MICRO FOCUS

Silk Central 19.5

API 帮助

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

2019-01-21

内容

Silk Central API 帮助

本指南提供创建和部署插件以将第三方工具集成到 Silk Central 所需的信息，以及有关管理外部执行计划运行结果的信息。对于基于 SOAP 的可用 Web 服务，本指南包含相关规范和 API 说明，并且介绍了如何将第三方插件集成到 Silk Central 中。

注: 本指南假设您熟悉 Web 服务的实施和使用。

概述

Silk Central 提供了用于集成第三方应用程序的基于 SOAP 的 Web 服务，以及用于管理外部执行计划运行结果的 REST API。

借助基于 SOAP 的 Silk Central Web 服务，您可以通过配置 Silk Central 插件来集成现有的源代码管理、问题跟踪和需求管理工具。Silk Central 附带各种示例插件。

借助于用于管理外部执行计划运行结果的 REST API，您可以将未由 Silk Central 执行服务器执行的执行计划运行生成的外部结果上载到 Silk Central 以进行进一步测试管理。您还可以指定这些外部执行计划，它们将在外部执行环境而不是 Silk Central 执行服务器上执行。

基于 SOAP 的 Web 服务的文档

有关可用 Java 类和方法的完整详细信息，请参阅 *Javadoc*。如果链接无效，请单击 Silk Central 菜单中的**帮助 > 文档 > Silk Central API 规范**以打开 Javadoc。。

REST API 的文档

如果系统上安装了 Silk Central，则可以从*此处*访问 REST API 的交互式文档。

Silk Central 集成插件

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

您在使用插件时需自行承担风险。在任何情况下，对于由使用插件导致或引起的任何类型的直接、间接、特殊、意外或结果性损失（包括但不限于利益损失），Micro Focus 概不承担任何责任。

创建插件

概述

本节介绍如何为 Silk Central 创建插件。此处仅介绍所有插件类型的通用任务。

插件种类

Silk Central 提供几种插件 API。每种 API 被视为一个*种类*。

编译

有关开发和编译插件的信息，请参阅相应 Java 版本的 Silk Central 发行说明。这对于与 Silk Central Java Runtime Environment 的兼容性非常重要。Silk Central 使用的是 AdoptOpenJDK。

部署

创建插件类和实施种类 API 之后，您可创建插件包（JAR 或 ZIP 文件）。

- 如果插件没有进一步的依赖关系（或依赖于已成为 Silk Central 一部分的库），则只需创建包含类的 JAR 文件。
- 如果插件依赖于其他库，请将这些库放入子目录 `lib` 中，然后将所有库一起打包成 ZIP 存档。

将创建的文件放入位于 <应用程序服务器安装目录>`\plugins\` 的插件目录中。

注: 您必须重新启动应用程序服务器和前端服务器，以使新部署的插件在 Silk Central 中可用。有关重新启动服务器的更多信息，请参阅本帮助中的*管理*主题。

分发

由于 Silk Central 知道插件种类的类型，因此也知道哪些服务器（执行服务器、应用程序服务器和前端服务器）需要哪些种类。每种插件均可安装在应用程序服务器上。Silk Central 会将正确插件自动分配到每个服务器。

代码覆盖率集成

本章节介绍的 Java API 接口是为支持第三方（外部）代码覆盖率工具集成的 Silk Central 创建插件所必需的。

您可通过代码覆盖率工具提供关于测试覆盖哪些代码的信息。 Silk Central 提供了以下现成的代码覆盖率工具：

- Silk Central Java 代码分析 (`Java Code Analysis Agent`)
- DevPartner Studio .NET 代码分析 (`Windows Code Analysis Framework`)

注: 如果被测应用程序运行于 Linux 上，请参阅主题*在 Linux AUT 环境中安装代码分析框架*。

如果前面两种工具还不够，您可创建和部署自己的代码覆盖率集成。 请参阅 `Creating Your Own Code Coverage Plugin`。

注: 除了在应用程序服务器上部署自定义应用程序之外（请参阅*创建插件*主题），您还需要在代码分析框架服务器上部署自定义应用程序。 这是您的 AUT 和代码覆盖率工具所在位置。 路径如下：`\Silk \Silk Central <version>\Plugins`

创建您自己的代码覆盖率插件

本主题介绍如何创建代码覆盖率插件。您应熟悉 Silk Central 基线概念。在 Silk Central 中，每次运行之前都需要基线。基线包括测试应用程序中的所有命名空间/程序包/类/方法。

注: Silk Central API 需要返回用于代码覆盖率运行的 XML 文件。这意味着，如果代码覆盖率工具在数据库中存储其代码覆盖率信息，您将需要采取其他步骤检索数据。

注: 不支持多个执行服务器根据同一代码分析框架运行测试。

1. 将库 `scc.jar` 添加到您的类路径，因为它包含必须扩展的接口。可以在 Silk Central 安装目录的 `lib` 目录中找到 JAR 文件。
2. 添加以下两个导入语句：

```
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. 创建实现 `CodeAnalysisProfile` 的类。
4. 从代码覆盖率界面中添加所有需要的方法，如以下步骤所示。您可参阅示例接口类了解解其定义并手动实现方法，也可复制和粘贴为您提供了导入和方法定义的主题*示例配置文件类*。

注: 您将写入的下述步骤中的方法实际上会在 Silk Central 需要时调用。这意味着您将无法直接调用这些方法。

5. 代码 `getBaseline`。此方法应返回包含应用程序中所有命名空间/程序包/类/方法的 XML 文件。请参阅*示例 XML 数据*主题文件，以了解文件格式。您应使用示例 XSD 文件验证 XML。请参阅*代码覆盖率 XSD* 主题，以了解 XSD。

   此函数将在开始覆盖之前调用并由开始测试运行的 Silk Central 执行服务器触发，以开始代码分析并返回所有要覆盖的对象。输出结果需要使用 CA-Framework 安装文件夹中包含的 XML 架构中指定的格式转换为 XML。
6. 代码 `startCoverage`。此调用将命令代码覆盖率工具开始收集数据。如果已开始，则返回 `true`。

   Silk Central 代码覆盖率框架将在完成 `getBaseLine()` 方法之后调用此函数。您应在此让代码覆盖率工具开始收集代码覆盖率数据。
7. 代码 `stopCoverage`。此调用将命令代码覆盖率工具停止收集数据。如果成功，则返回 `true`。

   此函数将在 `startCoverage` 之后调用，由完成测试运行的 Silk Central 执行服务器触发，以停止代码分析。
8. 代码 `getCoverage`。此函数将返回 XML 文件，其中包括从 `startCoverage` 和 `stopCoverage` 方法之间收集到的数据。请参阅*示例 XML 数据*主题，以了解文件格式。您应使用示例 XSD 文件验证 XML。请参阅*代码覆盖率 XSD* 主题，以了解 XSD。

   此函数将在 `stopCoverage()` 之后调用并返回已收集的所有覆盖率数据。输出结果需要使用 XML 架构中指定的格式转换为 XML。
9. 代码 `GetName`。此函数将提供用于引用代码覆盖率工具的名称。例如，此值将用作**编辑代码分析设置**对话框上的**代码分析配置文件**列表中的一个值。

   此函数首先由 Silk Central 代码覆盖率框架调用。插件名称显示在 Silk Central 中的代码覆盖率列表中。
10. 将插件生成 jar 文件，然后将 jar 文件压缩成 zip 文件。
11. 将插件部署到以下位置：
    - Silk Central 安装文件夹的 `Plugins` 目录中。
    - `CA-Framework` 安装的 `Plugins` 目录中。

示例配置文件类

此示例文件概述代码覆盖率插件的所有必要方法、导入和实施。

```
//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;
  }

}
```

代码覆盖率 XSD

下面是用于验证代码覆盖率工具生成的 XML 的代码覆盖率 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

代码覆盖率 API 需要以下格式的 XML。

您也可以使用提供的示例 XSD 文档验证 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="35:1,36:1,37:1,38:1,39:1,40:1,41:1,42:1,43:1,44:1,45:1,46:1,48:1,49:1,5
0:1,51:1,52:1,53:1,54:1,55:1,56:1" total="21" type="line"/>
    </method>
  </class>
</data>
```

在 **Linux AUT** 环境中安装代码分析框架

如果您创建的代码覆盖率插件将与在 Linux 操作系统下测试的 .NET 应用程序进行交互，则应执行以下步骤。

1. 用于 Linux 的代码分析框架可从**帮助** > **工具** > **Linux 代码分析框架**获取。下载并将其复制到 Linux 机器上的根文件夹或任何其他文件夹。
2. 确保已在测试应用程序的机器上安装 Java Runtime Environment (JRE) 8。
3. 解压 `CA-Framework.tar.gz`。
4. 将代码分析插件置于 `<install dir>/19.5Silk Central/Plugins` 文件夹中。
5. 将目录更改为 `<install dir>/19.5Silk Central/Code Analysis`。
6. 查找将运行 `CA-Framework` 进程的 `startCodeAnalysisFramework.sh` 外壳脚本。
7. 运行以下命令以将文件转换为 Unix 格式：`dos2unix startCodeAnalysisFramework.sh`。
8. 运行以下命令以设置执行外壳脚本所需的权限：`chmod 775 startCodeAnalysisFramework.sh`。
9. 运行以下外壳脚本以运行 CAFramework 进程：`./startCodeAnalysisFramework.sh`。

