MICRO FOCUS

Silk Central 18.5

API ヘルプ

Micro Focus
The Lawn
22-30 Old Bath Road
Newbury, Berkshire RG14 1QN
UK
http://www.microfocus.com

2017-10-24

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
サンプル プロファイル クラス 8
コード カバレッジ XSD 10
サンプル XML データ 11
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 12
ソース管理との統合 14
ソース管理との統合のインターフェイス 14
ソース管理との統合の慣習 15
問題追跡システムとの統合 16
Java インターフェイス 16
要件管理システムとの統合 17
Java インターフェイス 17
サードパーティ テスト タイプとの統合 18
プラグイン実装 18
パッケージング 18
プラグインへパラメータを渡す 19
API の構造 19
サンプル コード 20
設定 XML ファイル 23
プラグインのメタ情報 23
一般プロパティのメタ情報 24
文字列プロパティのメタ情報 24
ファイル プロパティのメタ情報 24
カスタム アイコン 25
デプロイメント 25
ビデオ キャプチャの開始と終了を示す 25
クラウド統合 27
Silk Central の Web サービス 28
Web サービス クイック スタート 29
前提条件 29
Web サービス入門 30
Web サービス クライアントの概要 30
サンプル事例：要件を追加する 31
セッション処理 33
利用可能な Web サービス 33
Services Exchange 34
reportData インターフェイス 34
TMAttach インターフェイス 35
createTestPlan インターフェイス 37
exportTestPlan インターフェイス 40
updateTestPlan インターフェイス 41
createRequirements インターフェイス 43
exportRequirements インターフェイス 45
updateRequirements インターフェイス 46
updateRequirementsByExtID インターフェイス 48
createExecutionDefinitions インターフェイス 50

exportExecutionDefinitions インターフェイス 53
updateExecutionDefinitions インターフェイス 54
createLibraries インターフェイス 57
exportLibraryStructure インターフェイス 59
exportLibraryStructureWithoutSteps インターフェイス 60
getLibraryInfoByName インターフェイス 61
Web サービス デモ クライアント 62

はじめに

本書では、サードパーティ ツールを Silk Central に統合するプラグインを、作成および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 Silk Central 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

Silk Central 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。 Silk Central 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、Silk Central プラグインを設定することで統合できます。 Silk Central 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、Silk Central メニューの **ヘルプ** > **ドキュメント** > **Silk Central API 仕様** を選択して、Javadoc を開いてください。

Silk Central 統合プラグイン

Silk Central 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。 Micro Focus 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、平穏享有権、平穏所有権、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。 Micro Focus 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、Silk Central 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

Silk Central 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、Silk Central のリリースノートを参照してください。 このことは、Silk Central 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 Silk Central 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、<application server installation directory>¥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 Silk Central 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。 サーバーの再起動の詳細については、この『ヘルプ』の「管理」トピックを参照してください。

配信

プラグインの種のタイプが Silk Central 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、 Silk Central 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、Silk Central 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。 Silk Central 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。

- Silk Central 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。 パスは次のようになります：¥Silk¥Silk Central <version>¥Plugins

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。Silk Central のベースラインの概念をよく理解している必要があります。Silk Central 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、Silk Central 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、Silk Central 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;
import com.segue.scc.published.api.codeanalysis.CodeAnalysisResult;
```

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 Silk Central 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 Silk Central 実行サーバーによってトリガされるとコード分析が開始され、カバー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、Silk Central 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。

   これは、startCoverage の後に呼び出され、コード分析を停止するために、テスト実行を完了した Silk Central 実行サーバーによってトリガーされます。
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 sstopCoverage() 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 Silk Central 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、Silk Central のコード カバレッジ リストに表示されます。
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。
    - Silk Central 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

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

public class myProfileClass implements CodeAnalysisProfile{

  // This function is called first by the Silk Central Code Coverage framework
  // The name of the plug-in is displayed in the code coverage drop down in Silk
Central
  @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 the Silk Central Execution Server starting a test run
  // 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 Silk Central 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 the Silk Central Execution Server finishing a test run
  //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(),
  // and 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 Silk CentralM 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 invokation 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:¥Users¥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:¥Users¥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>
```

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 Runtime Environment (JRE) 8 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>/Silk Central 18.5/Plugins フォルダに配置します。
5. ディレクトリを、<install dir>/Silk Central 18.5/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. 次のコマンドを実行して、ファイルを UNIX 形式に変換します：dos2unix startCodeAnalysisFramework.sh
8. 次のコマンドを実行して、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：chmod 775 startCodeAnalysisFramework.sh
9. 次のシェル スクリプトを実行して、CAFramework プロセスを実行します：./ startCodeAnalysisFramework.sh。

これで、Silk Central からコード分析フレームワークを使用できるようになります。

ソース管理との統合

ソース管理プロファイルにより、Silk Central を外部のソース管理システムと統合できます。

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、 上で構成できるようになります。これにより、Silk Central 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。

