



Silk Central 18.5

Aide de l'API

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

Copyright © Micro Focus 2004-2017. Tous droits réservés.

MICRO FOCUS, le logo Micro Focus et Silk Central sont des marques commerciales ou des marques déposées de Micro Focus IP Development Limited ou de ses filiales ou sociétés affiliées aux États-Unis, au Royaume-Uni et dans d'autres pays.

Toutes les autres marques appartiennent à leurs propriétaires respectifs.

2017-10-19

Table des matières

Introduction	5
Création de plug-ins	6
Intégration de couverture de code	7
Création de votre plug-in de couverture de code	7
Exemple de classe de profils	8
Fichier XSD de couverture de code	10
Exemple de données XML	11
Installation de l'atelier d'analyse de code dans un environnement d'application sous test Linux	12
Intégration de référentiels tiers	14
Interface d'intégration de référentiels tiers	14
Conventions relatives à l'intégration de référentiels tiers	15
Intégration du suivi des incidents	16
Interface Java	16
Intégration de la gestion des exigences	17
Interfaces Java	17
Intégration de types de tests tiers	18
Implémentation de plug-ins	18
Mise en package	18
Transmission de paramètres au plug-in	19
Structure de l'API	19
Exemple de code	20
Fichier XML de configuration	23
Méta-informations des plug-ins	23
Méta-informations des propriétés générales	23
Méta-informations des propriétés des chaînes	24
Méta-informations des propriétés des fichiers	24
Icônes personnalisées	24
Déploiement	25
Indication du début et de la fin de la capture vidéo	25
Intégration cloud	27
Services Web Silk Central	28
Démarrage rapide des services Web	29
Conditions préalables	29
Prise en main de services Web	30
Vue d'ensemble d'un client Web Services	30
Exemple de cas d'utilisation : ajout d'une exigence	31
Gestion des sessions	33
Services Web disponibles	33
Services Exchange	34
Interface reportData	34
Interface TMAAttach	35
Interface createTestPlan	37
Interface exportTestPlan	40
Interface updateTestPlan	40
Interface createRequirements	43
Interface exportRequirements	45
Interface updateRequirements	46
Interface updateRequirementsByExtID	47

Interface createExecutionDefinitions	49
Interface exportExecutionDefinitions	52
Interface updateExecutionDefinitions	53
Interface createLibraries	56
Interface exportLibraryStructure	58
Interface exportLibraryStructureWithoutSteps	59
Interface getLibraryInfoByName	60
Web Service Demo Client	61

Introduction

Ce guide fournit les informations dont vous avez besoin pour créer et déployer des plug-ins afin d'intégrer des outils tiers dans Silk Central. Il contient des spécifications de service Web et des descriptions d'API. Par ailleurs, il explique comment intégrer les plug-ins dans Silk Central.



Remarque: Ce guide part du principe que vous savez implémenter et utiliser les services Web.

Vue d'ensemble

Silk Central offre une API pour l'intégration d'applications tierces. Avec les API Silk Central, vous pouvez intégrer vos outils existants de référentiel tiers, de suivi des incidents et de gestion des exigences en configurant des plug-ins Silk Central. Différents exemples de plug-ins sont fournis avec Silk Central.

Reportez-vous au [Javadoc](#) pour connaître tous les détails sur les méthodes et classes Java disponibles. Si le lien ne fonctionne pas, cliquez sur **Aide > Documentation > Spécification de l'API de Silk Central** dans le menu Silk Central pour ouvrir le Javadoc.

Silk Central Plug-ins d'intégration de

Les plug-ins Silk Central sont fournis en l'état, avec tous les défauts qu'ils comportent et sans garantie d'aucune sorte. Micro Focus n'offre aucune garantie ou condition, qu'elle soit expresse, implicite ou légale sur quelque point que ce soit, y compris, mais sans s'y limiter, les garanties et conditions de qualité marchande, d'adéquation à un usage particulier, d'absence de virus, d'exactitude ou d'exhaustivité, de titre, de jouissance paisible et d'absence de contrefaçon.

Vous utilisez les plug-ins à vos propres risques. Micro Focus ne pourra en aucun cas être tenu responsable d'éventuels dommages directs, indirects, spéciaux, fortuits ou consécutifs de quelque nature que ce soit, y compris, mais sans s'y limiter, la perte de profits, résultant ou découlant de votre utilisation des plug-ins.

Création de plug-ins

Vue d'ensemble

Cette section décrit comment créer des plug-ins pour Silk Central. Seules les tâches communes à tous les types de plug-ins sont présentées ici.

Types de plug-ins

Silk Central fournit plusieurs API de plug-in. Chaque API est considérée comme une *espèce*.

Compilation

Pour les plug-ins de développement et de compilation, reportez-vous aux Notes de Release de Silk Central pour la version Java appropriée. Cela s'avère important pour la compatibilité avec l'environnement d'exécution Java de Silk Central.

Déploiement

Après avoir créé vos classes de plug-in et implémenté une API d'espèce, vous pouvez créer un package de plug-ins (fichier JAR ou ZIP).

- Si votre plug-in ne possède pas d'autres dépendances (ou dépend des bibliothèques faisant déjà partie de Silk Central), il vous suffit de créer un fichier JAR contenant vos classes.
- Si votre plug-in dépend de bibliothèques supplémentaires, placez les bibliothèques dans le sous-répertoire `lib` et regroupez toutes les bibliothèques dans une archive ZIP.

Placez le fichier créé dans le répertoire de plug-ins situé à l'emplacement `<application server installation directory>\plugins\`.



Remarque: Vous devez redémarrer le serveur d'application et le serveur de présentation de manière à ce que le nouveau plug-in déployé soit disponible dans Silk Central. Pour plus d'informations sur le redémarrage des serveurs, reportez-vous aux rubriques *Administration* de cette Aide.

Distribution

Étant donné que Silk Central connaît les types d'espèces de plug-ins, l'application est également capable de déterminer les espèces requises par chaque serveur (exécution, application et présentation). Chaque plug-in peut être installé sur le serveur d'application. Silk Central distribue automatiquement les plug-ins appropriés sur chaque serveur.

Intégration de couverture de code

L'interface de l'API Java évoquée dans ce chapitre est requise afin de créer des plug-ins pour Silk Central qui permettent d'activer l'intégration d'un outil de couverture de code tiers (externe).

Les outils de couverture de code permettent de fournir des informations sur le code couvert par les tests. Silk Central propose les outils de couverture de code prêts à l'emploi suivants :

- Analyse de code Java Silk Central (Java Code Analysis Agent)
- Analyse de code DevPartner Studio .NET (Windows Code Analysis Framework)



Remarque: Si votre application sous test est exécutée sous Linux, reportez-vous à la rubrique *Installation de l'atelier d'analyse de code dans un environnement d'application sous test Linux*.

Si les deux outils précédemment cités ne suffisent pas, vous pouvez créer et déployer votre propre intégration de couverture de code. Reportez-vous à *Creating Your Own Code Coverage Plugin*.



Remarque: En plus de déployer votre application personnalisée sur le serveur d'application (reportez-vous à la rubrique *Création de plug-ins*), vous devez aussi déployer votre application personnalisée sur le serveur de l'atelier d'analyse de code. Votre application sous test et votre outil de couverture de code se trouvent à cet emplacement. Le chemin d'accès est le suivant : `\Silk\Silk Central <version>\Plugins`

Création de votre plug-in de couverture de code

Cette rubrique décrit comment créer un plug-in de couverture de code. Vous devez maîtriser le concept de lotissement Silk Central. Dans Silk Central, un lotissement est requis avant chaque exécution. Un lotissement comprend tous les espaces de noms/packages/classes/méthodes dans l'application testée.



Remarque: L'API Silk Central requiert un fichier XML pour pouvoir être retournée dans le cadre d'exécutions de couverture de code. Cela signifie que si votre outil de couverture de code stocke les informations associées dans une base de données, vous devrez effectuer des étapes supplémentaires pour récupérer les données.



