MICRO FOCUS

Silk Central 21.1

API Help

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

2022-01-27

Contents

Silk Central-API-Hilfe 4
Erstellen von Plug-Ins 4
Integration der Codeabdeckung 5
Erstellen Ihres eigenen Codeabdeckungs-Plug-Ins 6
Installation des Codeanalyse-Systems in einer Linux-AUT-Umgebung 10
Integration der Versionsverwaltung 11
Schnittstelle der Versionsverwaltungsintegration 11
Konventionen für die Integration der Versionsverwaltung 12
Integration der Fehlerverfolgung 12
Java-Schnittstelle 13
Anforderungsverwaltungs-Integrationen 13
Java-Schnittstellen 14
Integration von Drittanbieter-Testtypen 14
Plug-In-Implementierung 15
Struktur der API 15
Beispielcode 16
XML-Konfigurationsdatei 19
Benutzerdefinierte Symbole 21
Testumgebung 21
Angeben von Start und Ende von Videoaufnahmen 22
Cloud Integration 22
Silk Central-Webdienste 23
Webdienste – Kurzanleitung 23
Webdienst-Authentifizierung 27
Verfügbare Webdienste 28
Services Exchange 28
Demo Client für Webdienste 55
Auslösen von Silk Central von einem CI-Server aus 56

Silk Central-API-Hilfe

In diesem Handbuch finden Sie Informationen zur Erstellung und Verteilung von Plug-Ins, um Tools von Drittanbietern in Silk Central zu integrieren. Es umfasst ebenfalls Informationen zur Verwaltung von Ergebnissen externer Testsuiteläufe. Für die verfügbaren SOAP-basierten Webdienste sind in diesem Handbuch Spezifikationen und API-Beschreibungen enthalten. Es wird erklärt, wie Sie Plug-Ins von Drittanbietern in Silk Central integrieren.

Hinweis: In dieser Hilfe wird davon ausgegangen, dass Sie mit der Implementierung und der Verwendung von Webdiensten vertraut sind.

Übersicht

Silk Central bietet SOAP-basierte Webdienste zur Integration von Drittanwendungen sowie eine REST-API zur Verwaltung der Ergebnisse externer Testsuiteläufe.

Mit den SOAP-basierten Webdiensten von Silk Central können Sie vorhandene Tools zur Versionsverwaltung, Fehlerverfolgung und Anforderungsverwaltung integrieren, indem Sie Silk Central-Plug-Ins konfigurieren. Silk Central wird mit einer Reihe von Beispiel-Plug-Ins ausgeliefert.

Mit der REST-API zur Verwaltung der Ergebnisse externer Testsuiteläufe können Sie externe Ergebnisse von Testsuiteläufen, die nicht von Silk Central-Ausführungsservern ausgeführt wurden, zum weiteren Testmanagement in Silk Central hochladen. Sie können bei externen Testsuiten auch angeben, dass sie auf einer externen Ausführungsumgebung anstelle der Silk Central-Ausführungsserver ausgeführt werden.

Dokumentation zu SOAP-basierten Webdiensten

Einzelheiten über die verfügbaren Java-Klassen und -Methoden finden Sie in *Javadoc*. Wenn der Link nicht funktioniert, klicken Sie im Silk Central-Menü auf **Hilfe** > **Dokumentation** > **Silk Central API Spezifikation**, um Javadoc zu öffnen..

Dokumentation zur REST-API

Wenn Silk Central auf Ihrem System installiert ist, können Sie von *hier* aus auf die interaktive Dokumentation der REST-API zugreifen.

Silk Central Integrations-Plug-Ins

Die Silk Central-Plug-Ins werden ohne Mängelgewähr und Gewährleistung jedweder Art geliefert. Micro Focus schließt hiermit jegliche ausdrückliche, konkludente, gesetzliche oder sonstige Gewährleistung aus, insbesondere die stillschweigende Gewährleistung der handelsüblichen Qualität, der Eignung für einen bestimmten Zweck, der Virenfreiheit, der Genauigkeit bzw. Vollständigkeit, des stillen Genusses, des ungestörten Besitzes und der Nichtbeeinträchtigung von Eigentumsrechten Dritter.

Die Verwendung der Plug-Ins erfolgt auf eigenes Risiko. Micro Focus schließt hiermit jegliche Haftung für unmittelbare, mittelbare oder spezielle Schäden, für Schadenersatz und für Folgeschäden jedweder Art, insbesondere entgangenen Gewinn, aus, die sich aus der Verwendung der Plug-Ins ergeben oder damit in Beziehung stehen.

Erstellen von Plug-Ins

Übersicht

In diesem Abschnitt wird beschrieben, wie Sie Plug-Ins für Silk Central erstellen. Hier werden nur die Aufgaben erläutert, die bei der Erstellung aller Arten von Plug-Ins ausgeführt werden müssen.

Plug-In-Art

Silk Central stellt mehrere Plug-In-APIs bereit. Jede API stellt eine *Art* dar.

Kompilierung

In den Versionshinweisen zu Silk Central finden Sie die zum Entwickeln und Kompilieren von Plug-Ins geeignete Java-Version. Dies ist für die Kompatibilität mit der Java-Laufzeitumgebung von Silk Central wichtig. Silk Central verwendet AdoptOpenJDK.

Testumgebung

Nach dem Entwickeln von Plug-In-Klassen und dem Implementieren einer Arten-API können Sie ein Plug-In-Paket (JAR- oder ZIP-Datei) erstellen.

- Wenn Ihr Plug-In keine weiteren Abhängigkeiten aufweist (oder von Bibliotheken abhängt, die bereits Teil von Silk Central sind), brauchen Sie nur eine JAR-Datei zu erstellen, die Ihre Klassen enthält.
- Hängt Ihr Plug-In von anderen Bibliotheken ab, kopieren Sie diese Bibliotheken in das Unterverzeichnis lib, und packen Sie alle Bibliotheken in einem ZIP-Archiv.

Legen Sie die ZIP-Datei im Plug-In-Verzeichnis <Installationsverzeichnis des Anwendungsservers>\plugins \ ab.

Hinweis: Damit das neue Plug-In in Silk Central verfügbar wird, müssen Sie den Anwendungsserver und den Front-End-Server neu starten. Weitere Informationen zum Neustart von Servern finden Sie in den Themen zur *Verwaltung* in diesem Hilfesystem.

Verteilung

Nachdem die Plug-In-Arten in Silk Central bekannt sind, steht auch fest, welche Arten von welchen Servern (Ausführungsserver, Anwendungsserver und Front-End-Server) benötigt werden. Jedes Plug-In kann auf dem Anwendungsserver installiert werden. Silk Central verteilt die Plug-Ins automatisch auf die richtigen Server.

Integration der Codeabdeckung

Die in diesem Kapitel besprochene Java API-Schnittstelle wird benötigt, um Plug-Ins für Silk Central zu erstellen, die die Integration eines externen Codeabdeckungs-Tools erlauben.

Die Codeabdeckungs-Tools liefern Informationen darüber, welcher Code durch Tests abgedeckt wird. Silk Central liefert standardmäßig die folgenden Codeabdeckungs-Tools:

- Silk Central Java Codeanalyse (Java Code Analysis Agent)
- DevPartner Studio .NET Codeanalyse (Windows Code Analysis Framework)

Hinweis: Wenn Ihre zu testende Anwendung unter Linux läuft, lesen Sie das Thema *Installation des Codeanalyse Systems in einer Linux-AUT-Umgebung*.

Falls die zwei vorher genannten Tools nicht ausreichend sind, können Sie Ihre eigene Integration zur Codeabdeckung erstellen und verteilen. Siehe Creating Your Own Code Coverage Plugin.

Hinweis: Zusätzlich zur Verteilung Ihrer maßgeschneiderten Anwendung auf dem Anwendungsserver (Details im Thema *Plug-Ins erstellen*) müssen Sie Ihre maßgeschneiderte Anwendung auf dem Codeanalyse Systemserver verteilen. Hier befindet sich Ihr AUT und Codeabdeckungs-Tool. Der Pfad lautet wie folgt: \Silk\Silk Central <version>\Plugins

Erstellen Ihres eigenen Codeabdeckungs-Plug-Ins

In diesem Thema wird beschrieben, wie ein Codeabdeckungs-Plug-In erstellt wird. Sie sollten mit dem Silk Central-Baseline-Konzept vertraut sein. In Silk Central wird vor jedem Lauf eine Baseline benötigt. Eine Baseline enthält alle Namensräume/Pakete/Klassen/Methoden der Testanwendung.

Hinweis: Das Silk Central-API erfordert die Rücksendung einer XML-Datei für Codeabdeckungsläufe. Dies bedeutet, dass Sie zusätzliche Schritte unternehmen müssen, um Ihre Daten abzurufen, wenn das Codeabdeckungs-Tool die Codeabdeckungsinformationen in einer Datenbank speichert.

Hinweis: Testläufe des Codeanalyse-Systems von verschiedenen Ausführungsservern zur gleichen Zeit werden nicht unterstützt.

1. Fügen Sie dem Klassenpfad die Bibliothek scc.jar hinzu. Diese Bibliothek enthält die Schnittstellen, die Sie erweitern müssen. Sie finden die jar-Datei im Verzeichnis lib des Silk Central-Installationsverzeichnisses.
2. Fügen Sie die folgenden zwei Importanweisungen hinzu:

   ```
   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. Erstellen Sie eine Klasse, die CodeAnalysisProfile implementiert.
4. Fügen Sie alle benötigten Methoden aus der Codeabdeckungsschnittstelle hinzu, welche in den folgenden Schritten aufgelistet sind. Sie können sich für dessen Definition entweder auf die Sample Interface Class beziehen und die Methoden manuell implementieren, oder Sie können den Inhalt von *Sample Profile Class* kopieren und einfügen, da es die Definitionen für Importe und Methoden bereits enthält.

   Hinweis: Die Methoden in den folgenden Schritten, die Sie schreiben werden, werden von Silk Central abgerufen, wenn sie benötigt werden. Das heißt, Sie werden diese nicht direkt abrufen.
5. Code getBaseline. Diese Methode sollte eine XML-Datei erzeugen, die alle Namensräume/Pakete/Klassen/Methoden der Anwendung enthält. Details über das Format der Datei finden Sie in der Themendatei *XML-Beispieldatei*. Überprüfen Sie die XML-Daten unter Verwendung der XSD-Beispieldatei. Details über XSD finden Sie im Thema *Codeabdeckung XSD*.

   Diese Funktion wird aufgerufen, bevor mit der Abdeckung begonnen wird, und wird entweder durch das Starten eines Testlaufs des Silk Central-Ausführungsservers ausgelöst, um die Codeanalyse zu starten und alle abzudeckenden Objekte darzustellen. Die Ausgabe muss unter Verwendung des im XML-Schema spezifizierten Formats, welches im Installationsorder CA-Framework enthalten ist, in XML konvertiert werden.
6. Code startCoverage. Dieser Aufruf sollte dem Codeabdeckungs-Tool mitteilen, mit der Datensammlung zu beginnen. Geben Sie true zurück, wenn die Operation gestartet wurde.

   Dies wird durch das Silk Central-Codeabdeckungssystem aufgerufen, nachdem die Methode getBaseLine() abgeschlossen ist. Jetzt sollten Sie das Codeabdeckungs-Tool zum Sammeln von Codeabdeckungsdaten starten.
7. Code stopCoverage. Dieser Aufruf sollte dem Codeabdeckungs-Tool mitteilen, die Datensammlung zu beenden. Geben Sie true zurück, wenn die Operation erfolgreich durchgeführt wurde.

   Dies wird nach startCoverage aufgerufen und wird ausgelöst wenn der Silk Central-Ausführungsserver einen Testlauf beendet hat.
8. Code getCoverage. Dies erzeugt eine XML-Datei mit den gesammelten Daten von den zwischen den Methoden startCoverage undstopCoverage gesammelten Daten. Details über das Format der Datei finden Sie im Thema *XML-Beispieldatei*. Überprüfen Sie die XML-Daten unter Verwendung der XSD-Beispieldatei. Details über XSD finden Sie im Thema *Codeabdeckung XSD*.

   Diese Funktion wird nach stopCoverage() aufgerufen und gibt alle gesammelten Abdeckungsdaten zurück. Die Ausgabe muss unter Verwendung des im XML-Schema spezifizierten Formats in XML konvertiert werden.

9. Code GetName. Dies sollte den Namen bereitstellen, der als Referenz für das Codeabdeckungs-Tool verwendet wird. So wird zum Beispiel dieser Wert als einer der Werte im Listenfeld **Codeanalyse Profil** im Dialog **Einstellungen für die Codeanalyse bearbeiten** verwendet.

   Dies wird zuerst durch das Silk Central-Codeabdeckungssystem aufgerufen. Der Name des Plug-Ins wird in der Codeabdeckungs-Liste in Silk Central angezeigt.
10. Integrieren Sie das Plug-In in ein Jar-Archiv, und platzieren Sie das Jar-Archiv in einer ZIP-Datei.
11. Verteilen Sie Ihr Plug-In auf die folgenden Servergruppen:
    - Im Verzeichnis Plugins des Silk Central-Installationsordners.
    - Im Verzeichnis Plugins der CA-Framework-Installation.

Beispielprofilklasse

Diese Beispieldatei stellt alle benötigten Methoden, Importe und Hilfsmittel für das Codeabdeckungs-Plug-In dar.

```
//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;
 }

}
```

Codeabdeckung-XSD

Im Folgenden finden Sie die Codeabdeckung-XSD, welche Sie verwenden sollten, um die von Ihrem Codeabdeckungs-Tool erstellte XML-Datei zu validieren. Dieses Dokument kann hier gefunden werden: <CA Framework installation>\CodeAnalysis\CodeCoverage.xsd.

```
<?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>
```

XML-Beispieldatei

Die Codeabdeckung API erwartet XML im folgenden Format.

Sie können auch das bereitgestellte XSD-Beispieldokument verwenden, um Ihre XML-Datei zu validieren.