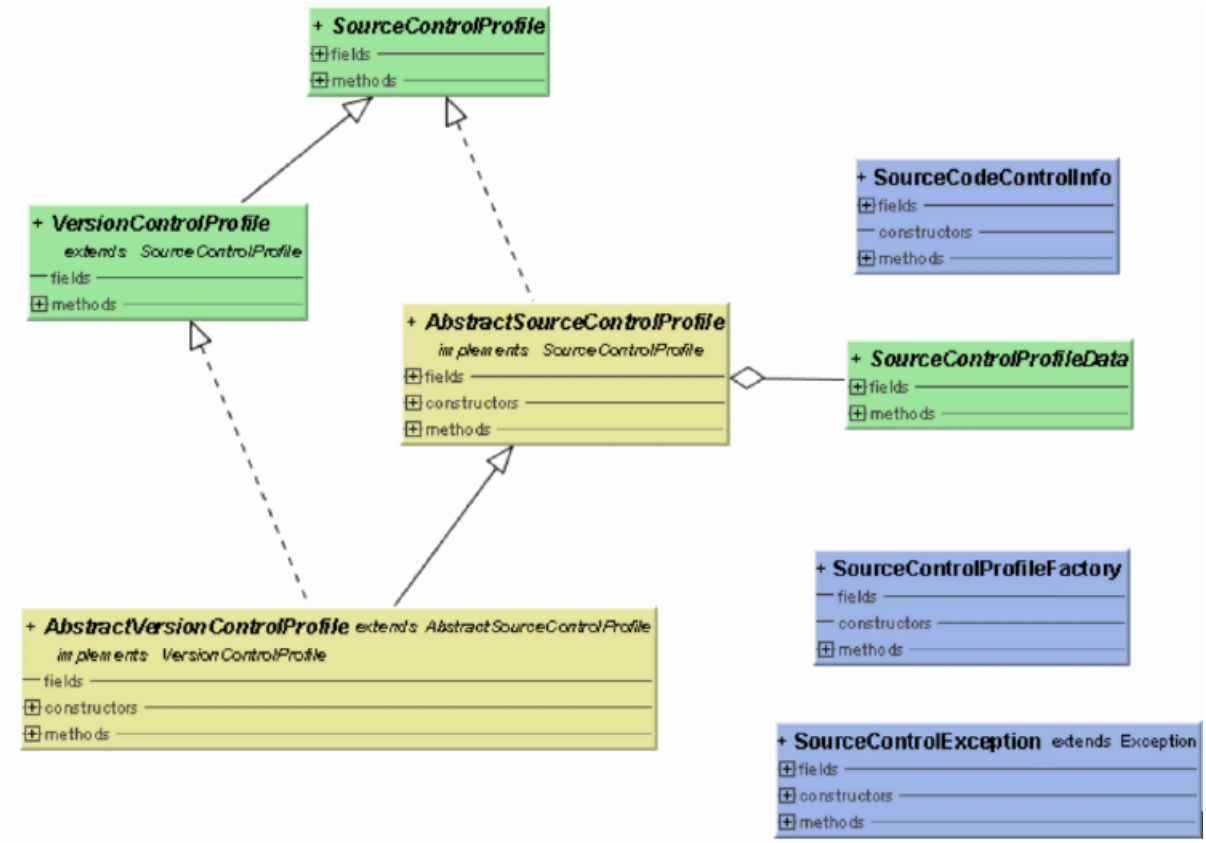
C:¥Program Files (x86)¥¥Silk¥instance_<インスタンス番号>_<インスタンス名>¥Plugins ¥subversion.zip にある com.segue.scc.vcs.subversion パッケージのソース コードを参照して、実際の実装方法の参考にしてください。

ソース管理との統合のインターフェイス

Silk Central 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 Silk Central 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、Silk Central メニューの **ヘルプ** > **ドキュメント** > **Silk Central 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、Silk Central 用プラグインを作成する際に必要となります。

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。

C:¥Program Files (x86)¥Silk¥Silk Central18.5¥instance_<インスタンス番号>_<インスタンス名>¥Plugins¥Bugzilla3.zip にある com.segue.scc.issuetracking.bugzilla3 パッケージのソース コードを参照して、実際の実装方法の参考にしてください。

Java 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、Silk Central メニューの **ヘルプ** > **ドキュメント** > **Silk Central API 仕様** を選択して、Javadoc を開いてください。

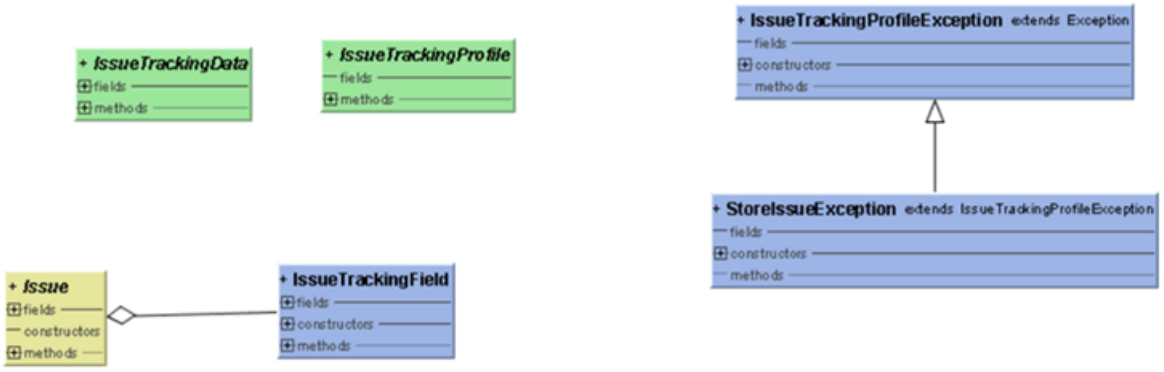
ビルド環境

ライブラリ scc.jar に拡張する必要の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、Silk Central 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

Silk Central 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、Silk Central 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、Silk Central 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 Silk Central 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、Silk Central メニューの **ヘルプ** > **ドキュメント** > **Silk Central 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。

Java 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、Silk Central メニューの **ヘルプ** > **ドキュメント** > **Silk Central 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.segue.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

