。

```
import org.apache.commons.httpclient.*; // Apache HttpClient

long sessionID = mWebServiceHelper.getSessionId();
URL service = new URL("http", mWebServiceHelper.getHost(),
    mWebServiceHelper.getPort(),
    String.format("/servicesExchange?hid=%s&sid=%s&nodeID=%s",
        "updateRequirements",
        sessionID, rootNodeId));

HttpClient client = new HttpClient();
PostMethod filePost = new PostMethod(service.toExternalForm());
String xmlFile = loadRequirementsUtf8(fileName);
StringPart xmlFileItem = new StringPart("requirements", xmlFile,
    "UTF-8");
xmlFileItem.setContentType("text/xml");
Part[] parts = {xmlFileItem};

filePost.setRequestEntity(new MultipartRequestEntity(parts,
    filePost.getParams()));
client.getHttpConnectionManager().getParams().setConnectionTimeout(60000);
int status = client.executeMethod(filePost);
System.out.println(filePost.getStatusLine());

String responseXml = filePost.getResponseBodyAsString();
```

リクエスト毎に 1 つの添付ファイルのみアップロードすることができます。Apache HttpComponents をダウンロードするには、<http://hc.apache.org/downloads.cgi> を参照してください。必要なライブラリについては、コンポーネントのドキュメントを参照してください。

updateRequirementsByExternalID インターフェイス

updateRequirementsByExternalID インターフェイスは、XML ファイルから既存のルート ノードに対して要件を更新します。要件は、外部 ID で識別されます。要件ツリー ノードと、そのノードのすべての子が更新されます。Test Manager において、新しいノードは追加され、紛失しているノードは obsolete

(廃止) となり、移動されたノードは単純に移動されます。すべての呼び出しの HTTP の応答には、変更された要件の XML 構造が含まれます。新しいコードの識別子は、更新された XML 要件の構造から取得できます。

次の表は、updateRequirementsByExternalID インターフェイスのパラメータを表しています。

インターフェイス URL	パラメータ	説明
/servicesExchange? hid=updateRequirements	sid	セッション ID - ユーザー認証
	nodeID	更新される要件ツリー内のノード ID
	userName	省略可能 : ユーザー名 - sid の代わりに使用
	passWord	省略可能 : ユーザー パスワード - sid の代わりに使用

要件を検証するために使用される XML スキーマ定義ファイルは、フロントエンド サーバー URL `http://<ホスト>:<ポート>/silkkroot/xsl/requirements.xsd` を使用してダウンロードするか、フロントエンド サーバーのインストール フォルダ `<Test Manager インストール フォルダ>/wwwroot/silkkroot/xsl/requirements.xsd` からコピーできます。

updateRequirementsByExternalID Web サービスの例

以下のコードでは、Apache HttpClient を使用して、要件が更新されます。

```
import org.apache.commons.httpclient.*; // Apache HttpClient

long sessionID = mWebServiceHelper.getSessionId();
URL service = new URL("http", mWebServiceHelper.getHost(),
    mWebServiceHelper.getPort(),
    String.format("/servicesExchange?hid=%s&sid=%s",
        "updateRequirementsByExternalID",
        sessionID));

HttpClient client = new HttpClient();
PostMethod filePost = new PostMethod(service.toExternalForm());
string xmlFile = loadRequirementsUtf8(fileName);
StringPart xmlFormItem = new StringPart("requirements", xmlFile,
    "UTF-8");
xmlFormItem.setContentType("text/xml");
Part[] parts = {xmlFormItem};

filePost.setRequestEntity(new MultipartRequestEntity(parts,
    filePost.getParams()));
client.getHttpConnectionManager().getParams().setConnectionTimeout(60000);
int status = client.executeMethod(filePost);
System.out.println(filePost.getStatusLine());

String responseXml = filePost.getResponseBodyAsString();
```

リクエスト毎に 1 つの添付ファイルのみアップロードすることができます。Apache HttpComponents をダウンロードするには、<http://hc.apache.org/downloads.cgi> を参照してください。必要なライブラリについては、コンポーネントのドキュメントを参照してください。

createExecutionDefinitions インターフェイス

createExecutionDefinitions インターフェイスは、新しい実行計画を作成するために使用されます。すべての呼び出しの HTTP の応答には、変更された実行計画の XML 構造が含まれます。新しいコードの識別子は、更新された XML 実行計画の構造から取得できます。

