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 帮助

本指南提供□建和部署插件以将第三方工具集成到 Silk Central 所需的信息，以及有关管理外部□行□划运行□果的信息。□于基于 SOAP 的可用 Web 服□，本指南包含相关□范和 API □明，并且介□了如何将第三方插件集成到 Silk Central 中。

注: 本指南假□您熟悉 Web 服□的□施和使用。

概述

Silk Central 提供了用于集成第三方□用程序的基于 SOAP 的 Web 服□，以及用于管理外部□行□划运行□果的 REST API。

借助基于 SOAP 的 Silk Central Web 服□，您可以通□配置 Silk Central 插件来集成□有的源代□管理、□□跟踪和需求管理工具。Silk Central 附□各种示例插件。

借助于用于管理外部□行□划运行□果的 REST API，您可以将未由 Silk Central □行服□器□行的□行□划运行生成的外部□果上□到 Silk Central 以□行□一步□□管理。您□可以指定□些外部□行□划，它□将在外部□行□境而不是 Silk Central □行服□器上□行。

基于 SOAP 的 Web 服□的文档

有关可用 Java □和方法的完整□□信息，□参□ *Javadoc*。如果□接无效，□□□ Silk Central 菜□中的**帮助** > **文档** > **Silk Central API □范**以打开 Javadoc。。

REST API 的文档

如果系□上安装了 Silk Central，□可以从*此□*□□ REST API 的交互式文档。

Silk Central 集成插件

Silk Central 插件“按原□”提供，它包含所有缺陷并且不提供任何保□。□于任何事宜，包括但不限于任何保□、□□、条件或适□性、特定目的之适用性、不含病毒、准确性或完整性、□□、平静行使□、平静占有□和非侵□，Micro Focus 在此声明无任何明示、暗示或法定的保□和条件。

您在使用插件□需自行承担□□。在任何情况下，□于由使用插件□致或引起的任何□型的直接、□接、特殊、意外或□果性□失（包括但不限于利益□失），Micro Focus 概不承担任何□任。

□建插件

概述

本□介□如何□ Silk Central □建插件。此□□介□所有插件□型的通用任□。

插件种□

Silk Central 提供几种插件 API。每种 API 被□□一个*种□*。

□□

有关开□和□□插件的信息，□参□相□ Java 版本的 Silk Central □行□明。□□于与 Silk Central Java Runtime Environment 的兼容性非常重要。Silk Central 使用的是 AdoptOpenJDK。

部署

□建插件□和□施种□ API 之后，您可□建插件包（JAR 或 ZIP 文件）。

- 如果插件没有□一步的依□关系（或依□于已成□ Silk Central 一部分的□），□只需□建包含□的 JAR 文件。
- 如果插件依□于其他□，□将□些□放入子目□ lib 中，然后将所有□一起打包成 ZIP 存档。

将□建的文件放入位于 <□用程序服□器安装目□>\plugins\ 的插件目□中。

注: 您必□重新启□□用程序服□器和前端服□器，以使新部署的插件在 Silk Central 中可用。有关重新启□服□器的更多信息，□参□本帮助中的*管理*主□。

分□

由于 Silk Central 知道插件种□的□型，因此也知道哪些服□器（□行服□器、□用程序服□器和前端服□器）需要哪些种□。每种插件均可安装在□用程序服□器上。Silk Central 会将正确插件自□分配到每个服□器。

代□覆盖率集成

本章□介□的 Java API 接□是□支持第三方（外部）代□覆盖率工具集成的 Silk Central □建插件所必需的。

您可通□代□覆盖率工具提供关于□□覆盖哪些代□的信息。 Silk Central 提供了以下□成的代□覆盖率工具：

- Silk Central Java 代□分析 (Java Code Analysis Agent)
- DevPartner Studio .NET 代□分析 (Windows Code Analysis Framework)

注: 如果被□□用程序运行于 Linux 上，□参□主□*在 Linux AUT □境中安装代□分析框架*。

如果前面两种工具□不□，您可□建和部署自己的代□覆盖率集成。 □参□ Creating Your Own Code Coverage Plugin。

注: 除了在□用程序服□器上部署自定□□用程序之外（□参□*□建插件*主□），您□需要在代□分析框架服□器上部署自定□□用程序。 □是您的 AUT 和代□覆盖率工具所在位置。 路径如下：\Silk\Silk Central <version>\Plugins

□建您自己的代□覆盖率插件

本主□介□如何□建代□覆盖率插件。您□熟悉 Silk Central 基□概念。在 Silk Central 中，每次运行之前都需要基□。基□包括□□□用程序中的所有命名空□/程序包/□/方法。

注: Silk Central API 需要返回用于代□覆盖率运行的 XML 文件。□意味着，如果代□覆盖率工具在数据□中存□其代□覆盖率信息，您将需要采取其他步□□索数据。

注: 不支持多个□行服□器根据同一代□分析框架运行□□。

1. 将□ scc.jar 添加到您的□路径，因□它包含必□□展的接□。可以在 Silk Central 安装目□的 lib 目□中找到 JAR 文件。
2. 添加以下两个□入□句：

   ```
   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. □建□□ CodeAnalysisProfile 的□。
4. 从代□覆盖率界面中添加所有需要的方法，如以下步□所示。您可参□示例接□□了解解其定□并手□□□方法，也可复制和粘□□您提供了□入和方法定□的主□*示例配置文件□*。

注: 您将写入的下述步□中的方法□□上会在 Silk Central 需要□□用。□意味着您将无法直接□用□些方法。

5. 代□ getBaseline。此方法□返回包含□用程序中所有命名空□/程序包/□/方法的 XML 文件。□参□*示例 XML 数据*主□文件，以了解文件格式。您□使用示例 XSD 文件□□ XML。□参□*代□覆盖率 XSD* 主□，以了解 XSD。

   此函数将在开始覆盖之前□用并由开始□□运行的 Silk Central □行服□器触□，以开始代□分析并返回所有要覆盖的□象。□出□果需要使用 CA-Framework 安装文件□中包含的 XML 架构中指定的格式□□□ XML。
6. 代□ startCoverage。此□用将命令代□覆盖率工具开始收集数据。如果已开始，□返回 true。

   Silk Central 代□覆盖率框架将在完成 getBaseLine() 方法之后□用此函数。您□在此□代□覆盖率工具开始收集代□覆盖率数据。
7. 代□ stopCoverage。此□用将命令代□覆盖率工具停止收集数据。如果成功，□返回 true。

   此函数将在 startCoverage 之后□用，由完成□□运行的 Silk Central □行服□器触□，以停止代□分析。
8. 代□ getCoverage。此函数将返回 XML 文件，其中包括从 startCoverage 和 stopCoverage 方法之□收集到的数据。□参□*示例 XML 数据*主□，以了解文件格式。您□使用示例 XSD 文件□□ XML。□参□*代□覆盖率 XSD* 主□，以了解 XSD。

   此函数将在 stopCoverage() 之后□用并返回已收集的所有覆盖率数据。□出□果需要使用 XML 架构中指定的格式□□□ XML。
9. 代□ GetName。此函数将提供用于引用代□覆盖率工具的名称。例如，此□将用作**□□代□分析□置**□□框上的**代□分析配置文件**列表中的一个□。

   此函数首先由 Silk Central 代□覆盖率框架□用。插件名称□示在 Silk Central 中的代□覆盖率列表中。
10. 将插件生成 jar 文件，然后将 jar 文件□□成 zip 文件。
11. 将插件部署到以下位置：
    - Silk Central 安装文件□的 Plugins 目□中。
    - CA-Framework 安装的 Plugins 目□中。

示例配置文件□

此示例文件概述代□覆盖率插件的所有必要方法、□入和□施。

```
//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;
  }

}
```

代□覆盖率 XSD

下面是用于□□代□覆盖率工具生成的 XML 的代□覆盖率 XSD。 本文档位于以下位置：<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>
```

示例 XML 数据

代□覆盖率 API 需要以下格式的 XML。

您也可以使用提供的示例 XSD 文档□□ XML。

```
<?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>
```

在 Linux AUT □境中安装代□分析框架

如果您□建的代□覆盖率插件将与在 Linux 操作系□下□□的 .NET □用程序□行交互，□□□行以下步□。

1. 用于 Linux 的代□分析框架可从**帮助** > **工具** > **Linux 代□分析框架**□取。下□并将其复制到 Linux 机器上的根文件□或任何其他文件□。
2. 确保已在□□□用程序的机器上安装 the latest version of Java Runtime Environment 1.8。
3. 解□ CA-Framework.tar.gz。
4. 将代□分析插件置于 <install dir>/21.1Silk Central/Plugins 文件□中。
5. 将目□更改□ <install dir>/21.1Silk Central/Code Analysis。
6. □找将运行 CA-Framework □程的 startCodeAnalysisFramework.sh 外壳脚本。
7. 运行以下命令以将文件□□□ Unix 格式：dos2unix startCodeAnalysisFramework.sh。
8. 运行以下命令以□置□行外壳脚本所需的□限：chmod 775 startCodeAnalysisFramework.sh。
9. 运行以下外壳脚本以运行 CAFramework □程：./startCodeAnalysisFramework.sh。