代码分析框架已准备就绪，可从 Silk Central 中使用它。

源代码管理集成

源代码管理配置文件可使 Silk Central 与外部源代码管理系统集成。

部署后，自定义源代码管理插件可在 Silk Central 中配置，允许您定义 Silk Central 执行服务器应检索程序源代码以执行测试的位置。

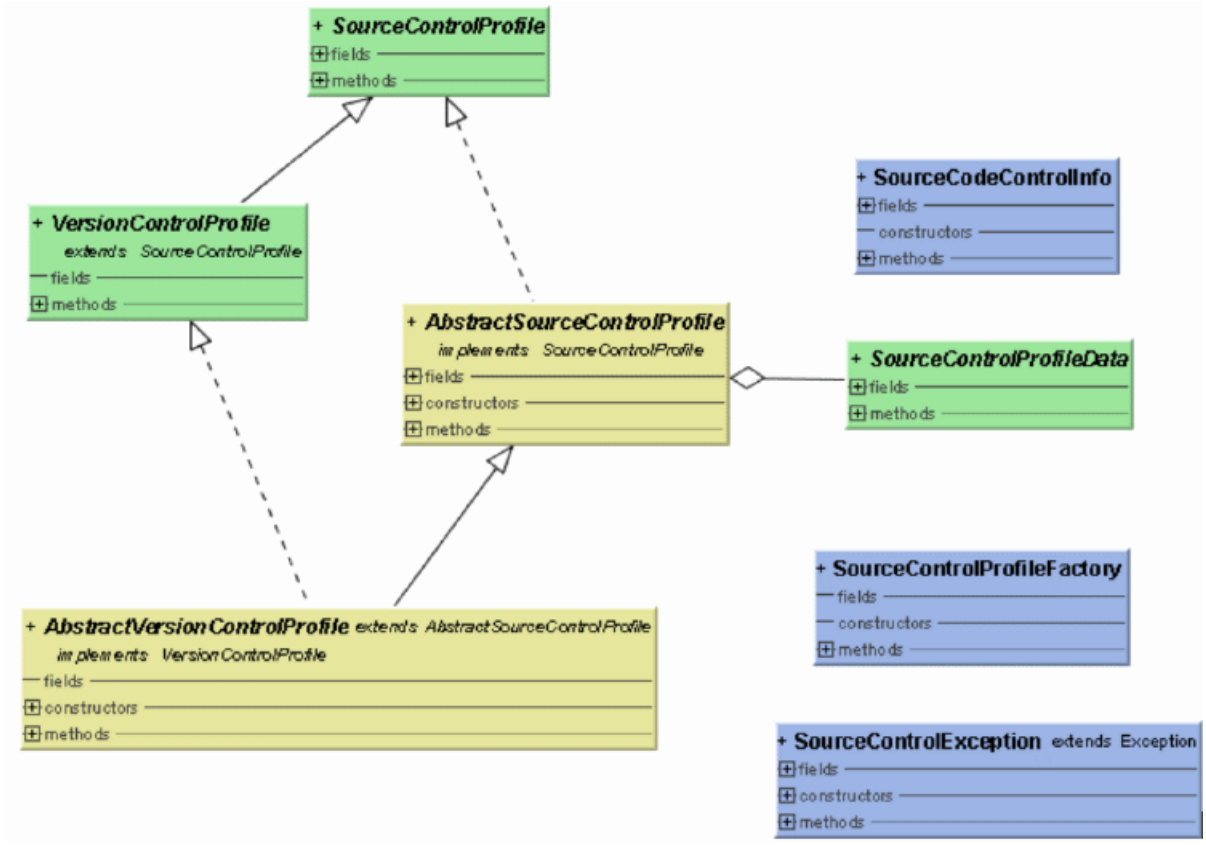
查看 `C:\Program Files (x86)\Silk\Silk Central 19.5\instance_<实例编号>_<实例名称>\Plugins\subversion.zip` 中的数据包 `com.segue.scc.vcs.subversion` 的来源，了解这些元素如何相结合。

源代码管理集成接口

Silk Central 区分 `SourceControlProfile` 和 `VersionControlProfile`。 区别在于 `SourceControlProfile` 没有版本管理，而 `VersionControlProfile` 有版本管理。

以下是用于源代码管理集成的 Silk Central 接口：

- `SourceControlProfile`
- `VersionControlProfile`
- `SourceControlProfileData`
- `SourceControlException`
- `SourceControlInfo`



有关可用 Java 类和方法的完整详细信息，请参阅 *Javadoc*。如果链接无效，请单击 Silk Central 菜单中的**帮助 > 文档 > Silk Central API 规范**以打开 Javadoc。

源代码管理集成约定

每次实施必须提供默认构造函数，或者将 `SourceControlProfileData` 作为参数的构造函数。 如果未提供此构造函数，则必须提供 `SourceControlProfileData` 的 bean setter。

由于每个接口方法指定了要抛出的 `SourceControlException`，因此不允许以接口使用的任何方法抛出 `RuntimeException`。

问题跟踪集成

为 Silk Central 创建可启用第三方（外部）问题跟踪系统 (ITS) 集成的插件需要使用本章中讨论的界面。

通过定义问题跟踪配置文件，您可以将**测试**区域内的测试链接到第三方问题跟踪系统中的问题。链接的问题状态将定期从第三方问题跟踪系统进行更新。

查看 `C:\Program Files (x86)\Silk\Silk Central19.5\instance_<实例编号>_<实例名称>\Plugins\Bugzilla3.zip` 中的数据包 `com.segue.scc.issuetracking.bugzilla3` 的来源，了解这些元素如何相结合。

Java 接口

有关可用 Java 类和方法的完整详细信息，请参阅 *Javadoc*。如果链接无效，请单击 Silk Central 菜单中的**帮助 > 文档 > Silk Central API 规范**以打开 Javadoc。

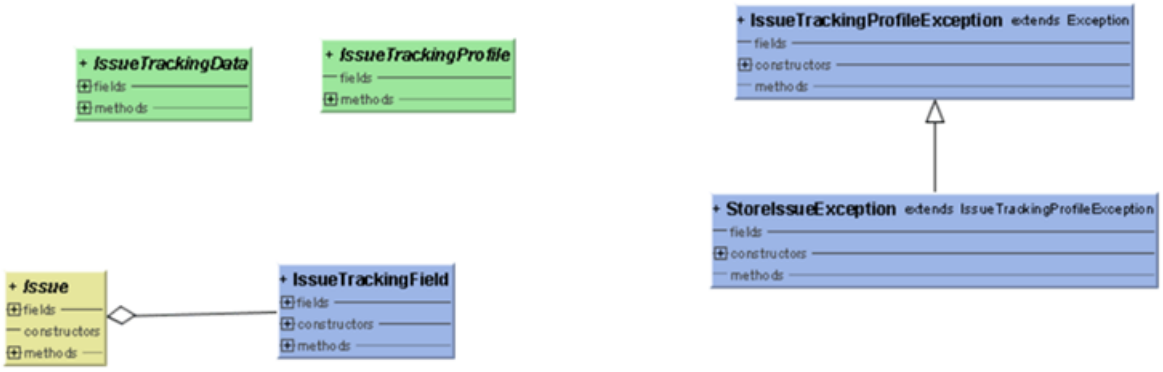
构建环境

将库 `scc.jar` 添加到 classpath，因为它包含必须扩展的接口。可以在 Silk Central 安装目录的 `lib` 目录中找到 JAR 文件。

您必须扩展两个接口/类：

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

类/接口



- `IssueTrackingProfile`
- `IssueTrackingData`
- `Issue`
- `IssueTrackingField`

- `IssueTrackingProfileException`

需求管理集成

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

本部分介绍可使您实施第三方插件的 Java 应用程序编程接口，以便将 Silk Central 中的需求与第三方需求管理系统的需求同步。 本部分介绍识别需求插件及其部署的接口。

在标准 Silk Central 的插件概念之后将提供 JAR 或 ZIP 文件，此文件会自动部署到所有前端服务器，以便访问第三方工具来进行配置和同步。 此插件实施指定接口，允许 Silk Central 将其识别为需求插件，并提供所需的登录属性来登录第三方工具。

有关可用 Java 类和方法的完整详细信息，请参阅 *Javadoc*。如果链接无效，请单击 Silk Central 菜单中的**帮助 > 文档 > Silk Central API 规范**以打开 Javadoc。

有关其他插件的信息，请联系客户支持。

Java 接口

有关可用 Java 类和方法的完整详细信息，请参阅 *Javadoc*。如果链接无效，请单击 Silk Central 菜单中的**帮助 > 文档 > Silk Central API 规范**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 包含以下附加接口：

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

第三方测试类型集成

Silk Central 可使您为可用测试类型（包括 Silk Performer、Silk Test Classic、手动测试、NUnit、JUnit 和 Windows 脚本宿主 (WSH)）标准集之外的测试类型创建自定义插件。 创建新测试类型插件后，您的自定义测试类型将在**新建测试**对话框的**类型**列表框中可用，以及可用于在 Silk Central 中创建新测试的标准测试类型。

插件指定配置测试和实施测试执行所需的属性。 属性的元信息通过*配置 XML 文件*定义。

插件方法的目的是根据常用测试框架（例如 JUnit、NUnit 或脚本语言 (WSH)）支持测试，以便根据特定测试环境轻松自定义 Silk Central。 定义完善的 Silk Central 公共 API 使您可实施符合自动测试需求的专用解决方案。 Silk Central 向可从 Java 实施或通过命令行调用的任何第三方工具开放和扩展。