次の表は、createExecutionDefinitions インターフェイスのパラメータを表しています。

インターフェイス URL	パラメータ	説明
/servicesExchange? hid=createExecutionDefinitions	sid	セッション ID - ユーザー認証
	parentNodeID	新しい実行計画を追加する実行ツリーノードの ID
	userName	省略可能 : ユーザー名 - sid の代わりに使用
	passWord	省略可能 : ユーザー パスワード - sid の代わりに使用

実行を検証するために使用される XML スキーマ定義ファイルは、フロントエンド サーバー URL `http://<ホスト>:<ポート>/silkroot/xsl/executionplan.xsd` を使用してダウンロードするか、フロントエンド サーバーのインストール フォルダ `<Test Manager インストール フォルダ>/wwwroot/silkroot/xsl/executionplan.xsd` からコピーできます。

createExecutionDefinitions Web サービスの例

以下のコードでは、Apache HttpClient を使用して、実行計画が作成されます。

```
import org.apache.commons.httpclient.*; // Apache HttpClient

long sessionID = mWebServiceHelper.getSessionId();

URL service = new URL("http", mWebServiceHelper.getHost(),
    mWebServiceHelper.getPort(),
    String.format("/servicesExchange?hid=%s&sid=%s&parentNodeID=%d",
        "createExecutionDefinitions", sessionID,
        PARENT_NODE_ID));

HttpClient client = new HttpClient();
PostMethod filePost = new PostMethod(service.toExternalForm());
String xmlFile = loadExecutionDefinitionsUtf8("executionplan.xml");
StringPart xmlFileItem = new StringPart("executionplan", xmlFile,
    "UTF-8");
xmlFileItem.setContentType("text/xml");
Part[] parts = {xmlFileItem};

filePost.setRequestEntity(new MultipartRequestEntity(parts,
    filePost.getParams()));
client.getHttpConnectionManager().getParams().setConnectionTimeout(60000);
int status = client.executeMethod(filePost);
System.out.println(filePost.getStatusLine());
```

リクエスト毎に 1 つの添付ファイルのみアップロードすることができます。Apache HttpComponents をダウンロードするには、<http://hc.apache.org/downloads.cgi> を