```
<?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>
```

Installation des Codeanalyse-Systems in einer Linux-AUT-Umgebung

Die folgenden Schritte sollten ausgeführt werden, wenn das von Ihnen zu erstellende Codeabdeckungs-Plug-In mit der zu testenden .NET-Anwendung auf dem Linux-Betriebssystem interagieren soll.

1. Das Codeanalyse System für Linux ist verfügbar unter **Hilfe** > **Tools** > **Linux Codeanalyse System**. Laden Sie dieses herunter und kopieren Sie es in die Stammkonten oder einen anderen Ordner auf der Linux-Maschine.

2. Stellen Sie sicher, dass the latest version of Java Runtime Environment 1.8 auf dem Computer installiert ist, wo die zu testende Anwendung läuft.
3. Entpacken Sie CA-Framework.tar.gz.
4. Legen Sie das Codeanalyse-Plug-In im Ordner <Installationsverzeichnis>/21.1 Silk Central/Plugins ab.
5. Wechseln Sie zum Verzeichnis <Installationsverzeichnis>/21.1 Silk Central/Code Analysis.
6. Finden Sie das Shellskript startCodeAnalysisFramework.sh, welches den Prozess CA-Framework ausführt.
7. Führen Sie den folgenden Befehl aus, um die Datei in das Unix-Format zu konvertieren: dos2unix startCodeAnalysisFramework.sh.
8. Führen Sie den folgenden Befehl aus, um die benötigten Berechtigungen einzustellen, die zur Ausführung des Shellskripts notwendig sind: chmod 775 startCodeAnalysisFramework.sh.
9. Führen Sie das folgende Shellskript aus, um den CAFramework-Prozess auszuführen: ./ startCodeAnalysisFramework.sh.

Das Codeanalyse-System ist zur Verwendung von Silk Central aus bereit.

Integration der Versionsverwaltung

Mithilfe von Versionsverwaltungsprofilen kann Silk Central in externe Versionsverwaltungssysteme integriert werden.

Sie können in benutzerdefinierte Versionsverwaltungs-Plug-Ins konfigurieren und festlegen, wo die Ausführungsserver von Silk Central den Programmcode für die Testausführung abrufen sollen.

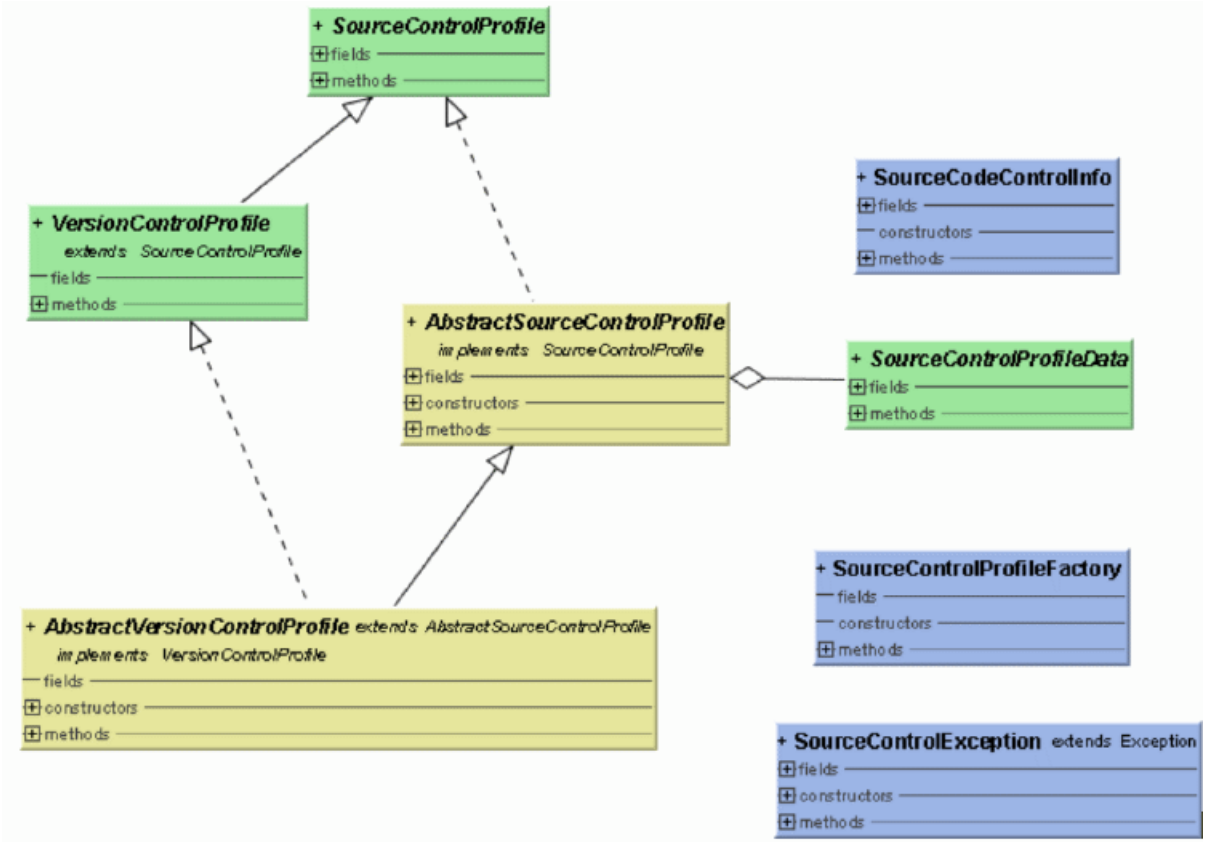
Sehen Sie sich die Sourcen des Pakets com.segue.scc.vcs.subversion auf C:\Program Files (x86)\\Silk \instance_<Instanznummer>_<Instanzname>\Plugins\subversion.zip an, um zu erfahren wie diese Elemente zusammenspielen.

Schnittstelle der Versionsverwaltungsintegration

Silk Central unterscheidet zwischen SourceControlProfile und VersionControlProfile. Der Unterschied besteht darin, dass SourceControlProfile im Gegensatz zu VersionControlProfile keine Versionsnummern hat.

Die folgenden Silk Central-Schnittstellen werden für die Versionsverwaltungsintegration verwendet:

- SourceControlProfile
- VersionControlProfile
- SourceControlProfileData
- SourceControlException
- SourceControlInfo



Einzelheiten über die verfügbaren Java-Klassen und -Methoden finden Sie in *Javadoc*. Wenn der Link nicht funktioniert, klicken Sie im Silk Central-Menü auf **Hilfe** > **Dokumentation** > **Silk Central API Spezifikation**, um Javadoc zu öffnen.

Konventionen für die Integration der Versionsverwaltung

Jede Implementierung muss einen Standardkonstruktor und (optional) einen Konstruktor mit einem SourceControlProfileData-Parameter zur Verfügung stellen. Fehlt der optionale Konstruktor, muss ein Grundgerüst für SourceControlProfileData bereitgestellt werden.

Da in jeder Schnittstellenmethode die auszulösende SourceControlException festgelegt wird, ist es nicht erlaubt, in einer von der Schnittstelle verwendeten Methode eine RuntimeException auszulösen.

Integration der Fehlerverfolgung

Die in diesem Kapitel besprochene Schnittstelle wird benötigt, um Plug-Ins für Silk Central zu erstellen, die die Integration eines externen Fehlerverfolgungssystems (Issue Tracking System, ITS) erlauben.

Durch die Definition von Fehlerverfolgungsprofilen können Sie Tests im Bereich **Tests** mit Fehlern in Verfolgungssystemen von Drittanbietern verknüpfen. Die Statuswerte der verknüpften Fehler werden von externen Verfolgungssystemen in regelmäßigen Abständen aktualisiert.

Sehen Sie sich die Sourcen des Pakets com.segue.scc.issuetracking.bugzilla4 auf C:\Program Files (x86)\Silk\Silk Central21.1\instance_<Instanznummer>_<Instanzname>\Plugins\IT-Bugzilla4.zip an, um zu erfahren wie diese Elemente zusammenspielen.

Java-Schnittstelle

Einzelheiten über die verfügbaren Java-Klassen und -Methoden finden Sie in *Javadoc*. Wenn der Link nicht funktioniert, klicken Sie im Silk Central-Menü auf **Hilfe** > **Dokumentation** > **Silk Central API Spezifikation**, um Javadoc zu öffnen.

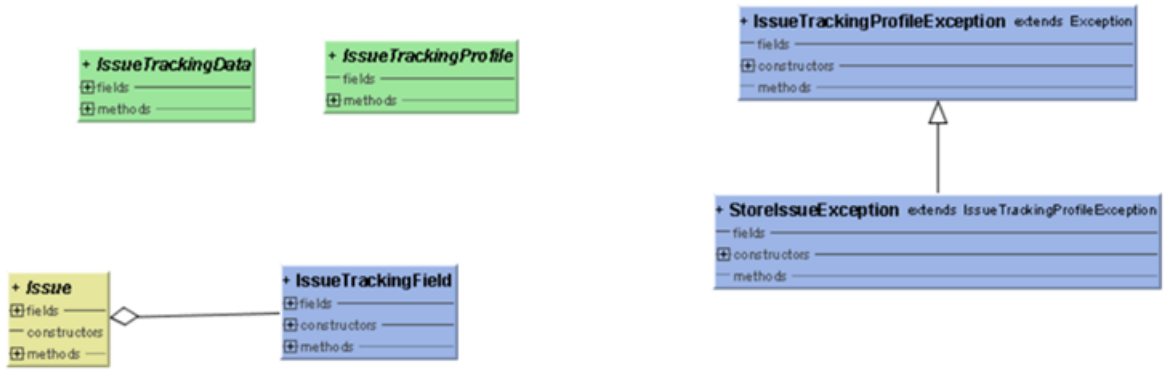
Build-Umgebung

Fügen Sie dem Klassenpfad die Bibliothek scc.jar hinzu. Diese Bibliothek enthält die Schnittstellen, die Sie erweitern müssen. Sie finden die jar-Datei im Verzeichnis lib des Silk Central-Installationsverzeichnisses.

Sie müssen zwei Schnittstellen/Klassen erweitern:

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

Klassen/Schnittstellen



- IssueTrackingProfile
- IssueTrackingData
- Issue
- IssueTrackingField
- IssueTrackingProfileException

Anforderungsverwaltungs-Integrationen

Anforderungsverwaltungssysteme (Requirements Management System, RMS) von Drittanbietern können in Silk Central integriert werden, um Anforderungen zu verknüpfen und zu synchronisieren.

In diesem Abschnitt erfahren Sie, wie sich mit Hilfe der Java-API Plug-Ins implementieren lassen, um die Anforderungen in Silk Central mit den Anforderungen des Anforderungsverwaltungssystems eines Drittanbieters zu synchronisieren. In diesem Abschnitt werden die Schnittstellen beschrieben, die ein Anforderungs-Plug-In und seine Bereitstellung identifizieren.

Die bereitgestellte JAR- oder ZIP-Datei unterstützt das Standard-Plug-In-Konzept von Silk Central. Sie wird automatisch an alle Front-End-Server verteilt und ermöglicht so den Zugriff auf Tools von Drittanbietern zum Zweck der Konfiguration und Synchronisation. Das Plug-In implementiert eine spezielle Schnittstelle, mit deren Hilfe es von Silk Central als Anforderungs-Plug-In erkannt wird. Es stellt die benötigten Daten für die Anmeldung bei einem Tool eines Drittanbieters bereit.

Einzelheiten über die verfügbaren Java-Klassen und -Methoden finden Sie in *Javadoc*. Wenn der Link nicht funktioniert, klicken Sie im Silk Central-Menü auf **Hilfe** > **Dokumentation** > **Silk Central API Spezifikation**, um Javadoc zu öffnen.

Wenn Sie Informationen zu weiteren Plug-Ins benötigen, wenden Sie sich an den Kundensupport.

Java-Schnittstellen

Einzelheiten über die verfügbaren Java-Klassen und -Methoden finden Sie in *Javadoc*. Wenn der Link nicht funktioniert, klicken Sie im Silk Central-Menü auf **Hilfe** > **Dokumentation** > **Silk Central API Spezifikation**, um Javadoc zu öffnen.

Die grundlegende Schnittstelle für den Anfang ist RMPluginProfile (com.segue.tm.published.api.requirements.javaplugin). RMPluginProfile gibt an, dass das Plug-In ein Anforderungs-Plug-In ist.

Die Anforderungs-Java-Plug-In-API umfasst die folgenden zusätzlichen Schnittstellen:

- RMAction
- RMAttachment
- RMDataProvider
- *Optional:* RMIconProvider
- RMNode
- RMNodeType
- RMPluginProfile
- RMProject
- RMTest
- RMTestParameter
- *Optional:* RMTestProvider
- RMTestStep

Integration von Drittanbieter-Testtypen

Silk Central erlaubt das Erstellen benutzerdefinierter Plug-Ins für Testtypen, die nicht zur Standardreihe der verfügbaren Testtypen gehören. Die Standardreihe umfasst Silk Performer, Silk Test Classic, manuelle Tests, NUnit, JUnit und Windows Scripting Host (WSH). Nach dem Erstellen eines neuen Testtyp-Plug-Ins wird der benutzerdefinierte Testtyp in dem Listenfeld **Typ** des Dialogfeldes **Neue Test** zusammen mit den Standard-Testtypen angezeigt, die in Silk Central für neue Tests zur Verfügung stehen.

Ein Plug-In legt fest, welche Eigenschaften benötigt werden, um eine Test zu konfigurieren und die Ausführung eines Tests zu implementieren. Die Metadaten der Eigenschaften werden über eine *XML-Konfigurationsdatei* definiert.

Das Ziel des Plug-In-Ansatzes besteht darin, Tests zu unterstützen, die auf gebräuchlichen Test-Frameworks wie JUnit und NUnit oder auf Skriptsprachen (WSH) beruhen. Dies erleichtert die Anpassung von Silk Central an eine spezielle Testumgebung. Die klar definierte öffentliche API von Silk Central erlaubt die Implementierung einer eigenen Lösung, die den Anforderungen automatisierter Tests entspricht. Silk Central kann durch jedes Drittanbieter-Tool erweitert werden, das über eine Java-Implementierung oder über die Befehlszeile aufgerufen werden kann.

Einzelheiten über die verfügbaren Java-Klassen und -Methoden finden Sie in *Javadoc*. Wenn der Link nicht funktioniert, klicken Sie im Silk Central-Menü auf **Hilfe** > **Dokumentation** > **Silk Central API Spezifikation**, um Javadoc zu öffnen.

Die in Javadoc beschriebenen Klassen sind in der Datei tm-testlaunchapi.jar enthalten.

Wenn Sie Informationen zu weiteren Plug-Ins benötigen, wenden Sie sich an den Kundensupport.

Dieser Abschnitt enthält ein Codebeispiel, das den Teststyp "Process Executor" implementiert. Mit "Process Executor" kann eine beliebige ausführbare Datei aufgerufen werden. Es erweitert die öffentliche

Klasse "Process Test Launcher". Weitere Informationen finden Sie im *Beispiel Teststart-Plug-In*, das Sie unter **Hilfe** > **Tools** herunterladen können, sowie in der Datei Readme.txt.

Plug-In-Implementierung

Die Richtlinien der API basieren auf dem weit verbreiteten Java Beans-Konzept. Entwickler können dadurch auf einfache Weise Teststart-Plug-Ins implementieren. Damit der Java-Code keine Textinformationen enthält, werden Metadaten von Eigenschaften in einer XML-Datei festgelegt.

Die Plug-In-Implementierung ist in einem .ZIP-Archiv komprimiert und implementiert eine Callback-Schnittstelle, um integriert werden zu können. Weitere Schnittstellen werden vom Plug-In-Framework bereitgestellt. Sie ermöglichen der Implementierung den Zugriff auf Informationen oder Rückgabewerte.

Komprimierung

Das Plug-In ist in einem ZIP-Archiv komprimiert, das die Java-Codebase und die XML-Konfigurationsdatei enthält. Im Archiv befinden sich außerdem einige Implementierungen von Teststart-Plug-Ins. Die Codebase kann in einem Java-Archiv (.jar) oder direkt in .class-Dateien enthalten sein, wobei die Ordner die Struktur des Java-Pakets darstellen.

Die TestLaunchBean-Plug-In-Klasse folgt dem Bean-Standard und implementiert die TestLaunchBean-Schnittstelle. Die XML-Konfigurationsdatei im zip-Archiv hat denselben Namen wie ihre Klasse. Dadurch können Sie mehrere Plug-Ins und XML-Dateien in einem einzigen Archiv komprimieren.

Übergabe von Parametern an das Plug-In

Wenn ein Plug-In auf der ExtProcessTestLaunchBean-Klasse basiert, wird jeder Parameter in dem vom Plug-In gestarteten Prozess automatisch als Umgebungsvariable festgelegt. Das ist auch dann der Fall, wenn der Parametername mit dem Namen einer Systemvariable übereinstimmt, sodass der Wert der Systemvariable durch den Parameterwert ersetzt wird (außer wenn der Parameterwert eine leere Zeichenfolge ist).

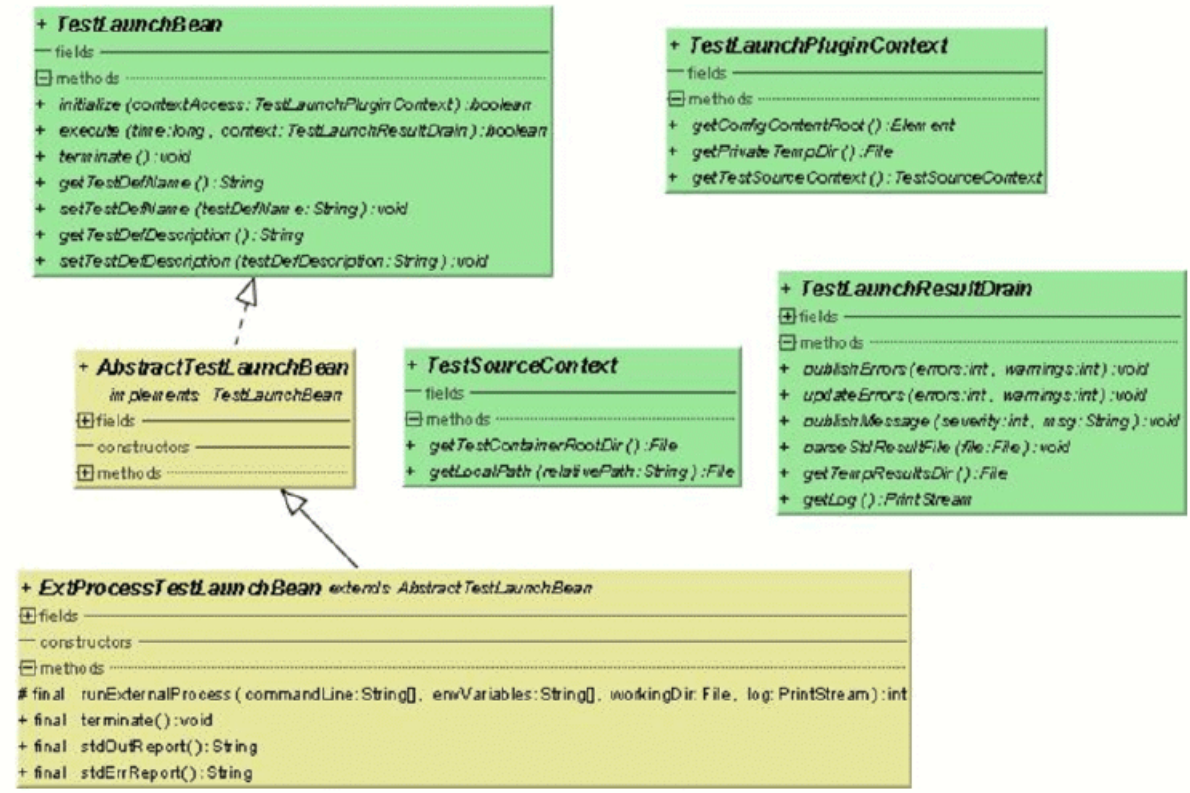
Die Plug-In-Schnittstelle erlaubt den uneingeschränkten Zugriff auf alle benutzerdefinierten Parameter, die im Bereich **Tests** von Silk Central definiert wurden. Für Testtypen von Drittanbietern werden nur benutzerdefinierte Parameter unterstützt. Das Plug-In kann keine vordefinierten Parameter verwenden. Die Implementierung entscheidet darüber, ob und wie Parameter für spezielle Tests definiert werden.

Mit der Methode getParameterValueStrings() der Schnittstelle TestLaunchPluginContext können Sie einen Container mit Zuordnungen zwischen Parameternamen (Schlüssel) und ihren Werten in der String-Darstellung abrufen.

Bei JUnit-Testtypen kann jede beliebige JUnit-Testklasse auf einen benutzerdefinierten Parameter des zugrunde liegenden Tests wie auf eine Java-Systemeigenschaft zugreifen. Das Startprogramm übergibt diese Parameter über das Argument "-D" an die ausführende VM.

Struktur der API

Diese Abbildung zeigt detailliert die Struktur der API für die Integration von Drittanbieter-Testtypen.



Einzelheiten über die verfügbaren Java-Klassen und -Methoden finden Sie in *Javadoc*. Wenn der Link nicht funktioniert, klicken Sie im Silk Central-Menü auf **Hilfe** > **Dokumentation** > **Silk Central API Spezifikation**, um Javadoc zu öffnen.

Verfügbare Schnittstellen

- TestLaunchBean
- ExtProcessTestLaunchBean
- TestLaunchPluginContext
- TestSourceContext
- TestLaunchResultDrain

Beispielcode

Dieser Beispielcodeblock implementiert den Testtyp "Process Executor", mit dem eine ausführbare Datei aufgerufen werden kann und das die öffentliche ExtProcessTestLaunchBean-Klasse erweitert.