Silk Central 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 --- Silk Performer、Silk Test Classic、手動テスト、NUnit、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、Silk Central 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、NUnit、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 Silk Central のカスタマイゼーションを可能にすることです。 Silk Central 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。 Silk Central 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、Silk Central メニューの **ヘルプ** > **ドキュメント** > **Silk Central 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

ExtProcessTestLaunchBean クラスに基づいたプラグインの場合、プラグインによって開始されたプロセスにおいて、各パラメータは環境変数として自動的に設定されます。 これはパラメータ名がシステム変数の名前と一致する場合にもあてはまるため、パラメータの値が空の文字列である場合の除き、システム変数の値はパラメータの値によって置き換えられます。

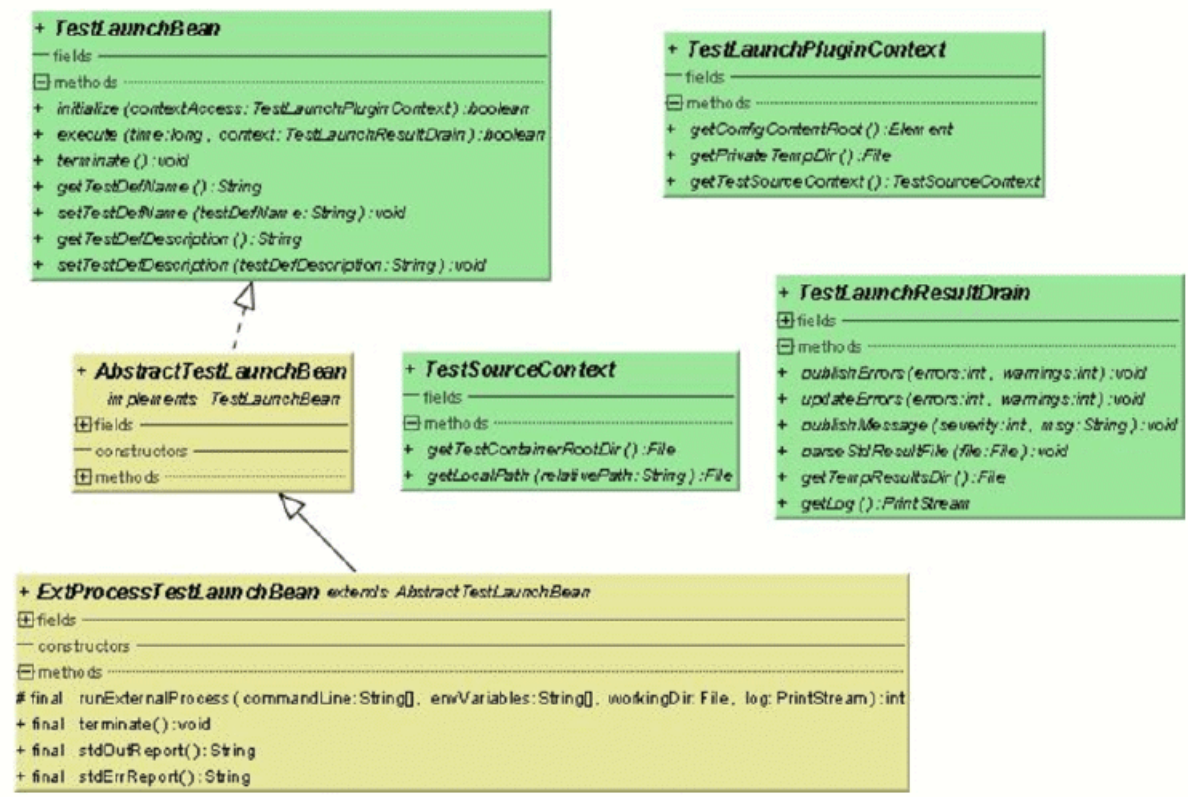
プラグイン インターフェイスは、Silk Central **テスト** 領域で定義されている、すべてのカスタム パラメータへのフルアクセスを提供します。 カスタム パラメータは、サードパーティのテスト タイプに対してのみサポートされています。 プラグインは、事前定義パラメータを指定することができないので、特定のテストに対して、パラメータが定義されているか、どのように定義されるかは、プラグイン実装によって決まります。

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、Silk Central メニューの **ヘルプ** > **ドキュメント** > **Silk Central 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.segue.tm.published.api.testlaunch.ExtProcessTestLaunchBean;
import com.segue.tm.published.api.testlaunch.TestLaunchResultDrain;

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

  // test properties that will be set by Silk Central 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 Silk Central 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 Silk Central 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 Silk Central 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 Silk Central 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 = getEnviromentVariables(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[] getEnviromentVariables(TestLaunchResultDrain context)
    throws IOException {
    String[] envVariables = {
      "SCTM_EXEC_RESULTSFOLDER="
      + context.getTempResultsDir().getAbsolutePath(),
      "SCTM_EXEC_SOURCESFOLDER="
      + sourceAccess().getTestContainerRootDir().getAbsolutePath(),
    };
    return envVariables;
}

/**
 * Let Silk Central parse the standard report xml file (output.xml) if exists.
 * See also Silk Central Web Help - Creating a Test Package. A XSD file
 * can be found in Silk Central Help -&gt; Tools -&gt; 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 exectued 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、ということを意味します。
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、バージョン管理化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。
<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。
- <Silk Central deploy folder>¥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、Silk Central 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、Silk Central 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、Silk Central メニューの **ヘルプ** > **ドキュメント** > **Silk Central API 仕様** を選択して、Javadoc を開いてください。