Remarque: L'exécution simultanée de tests dans le même atelier d'analyse de code par plusieurs serveurs d'exécution n'est pas prise en charge.

1. Ajoutez la bibliothèque `scc.jar` à votre classpath, car elle comporte les interfaces devant être étendues. Le fichier JAR se trouve dans le répertoire `lib` du répertoire d'installation de Silk Central.
2. Ajoutez les deux déclarations d'import suivantes :

```
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. Créez une classe qui implémente `CodeAnalysisProfile`.
4. Ajoutez toutes les méthodes requises à partir de l'interface de couverture de code et répertoriées dans les étapes suivantes. Vous pouvez vous reporter à l'élément `Exemple de classe d'interface` pour connaître sa définition et implémenter manuellement les méthodes. À défaut, vous pouvez copier et coller la rubrique [Exemple de classe d'interface](#), qui inclut les imports et les définitions de méthode.



Remarque: Les méthodes des étapes suivantes que vous écrirez sont appelées par Silk Central lorsqu'elles sont requises. Cela signifie que vous ne les appellerez pas directement.

5. Code `getBaseline`. Cette méthode doit renvoyer un fichier XML comprenant tous les espaces de noms/packages/classes/méthodes dans l'application. Reportez-vous au fichier de la rubrique [Exemple](#)

[de données XML](#) pour plus d'informations sur le format du fichier. Vous devez valider le fichier XML à l'aide de l'exemple de fichier XSD. Reportez-vous à la rubrique [Fichier XSD de couverture de code](#) pour en savoir plus sur le fichier XSD.

Cette fonction est appelée avant le démarrage de la couverture. Elle se déclenche lorsque le serveur d'exécution Silk Central démarre une exécution de test afin de lancer l'analyse de code et renvoyer tous les objets à couvrir. Le résultat doit être converti dans un fichier XML à l'aide du format spécifié dans le schéma XML inclus dans le dossier d'installation de CA-Framework.

6. Code `startCoverage`. Cet appel doit indiquer à votre outil de couverture de code de démarrer la collecte des données. Renvoie la valeur `Vrai` en cas de démarrage.

Cet appel est lancé par l'atelier de couverture de code Silk Central lorsque la méthode `getBaseLine()` est terminée. Vous devez y démarrer votre outil de couverture de code, qui collectera les données de couverture de code.

7. Code `stopCoverage`. Cet appel doit indiquer à votre outil de couverture de code d'arrêter la collecte des données. Renvoie la valeur `Vrai` en cas de réussite.

Cet appel est lancé après `startCoverage` ; il est déclenché par le serveur d'exécution Silk Central terminant une exécution de test afin d'arrêter l'analyse de code.

8. Code `getCoverage`. Renvoie un fichier XML comportant les données collectées entre les méthodes `startCoverage` et `stopCoverage`. Reportez-vous à la rubrique [Exemple de données XML](#) pour plus d'informations sur le format du fichier. Vous devez valider le fichier XML à l'aide de l'exemple de fichier XSD. Reportez-vous à la rubrique [Fichier XSD de couverture de code](#) pour en savoir plus sur le fichier XSD.

Cette fonction est appelée après `stopCoverage()` et renvoie toutes les données de couverture collectées. Le résultat doit être converti au format XML à l'aide du format spécifié dans le schéma XML.

9. Code `GetName`. Doit indiquer le nom qui sera utilisé pour désigner l'outil de couverture de code. Par exemple, cette valeur sera utilisée comme l'une des valeurs dans la liste **Profil d'Analyse de Code** de la boîte de dialogue **Modifier les Paramètres d'Analyse de Code**.

Celle-ci est appelée en premier lieu par l'atelier d'analyse de code Silk Central. Le nom du plug-in s'affiche dans la liste de couverture de code de Silk Central.

10. Placez le plug-in dans un fichier `.jar` et insérez ce dernier dans un fichier `.zip`.

11. Déployez votre plug-in aux emplacements suivants :

- Dans le répertoire `Plugins` du dossier d'installation de Silk Central.
- Dans le répertoire `Plugins` du dossier d'installation de CA-Framework.

Exemple de classe de profils

Ce fichier d'exemple présente toutes les méthodes, imports et implémentations requises pour un plug-in de couverture de code.

```
//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;
}
}

```

Fichier XSD de couverture de code

Le document suivant correspond au fichier XSD à utiliser pour valider le code XML généré par votre outil de couverture de code. Il est disponible à l'emplacement suivant : <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 invocation count, this will be 0 for the
definition file-->
        <xs:attribute name="inv" type="xs:string" />
    </xs:complexType>
</xs:element>
</xs:sequence>
    <xs:attribute name="name" type="xs:string" />
</xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>
</xs:element>
</xs:schema>

```

Exemple de données XML

L'API de couverture de code reconnaît les données XML au format suivant.

Vous pouvez utiliser l'exemple de document XSD fourni pour valider également vos données 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"/>
    </class>
</data>

```