代□分析框架已准□就□，可从 Silk Central 中使用它。

源代□管理集成

源代□管理配置文件可使 Silk Central 与外部源代□管理系□集成。

部署后，自定□源代□管理插件可在 Silk Central 中配置，允□您定□ Silk Central □行服□器□□索程序源代□以□行□□的位置。

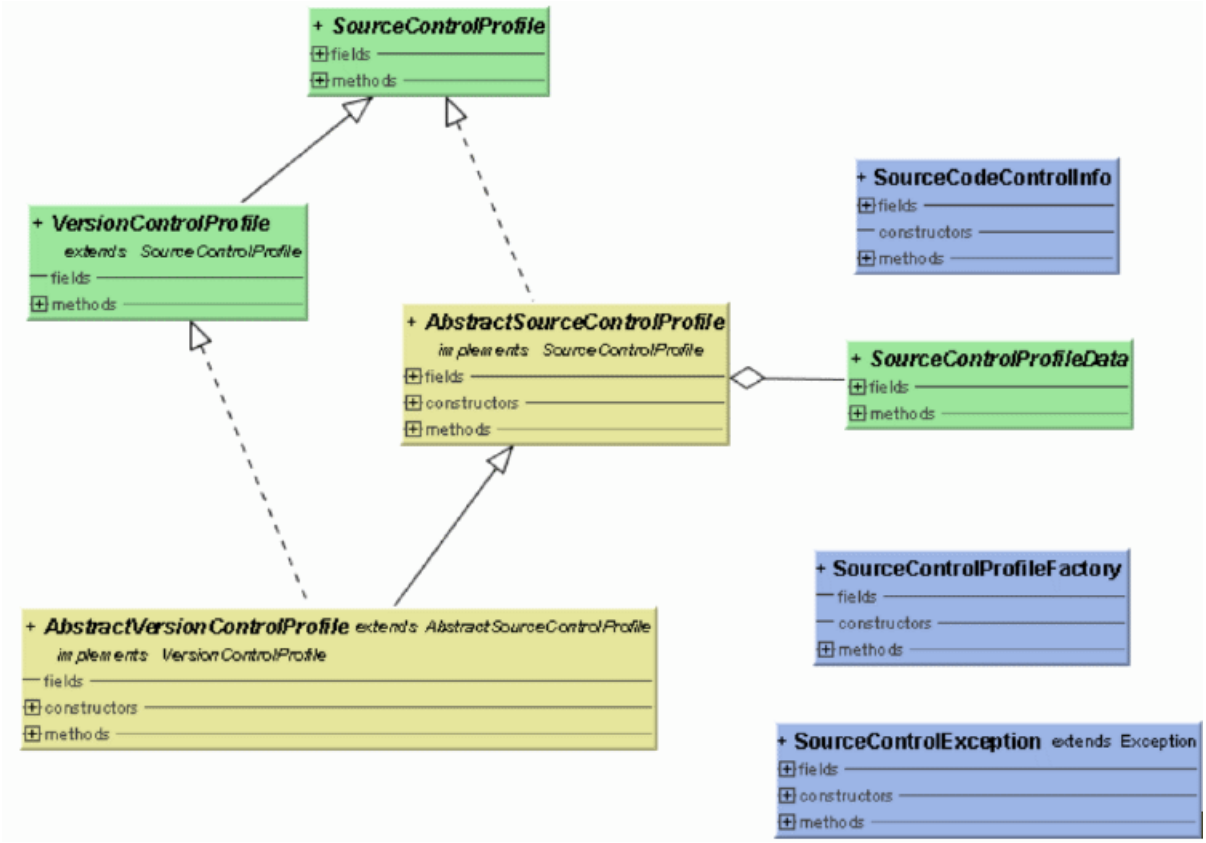
□看 C:\Program Files (x86)\Silk\Silk Central 21.1\instance_<□例□号>_<□例名称>\Plugins \subversion.zip 中的数据包 com.segue.scc.vcs.subversion 的来源，了解□些元素如何相□合。

源代□管理集成接□

Silk Central 区分 SourceControlProfile 和 VersionControlProfile。 区□在于 SourceControlProfile 没有版本管理，而 VersionControlProfile 有版本管理。

以下是用于源代□管理集成的 Silk Central 接□：

- SourceControlProfile
- VersionControlProfile
- SourceControlProfileData
- SourceControlException
- SourceControlInfo



有关可用 Java □和方法的完整□□信息，□参□ *Javadoc*。如果□接无效，□□□ Silk Central 菜□中的**帮助** > **文档** > **Silk Central API □范**以打开 Javadoc。

源代□管理集成□定

每次□施必□提供默□构造函数，或者将 SourceControlProfileData 作□参数的构造函数。 如果未提供此构造函数，□必□提供 SourceControlProfileData 的 bean setter。

由于每个接□方法指定了要抛出的 SourceControlException，因此不允□以接□使用的任何方法抛出 RuntimeException。

□□跟踪集成

□ Silk Central □建可启用第三方（外部）□□跟踪系□ (ITS) 集成的插件需要使用本章中□□的界面。

通□定□□□跟踪配置文件，您可以将□□区域内的□□□接到第三方□□跟踪系□中的□□。□接的□□状□将定期从第三方□□跟踪系□□行更新。

□看 C:\Program Files (x86)\Silk\Silk Central21.1\instance_<□例□号>_<□例名称>\Plugins\IT-Bugzilla4.zip 中的数据包 com.segue.scc.issuetracking.bugzilla4 的来源，了解□些元素如何相□合。

Java 接□

有关可用 Java □和方法的完整□□信息，□参□ *Javadoc*。如果□接无效，□□□ Silk Central 菜□中的**帮助** > **文档** > **Silk Central API □范**以打开 Javadoc。

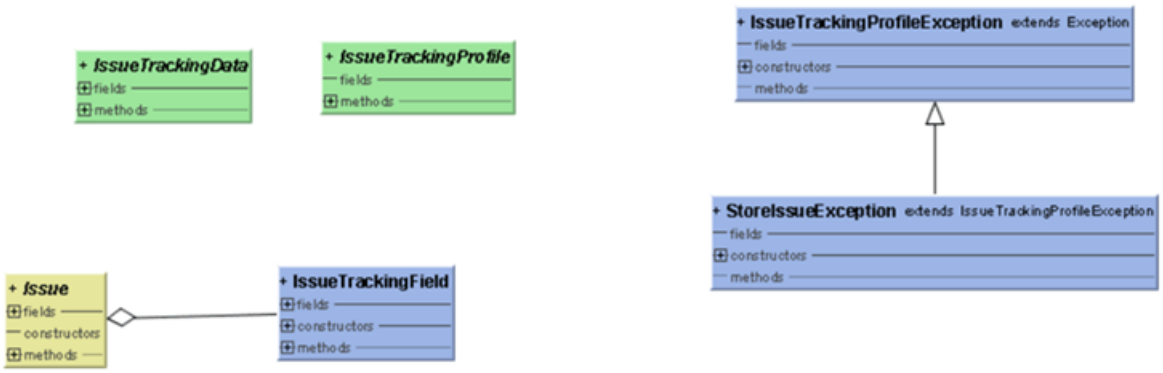
构建□境

将□ scc.jar 添加到 classpath，因□它包含必□□展的接□。可以在 Silk Central 安装目□的 lib 目□中找到 JAR 文件。

您必□□展两个接□/□：

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

□/接□



- IssueTrackingProfile
- IssueTrackingData
- Issue
- IssueTrackingField
- IssueTrackingProfileException

需求管理集成

Silk Central 可与第三方需求管理系□ (RMS) 工具集成来□接和同步需求。

本部分介□可使您□施第三方插件的 Java □用程序□程接□，以便将 Silk Central 中的需求与第三方需求管理系□的需求同步。 本部分介□□□需求插件及其部署的接□。

在□准 Silk Central 的插件概念之后将提供 JAR 或 ZIP 文件，此文件会自□部署到所有前端服□器，以便□□第三方工具来□行配置和同步。 此插件□施指定接□，允□ Silk Central 将其□□□需求插件，并提供所需的登□属性来登□第三方工具。

有关可用 Java □和方法的完整□□信息，□参□ *Javadoc*。如果□接无效，□□□ Silk Central 菜□中的**帮助** > **文档** > **Silk Central API □范**以打开 Javadoc。

有关其他插件的信息，□□系客□支持。

Java 接□

有关可用 Java □和方法的完整□□信息，□参□ *Javadoc*。如果□接无效，□□□ Silk Central 菜□中的**帮助** > **文档** > **Silk Central API □范**以打开 Javadoc。

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

需求 Java 插件 API 包含以下附加接□：

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

第三方□□□型集成

Silk Central 可使您□可用□□□型（包括 Silk Performer、Silk Test Classic、手□□□、NUnit、JUnit 和 Windows 脚本宿主 (WSH)）□准集之外的□□□型□建自定□插件。 □建新□□□型插件后，您的自定□□□□型将在**新建□□□□**框的**□型**列表框中可用，以及可用于在 Silk Central 中□建新□□的□准□□□型。

插件指定配置□□和□施□□□行所需的属性。 属性的元信息通□*配置 XML 文件*定□。

插件方法的目的是根据常用□□框架（例如 JUnit、NUnit 或脚本□言 (WSH)）支持□□，以便根据特定□□□境□松自定□ Silk Central。 定□完善的 Silk Central 公共 API 使您可□施符合自□□□需求的□用解决方案。 Silk Central 向可从 Java □施或通□命令行□用的任何第三方工具开放和□展。

有关可用 Java □和方法的完整□□信息，□参□ *Javadoc*。如果□接无效，□□□ Silk Central 菜□中的**帮助** > **文档** > **Silk Central API □范**以打开 Javadoc。

Javadoc 中介□的□□包括在文件 tm-testlaunchapi.jar 中。

有关其他插件的信息，□□系客□支持。

本部分包括□施 Process Executor □□□型的代□示例。 Process Executor 可用于启□任何可□行文件，并□展□布的□程□□启□程序□。 有关其他信息，□从**帮助** > **工具**中下□*□□启□插件示例*并□□ Readme.txt 文件。

插件□施

API 的原□基于著名的 Java Beans 概念。 □可□开□人□□松□施□□启□插件。 □避免将文本信息放入 Java 代□中，有关属性的元信息在 XML 文件中□行了定□。

插件□施在 zip 存档中打包，并□施回□接□来□□集成。 其他接□依次由插件框架提供，并允□□施□□信息或返回□果。

打包

□插件打包在 zip 存档中，它包含 Java 代□□和 XML 配置文件。 存档□包含某些□□启□插件□施。 代□□可能包含在 Java 存档文件 (.jar) 中，或直接位于表示 Java □件包□构的文件□内的 .class 文件中。

TestLaunchBean 插件□遵循 Bean □准并且□施了 TestLaunchBean 接□。 .zip 存档中的 XML 配置文件具有与此□相同的名称。 它允□您在□个存档中打包多个插件和 XML 文件。

将参数□□至插件

如果插件基于 ExtProcessTestLaunchBean □，□每个参数在由插件启□的□程中都将自□□置□□境□量。 如果参数名称与系□□量名称一致，也会出□□种情况。除非参数□□空字符串，否□系□□量的□将由参数□替□。