Silk Central で、新しいサード パーティ テスト 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 タイプの場合に、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 Silk Central 実行サーバーのポートに送信できます。使用するポート番号は、プラグイン内で *ExecutionContextInfo.ExecProperty#PORT_TESTCASE_START_FINISH* を使用して問い合わせできます。プラグインが ExtProcessTestLaunchBean を拡張している場合、テスト プロセスの *#sctm_portTestCaseStartFinish* という環境変数からポート番号を取得することもできます。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 には True または False を指定します。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 に False が設定されていれば、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。

テストを実行している外部プロセスが、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。このクラスは、tm-testlaunchapi.jar ファイルに含まれています。

クラウド統合

Silk Central は、クラウド プロバイダ プロファイルを設定することで、パブリックまたはプライベートなクラウド サービスのプロバイダと統合することができます。クラウド プロファイルは、プラグインの概念に基づいており、特定のクラウド プロバイダに対する独自のプラグインをユーザーが作成することができます。クラウド プロバイダ プラグインは、各自動テストが実行される前に仮想環境をデプロイします。

注: クラウド API は、Silk Central の将来のバージョンで変更される可能性があります。この API を使用する場合は、Silk Central の将来のバージョンにアップグレードする際に、実装も更新する必要となる可能性があります。

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、Silk Central メニューの **ヘルプ** > **ドキュメント** > **Silk Central 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 CloudProviderProfile（com.segue.scc.published.api.cloud）です。CloudProviderProfile は、外部のクラウド システムへのアクセスと制御を指定します。クラウド プロバイダ プラグインの実装には、以下の事項を行う必要があります。

- この種類のプロバイダにリモート アクセスするためにクラウド プロバイダ プロファイルで設定する必要のあるプロパティを公開する
- プロファイル プロパティを検証し、クラウド プロバイダとの接続を確認する
- 利用可能なイメージ テンプレートのリストをクラウド プロバイダから取得する
- 選択したイメージ テンプレートを基に仮想環境をデプロイし、外部からアクセス可能なホスト アドレスを公開する
- 仮想環境がデプロイされ実行中であることを確認する
- 特定の仮想環境を削除する

Silk Central の Web サービス

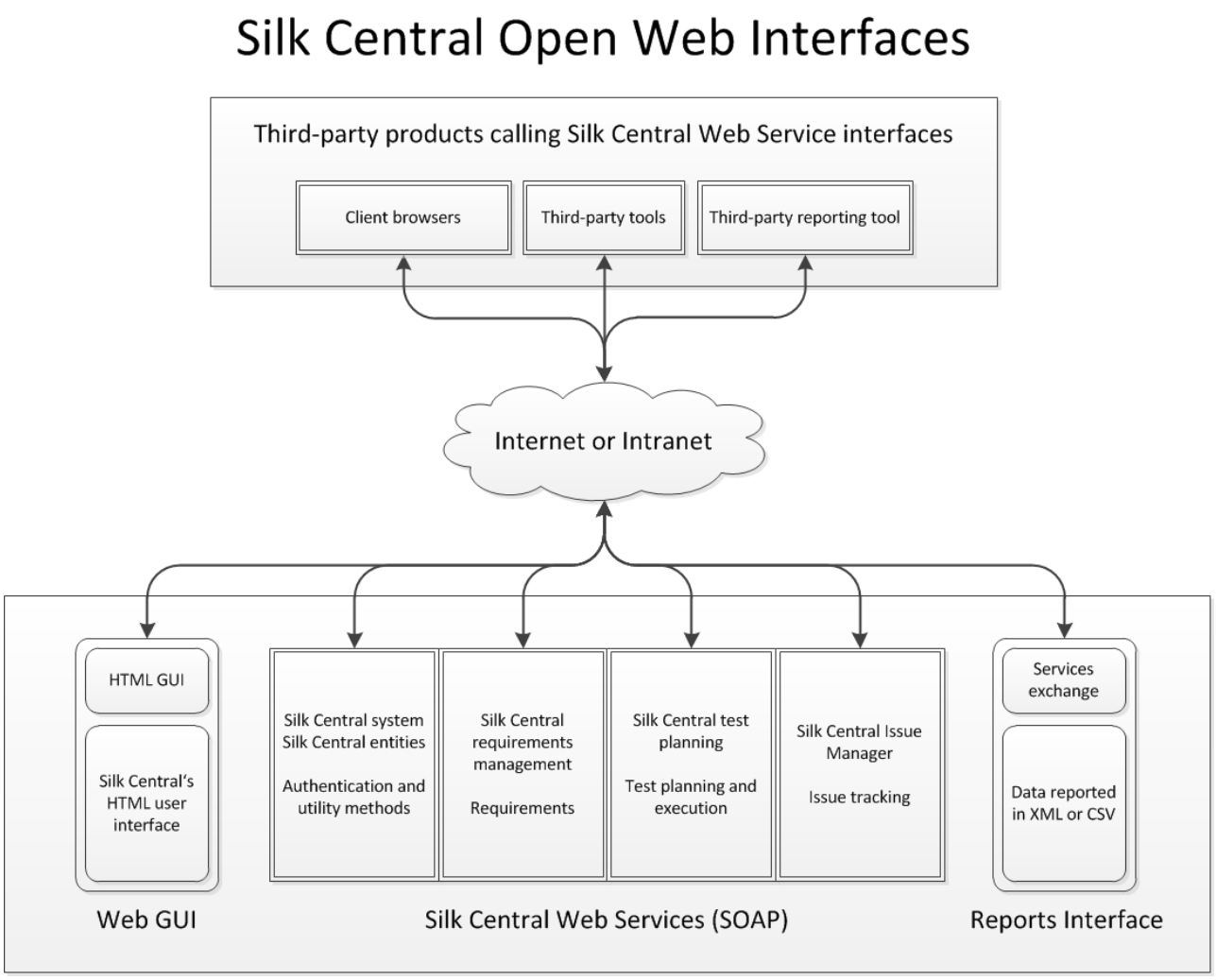
Web サービスにセットアップは必要なく、デフォルトで各フロントエンド サーバー上で有効になっています。たとえば、http://www.yourFrontend.com/login が Silk Central のアクセスに使用する URL である場合、http://www.yourFrontend.com/Services1.0/jaxws/system（従来のサービス： http://www.yourFrontend.com/Services1.0/services）および http://www.yourFrontend.com/AlmServices1.0/services が、利用可能な Web サービスのアクセスに使用するベース URL になります。