```

<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>

```

Installation de l'atelier d'analyse de code dans un environnement d'application sous test Linux

La procédure ci-dessous doit être exécutée si le plug-in de couverture de code que vous créez doit interagir avec l'application .NET sous test sur le système d'exploitation Linux.

1. Vous pouvez accéder à l'atelier d'analyse de code pour Linux en sélectionnant **Aide > Outils > Atelier d'Analyse de Code Linux**. Téléchargez-le et copiez-le dans le dossier racine ou tout autre dossier de la machine Linux.
2. Vérifiez que Java Runtime Environment (JRE) 8 est installé sur la machine où se trouve l'application sous test.

3. Extrayez `CA-Framework.tar.gz`.
4. Placez le plug-in d'analyse de code dans le dossier `<dossier d'installation>/18.5Silk Central/Plugins`.
5. Accédez au répertoire `<dossier d'installation>/Silk Central 18.5/Code Analysis`.
6. Recherchez le script de shell `startCodeAnalysisFramework.sh` qui exécutera le processus `CA-Framework`.
7. Exécutez la commande suivante afin de convertir le fichier au format Unix : `dos2unix startCodeAnalysisFramework.sh`.
8. Exécutez la commande suivante afin de définir les autorisations requises pour exécuter le script de shell : `chmod 775 startCodeAnalysisFramework.sh`.
9. Exécutez le script de shell suivant afin de mettre en œuvre le processus `CAFramework` : `./startCodeAnalysisFramework.sh`.

L'atelier d'analyse de code peut désormais être utilisé à partir de Silk Central.

Intégration de référentiels tiers

Les profils de référentiel tiers permettent à Silk Central de s'intégrer à des systèmes de référentiel tiers externes.

Une fois déployés, les plug-ins de référentiel tiers personnalisés peuvent être configurés dans Silk Central, ce qui vous permet de définir l'emplacement où les serveurs d'exécution Silk Central doivent récupérer les sources des programmes en vue de l'exécution des tests.

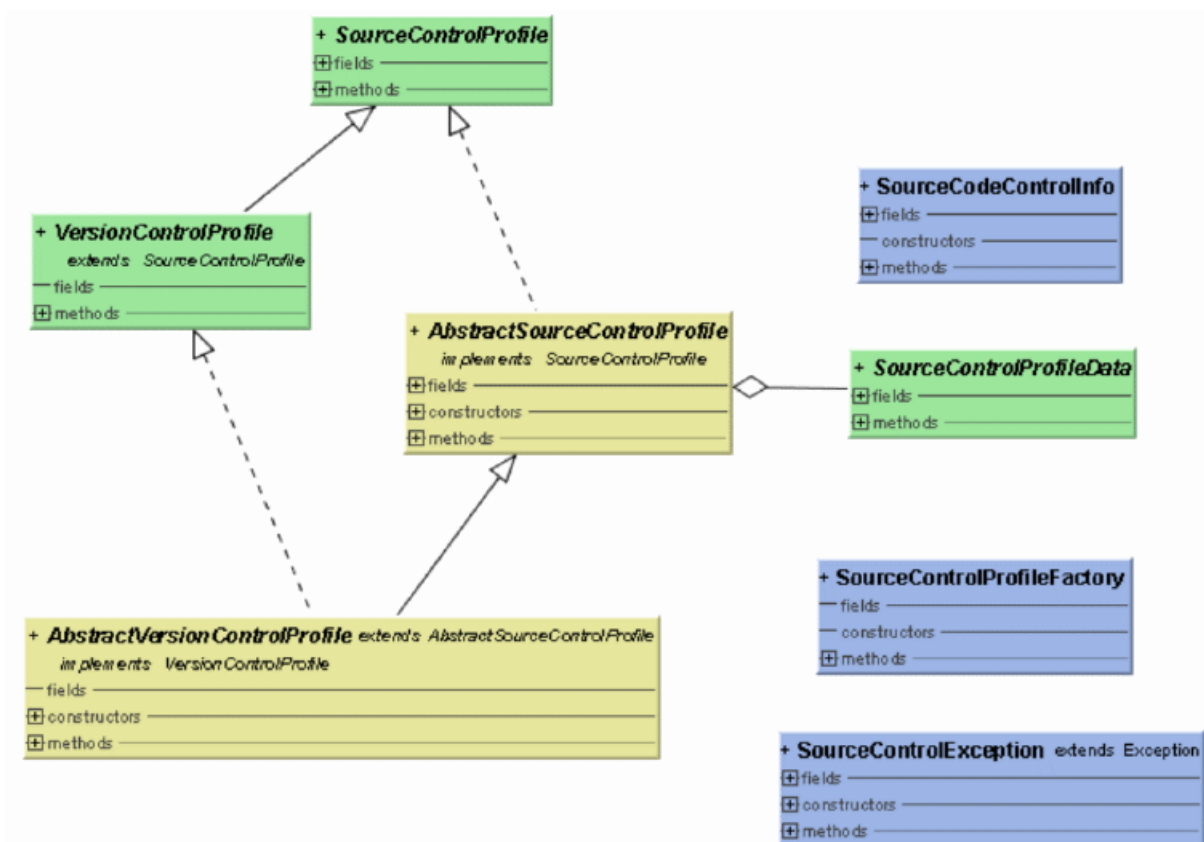
Voir les sources du package `com.segue.scc.vcs.subversion` dans `C:\Program Files (x86)\Silk\Silk Central 18.5\instance_<numéro d'instance>_<nom d'instance>\Plugins\subversion.zip` pour voir comment s'accordent ces éléments.

Interface d'intégration de référentiels tiers

Silk Central fait la distinction entre `SourceControlProfile` et `VersionControlProfile`. La différence est que `SourceControlProfile` n'est pas versionné, alors que `VersionControlProfile` l'est.

Voici les interfaces Silk Central utilisées pour l'intégration de référentiels tiers :

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



Reportez-vous au [Javadoc](#) pour connaître tous les détails sur les méthodes et classes Java disponibles. Si le lien ne fonctionne pas, cliquez sur **Aide > Documentation > Spécification de l'API de Silk Central** dans le menu Silk Central pour ouvrir le Javadoc.

Conventions relatives à l'intégration de référentiels tiers

Chaque implémentation doit fournir un constructeur par défaut et, éventuellement, un constructeur avec le paramètre `SourceControlProfileData`. Si ce constructeur n'est pas fourni, une méthode setter bean doit être indiquée pour le paramètre `SourceControlProfileData`.

Comme chaque méthode d'interface spécifie le paramètre `SourceControlException` à envoyer, il est interdit d'envoyer un paramètre `RuntimeException` dans une méthode utilisée par l'interface.

Intégration du suivi des incidents

L'interface présentée dans ce chapitre est requise afin de créer des plug-ins pour Silk Central qui permettent l'intégration d'un système de suivi des incidents tiers (externe).

La définition de profils de suivi d'incidents vous permet de lier des tests de l'espace **Tests** aux incidents des systèmes de suivi des incidents tiers. Les états des incidents liés sont mis à jour périodiquement à partir des systèmes de suivi des incidents tiers.

Reportez-vous aux sources du package `com.segue.scc.issuetracking.bugzilla3` dans `C:\Program Files (x86)\Silk\Silk Central 18.5\instance_<numéro d'instance>_<nom d'instance>\Plugins\Bugzilla3.zip` pour voir comment s'accordent ces éléments.

Interface Java

Reportez-vous au [Javadoc](#) pour connaître tous les détails sur les méthodes et classes Java disponibles. Si le lien ne fonctionne pas, cliquez sur **Aide > Documentation > Spécification de l'API de Silk Central** dans le menu Silk Central pour ouvrir le Javadoc.

Environnement de build

Ajoutez la bibliothèque `scc.jar` à votre classpath, car elle comporte les interfaces devant être étendues. Le fichier JAR se trouve dans le répertoire `lib` du répertoire d'installation de Silk Central.

Vous devez étendre deux interfaces/classes :

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

Classes/Interfaces



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

Intégration de la gestion des exigences

Silk Central peut intégrer des systèmes tiers de gestion des exigences (RMS) afin d'associer et de synchroniser les exigences.

Cette section décrit une interface de programmation d'application Java qui vous permet d'implémenter des plug-ins tiers en vue de la synchronisation des exigences dans Silk Central avec celles d'un système tiers. Cette section décrit les interfaces qui identifient un plug-in d'exigences et son déploiement.

Un fichier JAR ou ZIP est fourni selon le concept standard de plug-in Silk Central et déployé automatiquement sur tous les serveurs de présentation, permettant ainsi l'accès aux outils tiers à des fins de configuration et de synchronisation. Ce plug-in implémente une interface spécifiée qui permet à Silk Central de l'identifier en tant que plug-in d'exigences et fournit les propriétés de connexion requises pour accéder à l'outil tiers.

Reportez-vous au [Javadoc](#) pour connaître tous les détails sur les méthodes et classes Java disponibles. Si le lien ne fonctionne pas, cliquez sur **Aide > Documentation > Spécification de l'API de Silk Central** dans le menu Silk Central pour ouvrir le Javadoc.

Pour plus d'informations sur les plug-ins supplémentaires, contactez le support client.

Interfaces Java

Reportez-vous au [Javadoc](#) pour connaître tous les détails sur les méthodes et classes Java disponibles. Si le lien ne fonctionne pas, cliquez sur **Aide > Documentation > Spécification de l'API de Silk Central** dans le menu Silk Central pour ouvrir le Javadoc.