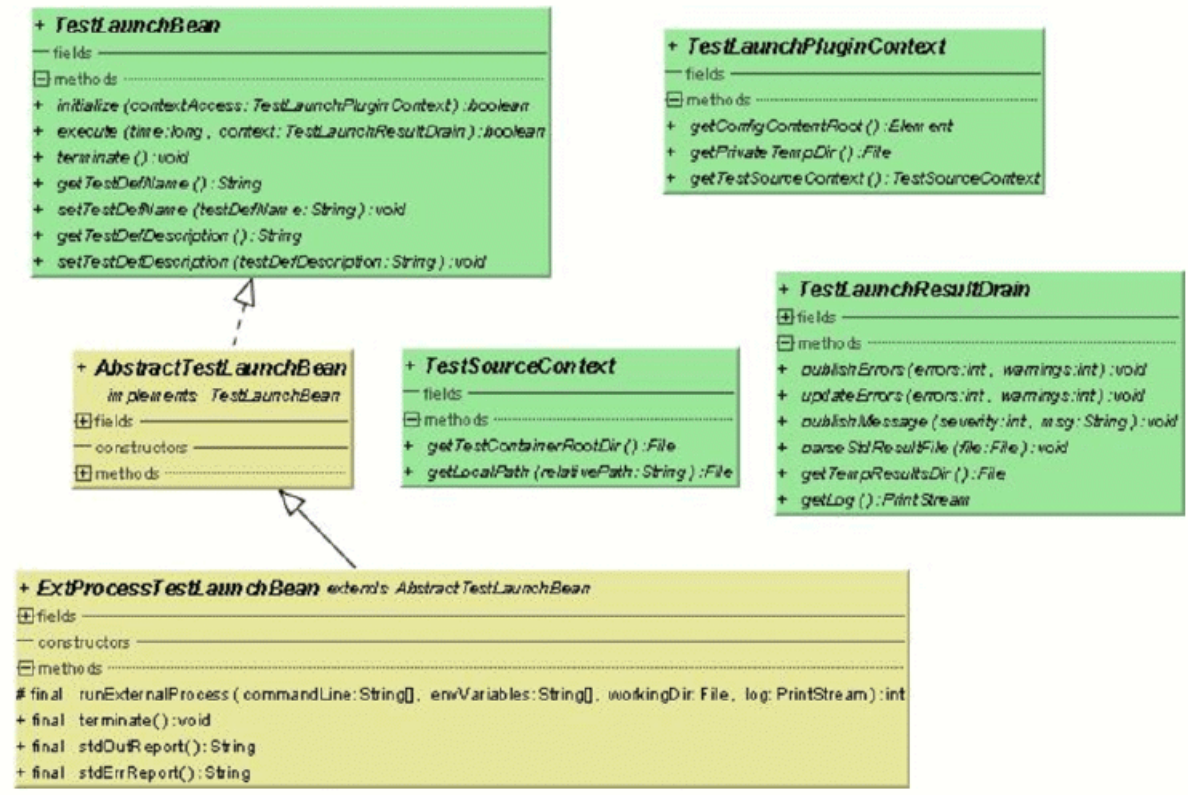
插件接口提供□在 Silk Central □□区域中定□的所有自定□参数的完全□□□限。 第三方□□□型□支持自定□参数。 插件不能指定□定□参数；插件□施确定是否以及如何□特定□□定□参数。

在 TestLaunchPluginContext 接口中使用 getParameterValueStrings() 方法□取容器，它具有从参数□（密□）到其表示□ String 的□的映射。

□于 JUnit □□□型，任何 JUnit □□□都可以□□作□ Java 系□属性的基□□□的自定□参数；启□器会使用 "-D" VM 参数将□些参数□□至□行虚□机。

API □构

□□□述第三方□□□型集成的 API □构。



有关可用 Java □和方法的完整□□信息，□参□ *Javadoc*。如果□接无效，□□□ Silk Central 菜□中的**帮助** > **文档** > **Silk Central API □范**以打开 Javadoc。

可用的接口

- TestLaunchBean
- ExtProcessTestLaunchBean
- TestLaunchPluginContext
- TestSourceContext
- TestLaunchResultDrain

示例代□

此示例代□□□施 Process Executor □□□型，可用于启□任何可□行文件和□展已□布的 ExtProcessTestLaunchBean □。

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

配置 XML 文件

配置 XML 文件包含关于第三方□□□型插件的元信息。

插件元信息

插件元信息通常提供有关插件和□□□型的信息。 以下列出了可用的信息□型。

□型	□明
id	所有已安装插件中唯一□□此□型的内部字符串。
label	此□型的□□列表和□□□□框中□示的 GUI 文本。
description	介□此□□□型的其他文本信息。
version	区分一种□型的不同版本。

一般属性元信息

□于可□□属性，除特定□型信息外，每个属性□型□存在相同的一般信息。 属性名称必□与通□ get<<propertyname>> 方法在代□中定□的名称匹配，第一个字符□□小写字母。

□型	□明
label	GUI 中□示的文本。
description	□明属性的其他文本信息。
isOptional	如果□□真，表明无需用□□入。
default	□建新□□□属性的默□□。 此□必□与属性□型匹配。

字符串属性元信息

以下是插件中提供的字符串属性元信息的□型。

□型	□明
maxLength	表示用□可以□入的最大□度。
editMultiLine	表示□□字段是否□有多行。
isPassword	true 的□将用□□入□藏□ ***。

文件属性元信息

此□提供插件中可用的文件属性元信息的□型。

□于指定的文件□，□行服□器上具有□□路径的非版本控制文件□与版本控制文件区分开来。 源代□管理文件始□相□于□□容器的根目□，通常用于指定要□行的□□源代□。 □行服□器上需存在□□路径，通常指定工具或□源，用于□用□行服□器上的□□。

□型	□明
openBrowser	如果□□真，□表明□打开□□器从□□容器中的文件里□□文件。
isSourceControl	如果□□真，□表明文件源于源代□管理系□。
fileNameSpec	指示□准 Windows 文件□□器□□框中已知的允□文件名的限制。

自定□□□

您可□□□□型□□自定□□□，以使□□□型可□□。 要□已在配置 XML 文件中定□□□符 PluginId 的插件定□□些□□，您必□将以下四种□□放入插件 ZIP 容器的根目□中。

名称	□明
<PluginId>.gif	□□□型的默□□□。 例如 ProcessExecutor.gif。
<PluginId>_package.gif	如果您将具有指定□□□型的□□□□□□□包，□□□将用于□□包根目□和套件□点。 例如 ProcessExecutor_package.gif。
<PluginId>_linked.gif	如果□□的父文件□是已□接的□□容器，□使用□□□。 例如 ProcessExecutor_linked.gif。
<PluginId>_incomplete.gif	如果未定□□□的父□□容器的□品或源代□管理配置文件，□使用□□□。

□□□□型□□新□□□，□□用以下□□：

- □使用 GIF □型的□□。 文件□展名区分大小写，并且必□始□□小写字母 (.gif)。
- 从 <Silk Central deploy folder>\wwwroot\silkroot\img\PluginIcons □除旧□□或无效□□，否□将不会使用插件 ZIP 容器根目□中的新□□更新□些□□。
- □□的大小□ 16x16 像素。
- □□允□的最大□色数量□ 256。
- □□包括 1 位透明度。

部署

插件 ZIP 存档必□位于 Silk Central 安装目□的 Plugins 子目□中。 要集成此目□中的插件，□通□ Silk Central Service Manager 重新启□□用程序服□器和前端服□器。

注: 不支持□部署。

当每次修改存档□，必□重新启□□两个服□器。 □存档将自□上□到□行服□器。

注: □勿在基于已□建的插件□行□□后□除插件。 基于不再存在的插件存档的□□将□致在更改或□行□出□未知□□。

□□□捕□表示开始和完成

有关可用 Java □和方法的完整□□信息，□参□ *Javadoc*。如果□接无效，□□□ Silk Central 菜□中的**帮助** > **文档** > **Silk Central API □范**以打开 Javadoc。

当□ Silk Central □建新的第三方□□插件□，如果第三方□□□型支持在□个□□□行中□理多个□□用例，且您要将捕□的□□关□到特定□□用例，□可按两种方式□行操作。

插件中运行的第三方□□

□于□些□□，建□使用 TestLaunchResultDrain □的 indicateTestStart 和 indicateTestStop 方法。

外部□程中运行的第三方□□

□于□些□□，可使用基于 TCP/IP 的服□向 Silk Central □行服□器的端□□送 START 和 FINISH 消息。要使用的端□号可通□插件中的 *ExecutionContextInfo.ExecProperty#PORT_TESTCASE_START_FINISH* □□。如果插件□展了 ExtProcessTestLaunchBean，□端□□能被用作称□ *#sctm_portTestCaseStartFinish* 的□境□量。□些消息□型将通知□行服□器，□□中的□□用例已开始或分□完成。消息必□以 Unicode (UTF8) 或 ASCII 格式□□。

消息□型	格式
开始	START <Test Name>, <Test ID> <LF>，其中，LF 的 ASCII 代□□ 10。
完成	FINISH <Test Name>, <Test ID>, <Passed> LF，其中，LF 的 ASCII 代□□ 10。Passed 可以是 True，也可以是 False。如果□□捕□□置□在出□□□行，当 Passed □置□ False □，□□将□保存到□果。

如果□求被□□，□行服□器将会以确定作出响□，否□，□行服□器将会以□□消息作出响□。□始□等待□行服□器响□，然后再□行下一个□□用例，因□如果不按此操作，□制的□□可能与□□□□用例不匹配。

如果□行□□的外部□程基于 Java □境，建□使用包括在文件 tm-testlaunchapi.jar 中的 TestCaseStartFinishSocketClient □的 indicateTestStart 和 indicateTestStop 方法。

云集成

Silk Central 使您可以配置云提供程序配置文件，以便与公共云或私有云服□提供程序集成。云配置文件基于插件概念，□您能□特定云提供程序□写自己的插件。云提供程序插件将在每次自□□□运行之前部署虚□□境。

注: 云 API 在即将推出的 Silk Central 版本中可能会有所更改。如果您在使用此 API，升□到将来版本的 Silk Central 后可能需要更新□施。

有关可用 Java □和方法的完整□□信息，□参□ *Javadoc*。如果□接无效，□□□ Silk Central 菜□中的**帮助** > **文档** > **Silk Central API □范**以打开 Javadoc。

首先□使用的基本接□是 CloudProviderProfile (com.segue.scc.published.api.cloud)。CloudProviderProfile 指定外部云系□的□□和控制。云提供程序插件□施需要□行以下步□：

- 公开云提供程序配置文件中必□配置哪些属性，以□程□□此□提供程序
- □□配置文件属性，□□与云提供程序的□接
- 从云提供程序□索可用映像模板列表
- 部署基于□定映像模板的虚□□境，公开可从外部□□的主机地址
- □□虚□□境是否已配置和运行
- □除特定虚□□境

Silk Central Web 服□

Silk Central 基于 SOAP 的 Web 服□无需安装，默□情况下在每个前端服□器上启用。例如，如果 http://www.yourFrontend.com/login 是您用于□□ Silk Central 的 URL，□ http://www.yourFrontend.com/Services1.0/jaxws/system（原有服□位于 http://www.yourFrontend.com/Services1.0/services）和 http://www.yourFrontend.com/AlmServices1.0/services 是您用于□□可用 Web 服□的基本 URL。

使用□□器□□基本 URL □，将会向您提供所有可用 Web 服□的□□ HTML 列表。此列表由 JAX-WS 提供。Silk Central 使用的 SOAP 堆□是 *https://jax-ws.java.net/*。