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、JAX-WS によって提供されています。 Silk Central が使用する SOAP スタックは、*https://jax-ws.java.net/* 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、Silk Performer 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、Silk Central メニューの **ヘルプ** > **ドキュメント** > **Silk Central 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 Silk Central 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。
- Silk Central 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、『*Silk Central* 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. Java SDK の bin ディレクトリが PATH に含まれていることを確認します。
2. 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。

   ```
   wsimport -s <生成したファイルを保存するパス> -Xnocompile
   -p <yourWebService クライアントで使用するパッケージ構造>
   http://<サービスの URL>/yourWebService?wsdl
   ```
3. 自動生成された Java クラス YourWebService 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できるようになっています。

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してはなりません。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。Silk Central では、SOAP メッセージを使用して通信する Web サービスやクライアントの構築に *Java API for XML Web Services (JAX-WS)* を使用しています。

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

JAX-WS 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、wsimport ツール（JDK にバンドル）を使用します。wsimport の使用方法の詳細については、『*JDK Tools and Utilities*』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。

- -s：クライアント スタブの出力ディレクトリ
- -p：ターゲットのパッケージ。このパッケージ構造にデプロイされます。

例：wsimport -s <生成するスタブの場所> -p <ターゲット パッケージ> <WSDL>

wsimport ツールは、Web サービス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になります。

- YourWebService：YourWebService を表すインターフェイス
- YourWebServiceService：利用可能なポートの一覧を取得するなど、YourWebService ロケーターを表す生成クラス（サービス エンドポイント インターフェイス）
- WSFaultException：wsdl:fault からマップされた例外クラス
- WsFaultBean：レスポンス wsdl:message から派生した非同期レスポンス
- 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

Silk Central Web サービスにアクセスする JAX-WS クライアントを生成するには、YourWebServiceClient という新しい Java クラスを作成します。Web サービスにはポートを介してバイ

ンドする必要があります。ポートは、リモート サービスのプロキシとして動作するローカル オブジェクトです。ポートは、上記手順で示した wsimport ツールの実行によって作成されます。このプロキシを取得するには、getRequirementsServicePort メソッドをサービスに対して呼び出します。

```
// Bind to the Requirements service
   RequirementsServiceService port = new RequirementsServiceService
      (new URL("http", mHost, mPort, "/Services1.0/jaxws/requirements?wsdl"));
   RequirementsService requirementsService = port.getRequirementsServicePort();
```

その後、ユーザー名とパスワードを指定してサービスの logonUser メソッドを呼び出してセッション ID を得ることができます。

```
// Login to Silk Central and get session ID
   String sessionId = requirementsService.logonUser(mUsername, mPassword);
```

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、Silk Central 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。

- requirements 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 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 Silk Central API ヘルプ トピックを読んでいる。

1. ユーザーの資格情報を指定するログイン メソッドを使用してサービスにバインドします。
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、プロジェクト ID、および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。

wsimport 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。

```
/** project ID of Silk Central project */
private static final int PROJECT_ID = 0;

/** 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";
```

```
// Get the Requirements service
RequirementsService service = getRequirementsService();

// Login to Silk Central
String sessionId = login(service);

// Construct Top Level Requirement
Requirement topLevelRequirement = new Requirement();
topLevelRequirement.setName("tmReqMgt TopLevelReq");
topLevelRequirement.setDescription("tmReqMgt TopLevel Desc");

PropertyValue propRisk = new PropertyValue();
propRisk.setPropertyId(PROPERTY_RISK);
propRisk.setValue("2");
PropertyValue propPriority = new PropertyValue();
propPriority.setPropertyId(PROPERTY_PRIORITY);
propPriority.setValue("3");
PropertyValue[] properties = new PropertyValue[] {propRisk, propPriority};

/*
* First add requirement skeleton, get its ID
* service is a binding stub, see above getRequirementsService() snippet
* sessionId is the stored session ID, see above login() snippet
*/
int requirementID = service.updateRequirement(sessionId, PROJECT_ID,
topLevelRequirement, -1);

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

// Add Child Requirement
Requirement childRequirement = new Requirement();
childRequirement.setName("tmReqMgt ChildReq");
childRequirement.setDescription("tmReqMgt ChildLevel Desc");
childRequirement.setParentId(requirementID);
propRisk = new PropertyValue();
propRisk.setPropertyId(PROPERTY_RISK);
propRisk.setValue("1");
propPriority = new PropertyValue();
propPriority.setPropertyId(PROPERTY_PRIORITY);
propPriority.setValue("1");
properties = new PropertyValue[] {propRisk, propPriority};

int childReqID = service.updateRequirement(sessionId, PROJECT_ID, childRequirement, -1);

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

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

注: このサンプル コードは、Web サービスのデモ クライアント クラスである com.microfocus.silkcentral.democlient.samples.AddingRequirement からも利用できます。

セッション処理

Silk Central 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 Silk Central 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、Silk Central 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 Silk Central 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 Silk Central 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、必要な 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 Silk Central 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 Silk Central 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 = systemService.logonUser("admin", "admin");
Project[] projects = sccentities.getProjects(sessionID);
```