```
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()]);
  }
}
```

XML-Konfigurationsdatei

Die XML-Konfigurationsdatei enthält Metadaten über das Testtyp-Plug-In eines Drittanbieters.

Plug-In-Metadaten

Plug-In-Metadaten stellen Informationen über das Plug-In und den Teststyp bereit. Die folgenden Metadatentypen sind verfügbar:

Typ	Beschreibung
id	Ein interner String, der den Typ identifiziert und in allen installierten Plug-Ins eindeutig ist.

Typ	Beschreibung
label	Der Text, der in Auswahllisten und Bearbeitungsdialogfeldern der Benutzeroberfläche für den Typ angezeigt wird.
description	Zusätzlicher Text, der den Teststyp beschreibt.
version	Zur Unterscheidung verschiedener Versionen eines Typs.

Allgemeine Metadaten für Eigenschaften

Für bearbeitbare Eigenschaften gibt es zusätzlich zu den typspezifischen Informationen allgemeine Informationen, die für jeden Eigenschaftstyp gleich sind. Der Name einer Eigenschaft muss den Namen entsprechen, die im Code durch get<<propertyname>>-Methoden definiert werden. Das erste Zeichen muss ein Kleinbuchstabe sein.

Typ	Beschreibung
label	Dieser Text wird in der Benutzeroberfläche angezeigt.
description	Zusätzlicher Text, der die Eigenschaft beschreibt.
isOptional	Der Wert "true" bedeutet, dass keine Benutzereingabe erforderlich ist.
default	Der voreingestellte Wert der Eigenschaft, wenn ein neuer Test erstellt wird. Der Wert muss dem Typ der Eigenschaft entsprechen.

Metadaten für String-Eigenschaften

Hier werden die Typen der Metadaten für String-Eigenschaften aufgeführt, die in dem Plug-In verfügbar sind.

Typ	Beschreibung
maxLength	Die maximale Zeichenanzahl, die der Benutzer eingeben kann.
editMultiLine	Gibt an, ob das Eingabefeld mehrere Zeilen haben soll.
isPassword	Bei true wird die Eingabe des Benutzers in *** umgewandelt.

Metadaten für Dateieigenschaften

Hier werden die Typen der Metadaten für Dateieigenschaften aufgeführt, die in dem Plug-In verfügbar sind.

Bei bestimmten Dateiwerten wird unterschieden, ob es sich um eine Datei ohne Versionsverwaltung mit absolutem Pfad auf dem Ausführungsserver oder um eine Datei mit Versionsverwaltung handelt. Der Pfad von Dateien unter Versionsverwaltung ist immer relativ zum Stammverzeichnis des Containers. Solche Dateien werden normalerweise verwendet, um eine auszuführende Testquelle anzugeben. Absolute Pfade auf dem Ausführungsserver verweisen normalerweise auf ein Tool oder eine Ressource, die den Test auf dem Ausführungsserver aufruft.

Typ	Beschreibung
openBrowser	Der Wert true bedeutet, dass ein Browser geöffnet werden soll, um die Datei aus den Dateien in einem Container auszuwählen.

Typ	Beschreibung
isSourceControl	Der Wert true bedeutet, dass die Datei aus einem Versionsverwaltungssystem stammt.
fileNameSpec	Bezeichnet eine Einschränkung für zulässige Dateinamen wie im Standard-Dialogfeld "Durchsuchen" von Windows.

Benutzerdefinierte Symbole

Sie können eigene Symbole für Ihren Testtyp entwerfen, damit der Testtyp schnell erkannt werden kann. Um diese Symbole für das Plug-In festzulegen, für das Sie in der XML-Konfigurationsdatei den Bezeichner PluginId definiert haben, müssen Sie die folgenden vier Symbole in das Stammverzeichnis des Plug-Ins kopieren.

Name	Beschreibung
<PluginId>.gif	Das Standardsymbol für Ihren Testtyp. Zum Beispiel: ProcessExecutor.gif.
<PluginId>_package.gif	Das Symbol für den Testpaketstamm und die Suiteknoten für den Fall, dass Sie einen Test des angegebenen Testtyps in ein Testpaket konvertieren. Zum Beispiel: ProcessExecutor_package.gif.
<PluginId>_linked.gif	Das Symbol wird verwendet, wenn der einem Test übergeordnete Ordner mit einem Container verknüpft wird. Zum Beispiel: ProcessExecutor_linked.gif.
<PluginId>_incomplete.gif	Das Symbol wird verwendet, wenn das Produkt oder das Versionsverwaltungsprofil des übergeordneten Containers des Tests nicht definiert ist.

Beachten Sie beim Erstellen eines neuen Symbols für einen Testtyp folgende Regeln:

- Verwenden Sie nur GIF-Symbole. Bei der Dateierweiterung wird die Groß-/Kleinschreibung berücksichtigt. Die Dateierweiterung muss immer in Kleinbuchstaben (.gif) angegeben werden.
- Entfernen Sie alte oder ungültige Symbole unter <Silk Central deploy folder>\wwwroot\silkroot\img \PluginIcons, da die Symbole andernfalls im Stammverzeichnis des Plug-Ins nicht durch die neuen Symbole ersetzt werden.
- Das Symbol hat eine Größe von 16x16 Pixel.
- Maximal sind 256 Farben für Symbole zulässig.
- Das Symbol beinhaltet 1 Bit für die Transparenz.

Testumgebung

Das ZIP-Archiv mit dem Plug-In muss sich im Unterverzeichnis Plugins im Installationsverzeichnis von Silk Central befinden. Um Plug-Ins zu integrieren, die sich in diesem Verzeichnis befinden, starten Sie den Anwendungsserver und den Front-End-Server mithilfe von Silk Central Service Manager neu.

Hinweis: Andere Integrationsverfahren werden nicht unterstützt.

Immer wenn ein Archiv geändert wird, müssen diese beiden Server neu gestartet werden. Das Archiv wird automatisch zu den Ausführungsservern hochgeladen.

Hinweis: Entfernen Sie niemals ein Plug-In-Archiv, nachdem ein auf diesem Plug-In basierender Test erstellt wurde. Ein Test, der auf einem nicht mehr vorhandenen Plug-In-Archiv beruht, löst bei Änderung oder Ausführung unbekannte Fehler aus.

Angeben von Start und Ende von Videoaufnahmen

Einzelheiten über die verfügbaren Java-Klassen und -Methoden finden Sie in *Javadoc*. Wenn der Link nicht funktioniert, klicken Sie im Silk Central-Menü auf **Hilfe** > **Dokumentation** > **Silk Central API Spezifikation**, um Javadoc zu öffnen.

Wenn Sie für Silk Central ein neues Drittanbieter-Test-Plug-In erstellen, das einen Drittanbieter-Testtyp zur Unterstützung mehrerer Testfälle in einer einzelnen Testausführung enthält, und Sie aufgezeichnete Videos zu bestimmten Testfällen zuordnen möchten, dann stehen Ihnen zwei Möglichkeiten zur Verfügung.

Drittanbieter-Tests, die in dem Plug-In ausgeführt werden

Für diese Tests empfehlen wir die Verwendung der Methoden indicateTestStart und indicateTestStop der Klasse TestLaunchResultDrain.

Drittanbieter-Tests, die in einem externen Prozess ausgeführt werden

Für diese Tests können Sie einen TCP/IP-basierten Dienst zum Senden von START- und FINISH-Meldungen an den Port des Silk Central-Ausführungsservers verwenden. Die zu verwendende Portnummer kann von *ExecutionContextInfo.ExecProperty#PORT_TESTCASE_START_FINISH* im Plug-In abgefragt werden. Der Port ist auch als Umgebungsvariable *#sctm_portTestCaseStartFinish* im Testprozess verfügbar, wenn das Plug-In ExtProcessTestLaunchBean erweitert. Anhand dieser Meldungstypen wird der Ausführungsserver informiert, dass ein Testfall in dem Test begonnen oder beendet wurde. Die Meldungen müssen im Unicode- (UTF8) oder ASCII-Format verschlüsselt sein.

Meldungstyp	Format
START	START <Test Name>, <Test ID> <LF>, wobei LF den ASCII-Code 10 hat.
FINISH	FINISH <Test Name>, <Test ID>, <Passed> LF, wobei LF den ASCII-Code 10 hat. Passed kann True oder False sein. Wenn die Videoaufzeichnung so eingestellt ist, dass sie Im Fehlerfall ausgeführt wird, wird das Video nur im Ergebnis gespeichert, wenn Passed auf Falsch gesetzt ist.

Wenn die Anfrage erkannt wurde, gibt der Ausführungsserver die Meldung OK zurück; andernfalls gibt er eine Fehlermeldung aus. Warten Sie immer erst die Reaktion des Ausführungsservers ab, bevor Sie den nächsten Testfall ausführen, da das aufgezeichnete Video und der tatsächliche Testfall sich sonst möglicherweise nicht entsprechen.

Basiert der externe Prozess, in dessen Rahmen der Test ausgeführt wird, auf einer Java-Umgebung, empfehlen wir die Verwendung der Methoden indicateTestStart und indicateTestStop der Klasse TestCaseStartFinishSocketClient, die in der Datei tm-testlaunchapi.jar enthalten ist.

Cloud Integration

Silk Central ermöglicht die Integration mit Anbietern von öffentlichen oder privaten Cloud Services, indem Cloud Profile konfiguriert werden. Cloud Profile basieren auf einem Plug-in Konzept, welches Ihnen die Möglichkeit bietet Ihr eigenes Plug-in für einen bestimmten Cloud Anbieter zu entwickeln. Ein Cloud-Anbieter Plug-in richtet vor jedem automatisierten Testlauf eine virtuelle Umgebung ein.

Hinweis: Die Cloud API wird voraussichtlich in einer zukünftigen Version von Silk Central Änderungen erhalten. Wenn Sie diese API verwenden, kann die Aktualisierung auf eine zukünftige Version von Silk Central bedeuten, dass Sie Ihre Plug-in Implementation anpassen müssen.

Einzelheiten über die verfügbaren Java-Klassen und -Methoden finden Sie in *Javadoc*. Wenn der Link nicht funktioniert, klicken Sie im Silk Central-Menü auf **Hilfe** > **Dokumentation** > **Silk Central API Spezifikation**, um Javadoc zu öffnen.

Die Basisschnittstelle ist CloudProviderProfile (com.segue.scc.published.api.cloud). Die Schnittstelle CloudProviderProfile spezifiziert den Zugang und den Umgang mit einem externen Cloud-System. Die Implementation eines Cloud-Anbieter Plug-Ins soll folgendes erreichen:

- Aufzeigen, welche Eigenschaften in einem Cloud-Anbieter-Profil konfiguriert werden müssen, um von extern auf einen Anbieter dieses Typs zugreifen zu können.
- Validieren von Profileigenschaften und Überprüfung der Verbindung zum Cloud-Anbieter.
- Eine Liste verfügbarer Image-Vorlagen vom Cloud-Anbieter beziehen.
- Einrichten einer virtuellen Umgebung, basierend auf der ausgewählten Image-Vorlage, und aufzeigen der extern erreichbaren Hostadressen.
- Überprüfen, ob eine virtuelle Umgebung eingerichtet und am Laufen ist.
- Löschen einer bestimmten virtuellen Umgebung.

Silk Central-Webdienste

SOAP-basierte Webdienste für Silk Central erfordern keine gesonderte Einrichtung. Sie werden auf jedem Front-End-Server standardmäßig aktiviert. Wenn z. B. http://www.yourFrontend.com/login der URL ist, über den Sie auf Silk Central zugreifen, sind http://www.yourFrontend.com/Services1.0/jaxws/system (Legacy-Dienste auf http://www.yourFrontend.com/Services1.0/services) und http://www.yourFrontend.com/AlmServices1.0/services die Basis-URLs für den Zugriff auf die verfügbaren Webdienste.

Wenn Sie mit Ihrem Browser auf den Basis-URL zugreifen, wird eine einfache HTML-Liste aller verfügbaren Webdienste angezeigt. Diese Liste wird von JAX-WS bereitgestellt. Silk Central verwendet den SOAP-Stack *https://jax-ws.java.net/*.

Der Basis-URL stellt Links zu XML-Dateien im WSDL (Web Service Definition Language)-Standard bereit, wobei jede Datei die Schnittstelle eines bestimmten Webdienstes beschreibt. Diese Dateien sind nur für Programme verständlich. Deshalb lesen SOAP-fähige Mandanten (z. B. Silk Performer Java Explorer) WSDL Dateien, wodurch Informationen abgerufen werden, die erforderlich sind, um Methoden der entsprechenden Webdienste aufzurufen.

Einzelheiten über die verfügbaren Java-Klassen und -Methoden finden Sie in *Javadoc*. Wenn der Link nicht funktioniert, klicken Sie im Silk Central-Menü auf **Hilfe** > **Dokumentation** > **Silk Central API Spezifikation**, um Javadoc zu öffnen.

Informationen zur Verwaltung von Testsuiteläufen, die nicht auf den Silk Central-Ausführungsservern, sondern auf externen Ausführungsumgebungen ausgeführt werden, sowie zur Verwaltung der Ergebnisse finden Sie in der REST-API-Dokumentation von Silk Central. Wenn Silk Central auf Ihrem System installiert ist, können Sie von *hier* aus auf die interaktive Dokumentation der REST-API zugreifen.

Webdienste – Kurzanleitung

Dieser Abschnitt enthält Voraussetzungen, ein Anwendungsfallbeispiel und weitere Themen zur Integration von Webdiensten.

Voraussetzungen

Bevor Sie versuchen, Webdienst-Clients zu erstellen, sollten Sie die folgenden Voraussetzungen erfüllen:

- Sie benötigen grundlegende Kenntnisse in der objektorientierten Programmierung (OOP). Erfahrung mit Java ist hilfreich, da die Beispiele in dieser Sprache vorliegen. Für Entwickler ohne Java-Kenntnisse, aber mit Erfahrung in C++, C#, Python oder Perl sollten die Beispiele kein Problem darstellen. Erfahrungen im Umgang mit Collections wie HashMaps und Listen sind von Vorteil.
- Gelegentlich werden JUnit-Tests erwähnt. Diese Java-Testumgebung ist nicht erforderlich, aber hilfreich.
- Einige praktische Kenntnisse der Webdienst-Technologie sollten vorhanden sein. Diese Hilfe stellt keinen Lehrgang über Webdienste oder SOAP dar. Der Leser sollte jedoch zumindest einen "Hello World!"-Webdienst-Client programmiert und erfolgreich ausgeführt haben.

- Machen Sie sich mit der Webdienst-Architektur von Silk Central vertraut.

Einführung in Webdienste

Stellen Sie anhand der *Silk Central Versionshinweise* sicher, dass Sie die geeignete Java SDK-Version installiert und in die PATH-Variable aufgenommen haben.

Hilfreich ist auch das JUnit-Archiv im Klassenpfad. Sie können die Datei JUnit.jar von *http://www.junit.org/index.htm* herunterladen.

1. Stellen Sie sicher, dass der bin-Ordner des Java SDK in Ihrem PATH eingetragen ist.
2. Führen Sie den folgenden Befehl aus, der auf die WSDL des gewünschten Webdienstes zeigt:

   ```
   wsimport -s <Pfad wo die generierten Dateien gespeichert werden sollen> -Xnocompile
   -p <Paketstruktur die Sie für Ihren yourWebService-Client verwenden möchten>
   http://<URL zu Ihrem Dienst>/yourWebService?wsdl
   ```
3. Suchen Sie nach der automatisch generierten Java-Klasse YourWebService.

   Diese Klasse ist bereits einsatzfähig und kann jede Methode ausführen, die YourWebService bereitstellt.

Webdienst-Client – Übersicht

Webdienste verwenden normalerweise SOAP anstelle des HTTP-Protokolls. In einem solchen Szenario werden SOAP-Pakete gesendet. Wenn Collections und andere komplexe Objekte in SOAP-Paketen gebündelt werden, ist es kaum möglich, die ASCII-Datenstrukturen zu lesen und zu bearbeiten. Unerfahrene Entwickler sollten nicht versuchen, einen Webdienst-Client durch direkte Bearbeitung von SOAP-Paketen zu erstellen. Erfahrene Entwickler erstellen normalerweise keine Webdienst-Clients auf SOAP-Paket-Ebene. Ein solches Vorgehen wäre mühsam und fehleranfällig. Aus diesem Grund stellen alle wichtigen Programmiersprachen Entwicklungs-Kits für Webdienste bereit. In Silk Central wird *Java API for XML Web Services (JAX-WS)* zur Erstellung von Webdiensten und Clients verwendet, die über SOAP-Nachrichten kommunizieren.

Unabhängig von der Programmiersprache (Java, C++, C#, Perl oder Python) folgt die Erstellung von Webdienst-Clients einem immer gleichen Muster:

1. Weisen Sie einem Entwicklungs-Kit die Webdienst-WSDL zu.
2. Ermitteln Sie einen Client-Rumpf.
3. Bearbeiten Sie den in Schritt 2 generierten Client-Rumpf so, dass Sie einen funktionsfähigen Client erhalten.

JAX-WS folgt diesem Muster. In unseren Beispielen wird das Tool "wsimport" (im JDK enthalten) verwendet, um Client-Stubs aus der WSDL zu erstellen. Weitere Informationen über die Verwendung von wsimport finden Sie auf *JDK Tools and Utilities documentation*. A brief description of the switches used in the above summary is as follows:

- -s: Das Ausgabeverzeichnis der Client-Rümpfe
- -p: Das Zielpaket. In dieser Paketstruktur einsetzen

Beispiel: wsimport -s <Speicherort der generierten Stubs> -p <Zielpaket> <WSDL>

Das Tool wsimport generiert mehrere Klassen, die einen Client für den Webdienst unterstützen. Wenn der Name des Dienstes YourWebService ist, werden die folgenden Klassen ausgegeben:

- YourWebService: Eine Schnittstelle, die YourWebService repräsentiert.
- YourWebServiceService: Eine generierte Klasse, die den Locator YourWebService repräsentiert, z.B. um eine Liste der verfügbaren Ports zu erhalten (Service-Endpunkt-Schnittstellen)
- WSFaultException: Eine Ausnahme-Klasse, die aus wsdl:fault abgebildet wird
- WsFaultBean: Asynchrones Antwort-Bean, abgeleitet von wsdl:message
- Serializeable Objects: Objekte auf Seiten des Clients, die Objekten entsprechen, die YourWebService verwendet.

Erstellen Sie eine neue Java-Klasse namens YourWebServiceClient, um einen JAX-WS-Client zu generieren, mit dem Silk Central-Webdienste verwendet werden können. Der Webdienst muss durch einen Port gebunden werden; ein Port ist ein logisches Objekt, das als Proxy für den Remote-Dienst auftritt. Beachten Sie, dass der Port durch die Ausführung des wsimport-Tools im oberen Schritt erstellt wurde. Um diesen Proxy zu erhalten, rufen Sie die Methode getRequirementsServicePort auf dem Dienst auf:

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

Um sich bei den Webdiensten zu authentifizieren, generieren Sie ein Webdienst-Token auf der Seite **Benutzereinstellungen** der Silk Central-Benutzeroberfläche. Um auf diese Seite zuzugreifen, zeigen Sie mit dem Mauszeiger auf den Benutzernamen im Menü Silk Central und wählen Sie **Benutzereinstellungen**.

Sie können ebenfalls die Methode logonUser des Dienstes aufrufen, indem Sie Benutzername und Kennwort übergeben, um eine Sitzungs-ID zu erhalten:

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

Anwendungsfallbeispiel: Hinzufügen einer Anforderung

Aufbauend auf den vorausgehenden Schritten dieses Abschnitts schließt dieses Thema den Anwendungsfall "Eine Anforderung zu Silk Central hinzufügen" ab.

Bevor Sie fortfahren, müssen Sie die folgenden Voraussetzungen erfüllt haben:

- Sie haben die Schritte für den Webdienst requirements abgeschlossen.
- Eine funktionierende POJO- oder JUnit-Klasse mit Bindungs- und Anmeldemethoden wurde erstellt.
- Sie haben die anderen Silk Central-API-Hilfethemen gelesen.

1. Generieren Sie auf der Seite **Benutzereinstellungen** ein Webdienst-Token.
   a) Klicken Sie auf den Benutzernamen im Silk Central-Menü.Die Seite **Benutzereinstellungen** wird geöffnet.
   b) Klicken Sie im Abschnitt **Webdienst-Token** der Seite auf **Token generieren**.
2. Konstruieren Sie ein Anforderungsobjekt, das die gewünschten Daten enthält.
3. Rufen Sie die Methode updateRequirement mit dem Webdienst-Token, der Projekt-ID und dem Anforderungsobjekt auf, die Sie generiert haben.
4. Speichern Sie die von der Methode updateRequirement zurückgegebene Anforderungs-ID.
5. Erstellen Sie ein PropertyValue-Array mit den Anforderungseigenschaften.
6. Rufen Sie die Methode updateProperties mit dem zuvor erzeugten Array auf.

wsimport erstellt die genannten Webdienst-Objekte:

- Requirement
- PropertyValue

Die OOP-Methoden der genannten Objekte können nun verwendet werden, um den Webdienst in Anspruch zu nehmen. Das umständliche Zusammenstellen von SOAP-Paketen ist dadurch überflüssig. Im Folgenden finden Sie Auszüge aus dem Code, der benötigt wird, um den Testfall auszuführen.