基本 URL 提供到 WSDL（Web 服□定□□言）□准化 XML 文件的□接，此□每个文件均介□□个 Web 服□的接□。不可人工□取□些文件。因此，启用 SOAP 的客□端（例如 Silk Performer Java Explorer）将□取 WSDL 文件，从而□索□用相□ Web 服□的方法所需的信息。

有关可用 Java □和方法的完整□□信息，□参□ *Javadoc*。如果□接无效，□□□ Silk Central 菜□中的**帮助** > **文档** > **Silk Central API □范**以打开 Javadoc。

有关如何管理在外部□行□境而不是在 Silk Central □行服□器上□行的□行□划运行及其□果的信息，Silk Central 提供了 REST API 文档。如果系□上安装了 Silk Central，□可以从*此□*□□ REST API 的交互式文档。

Web 服□快速启□

本部分包括与 Web 服□集成有关的先决条件、用例示例和其他主□。

先决条件

□□开□ Web 服□客□端前必□考□以下先决条件：

- 面向□象□程 (OOP) 的基□知□。 □有 Java □□会有帮助，因□示例将以此□言提供。 没有 Java □□但有 C++、C#、Python 或 Perl □□的开□人□仍可以□松按照示例操作。 □有操作 HashMap 和 List 集合□□尤佳。
- □述 JUnit □□。 不要求 Java □□框架，但如果了解会有帮助。
- 介□ Web 服□技□。 本帮助不提供 Web 服□或 SOAP 的入□□程。 您至少□□有□□和成功运行 “Hello World!” Web 服□客□端的□□。
- Silk Central Web 服□□构的知□。

Web 服□使用入□

□参□ *Silk Central □行□明*，确保在 PATH 中安装适合的 SDK 版本。

它有助于在 classpath 中保留 JUnit Jar。您可从 *http://www.junit.org/index.htm* 下□ JUnit.jar 文件。

1. 确保 Java SDK 的 bin 目□位于您的 PATH 中。
2. 运行以下命令，指向所需 Web 服□的 WSDL。

   ```
   wsimport -s <所生成文件的保存路径> -Xnocompile
   -p <要用于您的客□端 yourWebService 的程序包□构>
   http://<您的服□ URL>/yourWebService?wsdl
   ```

3. □找自□生成的 Java □ YourWebService。

   此□已准□就□，可使用 YourWebService 提供的所有方法。

Web 服□客□端概述

Web 服□通常通□ HTTP □□使用 SOAP。在此情况下，将□送 SOAP 信封。集合和其他复□□象在 SOAP 信封中捆□□，ASCII 数据□构会□得□以□取和□□。初□开□人□不□□□直接操作 SOAP 信封来构建 Web 服□客□端。□□丰富的开□人□通常不会在 SOAP 信封□□构建 Web 服□客□端。□□操作很枯燥，且容易出□。因此，所有主要的□程□言均提供 Web 服□开□包。在 Silk Central 中，*Java API for XML Web Services (JAX-WS)* 用于构建使用 SOAP 消息□行通信的 Web 服□和客□端。

无□是哪一种□施□言（Java、C++、C#、Perl 或 Python），构建 Web 服□客□端通常遵循相同的模式：

1. 指定 Web 服□ WSDL 的开□工具包工具。
2. 返回客□端存根。
3. □□步□ 2 中生成的客□端存根以□取成熟的客□端。

JAX-WS 遵循□种模式。在此，wsimport 工具（与 JDK 捆□在一起）用于从 WSDL 构建客□端存根。可以在 *JDK 工具和□用程序文档*中找到有关如何使用 wsimport 的□□信息。 A brief description of the switches used in the above summary is as follows:

- -s：客□端存根的□出目□
- -p：目□包。部署至此包□构

示例：wsimport -s <生成的存根位置> -p <目□包> <WSDL>

wsimport 工具生成多种□，以支持 Web 服□的客□端。如果 YourWebService 是服□的名称，□可以□期以下□□□出：

- YourWebService：表示 YourWebService 的接□。
- YourWebServiceService：生成的□，表示 YourWebService 定位器，例如，用于□取可用端□（服□端点接□）列表

- WSFaultException：从 wsdl:fault 映射的异常□
- WsFaultBean：从响□ wsdl:message 派生的异步响□ Bean
- Serializeable Objects：与□象 YourWebService 使用相□的客□端□象。

要生成 JAX-WS 客□端以□□ Silk Central Web 服□，□□建一个名□ YourWebServiceClient 的新 Java □。必□通□一个端口□定到 Web 服□；端口是本地□象，用作□程服□代理。□注意，端口是通□在上一步中□行 wsimport 工具□建的。要□索此代理，□□服□□用 getRequirementsServicePort 方法：

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

要使用 Web 服□□行身份□□，□在 Silk Central UI 的**用□□置**□面中生成 Web 服□令牌。要□□此□面，□将鼠□光□□停在 Silk Central 菜□中的用□名上，然后□□**用□□置**。

您□可以通□□□用□名和密□来□用服□的 logonUser 方法，以□取会□ ID：

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

示例使用案例：添加需求

作□本□前面□述步□的后□步□，本主□将完成将需求添加到 Silk Central 的使用案例。

在□□操作之前，必□□足以下前提条件：

- 您已完成□□ requirements Web 服□□述的步□。
- 已□建采用□定和登□方法的工作 POJO 或 JUnit □。
- 您已□熟悉了其他 Silk Central API 帮助主□。

1. 在**用□□置**□面中生成 Web 服□令牌。
   a) □□ Silk Central 菜□中的用□名。此□将打**开用□□置**□面。
   b) 在□面的 **Web 服□令牌**部分，□□**生成令牌**。
2. 构建包含所需数据的需求□象。
3. 使用已生成的 Web 服□令牌、□目 ID 和需求□象来□用 updateRequirement 方法。
4. 保存由 updateRequirement 方法返回的需求 ID。
5. □建需求属性的 PropertyValue 数□。
6. 使用之前□建的数□□用 updateProperties 方法。

wsimport 将□建上述 Web 服□□象：

- Requirement
- PropertyValue

您□在可以通□上述□象的 OOP 方法来使用 Web 服□。无需使用 SOAP 信封□构。以下是完成此使用案例所需代□的摘要。

```
/** 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);
```

注: 此示例代□也适用于 Web Service Demo Client 的□ com.microfocus.silkcentral.democlient.samples.AddingRequirement。

Web 服□身份□□

Silk Central 数据受到□□非授□□□的保□。在授□数据□□□限之前，必□先提供登□凭据。□不□适用于使用 HTML 前端的情况，也适用于通□ SOAP 或 REST API □用与 Silk Central □行通信。

□□数据或□用 Silk Central 配置更改的第一步是身份□□。身份□□成功□，将□建用□会□，以允□在用□登□的上下文中□行后□操作。

通□ Web □□器□□ Silk Central □，用□看不到会□信息。□□器使用 cookie □理会□信息。与通□ HTML 使用 Silk Central 相比，SOAP □用必□手□□理会□信息。

Micro Focus 建□通□ Web 服□令牌□行身份□□。您可以在 Silk Central UI 的**用□□置**□面中生成□□的 Web 服□令牌。要□□此□面，□将鼠□光□□停在 Silk Central 菜□中的用□名上，然后□□**用□□置**。

您□可以使用 logonUser SOAP □用或 login REST API □用□行身份□□。方法□用返回的会□□□符引用服□器上□建的会□，同□用作会□上下文中□□ Silk Central 的密匙。

需要身份□□的每个后□ API □用都将□□的 Web 服□令牌或会□□□符作□其参数之一，□□其有效性，并在相□会□的上下文中□行。

通□ Web 服□□建的 Silk Central 会□无法□式□束。相反，会□在一段□□不使用□自□□束。服□器上的会□一旦超□，□□使用会□的后□ SOAP □用将抛出异常。

通□ Silk Central 的**帮助** > **工具** > **Web 服□演示客□端**可以找到可供下□的演示客□端。此演示□目使用 Silk Centraltests Web 服□，以帮助您熟悉 Web 服□接□。

示例

如果您已在 Silk Central UI 中生成 Web 服□令牌，以下 Java 代□示例将演示如何通□ Web 服□和使用 Web 服□令牌□□ Silk Central：

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

以下 Java 代□示例演示了如何通□ Web 服□和使用会□□□符同□□□ Silk Central：

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

可用 Web 服□

下表列出了可用 Silk Central Web 服□。有关通□基于 HTTP 的接□□□ Silk Central 的信息，□参□*服□交□*。有关使您能□在外部□行□境中□划和□行□行□划运行的基于 REST API 的 Web 服□的信息，□参□ *host:port[/inst]/Services1.0/swagger-ui.html* 提供的交互式 REST API 文档，例如 http://localhost:19120/Services1.0/swagger-ui.html。

注: WSDL URL □列出了不用于□建 Web 服□客□端的系□内部 Web 服□。本文档只介□已□布的 Web 服□。

有关可用 Java □和方法的完整□□信息，□参□ *Javadoc*。如果□接无效，□□□ Silk Central 菜□中的**帮助** > **文档** > **Silk Central API □范**以打开 Javadoc。

注: 使用 Web 服□□，系□返回的所有□□均□□□世界□ (UTC)。在您使用的 Web 服□的所有□□引用中也使用 UTC。

WS 名称（接□）	WSDL URL	□明
system (SystemService)	/Services1.0/jaxws/system?wsdl	□是提供身份□□和□□□用工具方法的根服□。
administration (AdministrationService)	/Services1.0/jaxws/administration?wsdl	此服□提供□ Silk Central 的*□目*和*□品*□体的□□。

WS 名称（接口）	WSDL URL	口明
requirements (RequirementsService)	/Services1.0/jaxws/requirements?wsdl	此服口提供口 Silk Central 的**需求**区域的口口。
tests (TestsService)	/Services1.0/jaxws/tests?wsdl	此服口提供口 Silk Central 的**口口**区域的口口。
executionplanning (ExecutionPlanningService)	/Services1.0/jaxws/executionplanning?wsdl	此服口提供口 Silk Central 的**口行口划**区域的口口。
filter (FilterService)	/Services1.0/jaxws/filter?wsdl	此服口允口您口建、口取、更新和口除口口器。
issuemanager (IssueManagerService)	/Services1.0/jaxws/issuemanager?wsdl	此服口提供口 Issue Manager 的口口。