有关可用 Java 类和方法的完整详细信息，请参阅 *Javadoc*。如果链接无效，请单击 Silk Central 菜单中的**帮助 > 文档 > Silk Central API 规范**以打开 Javadoc。

Javadoc 中介绍的类别包括在文件 `tm-testlaunchapi.jar` 中。

有关其他插件的信息，请联系客户支持。

本部分包括实施 Process Executor 测试类型的代码示例。 Process Executor 可用于启动任何可执行文件，并扩展发布的进程测试启动程序类。 有关其他信息，请从**帮助** > **工具**中下载*测试启动插件示例*并阅读 `Readme.txt` 文件。

插件实施

API 的原则基于著名的 Java Beans 概念。 这可让开发人员轻松实施测试启动插件。 为避免将文本信息放入 Java 代码中，有关属性的元信息在 XML 文件中进行了定义。

插件实施在 zip 存档中打包，并实施回调接口来实现集成。 其他接口依次由插件框架提供，并允许实施访问信息或返回结果。

打包

该插件打包在 zip 存档中，它包含 Java 代码库和 XML 配置文件。 存档还包含某些测试启动插件实施。 代码库可能包含在 Java 存档文件 (`.jar`) 中，或直接位于表示 Java 软件包结构的文件夹内的 `.class` 文件中。

`TestLaunchBean` 插件类遵循 Bean 标准并且实施了 `TestLaunchBean` 接口。 `.zip` 存档中的 XML 配置文件具有与此类相同的名称。 它允许您在单个存档中打包多个插件和 XML 文件。

将参数传递至插件

如果插件基于 `ExtProcessTestLaunchBean` 类，则每个参数在由插件启动的进程中都将自动设置为环境变量。 如果参数名称与系统变量名称一致，也会出现这种情况。除非参数值为空字符串，否则系统变量的值将由参数值替换。

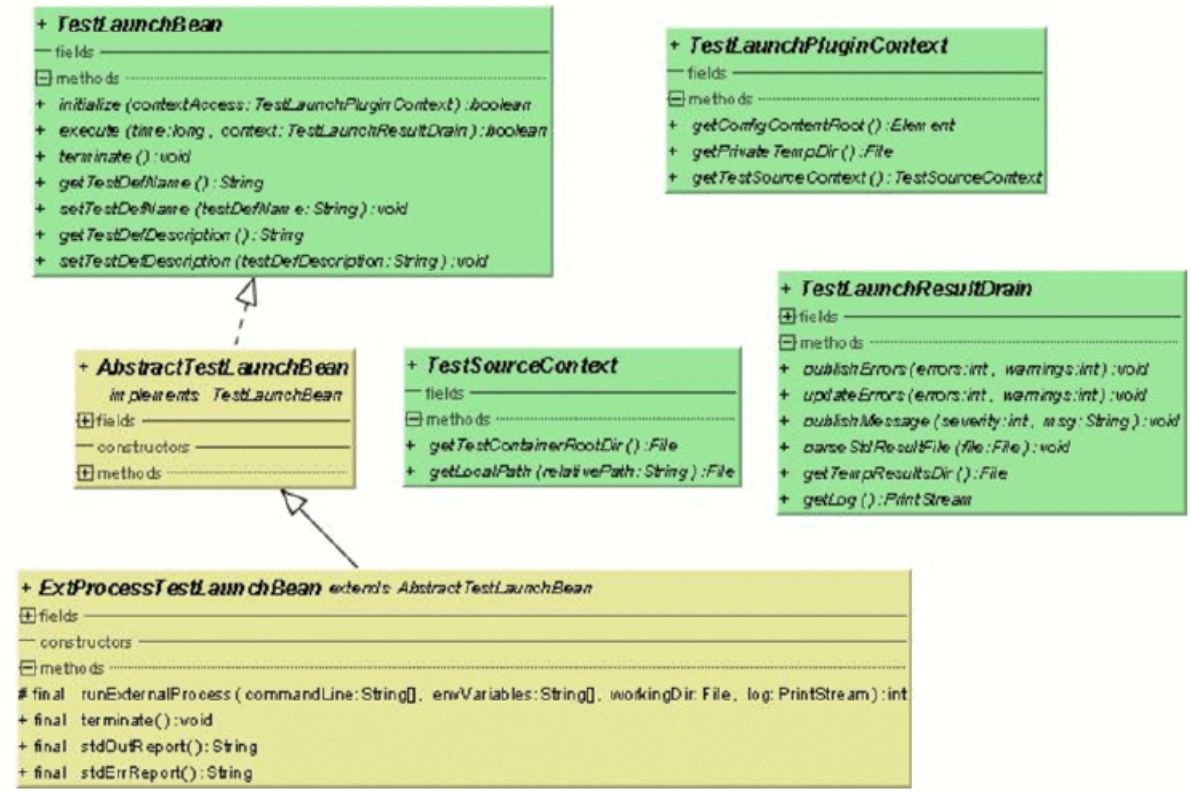
插件接口提供对在 Silk Central **测试**区域中定义的所有自定义参数的完全访问权限。 第三方测试类型仅支持自定义参数。 插件不能指定预定义参数；插件实施确定是否以及如何为特定测试定义参数。

在 `TestLaunchPluginContext` 接口中使用 `getParameterValueStrings()` 方法获取容器，它具有从参数值（密钥）到其表示为 `String` 的值的映射。

对于 JUnit 测试类型，任何 JUnit 测试类都可以访问作为 Java 系统属性的基础测试的自定义参数；启动器会使用 `"-D" VM` 参数将这些参数传递至执行虚拟机。

API 结构

该图详述第三方测试类型集成的 API 结构。



有关可用 Java 类和方法的完整详细信息，请参阅 *Javadoc*。如果链接无效，请单击 Silk Central 菜单中的**帮助 > 文档 > Silk Central API 规范**以打开 Javadoc。

可用的接口

- `TestLaunchBean`
- `ExtProcessTestLaunchBean`
- `TestLaunchPluginContext`
- `TestSourceContext`
- `TestLaunchResultDrain`