```
/** 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();

// The web-service token that you have generated in the UI. Required to authenticate when using
// a web service.
String webServiceToken = "e39a0b5b-45db-42db-84b2-b85028d954d5";

// 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()
*/
int requirementID = service.updateRequirement(webServiceToken, PROJECT_ID, topLevelRequirement,
-1);

// Now loop through and set properties
for (PropertyValue propValue : properties) {
propValue.setRequirementId(requirementID);
service.updateProperty(webServiceToken, 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(webServiceToken, PROJECT_ID, childRequirement, -1);

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

// Print Results
System.out.println("Login Successful with web-service token: " + webServiceToken);
```

```
System.out.println("Top Level Requirement ID: " + requirementID);
System.out.println("Child Requirement ID: " + childReqID);
```

Hinweis: Diesen Beispliel-Code finden Sie auch in der Klasse com.microfocus.silkcentral.democlient.samples.AddingRequirement des „Web Services Demo-Client"‘.

Webdienst-Authentifizierung

Silk Central-Daten sind gegen unberechtigten Zugriff geschützt. Bevor der Zugriff auf Daten erlaubt wird, müssen Anmeldeinformationen vorgelegt werden. Dies gilt nicht nur für die Arbeit mit dem HTML-Front-End, sondern auch für die Kommunikation mit Silk Central über SOAP- oder REST-API-Aufrufe.

Der erste Schritt beim Abfragen von Daten oder beim Ändern der Konfigurationseinstellungen von Silk Central besteht daher in der Authentifizierung. War die Authentifizierung erfolgreich, wird eine Benutzersitzung generiert, die die Ausführung der nachfolgenden Operationen im Kontext dieser Anmeldung erlaubt.

Wenn über einen Webbrowser auf Silk Central zugegriffen wird, sind die Sitzungsinformationen für den Benutzer nicht sichtbar. Der Browser verwaltet Sitzungsinformationen über Cookies. Im Gegensatz zum Zugriff auf Silk Central über HTML müssen SOAP-Aufrufe diese Sitzungsinformationen manuell verwalten.

Micro Focus empfiehlt die Authentifizierung über ein Webdienst-Token. Ein Webdienst-Token können Sie auf der Seite **Benutzereinstellungen** der Silk Central-Benutzeroberfläche generieren. Um auf diese Seite zuzugreifen, zeigen Sie mit dem Mauszeiger auf den Benutzernamen im Menü Silk Central und wählen Sie **Benutzereinstellungen**.

Sie können auch den SOAP-Aufruf logonUser oder den REST-API-Aufruf login zur Authentifizierung verwenden. Der Methodenaufruf gibt eine Sitzungs-ID zurück, die zur Identifizierung der auf dem Server generierten Sitzung und zugleich als Schlüssel für den Zugriff auf Silk Central im Kontext dieser Sitzung dient.

Jeder nachfolgende API-Aufruf, für den eine Authentifizierung erforderlich ist, übernimmt ein Webdienst-Token oder eine Sitzungs-ID als Parameter, prüft deren Gültigkeit und wird im Kontext der entsprechenden Sitzung ausgeführt.

Silk Central-Sitzungen, die über Webdienste generiert werden, können nicht explizit beendet werden. Sie enden stattdessen automatisch, wenn sie für einen bestimmten Zeitraum nicht benutzt werden. Nachdem eine Sitzung durch ein Timeout auf dem Server beendet wurde, lösen nachfolgende SOAP-Aufrufe beim Versuch, auf die Sitzung zuzugreifen, eine Ausnahme aus.

Eine Demo-Anwendung können Sie in Silk Central unter **Hilfe** > **Tools** > **Web Services Demo-Client** herunterladen. In diesem Demoprojekt wird der tests-Webdienst von Silk Central verwendet, damit Sie die Webdienst-Schnittstelle kennenlernen.

Beispiele

Für ein Webdienst-Token, das in der Benutzeroberfläche von Silk Central generiert wurde, veranschaulicht das folgende Java-Codebeispiel den Zugriff auf Silk Central über Webdienste und die Verwendung des Webdienst-Token:

```
string webServiceToken = "e39a0b5b-45db-42db-84b2-b85028d954d5";
Project[] projects = sccentities.getProjects(webServiceToken);
```

Mit dem folgenden Java-Codebeispiel wird derselbe Zugriff auf Silk Central über Webdienste sowie die Verwendung der Sitzungs-ID veranschaulicht:

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

Verfügbare Webdienste

Die folgende Tabelle zeigt die verfügbaren Silk Central-Webdienste. Informationen zum Zugriff auf Silk Central über HTTP-basierte Schnittstellen finden Sie unter *Services Exchange*. Informationen zu den REST-API-basierten Webdiensten, mit denen Sie Testsuiteläufe in externen Ausführungsumgebungen planen und ausführen können, finden Sie in der interaktiven REST-API-Dokumentation, die unter *host:port[/inst]/Services1.0/swagger-ui.html* verfügbar ist, z. B. http://localhost:19120/Services1.0/swagger-ui.html.

Hinweis: Unter der WSDL-URL finden Sie auch systeminterne Webdienste, die sich nicht zum Erstellen von Webdienst-Mandanten eignen. In diesem Dokument werden nur die öffentlichen Webdienste beschrieben.

Einzelheiten über die verfügbaren Java-Klassen und -Methoden finden Sie in *Javadoc*. Wenn der Link nicht funktioniert, klicken Sie im Silk Central-Menü auf **Hilfe** > **Dokumentation** > **Silk Central API Spezifikation**, um Javadoc zu öffnen.

Hinweis: Bei Verwendung von Webdiensten werden Zeiten vom System stets in der koordinierten Weltzeit (UTC) zurückgegeben. Geben Sie alle Zeiten in den verwendeten Webdiensten ebenfalls in UTC-Notation an.

Name des Webdiensts (Schnittstelle)	WSDL-URL	Beschreibung
system (SystemService)	/Services1.0/jaxws/system?wsdl	Der Stammdienst, der für die Authentifizierung sorgt und einfache Dienstmethoden bereitstellt.
administration (AdministrationService)	/Services1.0/jaxws/administration?wsdl	Dieser Dienst ermöglicht den Zugriff auf die Entitäten *Projekt* und *Produkt* von Silk Central.
requirements (RequirementsService)	/Services1.0/jaxws/requirements?wsdl	Dieser Dienst ermöglicht den Zugriff auf den Bereich **Anforderungen** von Silk Central.
tests (TestsService)	/Services1.0/jaxws/tests?wsdl	Dieser Dienst ermöglicht den Zugriff auf den Bereich **Tests** von Silk Central.
executionplanning (ExecutionPlanningService)	/Services1.0/jaxws/executionplanning?wsdl	Dieser Dienst ermöglicht den Zugriff auf den Bereich **Ausführungsplanung** von Silk Central.
filter (FilterService)	/Services1.0/jaxws/filter?wsdl	Dieser Dienst ermöglicht das Erstellen, Lesen, Aktualisieren und Löschen von Filtern.
issuemanager (IssueManagerService)	/Services1.0/jaxws/issuemanager?wsdl	Dieser Dienst ermöglicht den Zugriff auf Issue Manager.

Services Exchange

Dieser Abschnitt erläutert die HTTP-basierten Schnittstellen, die für Berichte, Anhänge, Testpläne, Ausführungen und Bibliotheken in Services Exchange zuständig sind.

Sie können auf Silk Central auch über Webdienste zugreifen. Weitere Informationen finden Sie unter *Verfügbare Webdienste*.

Hinweis: Bei Verwendung von Webdiensten werden Zeiten vom System stets in der koordinierten Weltzeit (UTC) zurückgegeben. Geben Sie alle Zeiten in den verwendeten Webdiensten ebenfalls in UTC-Notation an.

reportData-Schnittstelle

Über die Schnittstelle reportData werden die Daten eines Berichts angefordert. Die folgende Tabelle enthält die Parameter der Schnittstelle reportData.

Schnittstellen-URL	Parameter	Beschreibung
http://<front-end URL>/servicesExchange?hid=reportData	sid	Webdienst-Token oder Sitzungs-ID für die Benutzerauthentifizierung. Sie können das Webdienst-Token auf der **Einstellungsseite** der Silk Central-Benutzeroberfläche generieren. Um auf diese Seite zuzugreifen, zeigen Sie mit dem Mauszeiger auf den Benutzernamen im Menü Silk Central und wählen Sie **Benutzereinstellungen**. Sie können die Sitzungs-ID abrufen, indem Sie die Methode logonUser für einen der *verfügbaren Webdienste* aufrufen.
	reportFilterID	ID des Berichtsfilters
	type	Format des Antwortrumpfs: (csv oder xml)
	includeHeaders	Berichtskopfzeilen einschließen. (true oder false)
	projectID	Die eindeutige Kennung des Projekts

Beispiel: http://<front-end URL>/servicesExchange?hid=reportData&reportFilterID=<id>&type=<csv or xml>&includeHeaders=<true or false>&sid=<webServiceToken>&projectID=<id>

Beispiel für die reportData-Schnittstelle

```
String reportID = "<id>";
String webServiceToken = "e39a0b5b-45db-42db-84b2-b85028d954d5";
String host = "<any_host>";

URL report = new URL("http", host, 19120,
    "/servicesExchange?hid=reportData" +
    "&type=xml" + // or csv
    "&sid=" + webServiceToken +
    "&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);
```

Falls für den Bericht Parameter benötigt werden, müssen Sie den folgenden Code für jeden Parameter zu der URL des Berichts hinzufügen:

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

In diesem Beispiel ist der Parameter rp_execNode_Id_0 auf den Wert 1 gesetzt.

Hinweis: Den Namen von Parametern, die an den Dienst reportData übergeben werden, muss die Zeichenfolge rp_ vorangestellt werden. Beispiel: /servicesExchange?hid=reportData&type=xml&sid=<...>&reportFilterID=<...>&projectID=<...>&rp_TestID=<...>

TMAttach-Schnittstelle

Über die Schnittstelle TMAttach werden Anhänge zu einer Test oder Anforderung hochgeladen. Die folgende Tabelle enthält die Parameter der Schnittstelle TMAttach.

Schnittstellen-URL	Parameter	Beschreibung
http://<front-end URL>/servicesExchange?hid=TMAttach	sid	Webdienst-Token oder Sitzungs-ID für die Benutzerauthentifizierung. Sie können das Webdienst-Token auf der **Einstellungsseite** der Silk Central-Benutzeroberfläche generieren. Um auf diese Seite zuzugreifen, zeigen Sie mit dem Mauszeiger auf den Benutzernamen im Menü Silk Central und wählen Sie **Benutzereinstellungen**. Sie können die Sitzungs-ID abrufen, indem Sie die Methode logonUser für einen der *verfügbaren Webdienste* aufrufen.
	entityType	Typ der Zielentität: (Test, Anforderung oder TestStepParent)
	entityID	ID der Zielentität: (Test-ID, Anforderungs-ID oder ID des manuellen Tests)
	description	Beschreibung des Anhangs. URL-codierter Text, der den Anhang beschreibt.
	isURL	Bei true ist der Anhang ein URL. Bei false ist der Anhang eine Datei.
	URL	Optional - Anzuhängender URL.
	stepPosition	Optional – Der Name des Testschritts. Bezeichnet den Schritt eines manuellen Tests (z. B. 1 für den ersten Schritt). Die Reihenfolge ist obligatorisch, wenn entityType den Wert TestStepParent hat.

Beispiel: http://<front-end URL>/servicesExchange?hid=TMAttach&entityType=<test, requirement, or TestStepParent>&entityID=<id>&description=<text>&isURL=<true or false>&URL=<URL>&stepPosition=<number>&sid=<webServiceToken>

Beispiel für den Webdienst TMAttach

Der folgende Quelltext ruft zum Hochladen eines binären Anhangs mit Apache HtmlClient eine geeignete HTTP-POST API ab. Pro Anfrage kann nur ein Anhang hochgeladen werden.

Pro Anfrage kann nur ein Anhang hochgeladen werden. Apache HttpComponents können Sie von *http://hc.apache.org/downloads.cgi* herunterladen. Die erforderlichen Bibliotheken entnehmen Sie der Dokumentation der Komponente.

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

   String webServiceToken = "e39a0b5b-45db-42db-84b2-b85028d954d5"; // Token
generated in the UI
   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=" + webServiceToken +
      "&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) ...
```

Schnittstelle createTestPlan

Über die Schnittstelle createTestPlan werden neue Tests erstellt. Die HTTP-Antwort des Aufrufs enthält die XML-Struktur der geänderten Tests. Sie können die Kennungen der neuen Knoten der aktualisierten XML-Teststruktur entnehmen.

Die folgende Tabelle enthält die Parameter der Schnittstelle createTestPlan.

Schnittstellen-URL	Parameter	Beschreibung
http://<front-end URL>/ servicesExchange?hid=createTestPlan	sid	Webdienst-Token oder Sitzungs-ID für die Benutzerauthentifizierung. Sie können das Webdienst-Token auf der **Einstellungsseite** der Silk Central-Benutzeroberfläche generieren. Um auf diese Seite zuzugreifen, zeigen Sie mit dem Mauszeiger auf den Benutzernamen im Menü Silk Central

Schnittstellen-URL	Parameter	Beschreibung
		und wählen Sie **Benutzereinstellungen**. Sie können die Sitzungs-ID abrufen, indem Sie die Methode logonUser für einen der *verfügbaren Webdienste* aufrufen.
	parentNodeID	ID des Containers, dem der neue Test in der **Testhierarchie** hinzugefügt wird.

Beispiel: http://<front-end URL>/servicesExchange?hid=createTestPlan&parentNodeID=<id>&sid=<webServiceToken>

Die Definitionsdatei für das XML-Schema, mit der überprüft wird, ob Testpläne über den URL des Front-End-Servers http://<Host>/silkroot/xsl/testplan.xsd heruntergeladen oder aus dem Installationsordner <Silk Central-Installationsordner>/wwwroot/silkroot/xsl/testplan.xsd des Front-End-Servers kopiert werden können.

Beispiel für den Webdienst createTestPlan

Im folgenden Quelltext werden die Tests mithilfe von Apache HttpClient erstellt.

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

string webServiceToken = "e39a0b5b-45db-42db-84b2-b85028d954d5"; // The token
that you have generated in the UI

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

Pro Anfrage kann nur ein Anhang hochgeladen werden. Apache HttpComponents können Sie von *http://hc.apache.org/downloads.cgi* herunterladen. Die erforderlichen Bibliotheken entnehmen Sie der Dokumentation der Komponente.

Testbeispiel

Der folgende Code zeigt ein Beispiel eines Tests, der mithilfe des Dienstes Silk Central undcreateTestPlan in updateTestPlan hochgeladen werden kann.

```
<?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>
```

Schnittstelle exportTestPlan

Mithilfe der Schnittstelle exportTestPlan werden Testpläne als XML-Dateien exportiert. Die folgende Tabelle enthält die Parameter der Schnittstelle exportTestPlan.

Schnittstellen-URL	Parameter	Beschreibung
http://<front-end URL>/servicesExchange?hid=exportTestPlan	sid	Webdienst-Token oder Sitzungs-ID für die Benutzerauthentifizierung. Sie können das Webdienst-Token auf der **Einstellungsseite** der Silk Central-Benutzeroberfläche generieren. Um auf diese Seite zuzugreifen, zeigen Sie mit dem Mauszeiger auf den Benutzernamen im Menü Silk Central und wählen Sie **Benutzereinstellungen**. Sie können die Sitzungs-ID abrufen, indem Sie die Methode logonUser für einen der *verfügbaren Webdienste* aufrufen.
	nodeID	Der Knoten mit dieser ID und rekursiv alle untergeordneten Knoten werden exportiert.

Beispiel: http://<front-end URL>/servicesExchange?hid=exportTestPlan&nodeID=<id>&sid=<webServiceToken>

Beispiel für den Webdienst exportTestPlan

Im folgenden Quelltext werden die Tests mithilfe von Apache HttpClient exportiert.

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

string webServiceToken = "e39a0b5b-45db-42db-84b2-b85028d954d5";

URL service = new URL("http", mWebServiceHelper.getHost(),
 mWebServiceHelper.getPort(),
```

```
 String.format("/servicesExchange?hid=%s&sid=%s&nodeID=%d",
 "exportTestPlan", webServiceToken,
 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 können Sie von *http://hc.apache.org/downloads.cgi* herunterladen. Die erforderlichen Bibliotheken entnehmen Sie der Dokumentation der Komponente.

Schnittstelle updateTestPlan

Mithilfe der Schnittstelle updateTestPlan werden Tests durch bestehende Stammknoten aus XML-Dateien aktualisiert. Die HTTP-Antwort des Aufrufs enthält die XML-Struktur der geänderten Tests. Sie können die Kennungen der neuen Knoten der aktualisierten XML-Teststruktur entnehmen.

Die folgende Tabelle enthält die Parameter der Schnittstelle updateTestPlan.

Schnittstellen-URL	Parameter	Beschreibung
http://<front-end URL>/servicesExchange?hid=updateTestPlan	sid	Webdienst-Token oder Sitzungs-ID für die Benutzerauthentifizierung. Sie können das Webdienst-Token auf der **Einstellungsseite** der Silk Central-Benutzeroberfläche generieren. Um auf diese Seite zuzugreifen, zeigen Sie mit dem Mauszeiger auf den Benutzernamen im Menü Silk Central und wählen Sie **Benutzereinstellungen**. Sie können die Sitzungs-ID abrufen, indem Sie die Methode logonUser für einen der *verfügbaren Webdienste* aufrufen.

Beispiel: http://<front-end URL>/servicesExchange?hid=updateTestPlan&sid=<webServiceToken>

Die Definitionsdatei für das XML-Schema, mit der überprüft wird, ob Testpläne über den URL des Front-End-Servers http://<Host>/silkroot/xsl/testplan.xsd heruntergeladen oder aus dem Installationsordner <Silk Central-Installationsordner>/wwwroot/silkroot/xsl/testplan.xsd des Front-End-Servers kopiert werden können.

Beispiel für den Webdienst updateTestPlan

Im folgenden Quelltext werden die Tests mithilfe von Apache HttpClient aktualisiert.

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

String webServiceToken = "e39a0b5b-45db-42db-84b2-b85028d954d5";
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",
    webServiceToken));
```

```
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();
```

Pro Anfrage kann nur ein Anhang hochgeladen werden. Apache HttpComponents können Sie von *http://hc.apache.org/downloads.cgi* herunterladen. Die erforderlichen Bibliotheken entnehmen Sie der Dokumentation der Komponente.

Testbeispiel

Der folgende Code zeigt ein Beispiel eines Tests, der mithilfe des Dienstes Silk Central undcreateTestPlan in updateTestPlan hochgeladen werden kann.