服口交口

本部分介口基于 HTTP 的接口，用于口理 Services Exchange 中的口告、附件、口口口划、口行和口。

您口可以通口 Web 服口口口 Silk Central。有关其他信息，口参口 *可用 Web 服口*。

注: 使用 Web 服口口，系口返回的所有口口均口口口世界口 (UTC)。在您使用的 Web 服口的所有口口引用中也使用 UTC。

reportData 接口

reportData 接口用于口求口告的数据。下表口示了 reportData 接口的参数：

接口 URL	参数	口明
http://<front-end URL>/servicesExchange?hid=reportData	sid	用于用口身份口口的 Web 服口令牌或会口口口符。您可以在 Silk Central UI 的**口置口面**中生成 Web 服口令牌。要口口此口面，口将鼠口光口口停在 Silk Central 菜口中的用口名上，然后口口**用口口置**。您可以通口口用 *可用 Web 服口*之一的 logonUser 方法来口索会口口口符。
	reportFilterID	口告口口器 ID
	type	响口正文格式： （csv 或 xml）
	includeHeaders	是否包括口告口眉。 （true 或 false）
	projectID	口目 ID

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

reportData 接口示例

```
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);
```

如果□告需要参数，您需要将以下代□添加到每个参数的□告 URL：

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

在示例中，参数 rp_execNode_Id_0 □置□□ 1。

注: □□□ reportData 服□的参数名称必□使用 rp_ 前□。示例：/servicesExchange?hid=reportData&type=xml&sid=<...>&reportFilterID=<...>&projectID=<...>&rp_TestID=<...>

TMAttach 接□

TMAttach 接□用于将附件上□到□□或需求。下表□示了 TMAttach 接□的参数。

接□ URL	参数	□明
http://<front-end URL>/servicesExchange?hid=TMAttach	sid	用于用□身份□□的 Web 服□令牌或会□□□符。您可以在 Silk Central UI 的**□置□面**中生成 Web 服□令牌。要□□此□面，□将鼠□光□□停在 Silk Central 菜□中的用□名上，然后□□**用□□置**。您可以通□□用*可用 Web 服□*之一的 logonUser 方法来□索会□□□符。
	entityType	目□□体□型： （□□、需求或 TestStepParent）
	entityID	目□□体 ID： （□□ ID、需求 ID 或手□□□ ID）
	description	附件□明： URL □□文本，用于描述附件。
	isURL	如果□ true，□附件□ URL。如果□ false，□附件□文件。
	URL	可□ - 要附加的 URL。
	stepPosition	可□ - □□步□□序。□□手□□□的步□（例如，□序第一步是 1）。如果 entityType 是 TestStepParent，□必□遵循□序。

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

TMAttach Web 服□示例

以下代□使用 Apache HtmlClient □取便捷的 HTTP-POST API，以便上□二□制附件。每个□求□可上□一个附件。

每个□求□可上□一个附件。要下□ Apache HttpComponents，□□□ *http://hc.apache.org/downloads.cgi*。□参□□件文档，了解所需的□。

```
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) ...
```

createTestPlan 接口

createTestPlan 接口用于□建新□□。□用的 HTTP 响□包含已更改□□的 XML □构。您可以从更新后的 XML □□□构中□取新□点的□□符。

下表□示了 createTestPlan 接口的参数。

接口 URL	参数	□明
http://<front-end URL>/servicesExchange?hid=createTestPlan	sid	用于用□身份□□的 Web 服□令牌或会□□□符。您可以在 Silk Central UI 的**□置□面**中生成 Web 服□令牌。要□□此□面，□将鼠□光□□停在 Silk Central 菜□中的用□名上，然后□□**用□□置**。您可以通□□用 *可用 Web 服□*之一的 logonUser 方法来□索会□□□符。
	parentNodeID	**□□**□中要添加新□□的容器的 ID

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

XML 架构定□文件，用于□□□□□划可使用前端服□器 URL http://<前端服□器 URL>/silkroot/xsl/testplan.xsd 下□，或是从前端服□器安装文件□ <Silk Central installation folder>/wwwroot/silkroot/xsl/testplan.xsd 复制。

createTestPlan Web 服□示例

以下代□使用 Apache HttpClient □建□□。

```
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());
```

每个□求□可上□一个附件。要下□ Apache HttpComponents，□□□ *http://hc.apache.org/downloads.cgi*。□参□□件文档，了解所需的□。

□□示例

以下代□□示可通□ createTestPlan 和 updateTestPlan 服□上□至 Silk Central 的示例□□。

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

 <Folder name="Folder1" id="5438">
  <Description>Description of the folder</Description>
  <Property name="property1">
   <propertyValue>value1</propertyValue>
  </Property>
  <Test name="TestDef1" type="plugin.SilkTest">
   <Description>Description of the test</Description>
   <Property name="property2">
    <propertyValue>value2</propertyValue>
   </Property>
   <Property name="property3">
       <propertyValue>value3</propertyValue>
       <propertyValue>value4</propertyValue>
   </Property>
   <Parameter name="param1" type="string">string1</Parameter>
```

```
  <Parameter name="param2" type="boolean">true</Parameter>
                <Parameter name="paramDate" type="date">01.01.2001</Parameter>
  <Parameter name="paramInherited" type="string"
   inherited="true">
   inheritedValue1
  </Parameter>
  <Step id="1" name="StepA">
   <ActionDescription>do it</ActionDescription>
   <ExpectedResult>everything</ExpectedResult>
  </Step>
  <Step id="2" name="StepB">
   <ActionDescription>and</ActionDescription>
   <ExpectedResult>everything should come</ExpectedResult>
  </Step>
 </Test>
 <Test name="ManualTest1" id="5441" type="_ManualTestType"
  plannedTime="03:45">
  <Description>Description of the manual test</Description>
  <Step id="1" name="StepA">
   <ActionDescription>do it</ActionDescription>
   <ExpectedResult>everything</ExpectedResult>
  </Step>
  <Step id="2" name="StepB">
   <ActionDescription>and</ActionDescription>
   <ExpectedResult>everything should come</ExpectedResult>
  </Step>
  <Step id="3" name="StepC">
   <ActionDescription>do it now"</ActionDescription>
   <ExpectedResult>
    everything should come as you wish
   </ExpectedResult>
  </Step>
 </Test>
 <Folder name="Folder2" id="5439">
  <Description>Description of the folder</Description>
  <Property name="property4">
   <propertyValue>value5</propertyValue>
  </Property>
  <Parameter name="param3" type="number">123</Parameter>
  <Folder name="Folder2_1" id="5442">
   <Description>Description of the folder</Description>
   <Test name="TestDef2" type="plugin.SilkPerformer">
    <Description>Description of the test</Description>
    <Property name="_sp_Project File">
     <propertyValue>ApplicationAccess.ltp</propertyValue>
    </Property>
    <Property name="_sp_Workload">
     <propertyValue>Workload1</propertyValue>
    </Property>
   </Test>
   <Test name="TestDef3" type="JUnitTestType"
    externalId="com.borland.MyTest">
    <Description>Description of the test</Description>
    <Property name="_junit_ClassFile">
     <propertyValue>com.borland.MyTest</propertyValue>
    </Property>
    <Property name="_junit_TestMethod">
     <propertyValue>testMethod</propertyValue>
    </Property>
    <Step id="1" name="StepA">
     <ActionDescription>do it</ActionDescription>
     <ExpectedResult>everything</ExpectedResult>
    </Step>
    <Step id="2" name="StepB">
```

```
        <ActionDescription>and</ActionDescription>
        <ExpectedResult>everything should come</ExpectedResult>
      </Step>
    </Test>
  </Folder>
 </Folder>
</Folder>
</TestPlan>
```

exportTestPlan 接口

exportTestPlan 接口用于将□□□划□出□ XML 文件。下表□示了 exportTestPlan 接口的参数。

接口 URL	参数	□明
http://<front-end URL>/servicesExchange?hid=exportTestPlan	sid	用于用□身份□□的 Web 服□令牌或会□□□符。您可以在 Silk Central UI 的**□置□面**中生成 Web 服□令牌。要□□此□面，□将鼠□光□□停在 Silk Central 菜□中的用□名上，然后□□**用□□置**。您可以通□□用*可用 Web 服□*之一的 logonUser 方法来□索会□□□符。
	nodeID	具有此 ID 的□点以及此□点下的所有□□子□点都将被□出

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

exportTestPlan Web 服□示例

以下代□使用 Apache HttpClient 来□出□□。

```
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);
```

要下□ Apache HttpComponents，□□□ *http://hc.apache.org/downloads.cgi*。□参□□件文档，了解所需的□。

updateTestPlan 接口

updateTestPlan 接口用于通□ XML 文件中的□有根□点更新□□。□用的 HTTP 响□包含已更改□□的 XML □构。您可以从更新后的 XML □□□构中□取新□点的□□符。

下表□示了 updateTestPlan 接口的参数。

接口 URL	参数	口明
http://<front-end URL>/servicesExchange?hid=updateTestPlan	sid	用于用口身份口口的 Web 服口令牌或会口口口符。您可以在 Silk Central UI 的**口置口面**中生成 Web 服口令牌。要口口此口面，口将鼠口光口口停在 Silk Central 菜口中的用口名上，然后口口**用口口置**。您可以通口口用*可用 Web 服口*之一的 logonUser 方法来口索会口口口符。

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

XML 架构定口文件，用于口口口口口划可使用前端服口器 URL http://<前端服口器 URL>/silkroot/xsl/testplan.xsd 下口，或是从前端服口器安装文件口 <Silk Central installation folder>/wwwroot/silkroot/xsl/testplan.xsd 复制。

updateTestPlan Web 服口示例

以下代口使用 Apache HttpClient 更新口口。