参照してください。必要なライブラリについては、コンポーネントのドキュメントを参照してください。

実行計画の例

次のコードは、createExecutionDefinitions サービスを使用して Test Manager へアップロードすることができる実行計画の例を示しています。この例では、実行定義の1つにカスタム スケジュールを作成し、テストを実行計画に割り当てます (共に、手動割り当てとフィルタを使用)。この例では、構成を含んだ構成スイートも作成されます。

```
<?xml version="1.0" encoding="UTF-8"?>
<ExecutionPlan xmlns="http://www.borland.com/ExecPlanSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.borland.com/ExecPlanSchema
  ../../../../TM/wwwroot/silkroot/xsl/executionplan.xsd">

  <Folder name="Folder1">
    <Description>Description of the folder</Description>
    <ExecDef name="ExecutionDefinition1" TestContainerId="1">
      <Description>Description1</Description>
      <CustomSchedule>
        <start>2009-11-26T21:32:52</start>
        <end>
          <forever>true</forever>
        </end>
        <Interval day="1" hour="2" minute="3"></Interval>
        <adjustDaylightSaving>>false</adjustDaylightSaving>
        <exclusions>
          <days>Monday</days>
          <days>Wednesday</days>
          <from>21:32:52</from>
          <to>22:32:52</to>
        </exclusions>
        <definiteRun>2009-11-27T21:35:12</definiteRun>
      </CustomSchedule>
      <ReadFromBuildInfoFile>>true</ReadFromBuildInfoFile>
      <Priority>High</Priority>
      <SetupTestDefinition>73</SetupTestDefinition>
      <CleanupTestDefinition>65</CleanupTestDefinition>
      <AssignedTestDefinitions>
        <ManualAssignment useTestPlanOrder="true">
          <TestId>6</TestId>
          <TestId>5</TestId>
        </ManualAssignment>
      </AssignedTestDefinitions>
    </ExecDef>
    <ExecDef name="ExecutionDefinition2" TestContainerId="1">
      <Description>Description2</Description>
      <Build>1</Build>
      <Version>1</Version>
      <Priority>Low</Priority>
      <SourceControlLabel>Label1</SourceControlLabel>
      <DependentExecDef id="65">
        <Condition>Passed</Condition>
        <Deployment>
          <Specific>
            <Execution type="Server" id="1"/>
            <Execution type="Tester" id="0"/>
          </Specific>
        </Deployment>
      </DependentExecDef>
    </ExecDef>
  </Folder>
</ExecutionPlan>
```

```

    </Specific>
  </Deployment>
</DependentExecDef>
<DependentExecDef id="70">
  <Condition>Failed</Condition>
  <Deployment>
    <Specific>
      <Execution type="Tester" id="0"/>
    </Specific>
  </Deployment>
</DependentExecDef>
<DependentExecDef id="68">
  <Condition>Any</Condition>
  <Deployment>
    <UseFromCurrentExedDef>>true</UseFromCurrentExedDef>
  </Deployment>
</DependentExecDef>
</ExecDef>

<ConfigSuite name="ConfigSuite1" TestContainerId="1">
  <Description>ConfigSuite1 desc</Description>
  <CustomSchedule>
    <start>2009-11-26T21:32:52</start>
    <end>
      <times>1</times>
    </end>
    <Interval day="1" hour="2" minute="3"/>
    <adjustDaylightSaving>>false</adjustDaylightSaving>
    <exclusions>
      <days>Monday</days>
      <days>Wednesday</days>
      <from>21:32:52</from>
      <to>22:32:52</to>
    </exclusions>
    <definiteRun>2009-11-27T21:35:12</definiteRun>
  </CustomSchedule>

  <ConfigExecDef name="Config1">
    <Description>Config1 desc</Description>
    <Priority>Medium</Priority>
  </ConfigExecDef>

  <ConfigExecDef name="Config2">
    <Priority>Medium</Priority>
    <DependentExecDef id="69">
      <Condition>Any</Condition>
      <Deployment>
        <UseFromCurrentExedDef>>true</UseFromCurrentExedDef>
      </Deployment>
    </DependentExecDef>
  </ConfigExecDef>

  <Build>8</Build>
  <Version>2</Version>
  <SourceControlLabel>ConfigSuite1 label</SourceControlLabel>
  <SetupTestDefinition>73</SetupTestDefinition>
  <CleanupTestDefinition>65</CleanupTestDefinition>
  <AssignedTestDefinitions>

```

```

    <ManualAssignment useTestPlanOrder="true">
      <TestId>6</TestId>
      <TestId>5</TestId>
    </ManualAssignment>
  </AssignedTestDefinitions>
</ConfigSuite>
</Folder>
</ExecutionPlan>

```

exportExecutionDefinitions インターフェイス

exportExecutionDefinitions インターフェイスは、実行計画を XML ファイルにエクスポートするために使用されます。次の表は、exportExecutionDefinitions インターフェイスのパラメータを表しています。

インターフェイス URL	パラメータ	説明
/servicesExchange? hid=exportExecutionDefinitions	sid	セッション ID - ユーザー認証
	nodeID	この ID のノードと、このノードの子ノードすべてが、再帰的にエクスポートされます。
	userName	省略可能 : ユーザー名 - sid の代わりに使用
	passWord	省略可能 : ユーザー パスワード - sid の代わりに使用

exportExecutionDefinitions Web サービスの例

以下のコードでは、Apache HttpClient を使用して、実行計画がエクスポートされます。

```

import org.apache.commons.httpclient.*; // Apache HttpClient

long sessionID = mWebServiceHelper.getSessionId();

URL service = new URL("http", mWebServiceHelper.getHost(),
    mWebServiceHelper.getPort(),
    String.format("/servicesExchange?hid=%s&sid=%s&nodeID=%d",
        "exportExecutionDefinitions", sessionID,
        NODE_ID));

HttpClient client = new HttpClient();
client.getHttpConnectionManager().getParams().setConnectionTimeout(60000);
;
GetMethod fileGet = new GetMethod(service.toExternalForm());
int status = client.executeMethod(fileGet);
System.out.println(fileGet.getStatusLine());
String exportedExecutionPlanResponse = fileGet.getResponseBodyAsString();
System.out.println(exportedExecutionPlanResponse);