L'interface de base pour commencer est `RMPluginProfile` (`com.segue.tm.published.api.requirements.javaplugin`). `RMPluginProfile` spécifie le plug-in en tant que plug-in d'exigences.

L'API du plug-in d'exigences Java inclut également les interfaces suivantes :

- `RMAction`
- `RMAattachment`
- `RMDataProvider`
- *Facultatif* : `RMIconProvider`
- `RMNode`
- `RMNodeType`
- `RMPluginProfile`
- `RMProject`
- `RMTTest`
- `RMTTestParameter`
- *Facultatif* : `RMTTestProvider`
- `RMTTestStep`

Intégration de types de tests tiers

Silk Central vous permet de créer des plug-ins personnalisés pour des types de tests au-delà des types de tests standard disponibles, notamment Silk Performer, Silk Test Classic, tests manuels, NUnit, JUnit et Windows Scripting Host (WSH). Lors de la création d'un plug-in de type de test, le type de test personnalisé est disponible dans la zone de liste **Type** de la boîte de dialogue **Nouveau Test**, tout comme les types de tests standard disponibles pour la création de tests dans Silk Central.

Un plug-in désigne les propriétés requises pour la configuration d'un test et l'implémentation de l'exécution d'un test. Les méta-informations sur les propriétés sont définies dans un *Fichier XML de configuration*.

L'objectif du plug-in est la prise en charge des tests basés sur des infrastructures de test communes, telles que JUnit et NUnit, ou des langages de script (WSH) pour simplifier la personnalisation de Silk Central dans votre environnement de test. L'API publique bien définie de Silk Central vous permet d'implémenter une solution propriétaire répondant à vos besoins en matière de tests automatisés. Silk Central est ouvert et extensible pour n'importe quel outil tiers pouvant être appelé à partir d'une implémentation Java ou via un appel de la ligne de commande.

Reportez-vous au [Javadoc](#) pour connaître tous les détails sur les méthodes et classes Java disponibles. Si le lien ne fonctionne pas, cliquez sur **Aide > Documentation > Spécification de l'API de Silk Central** dans le menu Silk Central pour ouvrir le Javadoc.

Les classes décrites dans Javadoc sont comprises dans le fichier `tm-testlaunchapi.jar`.

Pour plus d'informations sur les plug-ins supplémentaires, contactez le support client.

Cette section inclut un exemple de code qui implémente le type de test de l'exécuteur de processus. L'exécuteur de processus peut être utilisé pour lancer un fichier exécutable et il étend la classe de lanceurs de tests de processus publiés. Pour plus d'informations, téléchargez l'*Exemple du Plug-in de Lancement de Test* dans **Aide > Outils** et prenez connaissance du fichier `Readme.txt`.

Implémentation de plug-ins

Les principes de l'API reposent sur des concepts Java Beans largement reconnus. Cela permet aux développeurs d'implémenter facilement des plug-ins de lancement de test. Pour éviter de placer des informations textuelles dans du code Java, les méta-informations relatives aux propriétés sont définies dans un fichier XML.

L'implémentation des plug-ins est mise en package dans une archive ZIP et inclut une interface de rappel en vue de son intégration. En retour, d'autres interfaces sont fournies par la structure de plug-ins et permettent à l'implémentation d'accéder aux informations ou de renvoyer des résultats.

Mise en package

Le plug-in est mis en package dans une archive ZIP qui contient le code source Java et le fichier de configuration XML. L'archive contient aussi des implémentations du plug-in de lancement de test. Le code source peut figurer dans un fichier d'archive Java (`.jar`) ou directement dans des fichiers `.class` situés à l'intérieur de dossiers qui représentent la structure des packages Java.

La classe de plug-in `TestLaunchBean` suit la norme Bean et implémente l'interface `TestLaunchBean`. Le fichier de configuration XML de l'archive `.zip` porte le même nom que cette classe. Cela vous permet de mettre plusieurs plug-ins et fichiers XML en package dans une seule archive.

Transmission de paramètres au plug-in

Si un plug-in est basé sur la classe `ExtProcessTestLaunchBean`, chaque paramètre est automatiquement défini comme une variable d'environnement dans le processus démarré par le plug-in. C'est également le cas si le nom du paramètre correspond au nom d'une variable système : ainsi, la valeur de la variable système est remplacée par celle du paramètre, sauf si la valeur du paramètre est une chaîne vide.

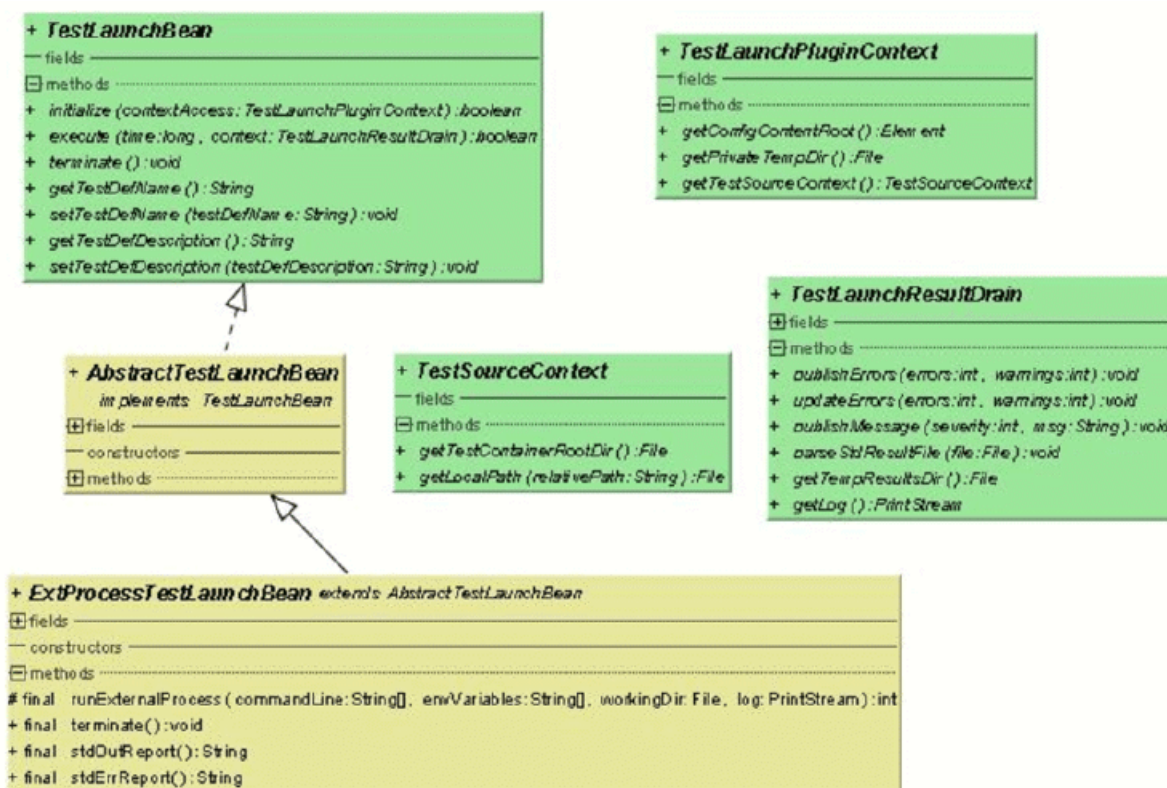
L'interface de plug-in propose un accès à tous les paramètres personnalisés ayant été définis dans l'espace **Tests** de Silk Central. Seuls les paramètres personnalisés sont pris en charge pour les types de tests tiers. Le plug-in ne peut pas spécifier des paramètres prédéfinis ; l'implémentation du plug-in détermine si des paramètres sont définis pour des tests spécifiques, et de quelle manière.

Utilisez la méthode `getParameterValueStrings()` dans l'interface `TestLaunchPluginContext` pour obtenir un conteneur comportant des mappages entre les noms de paramètres (clé) et leurs valeurs représentées sous la forme de `String`.

Pour les types de tests JUnit, toute classe de test JUnit peut accéder à un paramètre personnalisé du test sous-jacent en tant que propriété système Java ; le programme de lancement transmet ces paramètres à la machine virtuelle en cours d'exécution, à l'aide de l'argument `VM "-D"`.

Structure de l'API

Cette illustration détaille la structure de l'API pour l'intégration des types de tests tiers.



Reportez-vous au [Javadoc](#) pour connaître tous les détails sur les méthodes et classes Java disponibles. Si le lien ne fonctionne pas, cliquez sur **Aide > Documentation > Spécification de l'API de Silk Central** dans le menu Silk Central pour ouvrir le Javadoc.

Interfaces disponibles

- TestLaunchBean
- ExtProcessTestLaunchBean
- TestLaunchPluginContext
- TestSourceContext
- TestLaunchResultDrain

Exemple de code

Cet exemple de bloc de code implémente le type de test de l'exécuteur de processus, lequel peut être utilisé pour lancer un fichier exécutable et étendre la classe `ExtProcessTestLaunchBean` publiée.