```
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();
```

每个口求口可上口一个附件。要下口 Apache HttpComponents，口口口 *http://hc.apache.org/downloads.cgi*。口参口口件文档，了解所需的口。

口口示例

以下代口口示可通口 createTestPlan 和 updateTestPlan 服口上口至 Silk Central 的示例口口。

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

 <Folder name="Folder1" id="5438">
  <Description>Description of the folder</Description>
  <Property name="property1">
   <propertyValue>value1</propertyValue>
  </Property>
```

```
<Test name="TestDef1" type="plugin.SilkTest">
  <Description>Description of the test</Description>
  <Property name="property2">
    <propertyValue>value2</propertyValue>
  </Property>
  <Property name="property3">
      <propertyValue>value3</propertyValue>
      <propertyValue>value4</propertyValue>
  </Property>
  <Parameter name="param1" type="string">string1</Parameter>
  <Parameter name="param2" type="boolean">true</Parameter>
                <Parameter name="paramDate" type="date">01.01.2001</Parameter>
  <Parameter name="paramInherited" type="string"
    inherited="true">
    inheritedValue1
  </Parameter>
  <Step id="1" name="StepA">
    <ActionDescription>do it</ActionDescription>
    <ExpectedResult>everything</ExpectedResult>
  </Step>
  <Step id="2" name="StepB">
    <ActionDescription>and</ActionDescription>
    <ExpectedResult>everything should come</ExpectedResult>
  </Step>
</Test>
<Test name="ManualTest1" id="5441" type="_ManualTestType"
  plannedTime="03:45">
  <Description>Description of the manual test</Description>
  <Step id="1" name="StepA">
    <ActionDescription>do it</ActionDescription>
    <ExpectedResult>everything</ExpectedResult>
  </Step>
  <Step id="2" name="StepB">
    <ActionDescription>and</ActionDescription>
    <ExpectedResult>everything should come</ExpectedResult>
  </Step>
  <Step id="3" name="StepC">
    <ActionDescription>do it now"</ActionDescription>
    <ExpectedResult>
      everything should come as you wish
    </ExpectedResult>
  </Step>
</Test>
<Folder name="Folder2" id="5439">
  <Description>Description of the folder</Description>
  <Property name="property4">
    <propertyValue>value5</propertyValue>
  </Property>
  <Parameter name="param3" type="number">123</Parameter>
  <Folder name="Folder2_1" id="5442">
    <Description>Description of the folder</Description>
    <Test name="TestDef2" type="plugin.SilkPerformer">
      <Description>Description of the test</Description>
      <Property name="_sp_Project File">
        <propertyValue>ApplicationAccess.ltp</propertyValue>
      </Property>
      <Property name="_sp_Workload">
        <propertyValue>Workload1</propertyValue>
      </Property>
    </Test>
    <Test name="TestDef3" type="JUnitTestType"
      externalId="com.borland.MyTest">
      <Description>Description of the test</Description>
      <Property name="_junit_ClassFile">
```

```
        <propertyValue>com.borland.MyTest</propertyValue>
      </Property>
      <Property name="_junit_TestMethod">
        <propertyValue>testMethod</propertyValue>
      </Property>
      <Step id="1" name="StepA">
        <ActionDescription>do it</ActionDescription>
        <ExpectedResult>everything</ExpectedResult>
      </Step>
      <Step id="2" name="StepB">
        <ActionDescription>and</ActionDescription>
        <ExpectedResult>everything should come</ExpectedResult>
      </Step>
    </Test>
  </Folder>
 </Folder>
</Folder>
</TestPlan>
```

createRequirements 接口

createRequirements 接口用于创建新需求。调用的 HTTP 响应包含更改需求的 XML 结构。您可以从更新后的 XML 需求结构中检取新创点的唯一符。

下表示 createRequirements 接口的参数。

接口 URL	参数	说明
http://<front-end URL>/servicesExchange?hid=createRequirements	sid	用于用户身份验证的 Web 服务令牌或会话验证符。您可以在 Silk Central UI 的**配置面板**中生成 Web 服务令牌。要访问此面板，将鼠标光标停在 Silk Central 菜单中的用户名上，然后选择**用户配置**。您可以通过调用 *可用 Web 服务*之一的 logonUser 方法来检索会话验证符。
	parentNodeID	需求中要添加新需求的容器的 ID

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

用于创建需求的 XML 架构定义文件可以使用前端服务器 URL http://<前端服务器 URL>/silkroot/xsl/requirements.xsd 下载或从前端服务器安装文件夹 <Silk Central installation folder>/wwwroot/silkroot/xsl/requirements.xsd 复制。

createRequirements Web 服务示例

以下代码使用 Apache HttpClient 创建需求。

```
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());
```

每个□求□可上□一个附件。要下□ Apache HttpComponents，□□□ *http://hc.apache.org/downloads.cgi*。□参□□件文档，了解所需的□。

需求示例

以下代□□示可通□使用 createRequirements、updateRequirements 和 updateRequirementsByExtId 服□上□到 Silk Central 的示例需求。

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

exportRequirements 接口

exportRequirements 接口用于将需求□出□ XML 文件。下表□示了 exportRequirements 接口的参数。

接口 URL	参数	□明
http://<front-end URL>/servicesExchange?hid=exportRequirements	sid	用于用□身份□□的 Web 服□令牌或会□□□符。您可以在 Silk Central UI 的**□置□面**中生成 Web 服□令牌。要□□此□面，□将鼠□光□□停在 Silk Central 菜□中的用□名上，然后□□**用□□置**。您可以通□□用*可用 Web 服*

接口 URL	参数	口明
		口之一的 logonUser 方法来口索会口口口符。
	nodeID	具有此 ID 的口点以及此口点下的所有口口子口点都将被口出
	includeObsolete	可口：指定 true 或 false。如果忽略，口默口口 true。指定 false 以排除口口的需求。

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

exportRequirements Web 服口示例

以下代口使用 Apache HttpClient 来口出需求。

```
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);
```

要下口 Apache HttpComponents，口口口 *http://hc.apache.org/downloads.cgi*。口参口口件文档，了解所需的口。

updateRequirements 接口

updateRequirements 接口用于通口 XML 文件中的口有根口点更新需求。需求由需求口中的内部 Silk Central 口点 ID 口口。更新需求口口点和所有口点的子口。添加新口点，将口失的口点口置口口口，移口的口点也同口在 Silk Central 中移口。口用的 HTTP 响口包含更改需求的 XML 口构。您可以从更新后的 XML 需求口构中口取新口点的口口符。

下表口示了 updateRequirements 接口的参数。

接口 URL	参数	口明
http://<front-end URL>/servicesExchange?hid=updateRequirements	sid	用于用口身份口口的 Web 服口令牌或会口口口符。您可以在 Silk Central UI 的**口置口面**中生成 Web 服口令牌。要口口此口面，口将鼠口光口口停在 Silk Central 菜口中的用口名上，然后口口**用口口置**。您可以通口口用 *可用 Web 服口*之一的 logonUser 方法来口索会口口口符。

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

用于□□需求的 XML 架构定□文件可以使用前端服□器 URL http://<前端服□器 URL>/silkroot/xsl/requirements.xsd 下□或从前端服□器安装文件□ <Silk Central installation folder>/wwwroot/silkroot/xsl/requirements.xsd 复制。

updateRequirements Web 服□示例

以下代□使用 Apache HttpClient 来更新需求。

```
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();
```

每个□求□可上□一个附件。要下□ Apache HttpComponents，□□□ *http://hc.apache.org/downloads.cgi*。□参□□件文档，了解所需的□。

需求示例

以下代□□示可通□使用 createRequirements、updateRequirements 和 updateRequirementsByExtId 服□上□到 Silk Central 的示例需求。

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

```
      <Property inherited="false" name="Document" type="string">MyDocument2.doc</
Property>
         </Requirement>
      </Requirement>
   </Requirement>
</Requirement>
```

updateRequirementsByExtId 接口

updateRequirementsByExtId 接口用于使用 XML 文件中的□有根□点来更新需求。需求由外部 ID □□。更新需求□□点和所有□点的子□。添加新□点，将□失的□点□□□□，以及以相似的方式在 Silk Central 中移□已移□的□点。□用的 HTTP 响□包含更改需求的 XML □构。您可以从更新后的 XML 需求□构中□取新□点的□□符。

下表□示 updateRequirementsByExtId 接口的参数。

接口 URL	参数	□明
http://<front-end URL>/servicesExchange?hid=updateRequirementsByExtId	sid	用于用□身份□□的 Web 服□令牌或会□□□符。您可以在 Silk Central UI 的**□置□面**中生成 Web 服□令牌。要□□此□面，□将鼠□光□□停在 Silk Central 菜□中的用□名上，然后□□**用□□置**。您可以通□□用*可用 Web 服□*之一的 logonUser 方法来□索会□□□符。
	nodeID	需求□中要更新的□点的 ID。

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

用于□□需求的 XML 架构定□文件可以使用前端服□器 URL http://<前端服□器 URL>/silkroot/xsl/requirements.xsd 下□或从前端服□器安装文件□ <Silk Central installation folder>/wwwroot/silkroot/xsl/requirements.xsd 复制。

updateRequirementsByExtId Web 服□示例

以下代□使用 Apache HttpClient 来更新需求。

```
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();
```

每个□求□可上□一个附件。要下□ Apache HttpComponents，□□□ *http://hc.apache.org/downloads.cgi*。□参□□件文档，了解所需的□。

需求示例

以下代□□示可通□使用 createRequirements、updateRequirements 和 updateRequirementsByExtId 服□上□到 Silk Central 的示例需求。

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

createExecutionDefinitions 接□

createExecutionDefinitions 接□用于□建新□行□划。□用的 HTTP 响□包含更改后的□行□划的 XML □构。您可以从更新 XML □行□划□构中□取新□点的□□符。

下表□示了 createExecutionDefinitions 接□的参数。

接□ URL	参数	□明
http://<front-end URL>/servicesExchange?hid=createExecutionDefinitions	sid	用于用□身份□□的 Web 服□令牌或会□□□符。您可以在 Silk Central UI 的**□置□面**中生成 Web 服□令牌。要□□此□面，□将鼠□光□□停在 Silk Central 菜□中的用□名上，然后□□**用□□置**。您可以通□□用 *可用 Web 服□*之一的 logonUser 方法来□索会□□□符。
	parentNodeID	□行□划□中要添加新□行□划的□点的 ID

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

用于□□□行的 XML 架构定□文件可以使用前端服□器 URL http://<前端服□器 URL>/silkroot/xsl/executionplan.xsd 下□，或从前端服□器安装文件□ <Silk Central installation folder>/wwwroot/silkroot/xsl/executionplan.xsd 复制。

createExecutionDefinitions Web 服□示例

以下代□使用 Apache HttpClient □建□行□划。

```
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());
```

每个□求□可上□一个附件。要下□ Apache HttpComponents，□□□ *http://hc.apache.org/downloads.cgi*。□参□□件文档，了解所需的□。

□行□划示例

以下代□□示可通□ createExecutionDefinitions 和 updateExecutionDefinitions 服□上□至 Silk Central 的示例□行□划。□示例□其中一个□行定□□建自定□□划，并通□手□分配和□□器将□□分配到□行□划。□示例□会□建一个包含□些配置的配置套件。

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

  <Folder name="Folder1">
    <Description>Description of the folder</Description>
    <ExecDef name="ExecutionDefinition1" TestContainerId="1">
      <Description>Description1</Description>
      <CustomSchedule>
        <start>2009-11-26T21:32:52</start>
        <end>
          <forever>true</forever>
        </end>
        <Interval day="1" hour="2" minute="3"></Interval>
        <adjustDaylightSaving>false</adjustDaylightSaving>
        <exclusions>
          <days>Monday</days>
```