```
<?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>
```

Schnittstelle createRequirements

Über die Schnittstelle createRequirements werden neue Anforderungen erstellt. Die HTTP-Antwort des Aufrufs enthält die XML-Struktur der geänderten Anforderungen. Sie können die Kennungen der neuen Knoten der aktualisierten XML-Anforderungsstruktur entnehmen.

Die folgende Tabelle enthält die Parameter der Schnittstelle createRequirements.

Schnittstellen-URL	Parameter	Beschreibung
http://<front-end URL>/servicesExchange?hid=createRequirements	sid	Webdienst-Token oder Sitzungs-ID für die Benutzerauthentifizierung. Sie können das Webdienst-Token auf der **Einstellungsseite** der Silk Central-Benutzeroberfläche generieren. Um auf diese Seite zuzugreifen, zeigen Sie mit dem Mauszeiger auf den Benutzernamen im Menü Silk Central und wählen Sie **Benutzereinstellungen**. Sie können die Sitzungs-ID abrufen, indem Sie die Methode logonUser für einen der *verfügbaren Webdienste* aufrufen.
	parentNodeID	ID des Containers, dem die neue Anforderung in der Anforderungshierarchie hinzugefügt wird.

Beispiel: http://<front-end URL>/servicesExchange?hid=createRequirements&parentNodeID=<id>&sid=<webServiceToken>

Die Definitionsdatei für das XML-Schema, mit der überprüft wird, ob Anforderungen über den URL des Front-End-Servers http://<Host>/silkroot/xsl/requirements.xsd heruntergeladen oder aus dem Installationsordner <Silk Central-Installationsordner>/wwwroot/silkroot/xsl/requirements.xsd des Front-End-Servers kopiert werden können.

Beispiel für den Webdienst createRequirements

Im folgenden Quelltext werden die Anforderungen mithilfe von Apache HttpClient erstellt.

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

String webServiceToken = "e39a0b5b-45db-42db-84b2-b85028d954d5";

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

Pro Anfrage kann nur ein Anhang hochgeladen werden. Apache HttpComponents können Sie von *http://hc.apache.org/downloads.cgi* herunterladen. Die erforderlichen Bibliotheken entnehmen Sie der Dokumentation der Komponente.

Anforderungen – Beispiel

Der folgende Code zeigt ein Beispiel einer Anforderung, die mithilfe der Dienste createRequirements, updateRequirements und updateRequirementsByExtId in Silk Central hochgeladen werden kann.

```
<?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>
```

Schnittstelle exportRequirements

Mithilfe der Schnittstelle exportRequirements werden Anforderungen als XML-Dateien exportiert. Die folgende Tabelle enthält die Parameter der Schnittstelle exportRequirements.

Schnittstellen-URL	Parameter	Beschreibung
http://<front-end URL>/servicesExchange?hid=exportRequirements	sid	Webdienst-Token oder Sitzungs-ID für die Benutzerauthentifizierung. Sie können das Webdienst-Token auf der **Einstellungsseite** der Silk Central-Benutzeroberfläche generieren. Um auf diese Seite zuzugreifen, zeigen Sie mit dem Mauszeiger auf den Benutzernamen im Menü Silk Central und wählen Sie **Benutzereinstellungen**. Sie können die Sitzungs-ID abrufen, indem Sie die Methode logonUser für einen der *verfügbaren Webdienste* aufrufen.
	nodeID	Der Knoten mit dieser ID und rekursiv alle untergeordneten Knoten werden exportiert.

Schnittstellen-URL	Parameter	Beschreibung
	includeObsolete	*Optional:* Definieren Sie true oder false. Wird standardmäßig auf true gesetzt, wenn ausgelassen. Definieren Sie false um obsolete Anforderungen auszuschließen.

Beispiel: http://<front-end URL>/servicesExchange?hid=exportRequirements&nodeID=<id>&sid=<webServiceToken>

Beispiel für den Webdienst exportRequirements

Im folgenden Quelltext werden die Anforderungen mithilfe von Apache HttpClient exportiert.

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

String webServiceToken = "e39a0b5b-45db-42db-84b2-b85028d954d5";

URL service = new URL("http", mWebServiceHelper.getHost(),
  mWebServiceHelper.getPort(),
  String.format("/servicesExchange?hid=%s&sid=%s&nodeID=%d",
  "exportRequirements", webServiceToken,
  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 können Sie von *http://hc.apache.org/downloads.cgi* herunterladen. Die erforderlichen Bibliotheken entnehmen Sie der Dokumentation der Komponente.

Schnittstelle updateRequirements

Mithilfe der Schnittstelle updateRequirements werden Anforderungen durch bestehende Stammknoten aus XML-Dateien aktualisiert. Die Anforderungen werden in der Anforderungshierarchie durch die interne Knoten-ID von Silk Central definiert. Der Knoten der Anforderungshierarchie und alle untergeordneten Knoten werden aktualisiert. Neue Knoten werden hinzugefügt und fehlende Knoten als obsolet gekennzeichnet. Verschobene Knoten werden wie in Silk Central verschoben. Die HTTP-Antwort des Aufrufs enthält die XML-Struktur der geänderten Anforderungen. Sie können die Kennungen der neuen Knoten der aktualisierten XML-Anforderungsstruktur entnehmen.

Die folgende Tabelle enthält die Parameter der Schnittstelle updateRequirements.

Schnittstellen-URL	Parameter	Beschreibung
http://<front-end URL>/servicesExchange?hid=updateRequirements	sid	Webdienst-Token oder Sitzungs-ID für die Benutzerauthentifizierung. Sie können das Webdienst-Token auf der **Einstellungsseite** der Silk Central-Benutzeroberfläche generieren. Um auf diese Seite zuzugreifen, zeigen Sie mit dem Mauszeiger auf den Benutzernamen im Menü Silk Central

Schnittstellen-URL	Parameter	Beschreibung
		und wählen Sie **Benutzereinstellungen**. Sie können die Sitzungs-ID abrufen, indem Sie die Methode logonUser für einen der *verfügbaren Webdienste* aufrufen.

Beispiel: http://<front-end URL>/servicesExchange?hid=updateRequirements&sid=<webServiceToken>

Die Definitionsdatei für das XML-Schema, mit der überprüft wird, ob Anforderungen über den URL des Front-End-Servers http://<Host>/silkroot/xsl/requirements.xsd heruntergeladen oder aus dem Installationsordner <Silk Central-Installationsordner>/wwwroot/silkroot/xsl/requirements.xsd des Front-End-Servers kopiert werden können.

Beispiel für den Webdienst updateRequirements

Im folgenden Quelltext werden die Anforderungen mithilfe von Apache HttpClient aktualisiert.

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

String webServiceToken = "e39a0b5b-45db-42db-84b2-b85028d954d5";
URL service = new URL("http", mWebServiceHelper.getHost(),
  mWebServiceHelper.getPort(),
  String.format("/servicesExchange?hid=%s&sid=%s",
    "updateRequirements", webServiceToken));

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();
```

Pro Anfrage kann nur ein Anhang hochgeladen werden. Apache HttpComponents können Sie von *http://hc.apache.org/downloads.cgi* herunterladen. Die erforderlichen Bibliotheken entnehmen Sie der Dokumentation der Komponente.

Anforderungen – Beispiel

Der folgende Code zeigt ein Beispiel einer Anforderung, die mithilfe der Dienste createRequirements, updateRequirements und updateRequirementsByExtId in Silk Central hochgeladen werden kann.

```
<?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>
```

Schnittstelle "updateRequirementsByExtID"

Mithilfe der Schnittstelle updateRequirementsByExtId werden Anforderungen durch bestehende Stammknoten aus XML-Dateien aktualisiert. Die Anforderungen werden durch externe IDs identifiziert. Der Knoten der Anforderungshierarchie und alle untergeordneten Knoten werden aktualisiert. Neue Knoten werden hinzugefügt und fehlende Knoten als obsolet gekennzeichnet. Verschobene Knoten werden wie in Silk Central verschoben. Die HTTP-Antwort des Aufrufs enthält die XML-Struktur der geänderten Anforderungen. Sie können die Kennungen der neuen Knoten der aktualisierten XML-Anforderungsstruktur entnehmen.

Die folgende Tabelle enthält die Parameter der Schnittstelle updateRequirementsByExtId.

Schnittstellen-URL	Parameter	Beschreibung
http://<front-end URL>/servicesExchange?hid=updateRequirementsByExtId	sid	Webdienst-Token oder Sitzungs-ID für die Benutzerauthentifizierung. Sie können das Webdienst-Token auf der **Einstellungsseite** der Silk Central-Benutzeroberfläche generieren. Um auf diese Seite zuzugreifen, zeigen Sie mit dem Mauszeiger auf den Benutzernamen im Menü Silk Central und wählen Sie **Benutzereinstellungen**. Sie können die Sitzungs-ID abrufen, indem Sie die Methode logonUser für einen der *verfügbaren Webdienste* aufrufen.
	nodeID	Die ID des zu aktualisierenden Knotens der Anforderungshierarchie.

Beispiel: http://<front-end URL>/servicesExchange?hid=updateRequirementsByExtId&nodeID=<id>&sid=<webServiceToken>

Die Definitionsdatei für das XML-Schema, mit der überprüft wird, ob Anforderungen über den URL des Front-End-Servers http://<Host>/silkroot/xsl/requirements.xsd heruntergeladen oder aus dem Installationsordner <Silk Central-Installationsordner>/wwwroot/silkroot/xsl/requirements.xsd des Front-End-Servers kopiert werden können.

Beispiel für den Webdienst "updateRequirementsByExtId"

Im folgenden Quelltext werden die Anforderungen mithilfe von Apache HttpClient aktualisiert.

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

String webServiceToken = "e39a0b5b-45db-42db-84b2-b85028d954d5";
URL service = new URL("http", mWebServiceHelper.getHost(),
  mWebServiceHelper.getPort(),
  String.format("/servicesExchange?hid=%s&sid=%s",&nodeID=%s",
    "updateRequirementsByExtId",
    webServiceToken, 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();
```

Pro Anfrage kann nur ein Anhang hochgeladen werden. Apache HttpComponents können Sie von *http://hc.apache.org/downloads.cgi* herunterladen. Die erforderlichen Bibliotheken entnehmen Sie der Dokumentation der Komponente.

Anforderungen – Beispiel

Der folgende Code zeigt ein Beispiel einer Anforderung, die mithilfe der Dienste createRequirements, updateRequirements und updateRequirementsByExtId in Silk Central hochgeladen werden kann.

```
<?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>
```

Schnittstelle createExecutionDefinitions

Über die Schnittstelle createExecutionDefinitions werden neue Testsuiten erstellt. Die HTTP-Antwort des Aufrufs enthält die XML-Struktur der geänderten Testsuiten. Sie können die Kennungen der neuen Knoten der aktualisierten XML-Testsuitestruktur entnehmen.

Die folgende Tabelle enthält die Parameter der Schnittstelle createExecutionDefinitions.

Schnittstellen-URL	Parameter	Beschreibung
http://<front-end URL>/servicesExchange?hid=createExecutionDefinitions	sid	Webdienst-Token oder Sitzungs-ID für die Benutzerauthentifizierung. Sie können das Webdienst-Token auf der **Einstellungsseite** der Silk Central-Benutzeroberfläche generieren. Um auf diese Seite zuzugreifen, zeigen Sie mit dem Mauszeiger auf den Benutzernamen im Menü Silk Central und wählen Sie **Benutzereinstellungen**. Sie können die Sitzungs-ID abrufen, indem Sie die Methode logonUser für einen der *verfügbaren Webdienste* aufrufen.
	parentNodeID	ID des Knotens, dem die neue Testsuite in der Testsuitehierarchie hinzugefügt wird.

Beispiel: http://<front-end URL>/servicesExchange?hid=createExecutionDefinitions&parentNodeID=<id>&sid=<webServiceToken>

Die Definitionsdatei für das XML-Schema, mit der überprüft wird, ob Testsuiten über den URL des Front-End-Servers http://<Host>/silkroot/xsl/executionplan.xsd heruntergeladen oder aus dem Installationsordner <Silk Central-Installationsordner>/wwwroot/silkroot/xsl/executionplan.xsd des Front-End-Servers kopiert werden können.

Beispiel für den Webdienst createExecutionDefinitions

Im folgenden Quelltext werden die Testsuiten mithilfe von Apache HttpClient erstellt.

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

String webServiceToken = "e39a0b5b-45db-42db-84b2-b85028d954d5";

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

Pro Anfrage kann nur ein Anhang hochgeladen werden. Apache HttpComponents können Sie von *http://hc.apache.org/downloads.cgi* herunterladen. Die erforderlichen Bibliotheken entnehmen Sie der Dokumentation der Komponente.

Testsuite – Beispiel

Der folgende Code zeigt ein Beispiel eines Ausführungsplans, der mithilfe des Diensts Silk Central undcreateExecutionDefinitions in updateExecutionDefinitions hochgeladen werden kann. In diesem Fall wird ein benutzerdefinierter Ausführungstermin für eine der Testsuiten erstellt, und einer Testsuite werden Tests zugeordnet, sowohl manuell als auch mittels Filter. Im Beispiel wird zudem eine Konfigurationssuite mit Konfigurationen erstellt.

```
<?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>
```

Schnittstelle exportExecutionDefinitions

Mithilfe der Schnittstelle exportExecutionDefinitions werden Testsuiten als XML-Dateien exportiert. Die folgende Tabelle enthält die Parameter der Schnittstelle exportExecutionDefinitions.

Schnittstellen-URL	Parameter	Beschreibung
http://<front-end URL>/servicesExchange?hid=exportExecutionDefinitions	sid	Webdienst-Token oder Sitzungs-ID für die Benutzerauthentifizierung. Sie können das Webdienst-Token auf der **Einstellungsseite** der Silk Central-Benutzeroberfläche generieren. Um auf diese Seite zuzugreifen, zeigen Sie mit dem Mauszeiger auf den Benutzernamen im Menü Silk Central und wählen Sie **Benutzereinstellungen**. Sie können die Sitzungs-ID abrufen, indem Sie die Methode logonUser für einen der *verfügbaren Webdienste* aufrufen.
	nodeID	Der Knoten mit dieser ID und rekursiv alle untergeordneten Knoten werden exportiert.

Beispiel: http://<front-end URL>/servicesExchange?hid=exportExecutionDefinitions&nodeID=<id>&sid=<webServiceToken>

Beispiel für den Webdienst exportExecutionDefinitions

Im folgenden Quelltext werden die Testsuites mithilfe von Apache HttpClient exportiert.

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

String webServiceToken = "e39a0b5b-45db-42db-84b2-b85028d954d5";

URL service = new URL("http", mWebServiceHelper.getHost(),
  mWebServiceHelper.getPort(),
  String.format("/servicesExchange?hid=%s&sid=%s&nodeID=%d",
  "exportExecutionDefinitions", webServiceToken,
  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 können Sie von *http://hc.apache.org/downloads.cgi* herunterladen. Die erforderlichen Bibliotheken entnehmen Sie der Dokumentation der Komponente.

Schnittstelle updateExecutionDefinitions

Mithilfe der Schnittstelle updateExecutionDefinitions werden Testsuiten mittels XML-Dateien aktualisiert. Die HTTP-Antwort des Aufrufs enthält die XML-Struktur der geänderten Testsuiten. Sie können die Kennungen der neuen Knoten der aktualisierten XML-Testsuitestruktur entnehmen.

Die folgende Tabelle enthält die Parameter der Schnittstelle updateExecutionDefinitions.

Schnittstellen-URL	Parameter	Beschreibung
http://<front-end URL>/servicesExchange?hid=updateExecutionDefinitions	sid	Webdienst-Token oder Sitzungs-ID für die Benutzerauthentifizierung. Sie können das Webdienst-Token auf der **Einstellungsseite** der Silk Central-Benutzeroberfläche generieren. Um auf diese Seite zuzugreifen, zeigen Sie mit dem Mauszeiger auf den Benutzernamen im Menü Silk Central und wählen Sie **Benutzereinstellungen**. Sie können die Sitzungs-ID abrufen, indem Sie die Methode logonUser für einen der *verfügbaren Webdienste* aufrufen.

Beispiel: http://<front-end URL>/servicesExchange?hid=updateExecutionDefinitions&sid=<webServiceToken>

Die Definitionsdatei für das XML-Schema, mit der überprüft wird, ob Testsuiten über den URL des Front-End-Servers http://<Host>/silkroot/xsl/executionplan.xsd heruntergeladen oder aus dem Installationsordner <Silk Central-Installationsordner>/wwwroot/silkroot/xsl/executionplan.xsd des Front-End-Servers kopiert werden können.

Beispiel für den Webdienst updateExecutionDefinitions

Im folgenden Quelltext werden die Testsuites mithilfe von Apache HttpClient aktualisiert.

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

String webServiceToken = "e39a0b5b-45db-42db-84b2-b85028d954d5";
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",
    webServiceToken));
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();
```

Pro Anfrage kann nur ein Anhang hochgeladen werden. Apache HttpComponents können Sie von *http://hc.apache.org/downloads.cgi* herunterladen. Die erforderlichen Bibliotheken entnehmen Sie der Dokumentation der Komponente.