```

Apache HttpComponents をダウンロードするには、<http://hc.apache.org/downloads.cgi> を参照してください。必要なライブラリについては、コンポーネントのドキュメントを参照してください。

updateExecutionDefinitions インターフェイス

updateExecutionDefinitions インターフェイスは、XML ファイルから実行計画を更新するために使用します。すべての呼び出しの HTTP の応答には、変更された実行計画の XML 構造が含まれます。新しいコードの識別子は、更新された XML 実行計画の構造から取得できます。

次の表は、updateExecutionDefinitions インターフェイスのパラメータを表しています。

インターフェイス URL	パラメータ	説明
/servicesExchange? hid=updateExecutionDefinitions	sid	セッション ID - ユーザー認証
	userName	省略可能 : ユーザー名 - sid の代わりに使用
	passWord	省略可能 : ユーザー パスワード - sid の代わりに使用

実行を検証するために使用される XML スキーマ定義ファイルは、フロントエンド サーバー URL `http://<ホスト>:<ポート>/silkrout/xsl/executionplan.xsd` を使用してダウンロードするか、フロントエンド サーバーのインストール フォルダ `<Test Manager インストール フォルダ>/wwwroot/silkrout/xsl/executionplan.xsd` からコピーできます。

updateExecutionDefinitions Web サービスの例

以下のコードでは、Apache HttpClient を使用して、実行計画が更新されます。

```
import org.apache.commons.httpclient.*; // Apache HttpClient

long sessionID = mWebServiceHelper.getSessionId();
String xml = loadExecutionPlanUtf8(DEMO_EXECUTION_PLAN_XML);
HttpClient client = new HttpClient();

URL webServiceUrl = new URL("http", mWebServiceHelper.getHost(),
    mWebServiceHelper.getPort(),
    String.format("/servicesExchange?hid=%s&sid=%s",
        "updateExecutionDefinitions",
        sessionID));
StringPart ExecutionPlanXml = new
StringPart(DEMO_EXECUTION_PLAN_XML, xml,
    "UTF-8");
ExecutionPlanXml.setContentType("text/xml");
Part[] parts = {ExecutionPlanXml};
PostMethod filePost = new PostMethod(webServiceUrl.toExternalForm());
filePost.setRequestEntity(new MultipartRequestEntity(parts,
    filePost.getParams()));
client.getHttpConnectionManager().getParams().setConnectionTimeout(60000)
;
int status = client.executeMethod(filePost);
System.out.println(filePost.getStatusLine());

String responseXml = filePost.getResponseBodyAsString();
```

リクエスト毎に 1 つの添付ファイルのみアップロードすることができます。Apache HttpComponents をダウンロードするには、<http://hc.apache.org/downloads.cgi> を参照してください。必要なライブラリについては、コンポーネントのドキュメントを参照してください。

createLibraries インターフェイス

createLibraries インターフェイスは、新しいライブラリを作成するために使用されます。すべての呼び出しの HTTP の応答には、変更されたライブラリの XML 構造が含まれます。新しいコードの識別子は、更新された XML ライブラリの構造から取得できます。

次の表は、createLibraries インターフェイスのパラメータを表しています。

インターフェイス URL	パラメータ	説明
/servicesExchange? hid=createLibraries	sid	セッション ID - ユーザー認証
	userName	省略可能 : ユーザー名 - sid の代わりに使用
	passWord	省略可能 : ユーザー パスワード - sid の代わりに使用

ライブラリを検証するために使用される XML スキーマ定義ファイルは、フロントエンド サーバー URL `http://<ホスト>:<ポート>/silkroot/xsl/libraries.xsd` を使用してダウンロードするか、フロントエンドサーバーのインストール フォルダ `<Test Manager インストール フォルダ>/wwwroot/silkroot/xsl/libraries.xsd` からコピーできます。

createLibraries Web サービスの例

以下のコードでは、Apache HttpClient を使用して、ライブラリが作成されます。

```
import org.apache.commons.httpclient.*; // Apache HttpClient

long sessionID = mWebServiceHelper.getSessionId();

URL service = new URL("http", mWebServiceHelper.getHost(),
    mWebServiceHelper.getPort(), String.format("/servicesExchange?hid=
%s&sid=%s",
    "createLibraries", sessionID));

HttpClient client = new HttpClient();
PostMethod filePost = new PostMethod(service.toExternalForm());
String xmlFile = loadTestPlanUtf8("libraries.xml");
StringPart xmlFormItem = new StringPart("libraries", xmlFile, "UTF-8");
xmlFormItem.setContentType("text/xml");
Part[] parts = {xmlFormItem};

filePost.setRequestEntity(new MultipartRequestEntity(parts,
filePost.getParams()));
client.getHttpConnectionManager().getParams().setConnectionTimeout(60000);
int status = client.executeMethod(filePost);
System.out.println(filePost.getStatusLine());
```