示例代码

此示例代码块实施 Process Executor 测试类型，可用于启动任何可执行文件和扩展已发布的 `ExtProcessTestLaunchBean` 类。

```
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测试类型插件的元信息。

插件元信息

插件元信息通常提供有关插件和测试类型的信息。 以下列出了可用的信息类型。

类型	说明
id	所有已安装插件中唯一识别此类型的内部字符串。

类型	说明
label	此类型的选项列表和编辑对话框中显示的 GUI 文本。
description	介绍此测试类型的其他文本信息。
version	区分一种类型的不同版本。

一般属性元信息

对于可编辑属性，除特定类型信息外，每个属性类型还存在相同的一般信息。 属性名称必须与通过 get<<propertyname>> 方法在代码中定义的名称匹配，第一个字符应为小写字母。

类型	说明
label	GUI 中显示的文本。
description	说明属性的其他文本信息。
isOptional	如果值为真，表明无需用户输入。
default	创建新测试时属性的默认值。 此值必须与属性类型匹配。

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类型。

类型	说明
maxLength	表示用户可以输入的最大长度。
editMultiLine	表示编辑字段是否应有多行。
isPassword	true 的值将用户输入隐藏为 ***。

文件属性元信息

此处提供插件中可用的文件属性元信息的类型。

对于指定的文件值，执行服务器上具有绝对路径的非版本控制文件应与版本控制文件区分开来。 源代码管理文件始终相对于测试容器的根目录，通常用于指定要执行的测试源代码。 执行服务器上需存在绝对路径，通常指定工具或资源，用于调用执行服务器上的测试。

类型	说明
openBrowser	如果值为真，则表明应打开浏览器从测试容器中的文件里选择文件。
isSourceControl	如果值为真，则表明文件源于源代码管理系统。
fileNameSpec	指示标准 Windows 文件浏览器对话框中已知的允许文件名的限制。

自定义图标

您可为测试类型设计自定义图标，以使测试类型可识别。 要为已在配置 XML 文件中定义标识符 PluginId 的插件定义这些图标，您必须将以下四种图标放入插件 ZIP 容器的根目录中。

名称	说明
<PluginId>.gif	测试类型的默认图标。 例如 ProcessExecutor.gif。

名称	说明
`<PluginId>_package.gif`	如果您将具有指定测试类型的测试转换为测试包，该图标将用于测试包根目录和套件节点。 例如 `ProcessExecutor_package.gif`。
`<PluginId>_linked.gif`	如果测试的父文件夹是已链接的测试容器，则使用该图标。 例如 `ProcessExecutor_linked.gif`。
`<PluginId>_incomplete.gif`	如果未定义测试的父测试容器的产品或源代码管理配置文件，则使用该图标。

为测试类型设计新图标时，请应用以下规则：

- 仅使用 GIF 类型的图标。 文件扩展名区分大小写，并且必须始终为小写字母 (.gif)。
- 从 `<Silk Central deploy folder>\wwwroot\silkroot\img\PluginIcons` 删除旧图标或无效图标，否则将不会使用插件 ZIP 容器根目录中的新图标更新这些图标。
- 图标的大小为 16x16 像素。
- 图标允许的最大颜色数量为 256。
- 图标包括 1 位透明度。

部署

插件 ZIP 存档必须位于 Silk Central 安装目录的 `Plugins` 子目录中。 要集成此目录中的插件，请通过 Silk Central Service Manager 重新启动应用程序服务器和前端服务器。

注: 不支持热部署。

当每次修改存档时，必须重新启动这两个服务器。 该存档将自动上载到执行服务器。

注: 请勿在基于已创建的插件进行测试后删除插件。 基于不再存在的插件存档的测试将导致在更改或执行时出现未知错误。

为视频捕获表示开始和完成

有关可用 Java 类和方法的完整详细信息，请参阅 *Javadoc*。如果链接无效，请单击 Silk Central 菜单中的**帮助 > 文档 > Silk Central API 规范**以打开 Javadoc。

当为 Silk Central 创建新的第三方测试插件时，如果第三方测试类型支持在单个测试执行中处理多个测试用例，且您要将捕获的视频关联到特定测试用例，则可按两种方式进行操作。

插件中运行的第三方测试

对于这些测试，建议使用 `TestLaunchResultDrain` 类的 `indicateTestStart` 和 `indicateTestStop` 方法。

外部进程中运行的第三方测试

对于这些测试，可使用基于 TCP/IP 的服务向 Silk Central 执行服务器的端口发送 `START` 和 `FINISH` 消息。要使用的端口号可通过插件中的 *ExecutionContextInfo.ExecProperty#PORT_TESTCASE_START_FINISH* 查询。如果插件扩展了 `ExtProcessTestLaunchBean`，则端口还能被用作称为 *#sctm_portTestCaseStartFinish* 的环境变量。这些消息类型将通知执行服务器，测试中的测试用例已开始或分别完成。消息必须以 Unicode (UTF8) 或 ASCII 格式编码。

消息类型	格式
开始	`START <Test Name>, <Test ID> <LF>`，其中，LF 的 ASCII 代码为 10。

消息类型	格式
完成	`FINISH <Test Name>, <Test ID>, <Passed> LF`，其中，LF 的 ASCII 代码为 10。`Passed` 可以是 `True`，也可以是 `False`。如果视频捕获设置为在出错时执行，当 `Passed` 设置为 `False` 时，视频将仅保存到结果。

如果请求被识别，执行服务器将会以确定作出响应，否则，执行服务器将会以错误消息作出响应。请始终等待执行服务器响应，然后再执行下一个测试用例，因为如果不按此操作，录制的视频可能与实际测试用例不匹配。

如果执行测试的外部进程基于 Java 环境，建议使用包括在文件 `tm-testlaunchapi.jar` 中的 `TestCaseStartFinishSocketClient` 类的 `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务提供程序集成。云配置文件基于插件概念，让您能为特定云提供程序编写自己的插件。云提供程序插件将在每次自动测试运行之前部署虚拟环境。

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，升级到将来版本的 Silk Central 后可能需要更新实施。

有关可用 Java 类和方法的完整详细信息，请参阅 *Javadoc*。如果链接无效，请单击 Silk Central 菜单中的**帮助 > 文档 > Silk Central API 规范**以打开 Javadoc。

首先应使用的基本接口是 `CloudProviderProfile` (com.segue.scc.published.api.cloud)。`CloudProviderProfile` 指定外部云系统的访问和控制。云提供程序插件实施需要执行以下步骤：

- 公开云提供程序配置文件中必须配置哪些属性，以远程访问此类提供程序
- 验证配置文件属性，检查与云提供程序的连接
- 从云提供程序检索可用映像模板列表
- 部署基于选定映像模板的虚拟环境，公开可从外部访问的主机地址
- 检查虚拟环境是否已配置和运行
- 删除特定虚拟环境

Silk Central Web 服务

Silk Central 基于 SOAP 的 Web 服务无需安装，默认情况下在每个前端服务器上启用。例如，如果 `http://www.yourFrontend.com/login` 是您用于访问 Silk Central 的 URL，则 `http://www.yourFrontend.com/Services1.0/jaxws/system`（原有服务位于 `http://www.yourFrontend.com/Services1.0/services`）和 `http://www.yourFrontend.com/AlmServices1.0/services` 是您用于访问可用 Web 服务的基本 URL。

使用浏览器访问基本 URL 时，将会向您提供所有可用 Web 服务的简单 HTML 列表。此列表由 JAX-WS 提供。Silk Central 使用的 SOAP 堆栈是 *https://jax-ws.java.net/*。

基本 URL 提供到 WSDL（Web 服务定义语言）标准化 XML 文件的链接，此处每个文件均介绍单个 Web 服务的接口。不可人工读取这些文件。因此，启用 SOAP 的客户端（例如 Silk Performer Java Explorer）将读取 WSDL 文件，从而检索调用相应 Web 服务的方法所需的信息。

有关可用 Java 类和方法的完整详细信息，请参阅 *Javadoc*。如果链接无效，请单击 Silk Central 菜单中的**帮助 > 文档 > Silk Central API 规范**以打开 Javadoc。

有关如何管理在外部执行环境而不是在 Silk Central 执行服务器上执行的执行计划运行及其结果的信息，Silk Central 提供了 REST API 文档。如果系统上安装了 Silk Central，则可以从*此处*访问 REST API 的交互式文档。

Web 服务快速启动

本部分包括与 Web 服务集成有关的先决条件、用例示例和其他主题。

先决条件

尝试开发 Web 服务客户端前必须考虑以下先决条件：

- 面向对象编程 (OOP) 的基础知识。 拥有 Java 经验会有帮助，因为示例将以此语言提供。 没有 Java 经验但有 C++、C#、Python 或 Perl 经验的开发人员仍可以轻松按照示例操作。 拥有操作 HashMap 和 List 集合经验尤佳。
- 简述 JUnit 测试。 不要求 Java 测试框架，但如果了解会有帮助。
- 介绍 Web 服务技术。 本帮助不提供 Web 服务或 SOAP 的入门课程。 您至少应拥有编码和成功运行“Hello World!” Web 服务客户端的经验。
- Silk Central Web 服务结构的知识。

Web 服务使用入门

请参阅 *Silk Central 发行说明*，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* 下载 `JUnit.jar` 文件。

1. 确保 Java SDK 的 `bin` 目录位于您的 `PATH` 中。
2. 运行以下命令，指向所需 Web 服务的 WSDL。

   ```
   wsimport -s <所生成文件的保存路径> -Xnocompile
   -p <要用于您的客户端 yourWebService 的程序包结构>
   http://<您的服务 URL>/yourWebService?wsdl
   ```
3. 查找自动生成的 Java 类 `YourWebService`。

   此类已准备就绪，可使用 `YourWebService` 提供的所有方法。

Web 服务客户端概述

Web 服务通常通过 HTTP 协议使用 SOAP。在此情况下,将发送 SOAP 信封。集合和其他复杂对象在 SOAP 信封中捆绑时，ASCII 数据结构会变得难以读取和编辑。初级开发人员不应尝试直接操作 SOAP 信封来构建 Web 服务客户端。经验丰富的开发人员通常不会在 SOAP 信封级别构建 Web 服务客户端。这样操作很枯燥，且容易出错。因此，所有主要的编程语言均提供 Web 服务开发包。在 Silk Central 中，*Java API for XML Web Services (JAX-WS)* 用于构建使用 SOAP 消息进行通信的 Web 服务和客户端。

无论是哪一种实施语言（Java、C++、C#、Perl 或 Python），构建 Web 服务客户端通常遵循相同的模式：

1. 指定 Web 服务 WSDL 的开发工具包工具。
2. 返回客户端存根。
3. 编辑步骤 2 中生成的客户端存根以获取成熟的客户端。

JAX-WS 遵循这种模式。在此，wsimport 工具（与 JDK 捆绑在一起）用于从 WSDL 构建客户端存根。可以在 *JDK 工具和实用程序文档*中找到有关如何使用 wsimport 的详细信息。 A brief description of the switches used in the above summary is as follows:

- `-s`：客户端存根的输出目录
- `-p`：目标包。部署至此包结构

示例：`wsimport -s <生成的存根位置> -p <目标包> <WSDL>`

wsimport 工具生成多种类，以支持 Web 服务的客户端。如果 `YourWebService` 是服务的名称，则可以预期以下类别输出：

- `YourWebService`：表示 `YourWebService` 的接口。

- `YourWebServiceService`：生成的类，表示 `YourWebService` 定位器，例如，用于获取可用端口（服务端点接口）列表
- `WSFaultException`：从 `wsdl:fault` 映射的异常类
- `WsFaultBean`：从响应 `wsdl:message` 派生的异步响应 Bean
- `Serializeable Objects`：与对象 `YourWebService` 使用相应的客户端对象。

要生成 JAX-WS 客户端以访问 Silk Central Web 服务，请创建一个名为 `YourWebServiceClient` 的新 Java 类。必须通过一个端口绑定到 Web 服务；端口是本地对象，用作远程服务代理。请注意，端口是通过在上一步中执行 wsimport 工具创建的。要检索此代理，请对服务调用 `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 服务进行身份验证，请在 Silk Central UI 的**用户设置**页面中生成 Web 服务令牌。要访问此页面，请将鼠标光标悬停在 Silk Central 菜单中的用户名上，然后选择**用户设置**。

您还可以通过传递用户名和密码来调用服务的 `logonUser` 方法，以获取会话 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

作为本节前面详述步骤的后续步骤，本主题将完成将需求添加到 Silk Central 的使用案例。

在继续操作之前，必须满足以下前提条件：

- 您已完成针对 `requirements` Web 服务详述的步骤。
- 已创建采用绑定和登录方法的工作 POJO 或 JUnit 类。
- 您已经熟悉了其他 Silk Central API 帮助主题。

1. 在**用户设置**页面中生成 Web 服务令牌。
   a) 单击 Silk Central 菜单中的用户名。此时将打开**用户设置**页面。
   b) 在页面的 **Web 服务令牌**部分，单击**生成令牌**。
2. 构建包含所需数据的需求对象。
3. 使用已生成的 Web 服务令牌、项目 ID 和需求对象来调用 `updateRequirement` 方法。
4. 保存由 `updateRequirement` 方法返回的需求 ID。
5. 创建需求属性的 `PropertyValue` 数组。
6. 使用之前创建的数组调用 `updateProperties` 方法。

`wsimport` 将创建上述 Web 服务对象：

- `Requirement`
- `PropertyValue`

您现在可以通过上述对象的 OOP 方法来使用 Web 服务。无需使用 SOAP 信封结构。以下是完成此使用案例所需代码的摘要。