Testsuite – Beispiel

Der folgende Code zeigt ein Beispiel eines Ausführungsplans, der mithilfe des Diensts Silk Central undcreateExecutionDefinitions in updateExecutionDefinitions hochgeladen werden kann. In diesem Fall wird ein benutzerdefinierter Ausführungstermin für eine der Testsuiten erstellt, und einer Testsuite werden Tests zugeordnet, sowohl manuell als auch mittels Filter. Im Beispiel wird zudem eine Konfigurationssuite mit Konfigurationen erstellt.

```
<?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>
```

Schnittstelle createLibraries

Über die Schnittstelle createLibraries werden neue Bibliotheken erstellt. Die HTTP-Antwort des Aufrufs enthält die XML-Struktur der geänderten Bibliotheken. Sie können die Kennungen der neuen Knoten der aktualisierten XML-Bibliotheksstruktur entnehmen.

Die folgende Tabelle enthält die Parameter der Schnittstelle createLibraries.

Schnittstellen-URL	Parameter	Beschreibung
http://<front-end URL>/servicesExchange?hid=createLibraries	sid	Webdienst-Token oder Sitzungs-ID für die Benutzerauthentifizierung. Sie können das Webdienst-Token auf der **Einstellungsseite** der Silk Central-Benutzeroberfläche generieren. Um auf diese Seite zuzugreifen, zeigen Sie mit dem Mauszeiger auf den Benutzernamen im Menü Silk Central und wählen Sie **Benutzereinstellungen**. Sie können die Sitzungs-ID abrufen, indem Sie die Methode logonUser für einen der *verfügbaren Webdienste* aufrufen.

Beispiel: http://<front-end URL>/servicesExchange?hid=createLibraries&sid=<webServiceToken>

Die Definitionsdatei für das XML-Schema, mit der überprüft wird, ob Bibliotheken über den URL des Front-End-Servers http://<Host>/silkroot/xsl/libraries.xsd heruntergeladen oder aus dem Installationsordner <Silk Central-Installationsordner>/wwwroot/silkroot/xsl/libraries.xsd des Front-End-Servers kopiert werden können.

Beispiel für den Webdienst createLibraries

Im folgenden Quelltext werden die Bibliotheken mithilfe von Apache HttpClient erstellt.

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

String webServiceToken = "e39a0b5b-45db-42db-84b2-b85028d954d5";

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

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 können Sie von *http://hc.apache.org/downloads.cgi* herunterladen. Die erforderlichen Bibliotheken entnehmen Sie der Dokumentation der Komponente.

Bibliotheken – Beispiel

Der folgende Code zeigt ein Beispiel einer Bibliothek, die mithilfe des Dienstes Silk Central in createLibraries hochgeladen werden kann. Neue Bibliotheken können in beliebigen Projekten verwendet werden, sofern im Abschnitt GrantedProjects keine Projekte angegeben sind.

```
<?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>
```

Schnittstelle exportLibraryStructure

Über die Schnittstelle exportLibraryStructure können Bibliotheken, Ordner und Objekte mit gemeinsam verwendbaren Testschritten als XML-Dateien exportiert werden. Die folgende Tabelle enthält die Parameter der Schnittstelle exportLibraryStructure.

Schnittstellen-URL	Parameter	Beschreibung
http://<front-end URL>/servicesExchange?hid=exportLibraryStructure	sid	Webdienst-Token oder Sitzungs-ID für die Benutzerauthentifizierung. Sie

Schnittstellen-URL	Parameter	Beschreibung
		können das Webdienst-Token auf der **Einstellungsseite** der Silk Central-Benutzeroberfläche generieren. Um auf diese Seite zuzugreifen, zeigen Sie mit dem Mauszeiger auf den Benutzernamen im Menü Silk Central und wählen Sie **Benutzereinstellungen**. Sie können die Sitzungs-ID abrufen, indem Sie die Methode logonUser für einen der *verfügbaren Webdienste* aufrufen.
	nodeID	Der zu exportierende Bibliotheksknoten oder -ordner in der Bibliothekshierarchie. IDs von Knoten mit gemeinsam verwendbaren Testschritten sind unzulässig.

Beispiel: http://<front-end URL>/servicesExchange?hid=exportLibraryStructure&sid=<webServiceToken>&nodeID=<id>

Beispiel für den Webdienst exportLibraryStructure

Im folgenden Quelltext werden die Bibliotheken mithilfe von Apache HttpClient exportiert.

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

String webServiceToken = "e39a0b5b-45db-42db-84b2-b85028d954d5";

URL service = new URL("http", mWebServiceHelper.getHost(),
  mWebServiceHelper.getPort(), String.format("/servicesExchange?hid=%s&sid=
%s&nodeID=%d",
  "exportLibraryStructure", webServiceToken, 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 können Sie von *http://hc.apache.org/downloads.cgi* herunterladen. Die erforderlichen Bibliotheken entnehmen Sie der Dokumentation der Komponente.

Schnittstelle exportLibraryStructureWithoutSteps

Über die Schnittstelle exportLibraryStructureWithoutSteps können Bibliotheken, Ordner und Objekte mit gemeinsam verwendbaren Testschritten als XML-Dateien exportiert werden. Die Schritte der Objekte mit gemeinsam verwendbaren Testschritten werden nicht exportiert. Die folgende Tabelle enthält die Parameter der Schnittstelle exportLibraryStructureWithoutSteps.

Schnittstellen-URL	Parameter	Beschreibung
http://<front-end URL>/servicesExchange?hid=exportLibraryStructureWithoutSteps	sid	Webdienst-Token oder Sitzungs-ID für die Benutzerauthentifizierung. Sie können das Webdienst-Token auf der **Einstellungsseite** der Silk Central-Benutzeroberfläche generieren. Um auf diese Seite zuzugreifen, zeigen Sie mit dem Mauszeiger auf den Benutzernamen im Menü Silk Central und wählen Sie **Benutzereinstellungen**. Sie können die Sitzungs-ID abrufen, indem Sie die Methode logonUser für einen der *verfügbaren Webdienste* aufrufen.
	nodeID	Der zu exportierende Bibliotheksknoten oder -ordner in der Bibliothekshierarchie. IDs von Knoten mit gemeinsam verwendbaren Testschritten sind unzulässig.

Beispiel: http://<front-end URL>/servicesExchange?hid=exportLibraryStructureWithoutSteps&sid=<webServiceToken>&nodeID=<id>

Beispiel für den Webdienst exportLibraryStructureWithoutSteps

Im folgenden Quelltext werden die Bibliotheken mithilfe von Apache HttpClient exportiert.

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

String webServiceToken = "e39a0b5b-45db-42db-84b2-b85028d954d5";

URL service = new URL("http", mWebServiceHelper.getHost(),
  mWebServiceHelper.getPort(), String.format("/servicesExchange?hid=%s&sid=
%s&nodeID=%d",
  "exportLibraryStructureWithoutSteps", webServiceToken, 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 können Sie von *http://hc.apache.org/downloads.cgi* herunterladen. Die erforderlichen Bibliotheken entnehmen Sie der Dokumentation der Komponente.

Schnittstelle getLibraryInfoByName

Die Schnittstelle getLibraryInfoByName gibt die ID, den Namen und die Beschreibung aller benannten Bibliotheken zurück. Die Schnittstelle gibt nur die Eigenschaften von Bibliotheken, nicht aber ihre Hierarchie zurück. Die folgende Tabelle enthält die Parameter der Schnittstelle getLibraryInfoByName.

Schnittstellen-URL	Parameter	Beschreibung
http://<front-end URL>/servicesExchange?hid=getLibraryInfoByName	sid	Webdienst-Token oder Sitzungs-ID für die Benutzerauthentifizierung. Sie können das Webdienst-Token auf der **Einstellungsseite** der Silk Central-Benutzeroberfläche generieren. Um auf diese Seite zuzugreifen, zeigen Sie mit dem Mauszeiger auf den Benutzernamen im Menü Silk Central und wählen Sie **Benutzereinstellungen**. Sie können die Sitzungs-ID abrufen, indem Sie die Methode logonUser für einen der *verfügbaren Webdienste* aufrufen.
	libraryName	Der Name der Bibliothek

Beispiel: http://<front-end URL>/servicesExchange?hid=getLibraryInfoByName&sid=<webServicesToken>&libraryName=<name>

Beispiel für den Webdienst getLibraryInfoByName

Im folgenden Quelltext werden Bibliotheksinformationen mithilfe von Apache HttpClient abgerufen.

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

String webServiceToken = "e39a0b5b-45db-42db-84b2-b85028d954d5";

URL service = new URL("http", mWebServiceHelper.getHost(),
  mWebServiceHelper.getPort(), String.format("/servicesExchange?hid=%s&sid=%s",
  "getLibraryInfoByName", webServiceToken, 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 können Sie von *http://hc.apache.org/downloads.cgi* herunterladen. Die erforderlichen Bibliotheken entnehmen Sie der Dokumentation der Komponente.

Demo Client für Webdienste

Der Demo Client für Webdienste ist ein Tool zur Demonstration der Verwendung von Silk Central-Webdiensten. Sie können den Client über **Hilfe** > **Tools** herunterladen.

Der Demo Client für Webdienste zeigt alle unter **Tests** > **Testsattribute verwalten** verfügbaren Attribute für die einzelnen Tests und alle Eigenschaften für die einzelnen verfügbaren Testtypen an.

Achtung: Mit dem Demo Client für Webdienste kann die Verwendung von Webdiensten demonstriert werden. Verwenden Sie den Demo Client nicht in einer Produktionsumgebung.

Auslösen von Silk Central von einem CI-Server aus

Dieser Abschnitt beschreibt, wie Sie Silk Central besser in Ihre Continuous-Integration-Prozesse (CI-Prozesse) integrieren können, indem Sie von Ihrem CI-Server aus Ausführungen auf Silk Central mit einem Gradle-Skript auslösen.

Außerdem wird in diesem Abschnitt beschrieben, wie Sie Ergebnisse aus Silk Central abrufen und in Ihrem Build-Prozess verwenden können.

Um von einem CI-Server aus Ausführungen auf Silk Central auszulösen und die Ausführungsergebnisse aus Silk Central aufzubereiten, müssen Sie ein Gradle-Skript mit den entsprechenden Befehlen zu Ihrer Versionsverwaltung hinzufügen. Sie können die Datei silkcentral.gradle auf der Benutzeroberfläche von Silk Central herunterladen. Navigieren Sie zu **Hilfe** > **Tools** und klicken Sie auf **Gradle-Skript für die CI-Server-Integration**.

Sie können die folgenden Eigenschaften im Gradle-Skript konfigurieren:

Eigenschaft	Beschreibung
sc_executionNodeIds	Die kommagetrennte Liste der zu startenden Testsuiten. Diese Liste sollte keine Ordner enthalten. Beispiel: 22431,22432,22433.
sc_host	Der Silk Central-Host. Beispiel: http://[sc_server_name]:19120.
sc_token	Das Webdienst-Token für die Benutzerauthentifizierung. Sie können das Webdienst-Token auf der **Einstellungsseite** der Silk Central-Benutzeroberfläche generieren. Um auf diese Seite zuzugreifen, zeigen Sie mit dem Mauszeiger auf den Benutzernamen im Menü Silk Central und wählen Sie **Benutzereinstellungen**. Beispiel: 80827e02-cfda-4d2d-b0aa-2d5205eb6eq9.
sc_sourceControlBranch	*Optional:* Geben Sie diese Eigenschaft an, um einen bestimmten Branch auszuchecken. Wenn kein Branch angegeben ist, wird die Einstellung der Testsuite verwendet.
sc_ buildName	*Optional:* Der Build für den Lauf. Wenn kein Build angegeben ist, wird die Einstellung der Testsuite verwendet. Sie können diese Eigenschaft nur für Ausführungsplanknoten, nicht für Ordner, festlegen.
sc_StartOption	*Optional:* Die Tests, die ausgeführt werden sollen. Wird dies nicht angegeben, werden alle zugeordneten Tests ausgeführt. Die zulässigen Werte sind: • KOPIEREN • FEHLGESCHLAGEN • NOT_EXECUTED • NOT_EXECUTED_SINCE_BUILD • FAILED_NOTEXECUTED_SINCE_BUILD • HAVING_FIXED_ISSUES Der Standardwert lautet "ALL".
sc_sinceBuild	*Optional:* Der Name des Builds, seit dem keine Tests mehr durchgeführt wurden. Geben Sie diese Eigenschaft an, wenn die Eigenschaft sc_StartOption auf "FAILED_NOTEXECUTED_SINCE_BUILD" oder "NOT_EXECUTED_SINCE_BUILD" gesetzt ist.
sc_collectResults	*Optional:* Bei dem Wert "true" wartet das Skript, bis die Ausführung von Silk Central beendet ist, und schreibt die Ergebnisdateien im JUnit-Format. Bei dem Wert "false" löst das Skript die Ausführung aus und beendet den Vorgang, ohne auf die Ergebnisse zu warten. Die Dateien werden im Unterordner sc_results gespeichert. Der Standardwert lautet "true". Boolean.
sc_collectResultFiles	*Optional:* Wenn diese Eigenschaft "true" und sc_collectResults "true" ist, lädt das Skript die von den Tests erzeugten Ergebnisdateien herunter. Die Dateien werden im Unterordner sc_results gespeichert. Der Standardwert lautet "false". Boolean.

Eigenschaft	Beschreibung
sc_startDelay	*Optional:* Die Startverzögerung in Sekunden. Kann angegeben werden, wenn Sie mehr als eine Testsuite ausführen möchten. Die Testsuiten werden sequentiell mit der angegebenen Startverzögerung gestartet. Dies kann hilfreich sein, wenn Sie die Arbeitslast für die Testumgebung beim Start minimieren müssen, z. B. beim Starten einer virtuellen Maschine oder bei der Installation der zu testenden Anwendung. Der Standardwert lautet 0.