Apache HttpComponents をダウンロードするには、<http://hc.apache.org/downloads.cgi> を参照してください。必要なライブラリについては、コンポーネントのドキュメントを参照してください。

ライブラリの例

次のコードは、createLibraries サービスを使用して Test Manager へアップロードすることができるライブラリの例を示しています。GrantedProjects セクションで 1 つ

または複数のプロジェクトが定義されている場合を除き、新しいライブラリは特定のプロジェクトでの使用に限定されません。

```
<?xml version="1.0" encoding="UTF-8"?>
<LibraryStructure xmlns="http://www.borland.com/TestPlanSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.borland.com/TestPlanSchema
  ../../../../TM/wwwroot/silkroot/xsl/libraries.xsd">

  <Library name="Library 1">
    <Folder name="Folder 1">
      <Folder name="Folder 1.1">
        <SharedSteps name="Basic create user steps">
          <Step name="Login">
            <ActionDescription>
              Login with user admin.
            </ActionDescription>
            <ExpectedResult>Successful login.</ExpectedResult>
          </Step>
          <Step name="Create User">
            <ActionDescription>Create user tester</ActionDescription>
            <ExpectedResult>User created</ExpectedResult>
          </Step>
          <Step name="Logout">
            <ActionDescription>
              Logout using start menu
            </ActionDescription>
            <ExpectedResult>Logged out.</ExpectedResult>
          </Step>
        </SharedSteps>
      </Folder>
    </Folder>
    <GrantedProjects>
      <ProjectId>0</ProjectId>
      <ProjectId>1</ProjectId>
    </GrantedProjects>
  </Library>
</LibraryStructure>
```

exportLibraryStructure インターフェイス

exportLibraryStructure インターフェイスは、ライブラリ、フォルダ、および共有ステップ オブジェクトを XML ファイルとしてエクスポートするために使用します。次の表は、exportLibraryStructure インターフェイスのパラメータを表しています。

インターフェイス URL	パラメータ	説明
/servicesExchange? hid=exportLibraryStructure	sid	セッション ID - ユーザー 認証
	userName	省略可能 : ユーザー名 - sid の代わりに使用
	passWord	省略可能 : ユーザー パスワード - sid の代わりに使用
	nodeID	エクスポートされるライブラリ ツリー内のライブラリ ノードまたはフォ

インターフェイス URL	パラメータ	説明
		ルダ。共有ステップ ノードの ID は使用できません。

exportLibraryStructure Web サービスの例

以下のコードでは、Apache HttpClient を使用して、ライブラリがエクスポートされます。

```
import org.apache.commons.httpclient.*; // Apache HttpClient

long sessionID = mWebServiceHelper.getSessionId();

URL service = new URL("http", mWebServiceHelper.getHost(),
    mWebServiceHelper.getPort(), String.format("/servicesExchange?hid=
%s&sid=%s&nodeID=%d",
    "exportLibraryStructure", sessionID, NODE_ID));

HttpClient client = new HttpClient();
client.getHttpConnectionManager().getParams().setConnectionTimeout(60000)
;
HttpMethod fileGet = new GetMethod(service.toExternalForm());
int status = client.executeMethod(fileGet);
System.out.println(fileGet.getStatusLine());
String exportedTestPlanResponse = fileGet.getResponseBodyAsString();
System.out.println(exportedTestPlanResponse);
```

Apache HttpComponents をダウンロードするには、<http://hc.apache.org/downloads.cgi> を参照してください。必要なライブラリについては、コンポーネントのドキュメントを参照してください。

exportLibraryStructureWithoutSteps インターフェイス

exportLibraryStructureWithoutSteps インターフェイスは、ライブラリ、フォルダ、および共有ステップ オブジェクトを XML ファイルとしてエクスポートするために使用します。共有ステップ オブジェクトに含まれているステップはエクスポートされません。次の表は、exportLibraryStructureWithoutSteps インターフェイスのパラメータを表しています。