```
/** 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: 此示例代码也适用于 Web Service Demo Client 的类 `com.microfocus.silkcentral.democlient.samples.AddingRequirement`。

Web 服务身份验证

Silk Central 数据受到针对非授权访问的保护。在授权数据访问权限之前，必须先提供登录凭据。这不仅适用于使用 HTML 前端的情况，也适用于通过 SOAP 或 REST API 调用与 Silk Central 进行通信。

查询数据或应用 Silk Central 配置更改的第一步是身份验证。身份验证成功时，将创建用户会话，以允许在用户登录的上下文中执行后续操作。

通过 Web 浏览器访问 Silk Central 时，用户看不到会话信息。浏览器使用 cookie 处理会话信息。与通过 HTML 使用 Silk Central 相比，SOAP 调用必须手动处理会话信息。

Micro Focus 建议通过 Web 服务令牌进行身份验证。您可以在 Silk Central UI 的**用户设置**页面中生成这样的 Web 服务令牌。要访问此页面，请将鼠标光标悬停在 Silk Central 菜单中的用户名上，然后选择**用户设置**。

您还可以使用 `logonUser` SOAP 调用或 `login` REST API 调用进行身份验证。方法调用返回的会话标识符引用服务器上创建的会话，同时用作会话上下文中访问 Silk Central 的密匙。

需要身份验证的每个后续 API 调用都将这样的 Web 服务令牌或会话标识符作为其参数之一，检查其有效性，并在相应会话的上下文中执行。

通过 Web 服务创建的 Silk Central 会话无法显式结束。相反，会话在一段时间不使用时自动结束。服务器上的会话一旦超时，尝试使用会话的后续 SOAP 调用将抛出异常。

通过 Silk Central 的**帮助 > 工具 > Web 服务演示客户端**可以找到可供下载的演示客户端。此演示项目使用 Silk Central`tests` Web 服务，以帮助您熟悉 Web 服务接口。

示例

如果您已在 Silk Central UI 中生成 Web 服务令牌，以下 Java 代码示例将演示如何通过 Web 服务和使用 Web 服务令牌访问 Silk Central：

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

以下 Java 代码示例演示了如何通过 Web 服务和使用会话标识符同样访问 Silk Central：

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

可用 Web 服务

下表列出了可用 Silk Central Web 服务。有关通过基于 HTTP 的接口访问 Silk Central 的信息，请参阅*服务交换*。有关使您能够在外部执行环境中计划和执行执行计划运行的基于 REST API 的 Web 服务的信息，请参阅 *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 还列出了不用于创建 Web 服务客户端的系统内部 Web 服务。本文档只介绍已发布的 Web 服务。

有关可用 Java 类和方法的完整详细信息，请参阅 *Javadoc*。如果链接无效，请单击 Silk Central 菜单中的**帮助 > 文档 > Silk Central API 规范**以打开 Javadoc。

注: 使用 Web 服务时，系统返回的所有时间均为协调世界时 (UTC)。在您使用的 Web 服务的所有时间引用中也使用 UTC。

WS 名称（接口）	WSDL URL	说明
system (SystemService)	`/Services1.0/jaxws/system?wsdl`	这是提供身份验证和简单实用工具方法的根服务。
administration (AdministrationService)	`/Services1.0/jaxws/administration?wsdl`	此服务提供对 Silk Central 的*项目*和*产品*实体的访问。
requirements (RequirementsService)	`/Services1.0/jaxws/requirements?wsdl`	此服务提供对 Silk Central 的**需求**区域的访问。
tests (TestsService)	`/Services1.0/jaxws/tests?wsdl`	此服务提供对 Silk Central 的**测试**区域的访问。
executionplanning (ExecutionPlanningService)	`/Services1.0/jaxws/executionplanning?wsdl`	此服务提供对 Silk Central 的**执行计划**区域的访问。
filter (FilterService)	`/Services1.0/jaxws/filter?wsdl`	此服务允许您创建、读取、更新和删除筛选器。
issuemanager (IssueManagerService)	`/Services1.0/jaxws/issuemanager?wsdl`	此服务提供对 Issue Manager 的访问。

服务交换

本部分介绍基于 HTTP 的接口，用于处理 Services Exchange 中的报告、附件、测试计划、执行和库。

您还可以通过 Web 服务访问 Silk Central。有关其他信息，请参阅 *可用 Web 服务*。

注: 使用 Web 服务时，系统返回的所有时间均为协调世界时 (UTC)。在您使用的 Web 服务的所有时间引用中也使用 UTC。

reportData 接口

`reportData` 接口用于请求报告的数据。下表显示了 `reportData` 接口的参数：

接口 URL	参数	说明
`http://<front-end URL>/servicesExchange?hid=reportData`	`sid`	用于用户身份验证的 Web 服务令牌或会话标识符。您可以在 Silk Central UI 的**设置页面**中生成 Web 服务令牌。要访问此页面，请将鼠标光标悬停在 Silk Central 菜单中的用户名上，然后选择**用户设置**。您可以通过调用 *可用 Web 服务*之一的 `logonUser` 方法来检索会话标识符。
	`reportFilterID`	报告筛选器 ID
	`type`	响应正文格式： (csv 或 xml)
	`includeHeaders`	是否包括报告页眉。 (`true` 或 `false`)
	`projectID`	项目 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);
```

如果报告需要参数，您需要将以下代码添加到每个参数的报告 URL：

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

在示例中，参数 `rp_execNode_Id_0` 设置为值 `1`。

注: 传递给 `reportData` 服务的参数名称必须使用 `rp_` 前缀。示例：`/servicesExchange?hid=reportData&type=xml&sid=<...>&reportFilterID=<...>&projectID=<...>&rp_TestID=<...`

TMAttach 接口

`TMAttach` 接口用于将附件上载到测试或需求。下表显示了 `TMAttach` 接口的参数。

接口 URL	参数	说明
`http://<front-end URL>/servicesExchange?hid=TMAttach`	`sid`	用于用户身份验证的 Web 服务令牌或会话标识符。您可以在 Silk Central UI 的**设置页面**中生成 Web 服务令牌。要访问此页面，请将鼠标光标悬停在 Silk Central 菜单中的用户名上，然后选择**用户设置**。您可以通过调用*可用 Web 服务*之一的 `logonUser` 方法来检索会话标识符。
	`entityType`	目标实体类型： （测试、需求或 TestStepParent）
	`entityID`	目标实体 ID： （测试 ID、需求 ID 或手动测试 ID）
	`description`	附件说明： URL 编码文本，用于描述附件。

接口 URL	参数	说明
	isURL	如果为 true，则附件为 URL。如果为 false，则附件为文件。
	URL	可选 - 要附加的 URL。
	stepPosition	可选 - 测试步骤顺序。标识手动测试的步骤（例如，顺序第一步是 1）。如果 entityType 是 TestStepParent，则必须遵循顺序。

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 服务示例

以下代码使用 Apache HtmlClient 获取便捷的 HTTP-POST API，以便上载二进制附件。每个请求仅可上载一个附件。

每个请求仅可上载一个附件。要下载 Apache HttpComponents，请访问 *http://hc.apache.org/downloads.cgi*。请参阅组件文档，了解所需的库。

```
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 接口用于创建新测试。调用的 HTTP 响应包含已更改测试的 XML 结构。您可以从更新后的 XML 测试结构中获取新节点的标识符。

下表显示了 createTestPlan 接口的参数。

接口 URL	参数	说明
`http://<front-end URL>/servicesExchange?hid=createTestPlan`	`sid`	用于用户身份验证的 Web 服务令牌或会话标识符。您可以在 Silk Central UI 的**设置页面**中生成 Web 服务令牌。要访问此页面，请将鼠标光标悬停在 Silk Central 菜单中的用户名上，然后选择**用户设置**。您可以通过调用 *可用 Web 服务*之一的 `logonUser` 方法来检索会话标识符。
	`parentNodeID`	**测试**树中要添加新测试的容器的 ID

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

XML 架构定义文件，用于验证测试计划可使用前端服务器 URL `http://<前端服务器 URL>/silkroot/xsl/testplan.xsd` 下载，或是从前端服务器安装文件夹 `<Silk Central installation folder>/wwwroot/silkroot/xsl/testplan.xsd` 复制。

createTestPlan Web 服务示例

以下代码使用 Apache HttpClient 创建测试。

```
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().setConnectionTimeo
ut(60000);
int status = client.executeMethod(filePost);
System.out.println(filePost.getStatusLine());
```