Sie können die Eigenschaften entweder direkt im Skript angeben oder beim Auslösen des Skripts übergeben.

Alle zusätzlich angegebenen Projekteigenschaften werden beim Auslösen des Skripts als Parameter an Silk Central übergeben und für die Ausführung verwendet. Dadurch können Sie die Ausführungen in Silk Central mit Werten vom Build-Server parametrisieren.

Wenn mit dem Build beispielsweise ein Testserver in Docker gestartet wird, können Sie den URL an diesen Server übergeben, indem Sie die Eigenschaft in der Befehlszeile angeben:

```
-PmyServerUrl=http://docker:1234
```

Befehlszeilenbeispiel

Mit dem folgenden Befehl wird das Skript aus der Befehlszeile gestartet, indem die Knoten 22431, 22432 und 22433 der Ausführungshierarchie auf "localhost" gestartet werden. Dabei wird das Webdienst-Token "80827e02-cfda-4d2d-b0aa-2d5205eb6ea9" zur Authentifizierung verwendet:

```
gradle -b silkcentral.gradle
:silkCentralLaunch -Psc_executionNodeIds='22431,22432,22433'
-Psc_host='http://localhost:19120'
-Psc_token='80827e02-cfda-4d2d-b0aa-2d5205eb6ea9'
```

Spezifische Informationen zum Auslösen von Ausführungen auf Silk Central aus Jenkins finden Sie unter *Auslösen von Ausführungen aus Jenkins*. Spezifische Informationen zum Auslösen von Ausführungen auf Silk Central aus TeamCity finden Sie unter *Auslösen von Ausführungen aus TeamCity*.

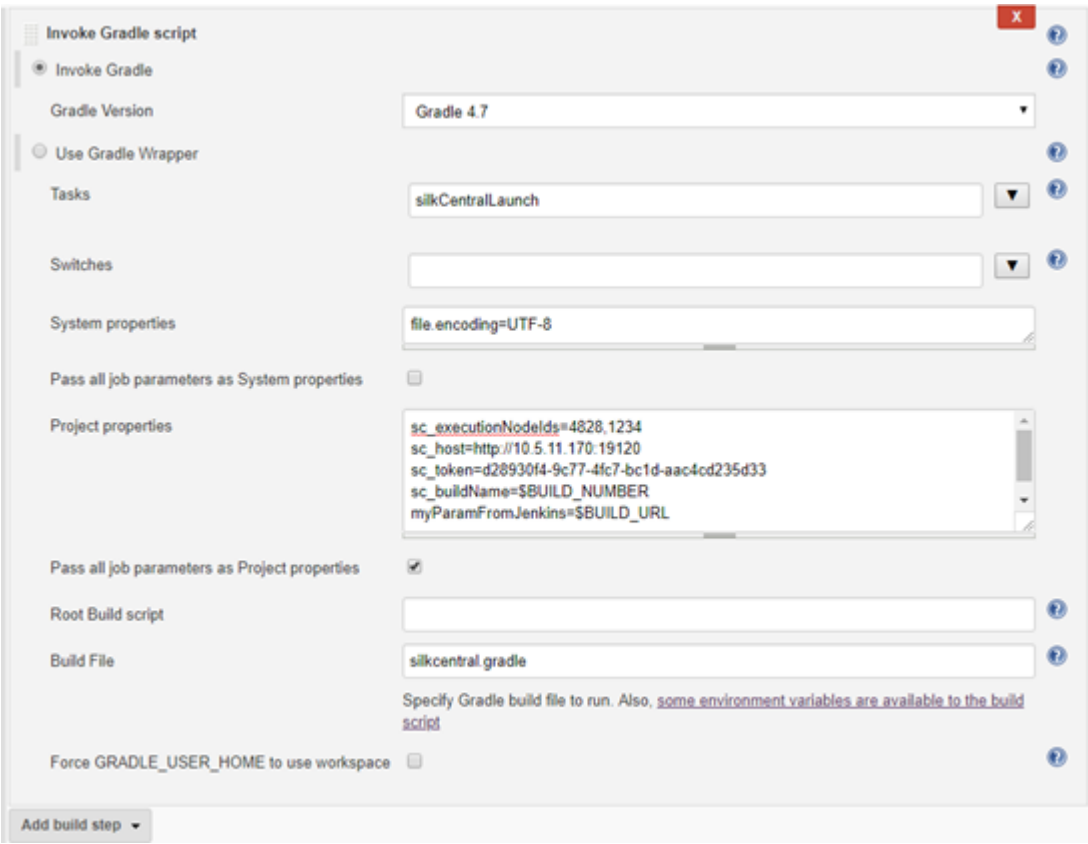
Auslösen von Ausführungen aus Jenkins

Wenn Sie im Build-Prozess nicht bereits Gradle verwenden, stellen Sie sicher, dass Jenkins Gradle-Skripte ausführen kann.

Gehen Sie wie folgt vor, um aus Jenkins Ausführungen in Silk Central auszulösen:

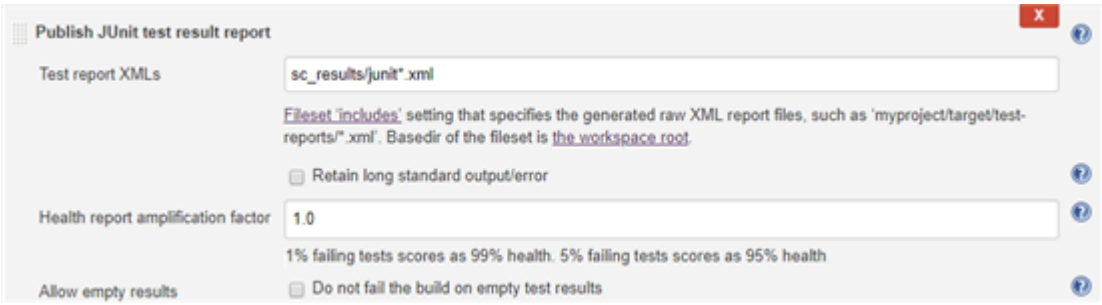
1. Installieren Sie Gradle in Jenkins unter **Manage Jenkins** > **Global Tool Configuration**.
2. Fügen Sie im Jenkins-Projekt den Build-Schritt **Invoke Gradle script** hinzu.

   Je nach Speicherort des Gradle-Skripts müssen Sie die Eigenschaft **Build File** anpassen. Konfigurieren Sie den Schritt wie im folgenden Screenshot:



a) Wie im Screenshot gezeigt, können Sie in Jenkins verfügbare Variablen wie z. B. *$BUILD_NUMBER* verwenden, um das Skript zu konfigurieren.
b) Wenn Ihr Jenkins-Projekt parametrisiert ist, können Sie alle Parameter direkt an Silk Central übergeben, indem Sie das Kontrollkästchen **Pass all job parameters as Project properties** aktivieren.

3. Um in Jenkins Testergebnisse anzuzeigen, fügen Sie die Post-Build-Aktion **Publish JUnit test result report** zum Jenkins-Projekt hinzu.
4. Geben Sie im Feld **Test report XMLs** den Speicherort an, in den das Skript die Dateien schreiben soll. Beispiel: sc_results/junit*.xml.



5. *Alternativ:* Sie können zum Konfigurieren von Jenkins und zum Auslösen von Ausführungen in Silk Central auch ein Pipeline-Skript verwenden.
Das folgende Beispiel-Pipeline-Skript löst zwei Ausführungen in Silk Central aus und sammelt die Ergebnisse. Die Gradle-Installation hat den Namen Gradle5.4.

```
node () {
  stage("Trigger Silk Central Executions") {
    def path = tool name: 'Gradle5.4', type: 'gradle'
    def scFile = new File(pwd(), "silkcentral.gradle")
```

```
    scFile.delete()
    scFile.getParentFile().mkdirs()
    writeFile([file: scFile.getAbsolutePath(), text: new URL ("http://scHost:19120/silkroot/tools/
silkcentral.gradle").getText()])
    def scTriggerInfo = '-Psc_executionNodeIds=6164,6123 -Psc_host=http://scHost:19120 -
Psc_token=d28930f4-9c77-4fc7-bc1d-aac4cd235d33'
    if (isUnix()) {
      sh "${path}/bin/gradle :silkCentralLaunch -b ${scFile} " + scTriggerInfo
    } else {
      bat "${path}/bin/gradle.bat :silkCentralLaunch -b ${scFile} " + scTriggerInfo
    }
    junit 'sc_results/junit*.xml'
  }
}
```

Auslösen von Ausführungen aus TeamCity

Gehen Sie wie folgt vor, um aus TeamCity Ausführungen in Silk Central auszulösen:

1. Fügen Sie dem Build in TeamCity einen Build-Schritt hinzu:
   a) Wählen Sie **Gradle** als **Runner Type** aus.
   b) Geben Sie silkCentralLaunch im Feld **Gradle task** an.
   c) Navigieren Sie im Feld **Gradle build file** zur Datei silkcentral.gradle und wählen Sie diese aus.
   d) Geben Sie zusätzliche Gradle-Befehlszeilenparameter im Feld **Additional Gradle command line parameters** an.

2. Um die Testergebnisse aus Silk Central in TeamCity zu verarbeiten, fügen Sie dem Build in TeamCity die Build-Funktion **XML report processing** hinzu.
3. Konfigurieren Sie die Build-Funktion **XML report processing**.
   a) Wählen Sie den Berichtstyp unter **Report type** aus.
   b) Geben Sie im Feld **Monitoring rules** den Speicherort an, in den das Skript die Dateien schreiben soll.

   Beispiel: sc_results/*.xml.

Index

A

Allgemeine Metadaten für Eigenschaften
    Drittanbieter-Testtyp-Plug-In 20
Anforderungs-Plug-Ins 14
Apache Axis 23
API
    Übersicht 4
API-Struktur
    Drittanbieter-Testtyp-Plug-In 15
APIs
    Integration Codeabdeckung 5
Arten
    Plug-Ins 4
Auslösen von Ausführungen
    CI-Server 56
    Jenkins 57
    TeamCity 59
Authentifizierung
    Webdienste 27

B

Beispielcode
    Drittanbieter-Testtyp-Plug-Ins 16
Benutzerdefinierte Symbole
    Drittanbieter-Testtyp-Plug-In 21
Bereitstellung
    Drittanbieter-Testtyp-Plug-In 21
    Plug-Ins 4

C

CI-Server
    Auslösen von Ausführungen, Gradle 56
Cloud Integration 22
Cloud Plug-In 22
Codeabdeckung
    APIs 5
createExecutionDefinitions
    Beispiele 44
    Schnittstellen 44
createLibraries
    Beispiele 51
    Schnittstellen 51
createRequirements
    Beispiele 37
    Schnittstellen 37
createTestPlan
    Beispiele 31
    Schnittstellen 31

D

Demo Client
    Webdienst-Schnittstelle 55
Demo Client für Webdienste 55
Drittanbieter-Testtyp-Plug-In
    Allgemeine Metadaten für Eigenschaften 20
    API-Struktur 15
    Benutzerdefinierte Symbole 21
    Implementierung 15
    Integration 14
    Metadaten für Dateieigenschaften 20
    Metadaten für String-Eigenschaften 20
    XML-Konfigurationsdatei 19, 21
Drittanbieter-Testtyp-Plug-Ins
    Beispielcode 16
    Komprimierung 15
    Metadaten 19
    Übergabe von vordefinierten Parametern 15

E

exportExecutionDefinitions
    Beispiele 47
    Schnittstellen 47
exportLibraryStructure
    Beispiele 52
    Schnittstellen 52
exportLibraryStructureWithoutSteps
    Beispiele 53
    Schnittstellen 53
exportRequirements
    Beispiele 39
    Schnittstellen 39
exportTestPlan
    Beispiele 34
    Schnittstellen 34
Externe Ergebnisse
    Hochladen 4

F

Fehlerverfolgung
    Plug-Ins 12

G

getLibraryInfoByName
    Beispiele 54
    Schnittstellen 54
Gradle
    Aufbereiten von Ergebnissen 56
    Ausführungen, Auslösen auf CI-Servern 56

H

Hochladen
    Externe Ergebnisse 4

I

Implementierung

Drittanbieter-Testtyp-Plug-In 15
Integration
Drittanbieter-Testtyp-Plug-In 14
Integration der Anforderungsverwaltung 13
Integration der Fehlerverfolgung
Übersicht 12

J

Java-Schnittstelle 13
Jenkins
Ausführungen, Auslösen 57
Auslösen von Ausführungen, Gradle 56

K

Klassen 13
Kompilierung von Plug-Ins 4
Komprimierung
Drittanbieter-Testtyp-Plug-Ins 15

M

Metadaten
Drittanbieter-Testtyp-Plug-Ins 19
Metadaten für Dateieigenschaften
Drittanbieter-Testtyp-Plug-In 20
Metadaten für String-Eigenschaften
Drittanbieter-Testtyp-Plug-In 20

P

Plug-Ins
Anforderungen 14
Anforderungsverwaltung 13
Arten 4
Bereitstellung 4
Cloud 22
Fehlerverfolgung 12
Kompilierung 4
Übersicht 4
Versionsverwaltung 11
Verteilung 4
Process Executor
Beispielcode 16

R

reportData
Beispiel 29
Schnittstelle 29
REST-API
Dokumentation 4

S

Schnittstellen
createExecutionDefinitions 44
createLibraries 51
createRequirements 37
createTestPlan 31
exportExecutionDefinitions 47
exportLibraryStructure 52
exportLibraryStructureWithoutSteps 53
exportRequirements 39
exportTestPlan 34
getLibraryInfoByName 54
Java-Schnittstelle 13
reportData 29
TMAttach 30
updateExecutionDefinitions 48
updateRequirements 40
updateRequirementsByExtId 42
updateTestPlan 35
Versionsverwaltung 11
Services Exchange 28
Sitzungen
Authentifizieren 27
SOAP
Pakete 24
Stack 23
Symbole
Benutzerdefiniert 21
Synchronisierung
Anforderungen 13

T

TeamCity
Ausführungen, Auslösen 59
TMAttach
Beispiel 30
Schnittstelle 30

U

updateExecutionDefinitions
Beispiele 48
Schnittstellen 48
updateRequirements
Beispiele 40
Schnittstellen 40
updateRequirementsByExtId
Beispiele 42
Schnittstellen 42
updateTestPlan
Beispiele 35
Schnittstellen 35

V

Versionsverwaltung
Integration 11
Plug-Ins 11
Schnittstellen 11
Schnittstellen-Konventionen 12
Verteilung
Plug-Ins 4
Videoaufnahme
Angeben des Starts 22
Angeben des Stopps 22

Vordefinierte Parameter
Übergabe an Drittanbieter-Testtyp-Plug-Ins 15

W

Webdienst
Anforderungsverwaltung 13
Cloud 22
Voraussetzungen 23
Webdienst-Schnittstelle
Kurzanleitung 23
Tutorial 24
Webdienste
Anmeldung, Benutzerdaten 27
Anwendungsfallbeispiel 25
Dokumentation 4
REST-API 23
Übersicht 23
Verfügbar 28