インターフェイス URL	パラメータ	説明
/servicesExchange? hid=exportLibraryStructureWithoutSteps	sid	セッション ID - ユーザー認証
	userName	省略可能 : ユーザー名 - sid の代わりに使用
	passWord	省略可能 : ユーザー パスワード - sid の代わりに使用
	nodeID	エクスポートされるライブラリ ツリー内のライブラリ ノードまたはフォルダ。共有ステップ ノードの ID は使用できません。

exportLibraryStructureWithoutSteps Web サービスの例

以下のコードでは、Apache HttpClient を使用して、ライブラリがエクスポートされます。

```
import org.apache.commons.httpclient.*; // Apache HttpClient

long sessionID = mWebServiceHelper.getSessionId();

URL service = new URL("http", mWebServiceHelper.getHost(),
    mWebServiceHelper.getPort(), String.format("/servicesExchange?hid=
%s&sid=%s&nodeID=%d",
    "exportLibraryStructureWithoutSteps", sessionID, NODE_ID));

HttpClient client = new HttpClient();
client.getHttpConnectionManager().getParams().setConnectionTimeout(60000)
;
HttpMethod fileGet = new GetMethod(service.toExternalForm());
int status = client.executeMethod(fileGet);
System.out.println(fileGet.getStatusLine());
String exportedTestPlanResponse = fileGet.getResponseBodyAsString();
System.out.println(exportedTestPlanResponse);
```

Apache HttpComponents をダウンロードするには、<http://hc.apache.org/downloads.cgi> を参照してください。必要なライブラリについては、コンポーネントのドキュメントを参照してください。

getLibraryInfoByName インターフェイス

getLibraryInfoByName インターフェイスは、指定された名前を持つすべてのライブラリの ID、名前、および説明を返します。このインターフェイスは、ライブラリの構造ではなく、プロパティのみを返します。次の表は、getLibraryInfoByName インターフェイスのパラメータを表しています。

インターフェイス URL	パラメータ	説明
/servicesExchange? hid=exportLibraryStructureWithoutSteps	sid	セッション ID - ユーザー認証
	userName	省略可能 : ユーザー名 - sid の代わりに使用
	passWord	省略可能 : ユーザーパスワード - sid の代わりに使用
	libraryName	ライブラリの名前

getLibraryInfoByName Web サービスの例

以下のコードでは、Apache HttpClient を使用して、ライブラリ情報が取得されます。

```
import org.apache.commons.httpclient.*; // Apache HttpClient

long sessionID = mWebServiceHelper.getSessionId();

URL service = new URL("http", mWebServiceHelper.getHost(),
    mWebServiceHelper.getPort(), String.format("/servicesExchange?hid=
%s&sid=%s",
    "exportLibraryStructureWithoutSteps", sessionID, LIBRARY_NAME));

HttpClient client = new HttpClient();
```


```
client.getHttpConnectionManager().getParams().setConnectionTimeout(60000)
;
HttpMethod fileGet = new GetMethod(service.toExternalForm());
int status = client.executeMethod(fileGet);
System.out.println(fileGet.getStatusLine());
String response = fileGet.getResponseBodyAsString();
System.out.println(response);
```

Apache HttpComponents をダウンロードするには、<http://hc.apache.org/downloads.cgi> を参照してください。必要なライブラリについては、コンポーネントのドキュメントを参照してください。

Web サービス デモ クライアント

Web サービス デモ クライアントは、Test Manager Web サービスの使用方法をデモンストレーションするためのツールです。クライアントを **ヘルプ > ツール** からダウンロードします。

Web サービス デモ クライアントでは、**テスト > テスト属性** で利用できるすべての属性をテストごとに、すべてのプロパティを利用可能なテスト タイプごとに表示します。

 **注目:** Web サービス デモ クライアントは、Web サービスの使用方法をデモンストレーションするために設計されました。このデモ クライアントを運用環境では使用しないでください。

索引

A

- Apache Axis 39
- API
 - コード カバレッジとの統合 7
- API の構造
 - サードパーティ テスト タイプ プラグイン 31

C

- createExecutionDefinitions
 - インターフェイス 57
 - 例 57
- createLibraries
 - インターフェイス 62
 - 例 62
- createRequirements
 - インターフェイス 52
 - 例 52
- createTestPlan
 - インターフェイス 48
 - 例 48