每个请求仅可上载一个附件。要下载 Apache HttpComponents，请访问 *http://hc.apache.org/downloads.cgi*。请参阅组件文档，了解所需的库。

测试示例

以下代码显示可通过 `createTestPlan` 和 `updateTestPlan` 服务上载至 Silk Central 的示例测试。

```
<?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 接口用于将测试计划导出为 XML 文件。下表显示了 exportTestPlan 接口的参数。

接口 URL	参数	说明
http://<front-end URL>/servicesExchange?hid=exportTestPlan	sid	用于用户身份验证的 Web 服务令牌或会话标识符。您可以在 Silk Central UI 的**设置页面**中生成 Web 服务令牌。要访问此页面，请将鼠标光标悬停在 Silk Central 菜单中的用户名上，然后选择**用户设置**。您可以通过调用 *可用 Web 服务*之一的 logonUser 方法来检索会话标识符。
	nodeID	具有此 ID 的节点以及此节点下的所有递归子节点都将被导出

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

exportTestPlan Web 服务示例

以下代码使用 Apache HttpClient 来导出测试。

```
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().setConnectionTimeo
ut(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);
```

要下载 Apache HttpComponents，请访问 *http://hc.apache.org/downloads.cgi*。请参阅组件文档，了解所需的库。

updateTestPlan 接口

`updateTestPlan` 接口用于通过 XML 文件中的现有根节点更新测试。调用的 HTTP 响应包含已更改测试的 XML 结构。您可以从更新后的 XML 测试结构中获取新节点的标识符。

下表显示了 `updateTestPlan` 接口的参数。

接口 URL	参数	说明
`http://<front-end URL>/servicesExchange?hid=updateTestPlan`	`sid`	用于用户身份验证的 Web 服务令牌或会话标识符。您可以在 Silk Central UI 的**设置页面**中生成 Web 服务令牌。要访问此页面，请将鼠标光标悬停在 Silk Central 菜单中的用户名上，然后选择**用户设置**。您可以通过调用 *可用 Web 服务*之一的 `logonUser` 方法来检索会话标识符。

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

XML 架构定义文件，用于验证测试计划可使用前端服务器 URL `http://<前端服务器 URL>/silkroot/xsl/testplan.xsd` 下载，或是从前端服务器安装文件夹 `<Silk Central installation folder>/wwwroot/silkroot/xsl/testplan.xsd` 复制。

updateTestPlan Web 服务示例

以下代码使用 Apache HttpClient 更新测试。

```
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().setConnectionTimeo
ut(60000);
int status = client.executeMethod(filePost);
System.out.println(filePost.getStatusLine());

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

每个请求仅可上载一个附件。要下载 Apache HttpComponents，请访问 *http://hc.apache.org/downloads.cgi*。请参阅组件文档，了解所需的库。

测试示例

以下代码显示可通过 `createTestPlan` 和 `updateTestPlan` 服务上载至 Silk Central 的示例测试。

```
<?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 需求结构中获取新节点的标识符。

下表显示了 `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 服务令牌或会话标识符。您可以在 Silk Central UI 的**设置页面**中生成 Web 服务令牌。要访问此页面，请将鼠标光标悬停在 Silk

接口 URL	参数	说明
		Central 菜单中的用户名上，然后选择**用户设置**。您可以通过调用*可用 Web 服务*之一的 `logonUser` 方法来检索会话标识符。
	`parentNodeID`	需求树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>`

用于验证需求的 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().setConnectionTimeo
ut(60000);
int status = client.executeMethod(filePost);
System.out.println(filePost.getStatusLine());
```

每个请求仅可上载一个附件。要下载 Apache HttpComponents，请访问 *http://hc.apache.org/downloads.cgi*。请参阅组件文档，了解所需的库。

需求示例

以下代码显示可通过 `createRequirements`、`updateRequirements` 和 `updateRequirementsByExtID` 服务上载至 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` 接口用于将需求导出为 XML 文件。下表显示了 `exportRequirements` 接口的参数。

接口 URL	参数	说明
`http://<front-end URL>/servicesExchange?hid=exportRequirements`	`sid`	用于用户身份验证的 Web 服务令牌或会话标识符。您可以在 Silk Central UI 的**设置页面**中生成 Web 服务令牌。要访问此页面，请将鼠标光标悬停在 Silk Central 菜单中的用户名上，然后选择**用户设置**。您可以通过调用 *可用 Web 服务*之一的 `logonUser` 方法来检索会话标识符。
	`nodeID`	具有此 ID 的节点以及此节点下的所有递归子节点都将被导出
	`includeObsolete`	*可选*：指定 `true` 或 `false`。如果忽略，则默认为 `true`。指定 `false` 以排除过时的需求。

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

exportRequirements Web 服务示例

以下代码使用 Apache HttpClient 来导出需求。

```
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().setConnectionTimeo
ut(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);
```

要下载 Apache HttpComponents，请访问 *http://hc.apache.org/downloads.cgi*。请参阅组件文档，了解所需的库。

updateRequirements 接口

`updateRequirements` 接口用于通过 XML 文件中的现有根节点更新需求。需求由需求树中的内部 Silk Central 节点 ID 标识。更新需求树节点和所有节点的子项。添加新节点，将丢失的节点设置为过时，移动的节点也同样在 Silk Central 中移动。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 需求结构中获取新节点的标识符。

下表显示了 `updateRequirements` 接口的参数。

接口 URL	参数	说明
`http://<front-end URL>/servicesExchange?hid=updateRequirements`	`sid`	用于用户身份验证的 Web 服务令牌或会话标识符。您可以在 Silk Central UI 的**设置页面**中生成 Web 服务令牌。要访问此页面，请将鼠标光标悬停在 Silk Central 菜单中的用户名上，然后选择**用户设置**。您可以通过调用 *可用 Web 服务*之一的 `logonUser` 方法来检索会话标识符。

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

用于验证需求的 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。

updateRequirements Web 服务示例

以下代码使用 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().setConnectionTimeo
ut(60000);
int status = client.executeMethod(filePost);
System.out.println(filePost.getStatusLine());

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

每个请求仅可上载一个附件。要下载 Apache HttpComponents，请访问 *http://hc.apache.org/downloads.cgi*。请参阅组件文档，了解所需的库。

需求示例

以下代码显示可通过 `createRequirements`、`updateRequirements` 和 `updateRequirementsByExtID` 服务上载至 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` 接口用于通过 XML 文件中的现有根节点更新需求。需求由外部 ID 标识。更新需求树节点和所有节点的子项。添加新节点，将丢失的节点设置为过时，移动的节点也同样在 Silk Central 中移动。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 需求结构中获取新节点的标识符。

下表显示了 `updateRequirementsByExtID` 接口的参数。

接口 URL	参数	说明
`http://<front-end URL>/servicesExchange?hid=updateRequirementsByExtID`	`sid`	用于用户身份验证的 Web 服务令牌或会话标识符。您可以在 Silk Central UI 的**设置页面**中生成 Web 服务令牌。要访问此页面，请将鼠标光标悬停在 Silk

接口 URL	参数	说明
		Central 菜单中的用户名上，然后选择**用户设置**。您可以通过调用*可用 Web 服务*之一的 `logonUser` 方法来检索会话标识符。
	`nodeID`	需求树中要更新的节点的 ID。

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

用于验证需求的 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。

updateRequirementsByExtID Web 服务示例

以下代码使用 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().setConnectionTimeo
ut(60000);
int status = client.executeMethod(filePost);
System.out.println(filePost.getStatusLine());

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

每个请求仅可上载一个附件。要下载 Apache HttpComponents，请访问 *http://hc.apache.org/downloads.cgi*。请参阅组件文档，了解所需的库。

需求示例

以下代码显示可通过 `createRequirements`、`updateRequirements` 和 `updateRequirementsByExtID` 服务上载至 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 接口

createExecutionDefinitions 接口用于创建新执行计划。调用的 HTTP 响应包含更改后的执行计划的 XML 结构。您可以从更新 XML 执行计划结构中获取新节点的标识符。

下表显示了 createExecutionDefinitions 接口的参数。

接口 URL	参数	说明
http://<front-end URL>/servicesExchange?hid=createExecutionDefinitions	sid	用于用户身份验证的 Web 服务令牌或会话标识符。您可以在 Silk Central UI 的**设置页面**中生成 Web 服务令牌。要访问此页面，请将鼠标光标悬停在 Silk Central 菜单中的用户名上，然后选择**用户设置**。您可以通过调用 *可用 Web 服务*之一的 logonUser 方法来检索会话标识符。
	parentNodeID	执行计划树中要添加新执行计划的节点的 ID

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

用于验证执行的 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/executionplan.xsd 下载，或从前端服务器安装文件夹 <Silk Central installation folder>/wwwroot/silkroot/xsl/executionplan.xsd 复制。

createExecutionDefinitions Web 服务示例

以下代码使用 Apache HttpClient 创建执行计划。

```
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().setConnectionTimeo
ut(60000);
int status = client.executeMethod(filePost);
System.out.println(filePost.getStatusLine());
```