Web サービスを介して作成された Silk Central 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 クライアントは、Silk Central のメニューの **ヘルプ** > **ツール** > **Web サービス デモ クライアント** ダウンロードできます。このデモ プロジェクトは Silk Central の tests 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 Silk Central Web サービスが一覧されています。HTTP ベースのインターフェイスを介して Silk Central にアクセスすることもできます。詳細については、「*Services Exchange*」を参照してください。

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、Silk Central メニューの **ヘルプ** > **ドキュメント** > **Silk Central 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
system (SystemService)	/Services1.0/jaxws/system?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。
administration (AdministrationService)	/Services1.0/jaxws/administration?wsdl	このサービスは、Silk Central の 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。
requirements (RequirementsService)	/Services1.0/jaxws/requirements?wsdl	このサービスは、Silk Central の **要件** 領域へのアクセスを提供します。
tests (TestsService)	/Services1.0/jaxws/tests?wsdl	このサービスは、Silk Central の **テスト** 領域へのアクセスを提供します。
executionplanning (ExecutionPlanningService)	/Services1.0/jaxws/executionplanning?wsdl	このサービスは、Silk Central の **実行計画** 領域へのアクセスを提供します。
filter (FilterService)	/Services1.0/jaxws/filter?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。
issuemanager (IssueManagerService)	/Services1.0/jaxws/issuemanager?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。

Web サービスを介して Silk Central にアクセスすることもできます。詳細については、「利用可能な *Web* サービス」を参照してください。

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。

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
http://<front-end URL>/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 : 次の代わりに使用 : sid

インターフェイス URL	パラメータ	説明
	passWord	(任意) ユーザー パスワード： 次の代わりに使用：sid
	projectID	プロジェクトの ID

例:http://<front-end URL>/servicesExchange?hid=reportData&reportFilterID=<id>&type=<csv or xml>&includeHeaders=<true or false>&userName=<user>&passWord=<password>&projectID=<id>

reportData インターフェイスの例

```
String reportID = "<id>";
    String user = "<user>";
    String pwd = "<pwd>";
    String host = "<any_host>";

     URL report = new URL("http", host, 19120,
     "/servicesExchange?hid=reportData" +
      "&type=xml" + // or csv
      "&userName=" + user +
      "&passWord=" + pwd +
      "&reportFilterID=" + reportID +
      "&includeHeaders=true" +
      "&rp_execNode_Id_0=1" +
      "&projectID=27);

    BufferedReader in = new BufferedReader(new
InputStreamReader(report.openStream(), "UTF-8"));

    StringBuilder builder = new StringBuilder();
    String line = "";

    while ((line = in.readLine()) != null) {
         builder.append(line + "¥n");
    }

    String text = builder.toString();
    System.out.println(text);
 }
```

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。

```
 "&rp_parametername=parametervalue"
```

この例で、パラメータ rp_execNode_Id_0 の値は 1 に設定されています。

注: reportData サービスに渡されるパラメータの名前は rp_ で始まる必要があります。例：/servicesExchange?hid=reportData&type=xml&sid=<...>&reportFilterID=<...>&projectID=<...>&rp_TestID=<...>

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
http://<front-end URL>/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： 次の代わりに使用：sid
	passWord	(任意) ユーザー パスワード： 次の代わりに使用：sid

例：http://<front-end URL>/servicesExchange?hid=TMAttach&entityType=<test, requirement, or TestStepParent>&entityID=<id>&description=<text>&isURL=<true or false>&URL=<URL>&stepPosition=<number>&userName=<user>&passWord=<password>

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 HtmlClient 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
http://<front-end URL>/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

例：http://<front-end URL>/servicesExchange?hid=createTestPlan&parentNodeID=<id>&userName=<user>&passWord=<password>

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://<フロントエンド URL>/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 <Silk Central 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");
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。

テストの例

次のコードは、createTestPlan サービス、および updateTestPlan サービスを使用して Silk Central 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://<front-end URL>/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
http://<front-end URL>/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

例 : http://<front-end URL>/servicesExchange?hid=exportTestPlan&nodeID=<id>&userName=<user>&passWord=<password>

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)
;
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。

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
http://<front-end URL>/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

例：http://<front-end URL>/servicesExchange?hid=updateTestPlan&userName=<user>&passWord=<password>

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://<フロントエンド URL>/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 <Silk Central 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。

テストの例

次のコードは、createTestPlan サービス、および updateTestPlan サービスを使用して Silk Central 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://<front-end URL>/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>
```

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
http://<front-end URL>/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

例：http://<front-end URL>/servicesExchange?hid=createRequirements&parentNodeID=<id>&userName=<user>&passWord=<password>

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://<フロントエンド URL>/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 <Silk Central 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。

要件の例

次のコードは、createRequirements サービス、updateRequirements サービス、および updateRequirementsByExtID サービスを使用して、Silk Central へアップロードすることができる要件の例を示しています。