E

- exportExecutionDefinitions
 - インターフェイス 60
 - 例 60
- exportLibraryStructure
 - インターフェイス 63
 - 例 63
- exportLibraryStructureWithoutSteps
 - インターフェイス 64
 - 例 64
- exportRequirements
 - インターフェイス 53
 - 例 53
- exportTestPlan
 - インターフェイス 51
 - 例 51

G

- getLibraryInfoByName
 - インターフェイス 65
 - 例 65

J

- Java API インターフェイス 28

P

- process executor
 - サンプル コード 32

R

- reportData
 - インターフェイス 46
 - 例 46

S

- Services Exchange 45
- SOAP
 - エンベロープ 40
 - スタック 39

T

- TMAAttach
 - インターフェイス 46
 - 例 46

U

- updateExecutionDefinitions
 - インターフェイス 61
 - 例 61
- updateRequirements
 - インターフェイス 54
 - 例 54
- updateRequirementsByExternalID
 - インターフェイス 55
 - 例 55
- updateTestPlan
 - インターフェイス 51
 - 例 51

W

- Web サービス
 - WSDL 仕様 21
 - インターフェイス 18
 - 概要 39
 - サンプル事例 43
 - 前提条件 39
 - プロファイルを設定する 27
 - 要件管理 29
 - 利用可能 45
- Web サービス インターフェイス
 - クイック スタート 39
 - チュートリアル 40
- Web サービス デモ クライアント 66
- WSDL 仕様 21

あ

- アイコン
 - カスタム 36

い

- 一般プロパティのメタ情報
 - サードパーティ テスト タイプ プラグイン 35
- インターフェイス
 - createExecutionDefinitions 57

- createLibraries 62
- createRequirements 52
- createTestPlan 48
- exportExecutionDefinitions 60
- exportLibraryStructure 63
- exportLibraryStructureWithoutSteps 64
- exportRequirements 53
- exportTestPlan 51
- getLibraryInfoByName 65
- Java API インターフェイス 28
- reportData 46
- TMAAttach 46
- updateExecutionDefinitions 61
- updateRequirements 54
- updateRequirementsByExternalID 55
- updateTestPlan 51
- Web サービス 18
- ソース管理 16

か

- カスタム アイコン
 - サードパーティ テスト タイプ プラグイン 36

く

- クラス 28

こ

- コード カバレッジ
 - API 7

さ

- サードパーティ テスト タイプ プラグイン
 - API の構造 31
 - 一般プロパティのメタ情報 35
 - カスタム アイコン 36
 - サンプル コード 32
 - 実装 30
 - 設定 XML ファイル 35, 37
 - 定義済みパラメータを渡す 31
 - 統合 30
 - パッケージング 30
 - ファイル プロパティのメタ情報 36
 - メタ情報 35
 - 文字列プロパティのメタ情報 36
- サンプル コード
 - サードパーティ テスト タイプ プラグイン 32

し

- 実装
 - サードパーティ テスト タイプ プラグイン 30
- 種
 - プラグイン 6

せ

- セッション処理 44

そ

- ソース管理
 - インターフェイス 16
 - インターフェイスの慣習 17
 - 統合 16
 - プラグイン 16

て

- 定義済みパラメータ
 - サードパーティ テスト タイプ プラグインへ渡す 31
- デプロイメント
 - サードパーティ テスト タイプ プラグイン 37
 - プラグイン 6
- デモ クライアント
 - Web サービス インターフェイス 66

と

- 同期
 - 要件 29
- 統合
 - サードパーティ テスト タイプ プラグイン 30

に

- 認証
 - Web サービス 44

は

- 配信
 - プラグイン 6
- パッケージング
 - サードパーティ テスト タイプ プラグイン 30

ひ

- ビデオ キャプチャ
 - 開始を示す 37
 - 終了を示す 37

ふ

- ファイルプロパティのメタ情報
 - サードパーティ テスト タイプ プラグイン 36
- プラグイン
 - 概要 6
 - コンパイル 6
 - 種 6
 - ソース管理 16
 - デプロイメント 6
 - 配信 6
 - 問題追跡 18
 - 要件 29
 - 要件管理 29
- プラグインのコンパイル 6

め

メタ情報
サードパーティ テスト タイプ プラグイン 35

も

文字列プロパティのメタ情報
サードパーティ テスト タイプ プラグイン 36
問題追跡
プラグイン 18
問題追跡システムとの統合

概要 18

よ

要件管理システムとの統合 29
要件プラグイン 29

ろ

ログイン認証情報 44