```
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 = getEnvironmentVariables(context);

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

        boolean outputXmlFound = handleOutputXmlIfExists(context);

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

/**
* Initializes environment variables to be set additionally to those
* inherited from the system environment of the Execution Server.
* @param context the test execution context
* @return String array containing the set environment variables
* @throws IOException

```

```

*/
private String[] 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 -> Tools -> Test Package XML
Schema
 * Definition File
 * @param context the test execution context
 * @return true if output.xml exists
 * @throws IOException
 */
private boolean handleOutputXmlIfExists(TestLaunchResultDrain context)
throws IOException {
    String outputFileName = context.getTempResultsDir().getAbsolutePath()
+ File.separator + TestLaunchResultDrain.OUTPUT_XML_RESULT_FILE;
    File outputfile = new File(outputFileName);
    boolean outputXmlExists = outputfile.exists();
    if (outputXmlExists) {
        context.parseStdResultFile(outputfile);
        context.publishMessage(TestLaunchResultDrain.SEVERITY_INFO,
            String.format("output.xml parsed from '%s'", outputFileName));
    }
    return outputXmlExists;
}

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

```

Fichier XML de configuration

Le fichier XML de configuration comprend des méta-informations sur le plug-in de type de test tiers.

Méta-informations des plug-ins

En général, les méta-informations des plug-ins fournissent des données sur les plug-ins et les types de tests. Les types d'informations disponibles sont répertoriés ci-dessous.

Type	Description
id	Chaîne interne unique parmi tous les plug-ins installés et identifiant ce type.
label	Interface utilisateur affichée dans la liste des choix et les boîtes de dialogue d'édition pour ce type.
description	Informations textuelles supplémentaires décrivant ce type de test.
version	Distingue les différentes versions d'un type.

Méta-informations des propriétés générales

Pour les propriétés modifiables, des informations générales sont identiques pour chaque type de propriété, en plus des informations spécifiques au type. Le nom d'une propriété doit correspondre à celles définies dans le code par les méthodes `get<<propertyname>>`, avec le premier caractère en minuscules.

Type	Description
label	Texte affiché dans l'interface utilisateur.
description	Informations textuelles supplémentaires décrivant la propriété.
isOptional	La valeur True indique qu'aucune saisie utilisateur n'est requise.
default	Valeur par défaut de la propriété lors de la création d'un test. Cette valeur doit correspondre au type de propriété.

Méta-informations des propriétés des chaînes

Voici les types de méta-informations des propriétés des chaînes disponibles dans le plug-in.

Type	Description
<code>maxLength</code>	Indique la longueur maximale de la chaîne pouvant être saisie par l'utilisateur.
<code>editMultiLine</code>	Indique si le champ d'édition doit s'étendre sur plusieurs lignes.
<code>isPassword</code>	La valeur <code>true</code> masque la saisie utilisateur sous la forme <code>***</code> .

Méta-informations des propriétés des fichiers

Voici les types de méta-informations des propriétés des fichiers disponibles dans le plug-in.

Pour les valeurs de fichiers spécifiées, les fichiers sans contrôle de version dotés de chemins d'accès absolus sur le serveur d'exécution sont distingués des fichiers avec contrôle de version. Les fichiers sous référentiel tiers sont toujours relatifs au répertoire racine du conteneur de tests et servent généralement à spécifier une source de test à exécuter. Les chemins d'accès absolus doivent exister sur le serveur d'exécution, en général en spécifiant un outil ou une ressource pour invoquer le test sur le serveur d'exécution.

Type	Description
<code>openBrowser</code>	Une valeur <code>True</code> indique qu'un navigateur doit être ouvert pour sélectionner le fichier dans le conteneur de tests.
<code>isSourceControl</code>	Une valeur <code>True</code> indique que le fichier provient d'un référentiel tiers.
<code>fileNameSpec</code>	Indique une restriction au niveau des noms de fichier autorisés par rapport à la boîte de dialogue standard Windows de navigation dans les fichiers.

Icônes personnalisées

Vous pouvez créer des icônes personnalisées pour votre type de test afin de pouvoir le différencier des autres types de tests. Pour définir ces icônes pour le plug-in (pour lequel vous avez défini l'identificateur `PluginId` dans le fichier XML de configuration), vous devez insérer les quatre icônes suivantes dans le répertoire racine du conteneur ZIP du plug-in.

Nom	Description
<code><PluginId>.gif</code>	Icône par défaut de votre type de test. Par exemple, <code>ProcessExecutor.gif</code> .
<code><PluginId>_package.gif</code>	Icône utilisée pour les nœuds du package de tests et de la suite, lorsque vous convertissez un test du type spécifié en un package de tests. Par exemple, <code>ProcessExecutor_package.gif</code> .
<code><PluginId>_linked.gif</code>	Icône utilisée si le dossier parent du test est un conteneur de tests lié. Par exemple, <code>ProcessExecutor_linked.gif</code> .

Nom	Description
<PluginId>_incomplete.gif	Icône utilisée si l'application ou le profil de référentiel tiers du conteneur de tests parent du test ne sont pas définis.

Lorsque vous créez une nouvelle icône pour un type de test, appliquez les règles suivantes :

- Utilisez uniquement des icônes de type GIF. L'extension de fichier est sensible à la casse et doit toujours figurer en minuscules (.gif).
- Supprimez les icônes anciennes ou non valides de <Silk Central deploy folder>\wwwroot\silkroot\img\PluginIcons, sinon les icônes ne seront pas mises à jour avec les nouvelles icônes dans le répertoire racine du conteneur ZIP du plug-in.
- L'icône est au format 16x16 pixels.
- Au maximum, 256 couleurs sont autorisées pour une icône.
- L'icône inclut une transparence de 1 bit.

Déploiement

L'archive ZIP du plug-in doit figurer dans le sous-répertoire `Plugins` du répertoire d'installation de Silk Central. Pour intégrer des plug-ins se trouvant dans ce répertoire, redémarrez votre serveur d'application et de présentation via le Gestionnaire de services Silk Central.



Remarque: Le déploiement à chaud n'est pas pris en charge.

Ces deux serveurs doivent être redémarrés à chaque modification d'une archive. L'archive est téléchargée automatiquement sur les serveurs d'exécution.



Remarque: Ne supprimez jamais une archive de plug-in après avoir créé un test basé sur ce plug-in. Un test basé sur une archive de plug-in qui n'existe plus renverra des erreurs inconnues si celui-ci est modifié ou exécuté.

Indication du début et de la fin de la capture vidéo

Reportez-vous au [Javadoc](#) pour connaître tous les détails sur les méthodes et classes Java disponibles. Si le lien ne fonctionne pas, cliquez sur **Aide > Documentation > Spécification de l'API de Silk Central** dans le menu Silk Central pour ouvrir le Javadoc.

Quand vous créez un nouveau plug-in de test de tiers pour Silk Central, que le type de test tiers prend en charge le traitement de plusieurs cas de tests lors d'une seule exécution et que vous voulez associer les vidéos capturées à des cas de tests spécifiques, vous avez le choix entre deux méthodes.