```
<?xml version="1.0" encoding="UTF-8"?>
<Requirement id="0" name="name" xmlns="http://www.borland.com/
RequirementsSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://<front-end URL>/silkroot/xsl/
requirements.xsd">
      <ExternalId>myExtId1</ExternalId>
      <Description>Description</Description>
      <Priority value="Low" inherited="false"/>
      <Risk value="Critical" inherited="false"/>
      <Reviewed value="true" inherited="false"/>
      <Property inherited="false" name="Document"
type="string">MyDocument1.doc</Property>
      <Requirement id="1" name="name" />
      <Requirement id="2" name="name1">
            <Requirement id="3" name="name" />
            <Requirement id="4" name="name1">
                  <Requirement id="5" name="name" />
        <Requirement id="6" name="name1">
            <ExternalId>myExtId2</ExternalId>
            <Description>Another Description</Description>
            <Priority value="Medium" inherited="false"/>
            <Risk value="Critical" inherited="false"/>
            <Reviewed value="true" inherited="false"/>
            <Property inherited="false" name="Document"
type="string">MyDocument2.doc</Property>
                  </Requirement>
            </Requirement>
      </Requirement>
</Requirement>
```

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す。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
http://<front-end URL>/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

インターフェイス URL	パラメータ	説明
	includeObsolete	省略可能： true または false を指定します。省略した場合はデフォルトで true に設定されます。廃止した要件を除外する場合は false を指定します。

例：http://<front-end URL>/servicesExchange?hid=exportRequirements&nodeID=<id>&userName=<user>&passWord=<password>

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 Silk Central 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。Silk Central 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廃止 (obsolete) 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
http://<front-end URL>/servicesExchange?hid=updateRequirements	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

インターフェイス URL	パラメータ	説明
	passWord	省略可能： ユーザー パスワード - sid の代わりに使用

例：http://<front-end URL>/servicesExchange?hid=updateRequirements&userName=<user>&passWord=<password>

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://<フロントエンド URL>/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 <Silk Central 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",
      "updateRequirements", sessionID));

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。

要件の例

次のコードは、createRequirements サービス、updateRequirements サービス、および updateRequirementsByExtID サービスを使用して、Silk Central へアップロードすることができる要件の例を示しています。

```
<?xml version="1.0" encoding="UTF-8"?>
<Requirement id="0" name="name" xmlns="http://www.borland.com/
RequirementsSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://<front-end URL>/silkroot/xsl/
requirements.xsd">
      <ExternalId>myExtId1</ExternalId>
```

```
    <Description>Description</Description>
    <Priority value="Low" inherited="false"/>
    <Risk value="Critical" inherited="false"/>
    <Reviewed value="true" inherited="false"/>
    <Property inherited="false" name="Document"
type="string">MyDocument1.doc</Property>
    <Requirement id="1" name="name" />
    <Requirement id="2" name="name1">
        <Requirement id="3" name="name" />
        <Requirement id="4" name="name1">
            <Requirement id="5" name="name" />
      <Requirement id="6" name="name1">
        <ExternalId>myExtId2</ExternalId>
        <Description>Another Description</Description>
        <Priority value="Medium" inherited="false"/>
        <Risk value="Critical" inherited="false"/>
        <Reviewed value="true" inherited="false"/>
        <Property inherited="false" name="Document"
type="string">MyDocument2.doc</Property>
            </Requirement>
        </Requirement>
    </Requirement>
</Requirement>
```

updateRequirementsByExtID インターフェイス

updateRequirementsByExtID 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。Silk Central 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廃止 (obsolete) 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、updateRequirementsByExtID 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
http://<front-end URL>/servicesExchange?hid=updateRequirementsByExtID	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

例：http://<front-end URL>/servicesExchange?hid=updateRequirementsByExtID&nodeID=<id>&userName=<user>&passWord=<password>

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://<フロントエンド URL>/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 <Silk Central インストール フォルダ>/wwwroot/silkroot/xsl/requirements.xsd からコピーできます。

updateRequirementsByExtID 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",
      "updateRequirementsByExtID",
      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。

要件の例

次のコードは、createRequirements サービス、updateRequirements サービス、および updateRequirementsByExtID サービスを使用して、Silk Central へアップロードすることができる要件の例を示しています。

```
<?xml version="1.0" encoding="UTF-8"?>
<Requirement id="0" name="name" xmlns="http://www.borland.com/
RequirementsSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://<front-end URL>/silkroot/xsl/
requirements.xsd">
      <ExternalId>myExtId1</ExternalId>
      <Description>Description</Description>
      <Priority value="Low" inherited="false"/>
      <Risk value="Critical" inherited="false"/>
      <Reviewed value="true" inherited="false"/>
      <Property inherited="false" name="Document"
type="string">MyDocument1.doc</Property>
      <Requirement id="1" name="name" />
      <Requirement id="2" name="name1">
            <Requirement id="3" name="name" />
            <Requirement id="4" name="name1">
                  <Requirement id="5" name="name" />
```

```
        <Requirement id="6" name="name1">
          <ExternalId>myExtId2</ExternalId>
          <Description>Another Description</Description>
          <Priority value="Medium" inherited="false"/>
          <Risk value="Critical" inherited="false"/>
          <Reviewed value="true" inherited="false"/>
          <Property inherited="false" name="Document"
type="string">MyDocument2.doc</Property>
                </Requirement>
          </Requirement>
      </Requirement>
</Requirement>
```

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
http://<front-end URL>/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

例 : http://<front-end URL>/servicesExchange?hid=createExecutionDefinitions&parentNodeID=<id>&userName=<user>&passWord=<password>

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://<フロントエンド URL>/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 <Silk Central 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、および updateExecutionDefinitions サービスを使用して Silk Central 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、実行計画の 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://<front-end URL>/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>
        <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
http://<front-end URL>/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

例：http://<front-end URL>/servicesExchange?hid=exportExecutionDefinitions&nodeID=<id>&userName=<user>&passWord=<password>

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)
;
HttpMethod 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
http://<front-end URL>/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

例：http://<front-end URL>/servicesExchange?hid=updateExecutionDefinitions&userName=<user>&passWord=<password>

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://<フロントエンド URL>/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 <Silk Central インストール フォルダ>/wwwroot/silkroot/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。

実行計画の例

次のコードは、createExecutionDefinitions サービス、および updateExecutionDefinitions サービスを使用して Silk Central 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、実行計画の 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://<front-end URL>/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>
    <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>
```