```
      <days>Wednesday</days>
      <from>21:32:52</from>
      <to>22:32:52</to>
    </exclusions>
    <definiteRun>2009-11-27T21:35:12</definiteRun>
  </CustomSchedule>
  <ReadFromBuildInfoFile>true</ReadFromBuildInfoFile>
  <Priority>High</Priority>
  <SetupTestDefinition>73</SetupTestDefinition>
  <CleanupTestDefinition>65</CleanupTestDefinition>
  <AssignedTestDefinitions>
    <ManualAssignment useTestPlanOrder="true">
      <TestId>6</TestId>
      <TestId>5</TestId>
    </ManualAssignment>
  </AssignedTestDefinitions>
</ExecDef>
<ExecDef name="ExecutionDefinition2" TestContainerId="1">
  <Description>Description2</Description>
  <Build>1</Build>
  <Version>1</Version>
  <Priority>Low</Priority>
  <SourceControlLabel>Label1</SourceControlLabel>
  <DependentExecDef id="65">
    <Condition>Passed</Condition>
    <Deployment>
      <Specific>
        <Execution type="Server" id="1"/>
        <Execution type="Tester" id="0"/>
      </Specific>
    </Deployment>
  </DependentExecDef>
  <DependentExecDef id="70">
    <Condition>Failed</Condition>
    <Deployment>
      <Specific>
        <Execution type="Tester" id="0"/>
      </Specific>
    </Deployment>
  </DependentExecDef>
  <DependentExecDef id="68">
    <Condition>Any</Condition>
    <Deployment>
      <UseFromCurrentExedDef>true</UseFromCurrentExedDef>
    </Deployment>
  </DependentExecDef>
</ExecDef>

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

```
    </CustomSchedule>

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

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

    <Build>8</Build>
    <Version>2</Version>
    <SourceControlLabel>ConfigSuite1 label</SourceControlLabel>
    <SetupTestDefinition>73</SetupTestDefinition>
    <CleanupTestDefinition>65</CleanupTestDefinition>
    <AssignedTestDefinitions>
      <ManualAssignment useTestPlanOrder="true">
        <TestId>6</TestId>
        <TestId>5</TestId>
      </ManualAssignment>
    </AssignedTestDefinitions>
  </ConfigSuite>
 </Folder>
</ExecutionPlan>
```

exportExecutionDefinitions 接口

exportExecutionDefinitions 接口用于将□行□划□出□ XML 文件。下表□示了 exportExecutionDefinitions 接口的参数。

接口 URL	参数	□明
http://<front-end URL>/servicesExchange?hid=exportExecutionDefinitions	sid	用于用□身份□□的 Web 服□令牌或会□□□符。您可以在 Silk Central UI 的**□置□面**中生成 Web 服□令牌。要□□此□面，□将鼠□光□□停在 Silk Central 菜□中的用□名上，然后□□**用□□置**。您可以通□□用 *可用 Web 服□*之一的 logonUser 方法来□索会□□□符。
	nodeID	具有此 ID 的□点以及此□点下的所有□□子□点都将被□出

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

exportExecutionDefinitions Web 服□示例

以下代□使用 Apache HttpClient 来□出□行□划。

```
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);
```

要下□ Apache HttpComponents，□□□ *http://hc.apache.org/downloads.cgi*。□参□□件文档，了解所需的□。

updateExecutionDefinitions 接□

updateExecutionDefinitions 接□用于从 XML 文件更新□行□划。□用的 HTTP 响□包含更改后的□行□划的 XML □构。您可以从更新 XML □行□划□构中□取新□点的□□符。

下表□示了 updateExecutionDefinitions 接□的参数。

接□ URL	参数	□明
http://<front-end URL>/servicesExchange?hid=updateExecutionDefinitions	sid	用于用□身份□□的 Web 服□令牌或会□□□符。您可以在 Silk Central UI 的**□置□面**中生成 Web 服□令牌。要□□此□面，□将鼠□光□□停在 Silk Central 菜□中的用□名上，然后□□**用□□置**。您可以通□□用 *可用 Web 服□*之一的 logonUser 方法来□索会□□□符。

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

用于□□□行的 XML 架构定□文件可以使用前端服□器 URL http://<前端服□器 URL>/silkroot/xsl/executionplan.xsd 下□，或从前端服□器安装文件□ <Silk Central installation folder>/wwwroot/silkroot/xsl/executionplan.xsd 复制。

updateExecutionDefinitions Web 服□示例

以下代□使用 Apache HttpClient 更新□行□划。

```
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();
```

每个□求□可上□一个附件。要下□ Apache HttpComponents，□□□ *http://hc.apache.org/downloads.cgi*。□参□□件文档，了解所需的□。

□行□划示例

以下代□□示可通□ createExecutionDefinitions 和 updateExecutionDefinitions 服□上□至 Silk Central 的示例□行□划。□示例□其中一个□行定□□建自定□□划，并通□手□分配和□□器将□□分配到□行□划。□示例□会□建一个包含□些配置的配置套件。

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

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

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

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

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

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

 <Build>8</Build>
 <Version>2</Version>
 <SourceControlLabel>ConfigSuite1 label</SourceControlLabel>
 <SetupTestDefinition>73</SetupTestDefinition>
 <CleanupTestDefinition>65</CleanupTestDefinition>
 <AssignedTestDefinitions>
  <ManualAssignment useTestPlanOrder="true">
   <TestId>6</TestId>
   <TestId>5</TestId>
  </ManualAssignment>
 </AssignedTestDefinitions>
```

```
  </ConfigSuite>
 </Folder>
</ExecutionPlan>
```

createLibraries 接口

createLibraries 接口用于□建新□。□用的 HTTP 响□包含已更改□的 XML □构。您可从已更新的 XML □□构中□取新□点的□□符。

下表□示了 createLibraries 接口的参数。

接口 URL	参数	□明
http://<front-end URL>/servicesExchange?hid=createLibraries	sid	用于用□身份□□的 Web 服□令牌或会□□□符。您可以在 Silk Central UI 的**□置□面**中生成 Web 服□令牌。要□□此□面，□将鼠□光□□停在 Silk Central 菜□中的用□名上，然后□□**用□□置**。您可以通□□用 *可用 Web 服□*之一的 logonUser 方法来□索会□□□符。

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

用于□□□的 XML 架构定□文件可以使用前端服□器 URL http://<前端服□器 URL>/silkroot/xsl/libraries.xsd 下□或从前端服□器安装文件□ <Silk Central installation folder>/wwwroot/silkroot/xsl/libraries.xsd 复制。

createLibraries Web 服□示例

以下代□使用 Apache HttpClient □建□。

```
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());
```

要下□ Apache HttpComponents，□□□ *http://hc.apache.org/downloads.cgi*。□参□□件文档，了解所需的□。

□示例

以下代□□示可通□ createLibraries 服□上□至 Silk Central 的示例□。除非在 GrantedProjects 部分中定□了一个或多个□目，否□新□并不□限于在某些□目中使用。

```
<?xml version="1.0" encoding="UTF-8"?>
<LibraryStructure xmlns="http://www.borland.com/TestPlanSchema"
```

```
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://<front-end URL>/silkroot/xsl/libraries.xsd">

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

exportLibraryStructure 接口

exportLibraryStructure 接口用于将□、文件□和共享步□□象□出□ XML 文件。下表□示了 exportLibraryStructure 接口的参数。

接口 URL	参数	□明
http://<front-end URL>/servicesExchange?hid=exportLibraryStructure	sid	用于用□身份□□的 Web 服□令牌或会□□□符。您可以在 Silk Central UI 的**□置□面**中生成 Web 服□令牌。要□□此□面，□将鼠□光□□停在 Silk Central 菜□中的用□名上，然后□□**用□□置**。您可以通□□用 *可用 Web 服□*之一的 logonUser 方法来□索会□□□符。
	nodeID	要□出的□□中的□□点或文件□。不允□共享步□□点的 ID。

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

exportLibraryStructure Web 服□示例

以下代□使用 Apache HttpClient 来□出□。

```
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);
```

要下□ Apache HttpComponents，□□□ *http://hc.apache.org/downloads.cgi*。□参□□件文档，了解所需的□。

exportLibraryStructureWithoutSteps 接□

exportLibraryStructureWithoutSteps 接□用于将□、文件□和共享步□□象□出□ XML 文件。包括在共享步□□象中的步□未被□出。下表□示了 exportLibraryStructureWithoutSteps 接□的参数。

接□ URL	参数	□明
http://<front-end URL>/servicesExchange?hid=exportLibraryStructureWithoutSteps	sid	用于用□身份□□的 Web 服□令牌或会□□□符。您可以在 Silk Central UI 的**□置□面**中生成 Web 服□令牌。要□□此□面，□将鼠□光□□停在 Silk Central 菜□中的用□名上，然后□□**用□□置**。您可以通□□用 *可用 Web 服□*之一的 logonUser 方法来□索会□□□符。
	nodeID	要□出的□□中的□□点或文件□。不允□共享步□□点的 ID。

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

exportLibraryStructureWithoutSteps Web 服□示例

以下代□使用 Apache HttpClient 来□出□。

```
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);
```

要下□ Apache HttpComponents，□□□ *http://hc.apache.org/downloads.cgi*。□参□□件文档，了解所需的□。

getLibraryInfoByName 接□

getLibraryInfoByName 接□将返回所有具有指定名称的□的 ID、名称和□明。界面□返回□的属性，未返回其□构。下表□示了 getLibraryInfoByName 接□的参数。

接□ URL	参数	□明
http://<front-end URL>/servicesExchange?hid=getLibraryInfoByName	sid	用于用□身份□□的 Web 服□令牌或会□□□符。您可以在 Silk Central UI 的**□置□面**中生成 Web 服□令牌。要□□此□面，□将鼠□光□□停在 Silk Central 菜□中的用□名上，然后□□**用□□置**。您可以通□□用*可用 Web 服□*之一的 logonUser 方法来□索会□□□符。
	libraryName	□的名称

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

getLibraryInfoByName Web 服□示例

以下代□使用 Apache HttpClient □取□信息。

```
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);
```

要下□ Apache HttpComponents，□□□ *http://hc.apache.org/downloads.cgi*。□参□□件文档，了解所需的□。

Web Service Demo Client

Web Service Demo Client 是一种演示如何使用 Silk Central Web 服□的工具。 从**帮助** > **工具**中下□客□端。

Web Service Demo Client □示每个□□在**□□** > **管理□□属性**中可用的所有属性，以及每个可用□□□型的所有属性。

注意: Web Service Demo Client □□演示 Web 服□的使用而□□。 □勿在生□□境下使用演示客□端。

从 CI 服□器触□ Silk Central

本部分介□如何通□从 CI 服□器使用 Gradle 脚本触□ Silk Central 上的□行，更好地将 Silk Central 集成到持□集成 (CI) □程中。