Tests tiers exécutés dans le plug-in

Pour ces tests, nous recommandons de suivre les méthodes `indicateTestStart` et `indicateTestStop` de la classe `TestLaunchResultDrain`.

Tests tiers exécutés dans un processus externe

Pour ces tests, vous pouvez utiliser un service TCP/IP pour envoyer les messages `START` (démarrer) et `FINISH` (terminer) au port du serveur d'exécution Silk Central. Le numéro de port à utiliser peut être interrogé à partir de `ExecutionContextInfo.ExecProperty#PORT_TESTCASE_START_FINISH` dans le plug-in. Le port est également disponible sous la forme de la variable `#sctm_portTestCaseStartFinish` dans le processus de test, si le plug-in dépasse `ExtProcessTestLaunchBean`. Ces types de messages indiquent respectivement au serveur d'exécution qu'un cas de test a démarré ou est terminé. Ces messages doivent être codés au format Unicode (UTF8) ou ASCII.

Type de message	Format
START	START <Test Name>, <Test ID> <LF>, où le caractère de retour chariot est représenté par le code ASCII 10.
FINISH	FINISH <Test Name>, <Test ID>, <Passed> LF, où le caractère de retour chariot est représenté par le code ASCII 10. Passed peut être True ou False. Si la configuration de la capture vidéo fait démarrer celle-ci En cas d'erreur, la vidéo est enregistrée avec les résultats uniquement si Passed est défini sur False.

Le serveur d'exécution répond par le message OK si la requête est reconnue. Dans le cas contraire, le serveur d'exécution répond par un message d'erreur. Attendez toujours la réponse du serveur d'exécution avant d'exécuter le cas de test suivant. Autrement, il est possible que la vidéo enregistrée ne corresponde pas au cas de test réel.

Si le processus externe, l'emplacement d'exécution du test, est basé sur un environnement Java, nous recommandons de suivre les méthodes `indicateTestStart` et `indicateTestStop` de la classe `TestCaseStartFinishSocketClient`, qui est incluse dans le fichier `tm-testlaunchapi.jar`.

Intégration cloud

Silk Central vous permet de procéder à l'intégration dans des services cloud publics et privés en configurant des profils de fournisseur cloud. Les profils de cloud reposent sur un concept de plug-in. Vous pouvez ainsi écrire votre propre plug-in pour un fournisseur cloud spécifique. Un plug-in de fournisseur cloud déploie un environnement virtuel avant chaque exécution de tests automatisés.



Remarque: L'API de cloud va être modifiée dans les prochaines versions de Silk Central. Si vous utilisez cette API, la mise à niveau vers une version ultérieure de Silk Central peut vous amener à mettre à jour votre implémentation.

Reportez-vous au [Javadoc](#) pour connaître tous les détails sur les méthodes et classes Java disponibles. Si le lien ne fonctionne pas, cliquez sur **Aide > Documentation > Spécification de l'API de Silk Central** dans le menu Silk Central pour ouvrir le Javadoc.

L'interface de base par laquelle il convient de commencer est `CloudProviderProfile` (`com.segue.scc.published.api.cloud`).`CloudProviderProfile` spécifie l'accès et le contrôle d'un système de cloud externe. Pour implémenter un plug-in de fournisseur cloud, procédez comme suit :

- Déterminer les propriétés devant être configurées dans un profil de fournisseur cloud pour accéder à distance à un fournisseur de ce type
- Valider les propriétés de profil et vérifier la connexion au fournisseur cloud
- Récupérer la liste des modèles d'image disponibles à partir du fournisseur cloud
- Déployer un environnement virtuel en fonction d'un modèle d'image sélectionné et exposer les adresses d'hôte accessibles de l'extérieur
- Vérifier si un environnement virtuel est déployé et en cours d'exécution
- Supprimer un environnement virtuel spécifique

Services Web Silk Central

Les services Web ne nécessitent pas de configuration, ils sont activés par défaut sur chaque serveur de présentation. Par exemple, si vous utilisez l'URL `http://www.yourFrontend.com/login` pour accéder à Silk Central, `http://www.yourFrontend.com/Services1.0/jaxws/system` (services hérités à `http://www.yourFrontend.com/Services1.0/services`) et `http://www.yourFrontend.com/AlmServices1.0/services` sont les URL de base que vous utilisez pour accéder aux services Web disponibles.

Lorsque vous accédez à l'URL de base à l'aide d'un navigateur, vous voyez une simple liste HTML de tous les services Web disponibles. Cette liste est fournie par JAX-WS. La pile SOAP utilisée par Silk Central est <https://jax-ws.java.net/>.

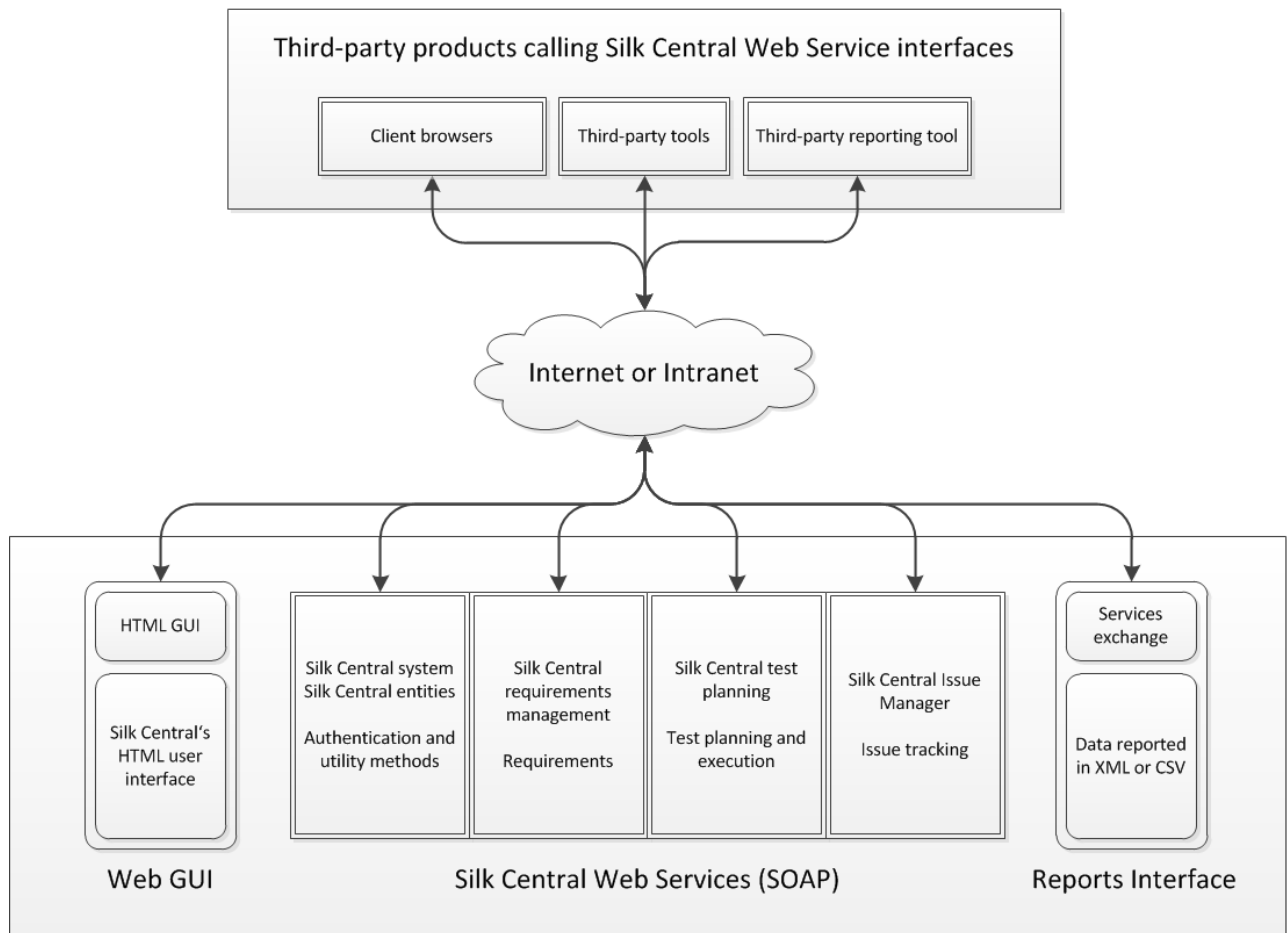
L'URL de base propose des liens vers des fichiers XML normalisés au format WSDL (Web Service Definition Language), où chaque fichier décrit l'interface d'un seul service Web. Ces fichiers ne sont pas contrôlables de visu. C'est la raison pour laquelle des clients compatibles avec SOAP (par exemple, Silk Performer Java Explorer) lisent les fichiers WSDL et extraient ainsi les informations requises pour invoquer des méthodes sur les services Web correspondants.

Reportez-vous au [Javadoc](#) pour connaître tous les détails sur les méthodes et classes Java disponibles. Si le lien ne fonctionne pas, cliquez sur **Aide > Documentation > Spécification de l'API de Silk Central** dans le menu Silk Central pour ouvrir le Javadoc.



Remarque: Les services Web d'origine implémentés dans Silk Central avant la version 2008 sont encore disponibles à l'emplacement `http://<nom d'hôte>:<port>/services`.

Silk Central Open Web Interfaces



Démarrage rapide des services Web

Cette section contient les conditions préalables, un exemple de cas d'usage et d'autres rubriques relatives à l'intégration des services Web.

Conditions préalables

Vous devez tenir compte des conditions préalables suivantes avant d'essayer de développer un client de services Web :

- Connaissances de base de la programmation orientée objet (POO). Une expérience du Java est utile, car les exemples donnés utilisent ce langage. Un développeur sans expérience Java, mais connaissant les langages C++, C#, Python ou Perl doit malgré tout être capable de suivre aisément les exemples. Une expérience de la manipulation des collections telles que HashMaps et Lists est souhaitable.
- Les tests JUnit sont mentionnés brièvement. Cette structure de test Java n'est pas requise, mais la connaître s'avère utile.
- Connaissances de base de la technologie des services Web. La présente Aide ne propose pas de cours d'introduction aux services Web ou à SOAP. Au minimum, vous devez avoir de l'expérience dans le codage et l'exécution d'un client de services Web « Hello World! ».
- Connaissance de l'architecture de Silk Central Web Services.

Prise en main de services Web

Reportez-vous aux [Notes de mise à jour de Silk Central](#) afin de vérifier que vous disposez de la version appropriée du kit de développement logiciel (SDK) installée et qu'il se trouve dans votre PATH.

Il est utile de disposer du fichier JAR JUnit dans votre classpath. Vous pouvez télécharger le fichier `JUnit.jar` depuis le site <http://www.junit.org/index.htm>.

1. Assurez-vous que le répertoire `bin` du SDK Java se trouve dans votre PATH.
2. Exécutez la commande suivante, en pointant vers le fichier WSDL du service Web souhaité.