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
http://<front-end URL>/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

例：http://<front-end URL>/servicesExchange?hid=createLibraries&userName=<user>&passWord=<password>

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://<フロントエンド URL>/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 <Silk Central 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 xmlFileItem = new StringPart("libraries", 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());
```

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 Silk Central 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://<front-end URL>/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>Succesful login.</ExpectedResult>
            <CustomStepProperty name="Step Property 1">
                    <propertyValue>Step Property Value</
propertyValue>
            </CustomStepProperty>
          </Step>
          <Step name="Create User">
            <ActionDescription>Create user tester</ActionDescription>
```

```
                <ExpectedResult>User created</ExpectedResult>
                <CustomStepProperty name="Step Property 1">
                        <propertyValue>Step Property Value</
propertyValue>
                </CustomStepProperty>
              </Step>
              <Step name="Logout">
                <ActionDescription>
                  Logout using start menu
                </ActionDescription>
                <ExpectedResult>Logged out.</ExpectedResult>
                <CustomStepProperty name="Step Property 1">
                        <propertyValue>Step Property Value</
propertyValue>
                </CustomStepProperty>
              </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
http://<front-end URL>/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 ノードまたはフォルダ。共有ステップ ノードの ID は使用できません。

例：http://<front-end URL>/servicesExchange?hid=exportLibraryStructure&userName=<user>&passWord=<password>&nodeID=<id>

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
http://<front-end URL>/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。

例：http://<front-end URL>/servicesExchange?hid=exportLibraryStructureWithoutSteps&userName=<user>&passWord=<password>&nodeID=<id>

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
http://<front-end URL>/servicesExchange?hid=getLibraryInfoByName	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

例：http://<front-end URL>/servicesExchange?hid=getLibraryInfoByName&userName=<user>&passWord=<password>&libraryName=<name>

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",
  "getLibraryInfoByName", 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、Silk Central 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 28
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 19

C

createExecutionDefinitions
    インターフェイス 50
    例 50
createLibraries
    インターフェイス 57
    例 57
createRequirements
    インターフェイス 43
    例 43
createTestPlan
    インターフェイス 37
    例 37

E

exportExecutionDefinitions
    インターフェイス 53
    例 53
exportLibraryStructure
    インターフェイス 59
    例 59
exportLibraryStructureWithoutSteps
    インターフェイス 60
    例 60
exportRequirements
    インターフェイス 45
    例 45
exportTestPlan
    インターフェイス 40
    例 40

G

getLibraryInfoByName
    インターフェイス 61
    例 61

J

Java インターフェイス 16

P

process executor
    サンプル コード 20

R

reportData
    インターフェイス 34
    例 34

S

Services Exchange 34
SOAP
    エンベロープ 30
    スタック 28

T

TMAttach
    インターフェイス 35
    例 35

U

updateExecutionDefinitions
    インターフェイス 54
    例 54
updateRequirements
    インターフェイス 46
    例 46
updateRequirementsByExtID
    インターフェイス 48
    例 48
updateTestPlan
    インターフェイス 41
    例 41

W

Web サービス インターフェイス
    クイック スタート 29
    チュートリアル 30
Web サービス
    概要 28
    クラウド 27
    サンプル事例 31
    前提条件 29
    要件管理 17
    利用可能 33
Web サービス デモ クライアント 62

あ

アイコン
    カスタム 25

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 24
インターフェイス
    createExecutionDefinitions 50
    createLibraries 57
    createRequirements 43
    createTestPlan 37

exportExecutionDefinitions 53
exportLibraryStructure 59
exportLibraryStructureWithoutSteps 60
exportRequirements 45
exportTestPlan 40
getLibraryInfoByName 61
Java インターフェイス 16
reportData 34
TMAttach 35
updateExecutionDefinitions 54
updateRequirements 46
updateRequirementsByExtID 48
updateTestPlan 41
ソース管理 14

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 25

く

クラウド統合 27
クラウド プラグイン 27
クラス 16

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 19
一般プロパティのメタ情報 24
カスタム アイコン 25
サンプル コード 20
実装 18
設定 XML ファイル 23, 25
定義済みパラメータを渡す 19
統合 18
パッケージング 18
ファイル プロパティのメタ情報 24
メタ情報 23
文字列プロパティのメタ情報 24
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 20

し

実装
サードパーティ テスト タイプ プラグイン 18
種
プラグイン 6

せ

セッション処理 33

そ

ソース管理
インターフェイス 14
インターフェイスの慣習 15
統合 14
プラグイン 14

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 19
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 25
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 62

と

同期
要件 17
統合
サードパーティ テスト タイプ プラグイン 18

に

認証
Web サービス 33

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 18

ひ

ビデオ キャプチャ
開始を示す 25
終了を示す 25

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 24
プラグイン
概要 6
クラウド 27
コンパイル 6
種 6
ソース管理 14
デプロイメント 6
配信 6
問題追跡 16
要件管理 17
要件 17
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 23

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 24
問題追跡
    プラグイン 16
問題追跡システムとの統合
    概要 16

よ

要件管理システムとの統合 17
要件プラグイン 17

ろ

ログイン資格情報 33