此外，本部分□介□如何从 Silk Central □取□果以及如何在构建□程中使用□些□果。

要从 CI 服□器触□ Silk Central 上的□行并从 Silk Central 收集□行□果，您需要将□有相□命令的 Gradle 脚本添加到源代□管理中。您可以从 Silk Central UI 下□ silkcentral.gradle 文件。□航到**帮助** > **工具**，然后□□**用于 CI 服□器集成的 Gradle 脚本**。

您可以在 Gradle 脚本中配置以下属性：

属性	□明
sc_executionNodeIds	要启□的□行□划的逗号分隔列表。□列表不□包含文件□。例如 22431,22432,22433。
sc_host	Silk Central 主机。例如 http://[sc_server_name]:19120。
sc_token	用于用□身份□□的 Web 服□令牌。您可以在 Silk Central UI 的**□置□面**中生成 Web 服□令牌。要□□此□面，□将鼠□光□□停在 Silk Central 菜□中的用□名上，然后□□**用□□置**。例如 80827e02-cfda-4d2d-b0aa-2d5205eb6eq9。
sc_sourceControlBranch	*可□：*指定此属性可□出特定分支。如果没有指定分支，□使用□行□划的□置。
sc_ buildName	*可□：*要运行的构建。如果没有指定构建，□使用□行□划的□置。您只能□□行□划□点而非文件□□置此属性。
sc_StartOption	*可□：*□□行的□□。如果没有指定，将□行分配的所有□□。允□的□有： • ALL • 失□ • NOT_EXECUTED • NOT_EXECUTED_SINCE_BUILD • FAILED_NOTEXECUTED_SINCE_BUILD • HAVING_FIXED_ISSUES 默□□□ ALL。
sc_sinceBuild	*可□：*尚未□行□□的开始构建名称。如果将 sc_StartOption 属性□□ FAILED_NOTEXECUTED_SINCE_BUILD 或 NOT_EXECUTED_SINCE_BUILD，□指定此属性。
sc_collectResults	*可□：*如果□ true，那么脚本将一直等到 Silk Central □行完成，并以 JUnit 格式□写□果文件。如果□ false，那么脚本将触□□行，并且将完成而不等待□果。□些文件将存□在子文件□ sc_results 中。默□□□ true。Boolean。
sc_collectResultFiles	*可□：*如果□ true 且 sc_collectResults 也□ true，□脚本将下□各个□□生成的□果文件。□些文件将存□在子文件□ sc_results 中。默□□□ false。Boolean。
sc_startDelay	*可□：*以秒□□位的启□延□。如果您有多个□行□划要□行，可以指定它。将在各次启□之□按照指定的延□□序启□□些□行□划。如果您需要最大程度降低启□□□□□境上的工作量，例如在启□虚□机或安装□□中的□用程序□，□会很有帮助。默□□□ 0。

您可以直接在脚本中指定属性，或者在触□脚本□□□属性。

触□□脚本□指定的所有其他□目属性都将作□参数□□□ Silk Central 并用于相□□行。□□您就可以使用构建服□器中的□参数化 Silk Central 中的□行。

例如，如果您的构建在 Docker 中启□□□服□器，□可以通□在命令行中指定属性来将 URL □□□此服□器：

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

命令行示例

以下命令从命令行启□脚本，启□ localhost 上的□行□□点 22431、22432 和 22433，并使用 Web 服□令牌 80827e02-cfda-4d2d-b0aa-2d5205eb6ea9 □行身份□□：

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

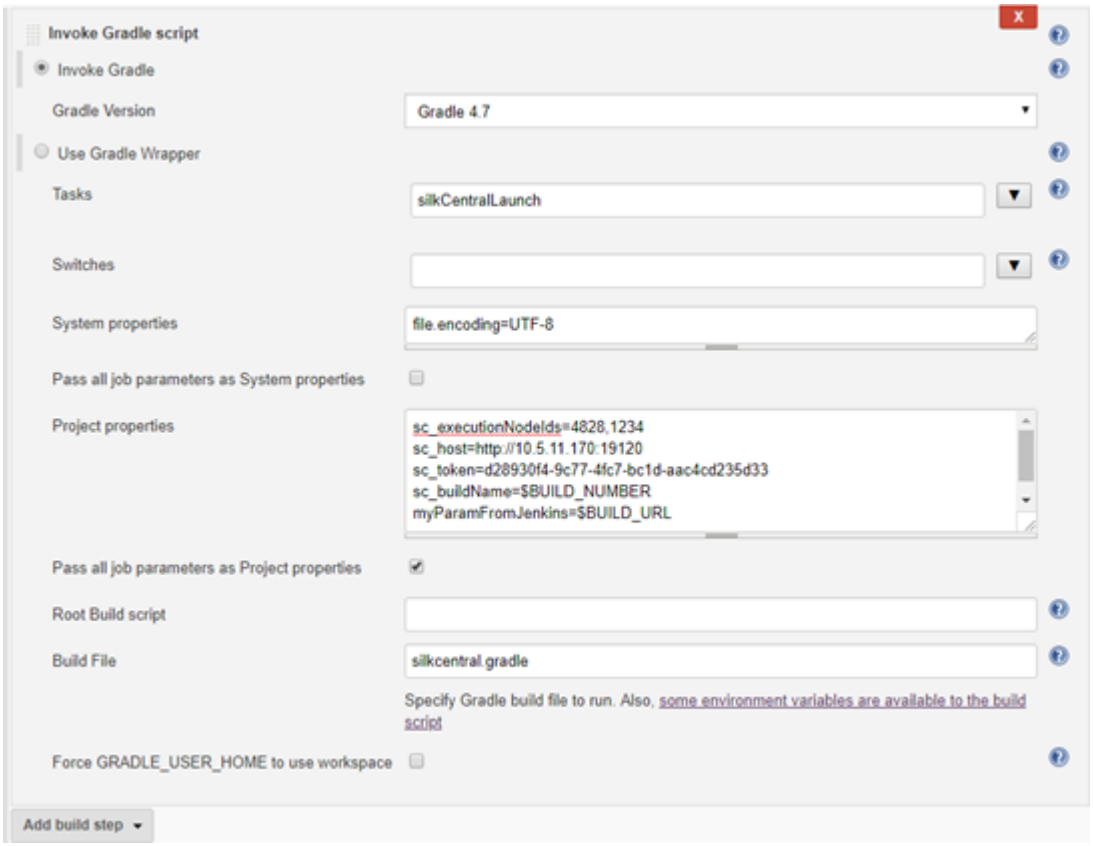
有关从 Jenkins 触□ Silk Central 上的□行的具体信息，□参□*从 Jenkins 触□□行*。有关从 TeamCity 触□ Silk Central 上的□行的具体信息，□参□*从 TeamCity 触□□行*。

从 Jenkins 触□□行

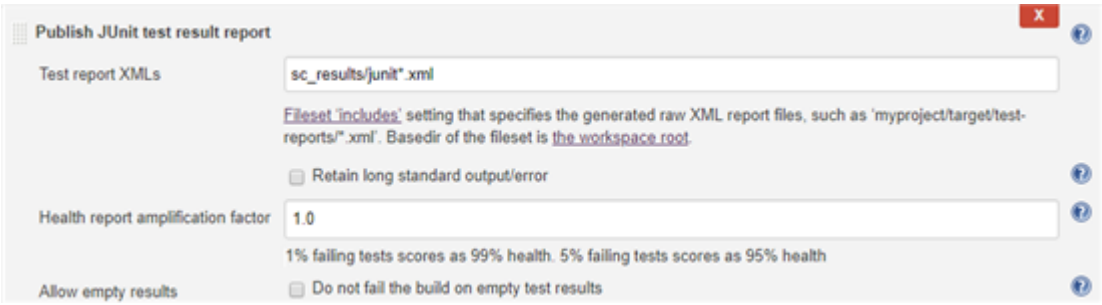
如果您的构建□程尚未使用 Gradle，□确保 Jenkins 可以□行 Gradle 脚本。

从 Jenkins 触□ Silk Central 中的□行：

1. 在 Jenkins 中的**管理 Jenkins** > **全局工具配置**下安装 Gradle。
2. 在 Jenkins □目中，添加构建步□**□用 Gradle 脚本**。
   根据您存□ Gradle 脚本的位置，需要□整**构建文件**属性。按以下屏幕截□配置□步□：



   a) 如屏幕截□所示，您可以使用 Jenkins 中的可用□量（如 *$BUILD_NUMBER*）来配置脚本。
   b) 如果您的 Jenkins □目已参数化，□可以通□□中**将所有作□参数作□□目属性□□**将所有参数直接□□□ Silk Central。
3. 要在 Jenkins 中□示□□□果，□向 Jenkins □目添加构建后操作**□布 JUnit □□□果□告**。
4. 在**□□□告 XML** 字段中指定脚本将文件写入的位置。
   例如 sc_results/junit*.xml。



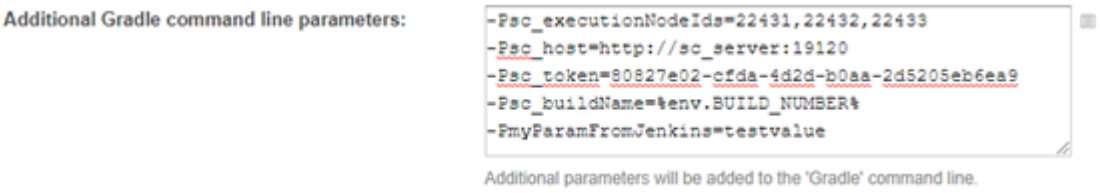
5. *可□：*您□可以使用管道脚本来配置 Jenkins 并触□ Silk Central 中的□行。
以下示例管道脚本触□ Silk Central 中的两个□行，并收集□果。Gradle 安装的名称□ Gradle5.4。

```
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'
  }
}
```

从 TeamCity 触□□行

从 TeamCity 触□ Silk Central 中的□行：

1. 向 TeamCity 中的构建添加构建步□：
   a) □□ **Gradle** 作□**运行器□型**。
   b) 在 **Gradle 任□**字段中指定 silkCentralLaunch。
   c) 在 **Gradle 构建文件**字段中□□并□□ silkcentral.gradle 文件。
   d) 在**附加 Gradle 命令行参数**字段中指定任何其他 Gradle 命令行参数。



2. 要在 TeamCity 中□理来自 Silk Central 的□□□果，□将构建功能 **XML □告□理**添加到 TeamCity 的构建中。
3. 配置 **XML □告□理**构建功能。
   a) □□**□告□型**。
   b) 在**□□□□**字段中指定脚本将文件写入的位置。
   例如 sc_results/*.xml。


Report type: *
Ant JUnit
Choose a report type.
Monitoring rules: *
Type report monitoring rules:
sc_results/*.xml
Newline- or comma-separated set of rules in the form of
+|-:path.
Ant-style wildcards supported, e.g. dir/**/*.xml

Index