```
wsimport -s <chemin dans lequel enregistrer les fichiers générés> -Xnocompile -p <structure de package à utiliser pour votre client yourWebService> http://<URL de votre service>/yourWebService?wsdl
```

3. Recherchez la classe Java `YourWebService` générée automatiquement.

Cette classe permet d'utiliser toutes les méthodes fournies par `YourWebService`.

Vue d'ensemble d'un client Web Services

En général, les services Web utilisent un protocole SOAP sur HTTP. Dans ce scénario, les enveloppes SOAP sont envoyées. Lorsque les collections et les autres objets complexes sont regroupés dans des enveloppes SOAP, vous pouvez avoir des difficultés à lire et à modifier les structures de données ASCII. Les développeurs débutants ne doivent pas essayer de créer un client Web Services en manipulant directement les enveloppes SOAP. En général, les développeurs confirmés ne procèdent pas de cette façon, car cette opération est fastidieuse et susceptible d'engendrer des erreurs. C'est pour cette raison que tous les principaux langages de programmation proposent des kits de développement Web Services. Dans Silk Central, l'API *Java API for XML Web Services (JAX-WS)* permet de créer des services Web et des clients communiquant à l'aide de messages SOAP.

Quel que soit le langage de programmation (Java, C++, C#, Perl ou Python), la procédure de création des clients Web Services est généralement la même :

1. Pointer un kit de développement sur le WSDL Web Services
2. Obtenir l'élément de remplacement d'un client en retour
3. Modifier l'élément de remplacement du client généré à l'étape 2 pour obtenir un client à part entière

C'est le cas pour JAX-WS. L'outil `wsimport` (accompagné du JDK) sert à créer les éléments de remplacement du client à partir de WSDL. Vous trouverez des informations détaillées sur le mode d'utilisation de l'outil `wsimport` dans la [Documentation des outils et utilitaires JDK](#). Voici une courte description des commutateurs utilisés ci-dessus :

- `-s` : répertoire de sortie des éléments de remplacement du client
- `-p` : Package cible. Déploiement sur cette structure de package

Exemple : `wsimport -s <emplacement des éléments de remplacement générés> -p <package cible> <WSDL>`

L'outil `wsimport` génère plusieurs classes pour garantir la prise en charge d'un client avec le service Web. Si `YourWebService` est le nom du service, les classes ci-dessous sont générées :

- `YourWebService` : interface représentant `YourWebService`.
- `YourWebServiceService` : Classe générée représentant le localisateur `YourWebService`, par exemple pour obtenir la liste des ports disponibles (interfaces de points de terminaison de service)
- `WSFaultException` : Classe d'exception mappée à partir de `wsdl:fault`
- `WsFaultBean` : Bean de réponse asynchrone provenant de la réponse `wsdl:message`

- `Serializable Objects` : objets côté client correspondant aux objets utilisés par `YourWebService`.

Pour générer un client JAX-WS pour l'accès aux services Web Silk Central, créez une classe Java appelée `YourWebServiceClient`. Vous avez établi une liaison au service Web via un port ; un port est un objet local assurant une fonction de proxy pour le service distant. Notez que le port est créé par l'exécution de l'outil `wsimport` dans l'étape ci-dessus. Pour récupérer ce proxy, appelez la méthode `getRequirementsServicePort` sur le service :

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

Vous pouvez ensuite appeler la méthode du service `logonUser` en transférant un nom d'utilisateur et un mot de passe afin d'obtenir un ID de session :

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

Exemple de cas d'utilisation : ajout d'une exigence

Dans le prolongement des étapes précédentes détaillées dans cette section, cette rubrique présente un cas d'utilisation consistant à ajouter une exigence à Silk Central.

Avant de continuer, les conditions préalables suivantes doivent être remplies :

- Vous avez effectué les étapes détaillées pour le service Web `requirements`.
 - Une classe POJO ou JUnit de travail possédant des méthodes de liaison et de connexion a été créée.
 - Vous avez pris connaissance des autres rubriques d'aide de l'API Silk Central.
1. Établissez une liaison au service à l'aide de la méthode de connexion incluant les données d'authentification de l'utilisateur.
 2. Enregistrez l'ID de session obtenu à l'étape précédente.
 3. Créez un objet d'exigence comprenant les données souhaitées.
 4. Appelez la méthode `updateRequirement` à l'aide de l'ID de session, l'ID de projet et de l'objet d'exigence généré.
 5. Enregistrez l'ID d'exigence renvoyé par la méthode `updateRequirement`.
 6. Créez un tableau `PropertyValue` des propriétés d'exigence.
 7. Appelez la méthode `updateProperties` à l'aide du tableau créé précédemment.

L'outil `wsimport` créera les objets de service Web mentionnés ci-dessus :

- `Requirement`
- `PropertyValue`

Vous pouvez à présent utiliser les méthodes OOP des objets ci-dessus afin d'utiliser le service Web. Aucune construction d'enveloppe SOAP n'est requise. Voici ci-après des extraits du code requis pour effectuer ce cas d'utilisation.

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

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

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

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