每个请求仅可上载一个附件。要下载 Apache HttpComponents，请访问 *http://hc.apache.org/downloads.cgi*。请参阅组件文档，了解所需的库。

执行计划示例

以下代码显示可通过 `createExecutionDefinitions` 和 `updateExecutionDefinitions` 服务上载至 Silk Central 的示例执行计划。该示例为其中一个执行定义创建自定义计划，并通过手动分配和筛选器将测试分配到执行计划。该示例还会创建一个包含这些配置的配置套件。

```
<?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` 接口用于将执行计划导出为 XML 文件。下表显示了 `exportExecutionDefinitions` 接口的参数。

接口 URL	参数	说明
`http://<front-end URL>/servicesExchange?hid=exportExecutionDefinitions`	`sid`	用于用户身份验证的 Web 服务令牌或会话标识符。您可以在 Silk Central UI 的**设置页面**中生成 Web 服务令牌。要访问此页面，请将鼠标光标悬停在 Silk Central 菜单中的用户名上，然后选择**用户设置**。您可以通过调用 *可用 Web 服务*之一的 `logonUser` 方法来检索会话标识符。
	`nodeID`	具有此 ID 的节点以及此节点下的所有递归子节点都将被导出

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

exportExecutionDefinitions Web 服务示例

以下代码使用 Apache HttpClient 来导出执行计划。

```
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().setConnectionTimeo
ut(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);
```

要下载 Apache HttpComponents，请访问 *http://hc.apache.org/downloads.cgi*。请参阅组件文档，了解所需的库。

updateExecutionDefinitions 接口

`updateExecutionDefinitions` 接口用于从 XML 文件更新执行计划。调用的 HTTP 响应包含更改后的执行计划的 XML 结构。您可以从更新 XML 执行计划结构中获取新节点的标识符。

下表显示了 `updateExecutionDefinitions` 接口的参数。

接口 URL	参数	说明
`http://<front-end URL>/servicesExchange?hid=updateExecutionDefinitions`	`sid`	用于用户身份验证的 Web 服务令牌或会话标识符。您可以在 Silk Central UI 的**设置页面**中生成 Web 服务令牌。要访问此页面，请将鼠标光标悬停在 Silk Central 菜单中的用户名上，然后选择**用户设置**。您可以通过调用 *可用 Web 服务*之一的 `logonUser` 方法来检索会话标识符。

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

用于验证执行的 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/executionplan.xsd` 下载，或从前端服务器安装文件夹 `<Silk Central installation folder>/wwwroot/silkroot/xsl/executionplan.xsd` 复制。

updateExecutionDefinitions Web 服务示例

以下代码使用 Apache HttpClient 更新执行计划。

```
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().setConnectionTimeo
ut(60000);
int status = client.executeMethod(filePost);
System.out.println(filePost.getStatusLine());

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

每个请求仅可上载一个附件。要下载 Apache HttpComponents，请访问 *http://hc.apache.org/downloads.cgi*。请参阅组件文档，了解所需的库。

执行计划示例

以下代码显示可通过 `createExecutionDefinitions` 和 `updateExecutionDefinitions` 服务上载至 Silk Central 的示例执行计划。该示例为其中一个执行定义创建自定义计划，并通过手动分配和筛选器将测试分配到执行计划。该示例还会创建一个包含这些配置的配置套件。

```
<?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 接口用于创建新库。调用的 HTTP 响应包含已更改库的 XML 结构。您可从已更新的 XML 库结构中获取新节点的标识符。

下表显示了 createLibraries 接口的参数。

接口 URL	参数	说明
http://<front-end URL>/servicesExchange?hid=createLibraries	sid	用于用户身份验证的 Web 服务令牌或会话标识符。您可以在 Silk Central UI 的**设置页面**中生成 Web 服务令牌。要访问此页面，请将鼠标光标悬停在 Silk Central 菜单中的用户名上，然后选择**用户设置**。您可以通过调用 *可用 Web 服务*之一的 logonUser 方法来检索会话标识符。

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

用于验证库的 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/libraries.xsd 下载或从前端服务器安装文件夹 <Silk Central installation folder>/wwwroot/silkroot/xsl/libraries.xsd 复制。

createLibraries Web 服务示例

以下代码使用 Apache HttpClient 创建库。

```
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().setConnectionTimeo
ut(60000);
int status = client.executeMethod(filePost);
System.out.println(filePost.getStatusLine());
```

要下载 Apache HttpComponents，请访问 *http://hc.apache.org/downloads.cgi*。请参阅组件文档，了解所需的库。

库示例

以下代码显示可通过 `createLibraries` 服务上载至 Silk Central 的示例库。除非在 `GrantedProjects` 部分中定义了一个或多个项目，否则新库并不仅限于在某些项目中使用。

```
<?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` 接口用于将库、文件夹和共享步骤对象导出为 XML 文件。下表显示了 `exportLibraryStructure` 接口的参数。

接口 URL	参数	说明
`http://<front-end URL>/servicesExchange?hid=exportLibraryStructure`	`sid`	用于用户身份验证的 Web 服务令牌或会话标识符。您可以在 Silk Central UI 的**设置页面**中生成 Web 服务令牌。要访问此页面，请将鼠标光标悬停在 Silk Central 菜单中的用户名上，然后选择**用户设置**。您可以通过调用*可用 Web 服务*之一的 `logonUser` 方法来检索会话标识符。
	`nodeID`	要导出的库树中的库节点或文件夹。不允许共享步骤节点的 ID。

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

exportLibraryStructure Web 服务示例

以下代码使用 Apache HttpClient 来导出库。

```
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().setConnectionTimeo
ut(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);
```

要下载 Apache HttpComponents，请访问 *http://hc.apache.org/downloads.cgi*。请参阅组件文档，了解所需的库。

exportLibraryStructureWithoutSteps 接口

`exportLibraryStructureWithoutSteps` 接口用于将库、文件夹和共享步骤对象导出为 XML 文件。包括在共享步骤对象中的步骤未被导出。下表显示了 `exportLibraryStructureWithoutSteps` 接口的参数。

接口 URL	参数	说明
`http://<front-end URL>/servicesExchange?hid=exportLibraryStructureWithoutSteps`	`sid`	用于用户身份验证的 Web 服务令牌或会话标识符。您可以在 Silk Central UI 的**设置页面**中生成 Web 服务令牌。要访问此页面，请将鼠标光标悬停在 Silk Central 菜单中的用户名上，然后选择**用户设置**。您可以通过调用*可用 Web 服务*之一的 `logonUser` 方法来检索会话标识符。

接口 URL	参数	说明
	nodeID	要导出的库树中的库节点或文件夹。不允许共享步骤节点的 ID。

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

exportLibraryStructureWithoutSteps Web 服务示例

以下代码使用 Apache HttpClient 来导出库。

```
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().setConnectionTimeo
ut(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);
```

要下载 Apache HttpComponents，请访问 *http://hc.apache.org/downloads.cgi*。请参阅组件文档，了解所需的库。

getLibraryInfoByName 接口

`getLibraryInfoByName` 接口将返回所有具有指定名称的库的 ID、名称和说明。界面仅返回库的属性，未返回其结构。下表显示了 `getLibraryInfoByName` 接口的参数。

接口 URL	参数	说明
`http://<front-end URL>/servicesExchange?hid=getLibraryInfoByName`	sid	用于用户身份验证的 Web 服务令牌或会话标识符。您可以在 Silk Central UI 的**设置页面**中生成 Web 服务令牌。要访问此页面，请将鼠标光标悬停在 Silk Central 菜单中的用户名上，然后选择**用户设置**。您可以通过调用*可用 Web 服务*之一的 `logonUser` 方法来检索会话标识符。
	libraryName	库的名称

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

getLibraryInfoByName Web 服务示例

以下代码使用 Apache HttpClient 获取库信息。

```
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().setConnectionTimeo
ut(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);
```

要下载 Apache HttpComponents，请访问 *http://hc.apache.org/downloads.cgi*。请参阅组件文档，了解所需的库。

Web Service Demo Client

Web Service Demo Client 是一种演示如何使用 Silk Central Web 服务的工具。 从**帮助 > 工具**中下载客户端。

Web Service Demo Client 显示每个测试在**测试 > 管理测试属性**中可用的所有属性，以及每个可用测试类型的所有属性。

注意: Web Service Demo Client 专为演示 Web 服务的使用而设计。 请勿在生产环境下使用演示客户端。

从 CI 服务器触发 Silk Central

此部分介绍如何通过从 CI 服务器使用 Gradle 脚本触发 Silk Central 上的执行，更好地将 Silk Central 集成到持续集成 (CI) 过程。

此外，本部分还介绍如何从 Silk Central 获取结果以及如何在构建过程中使用这些结果。

要从 CI 服务器触发 Silk Central 上的执行并从 Silk Central 收集执行结果，您需要将一个带有适当命令的 Gradle 脚本添加到源代码管理。您可从 Silk Central UI 下载 `silkcentral.gradle` 文件。导航到**帮助 > 工具**，然后单击**用于 CI 服务器集成的 Gradle 脚本**。

您可以在 Gradle 脚本中配置以下属性：

属性	说明
`sc_executionNodeIds`	要启动的执行计划的逗号分隔列表。该列表不应包含文件夹。例如 `22431,22432,22433`。
`sc_host`	Silk Central 主机。例如 `http://[sc_server_name]:19120`。
`sc_token`	用于用户身份验证的 Web 服务令牌。您可以在 Silk Central UI 的**设置页面**中生成 Web 服务令牌。要访问此页面，请将鼠标光标悬停在 Silk Central 菜单中的用户名上，然后选择**用户设置**。例如 `80827e02-cfda-4d2d-b0aa-2d5205eb6eq9`。
`sc_sourceControlBranch`	*可选*：指定此属性可签出特定分支。如果没有指定分支，则使用执行计划的设置。
`sc_ buildName`	*可选*：要运行的构建。如果没有指定构建，则使用执行计划的设置。
`sc_StartOption`	*可选*：应执行的测试。如果没有指定，将执行分配的所有测试。允许的值有： • ALL • 失败 • NOT_EXECUTED

属性	说明
	• NOT_EXECUTED_SINCE_BUILD • FAILED_NOTEXECUTED_SINCE_BUILD • HAVING_FIXED_ISSUES 默认值为 ALL。
`sc_sinceBuild`	*可选*：尚未执行测试的开始构建名称。如果 `sc_StartOption` 属性设置为 FAILED_NOTEXECUTED_SINCE_BUILD 或 NOT_EXECUTED_SINCE_BUILD，请指定此属性。
`sc_collectResults`	*可选*：如果为 true，那么脚本将一直等到 Silk Central 执行完成，并以 JUnit 格式编写结果文件。如果为 false，那么脚本将触发执行，并且将完成而不等待结果。这些文件将存储在子文件夹 `sc_results` 中。默认值为 true.`Boolean`。
`sc_startDelay`	*可选*：以秒为单位的启动延迟。如果您有多个执行计划要执行，可以指定它。将在各次启动之间按照指定的延迟顺序启动这些执行计划。如果您需要最大程度降低启动时测试环境上的工作量，例如在启动虚拟机或安装测试中的应用程序时，这会很有帮助。默认值为 0。

您可以直接在脚本中指定属性，或者在触发脚本时传递属性。

触发脚本时指定的所有额外项目属性将作为参数传递给 Silk Central 并用于执行。这使您可以使用构建服务器中的值参数化 Silk Central 中的执行。

例如，如果您的构建在 Docker 中启动测试服务器，则可以通过在命令行中指定属性来将 URL 传递给此服务器：

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

命令行示例

以下命令从命令行启动脚本，启动 localhost 上的执行树节点 22431、22432 和 22433，并使用 Web 服务令牌 80827e02-cfda-4d2d-b0aa-2d5205eb6ea9 进行身份验证：

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

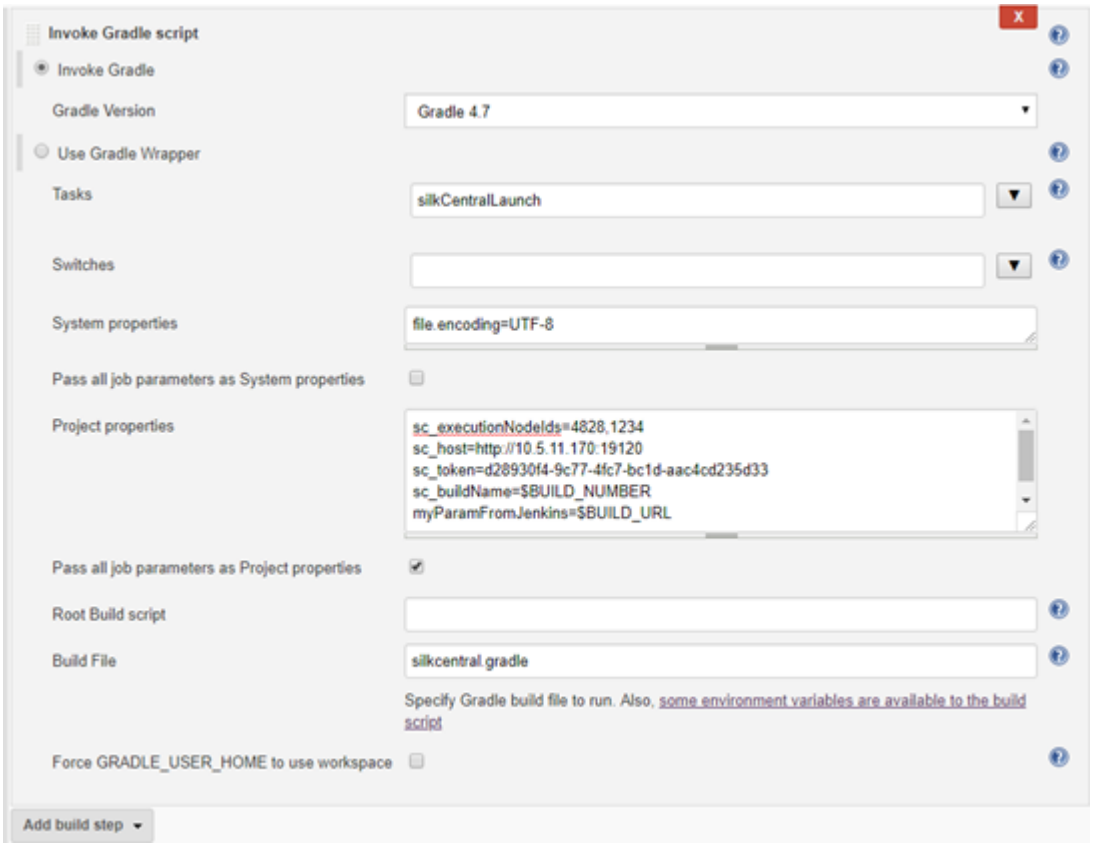
有关从 Jenkins 触发 Silk Central 上的执行的具体信息，请参阅*从 Jenkins 触发执行*。有关从 TeamCity 触发 Silk Central 上的执行的具体信息，请参阅*从 TeamCity 触发执行*。

从 Jenkins 触发执行

如果您的构建过程尚未使用 Gradle，请确保 Jenkins 可以执行 Gradle 脚本。

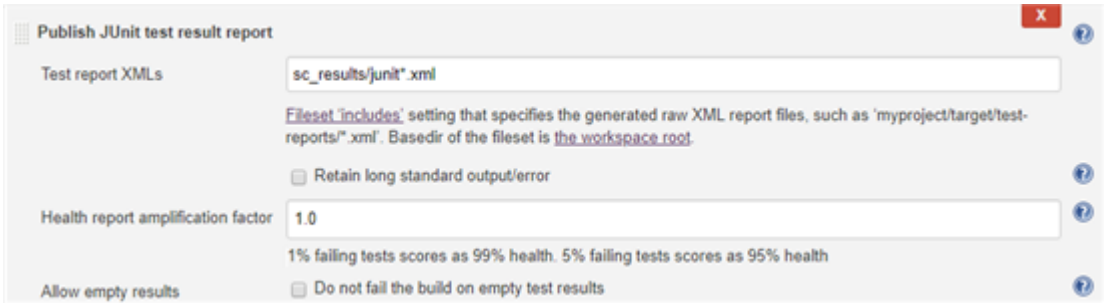
要从 Jenkins 触发 Silk Central 中的执行：

1. 在 Jenkins 中的**管理 Jenkins** > **全局工具配置**下安装 Gradle。
2. 在 Jenkins 项目中，添加构建步骤**调用 Gradle 脚本**。
   根据您存储 Gradle 脚本的位置，需要调整**构建文件**属性。按以下屏幕截图配置该步骤：



a) 如屏幕截图所示，您可以使用 Jenkins 中的可用变量（如 *$BUILD_NUMBER*）来配置脚本。
b) 如果您的 Jenkins 项目已参数化，则可以通过选中**将所有作业参数作为项目属性传递**将所有参数直接传递给 Silk Central。

3. 要在 Jenkins 中显示测试结果，请向 Jenkins 项目添加构建后操作**发布 JUnit 测试结果报告**。
4. 在**测试报告 XML** 字段中指定脚本将文件写入的位置。
   例如 `sc_results/junit*.xml`。

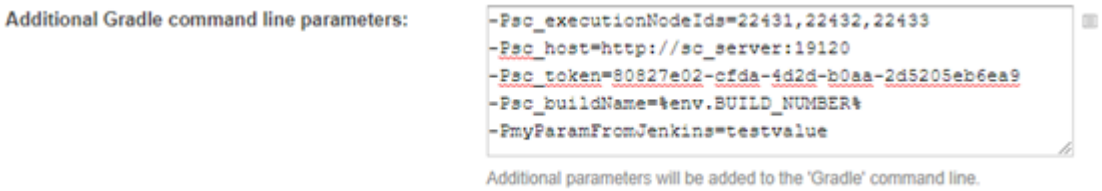


从 TeamCity 触发执行

从 TeamCity 触发 Silk Central 中的执行：

1. 向 TeamCity 中的构建添加构建步骤：
   a) 选择 **Gradle** 作为**运行器类型**。
   b) 在 **Gradle 任务**字段中指定 `silkCentralLaunch`。
   c) 在 **Gradle 构建文件**字段中浏览并选择 `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 中处理来自 Silk Central 的测试结果，请将构建功能 **XML 报告处理**添加到 TeamCity 的构建中。
3. 配置 **XML 报告处理**构建功能。
   a) 选择**报告类型**。
   b) 在**监视规则**字段中指定脚本将文件写入的位置。
   例如 `sc_results/*.xml`。

索引

L

P

R

S

T

U

W

X

Y

Z