```

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

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

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

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

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

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

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

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

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

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

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

```




Remarque: Cet exemple de code est également disponible dans la classe Client de démo du service `Web com.microfocus.silkcentral.democlient.samples.AddingRequirement`.

Gestion des sessions

Les données Silk Central sont protégées contre les accès non autorisés. Vous devez indiquer vos données d'authentification pour que l'accès aux données vous soit octroyé. Cela est vrai non seulement lorsque vous travaillez sur le serveur de présentation HTML, mais également pour la communication avec Silk Central via des appels SOAP.

L'authentification constitue la première étape pour interroger des données ou modifier la configuration de Silk Central. Une fois l'authentification réussie, une session utilisateur est créée, autorisant l'exécution d'opérations ultérieures dans le contexte de cette connexion utilisateur.

Lorsque vous accédez à Silk Central via un navigateur Web, les données de session ne sont pas visibles par l'utilisateur. Le navigateur gère les données de session à l'aide de cookies. Contrairement à l'utilisation de Silk Central via HTML, les appels SOAP doivent traiter manuellement les données de session.

L'authentification via les services Web s'effectue à travers l'appel SOAP `logonUser` du service Web requis. L'appel de la méthode renvoie un identificateur de session qui référence la session créée sur le serveur et sert également de clé d'accès à Silk Central dans le contexte de cette session.

Chaque appel SOAP ultérieur requiert une session active pour s'exécuter. Il utilise l'identificateur ainsi renvoyé comme paramètre, en vérifie sa validité et s'exécute dans le contexte de la session correspondante.

L'exemple de code Java montre un accès simple à Silk Central via les services Web et l'utilisation de l'identificateur de session :

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

Une session Silk Central créée via les services Web ne peut pas être arrêtée explicitement. Les sessions se terminent automatiquement lorsqu'elles restent inactives durant un certain laps de temps. Dès qu'une session expire sur un serveur, les appels SOAP ultérieurs qui tentent d'utiliser la session génèrent des exceptions.

Un client de démonstration est disponible en téléchargement dans Silk Central dans le menu **Aide > Outils > Client de Démo des Web Services**. Ce projet de démonstration utilise le service Web d'exécution de Silk Central `tests`, vous aidant à vous familiariser avec l'interface de service Web.

Services Web disponibles

Le tableau suivant répertorie les services Web Silk Central disponibles. Vous pouvez également accéder à Silk Central depuis des interfaces HTTP. Pour plus d'informations, reportez-vous à la rubrique [Services Exchange](#).



Remarque: L'URL WSDL répertorie aussi les services Web internes au système qui ne sont pas destinés à créer des clients de services Web. Ce document décrit uniquement les services Web publiés.

Reportez-vous au [Javadoc](#) pour connaître tous les détails sur les méthodes et classes Java disponibles. Si le lien ne fonctionne pas, cliquez sur **Aide > Documentation > Spécification de l'API de Silk Central** dans le menu Silk Central pour ouvrir le Javadoc.




Remarque: Avec les services Web, toutes les heures qui sont indiquées par le système sont exprimées par rapport au Temps Universel Coordonné (UTC). Utilisez également l'heure UTC pour toutes les références horaires au sein des services Web que vous utilisez.

Nom de service Web (interface)	URL WSDL	Description
system (SystemService)	/Services1.0/jaxws/system?wsdl	Il s'agit du service racine qui fournit des méthodes d'authentification et d'utilitaire simple.
administration (AdministrationService)	/Services1.0/jaxws/administration?wsdl	Ce service donne accès aux entités <i>Projet</i> et <i>Application</i> de Silk Central.
requirements (RequirementsService)	/Services1.0/jaxws/requirements?wsdl	Ce service donne accès à l'espace Exigences de Silk Central.
tests (TestsService)	/Services1.0/jaxws/tests?wsdl	Ce service donne accès à l'espace Tests de Silk Central.
executionplanning (ExecutionPlanningService)	/Services1.0/jaxws/executionplanning?wsdl	Ce service donne accès à l'espace Organisation des exécutions de Silk Central.
filter (FilterService)	/Services1.0/jaxws/filter?wsdl	Ce service permet de créer, de lire, de mettre à jour et de supprimer des filtres.
issuemanager (IssueManagerService)	/Services1.0/jaxws/issuemanager?wsdl	Ce service donne accès à Issue Manager.

Services Exchange

Cette section décrit les interfaces HTTP qui sont utilisées pour gérer les rapports, les plans de test, les exécutions et les bibliothèques dans Services Exchange.

Vous pouvez également accéder à Silk Central via les services Web. Pour plus d'informations, reportez-vous à la rubrique [Services Web disponibles](#).

 **Remarque:** Avec les services Web, toutes les heures qui sont indiquées par le système sont exprimées par rapport au Temps Universel Coordonné (UTC). Utilisez également l'heure UTC pour toutes les références horaires au sein des services Web que vous utilisez.

Interface reportData

L'interface `reportData` permet d'interroger les données d'un rapport. Le tableau ci-dessous présente les paramètres de l'interface `reportData`.

URL de l'interface	Paramètres	Descriptions
http://<front-end URL>/servicesExchange?hid=reportData	sid	ID de session : Authentification de l'utilisateur
	reportFilterID	ID du filtre des rapports
	type	Format du corps de réponse : (csv ou xml)
	includeHeaders	Inclut ou non des en-têtes de rapport. (true ou false)
	userName	Nom d'utilisateur (optionnel) : Utilisé à la place de sid

URL de l'interface	Paramètres	Descriptions
	passWord	Mot de passe utilisateur (optionnel) : Utilisé à la place de sid
	projectID	ID du projet

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

Exemple d'interface reportData

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

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

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

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

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

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

Si le rapport nécessite des paramètres, vous devez ajouter le code ci-dessous à l'URL du rapport pour chaque paramètre :

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

Dans l'exemple, le paramètre `rp_execNode_Id_0` est défini sur la valeur 1.



Remarque: Les noms des paramètres transmis au service `reportData` doivent porter le préfixe `rp_`. Exemple : `/servicesExchange?hid=reportData&type=xml&sid=<...>&reportFilterID=<...>&projectID=<...>&rp_TestID=<...>`

Interface TMAAttach

L'interface `TMAAttach` sert à télécharger des pièces jointes à un test ou une exigence. Le tableau ci-dessous présente les paramètres de l'interface `TMAAttach`.

URL de l'interface	Paramètre	Descriptions
http://<front-end URL>/servicesExchange?hid=TMAttach	sid	ID de session : Authentification de l'utilisateur
	entityType	Type d'entité cible : test, exigence ou parent du pas de test
	entityID	ID d'entité cible : ID de test, d'exigence ou de test manuel
	description	Description de la pièce jointe : Texte codé dans une URL, utilisé pour décrire la pièce jointe
	isURL	Si la valeur est true, la pièce jointe est une URL. Si la valeur est false, la pièce jointe est un fichier.
	URL	Optionnel : URL à joindre
	stepPosition	Optionnel : Ordre du pas dans le test. Identifie le pas d'un test manuel (par exemple, l'ordre du premier pas est 1). L'ordre est obligatoire si entityType est TestStepParent.
	userName	Nom d'utilisateur (optionnel) : Utilisé à la place de sid
	passWord	Mot de passe utilisateur (optionnel) : Utilisé à la place de sid

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

Exemple de service Web TMAttach

Le code ci-dessous utilise Apache HttpClient pour obtenir une API HTTP-POST pratique afin de télécharger une pièce jointe binaire. Il n'est possible de télécharger qu'une seule pièce jointe par demande.

Il n'est possible de télécharger qu'une seule pièce jointe par demande. Pour télécharger Apache HttpComponents, visitez le site <http://hc.apache.org/downloads.cgi>. Reportez-vous à la documentation du composant pour connaître les bibliothèques requises.

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

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

HttpClient client = new HttpClient();
String formURL = "http://localhost:19120/
```

```

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

```

Interface createTestPlan

L'interface `createTestPlan` permet de créer des tests. La réponse HTTP de l'appel comprend la structure XML des tests modifiés. Les identifiants des nouveaux nœuds sont disponibles dans la structure du test XML mis à jour.

Le tableau ci-dessous présente les paramètres de l'interface `createTestPlan`.

URL de l'interface	Paramètre	Descriptions
http://<front-end URL>/servicesExchange?hid=createTestPlan	sid	ID de session – Authentification de l'utilisateur
	parentNodeID	ID du conteneur auquel le nouveau test est ajouté dans l'arborescence Tests
	userName	<i>Facultatif</i> : nom d'utilisateur – Utilisé à la place de sid
	passWord	<i>Facultatif</i> : mot de passe utilisateur – Utilisé à la place de sid

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

Le fichier de définition du schéma XML qui sert à valider les plans de test peut être téléchargé au moyen de l'URL du serveur de présentation, `http://<URL serveur de présentation>/silkroot/xsl/testplan.xsd`, ou copié à partir du dossier d'installation du serveur de présentation, `<dossier d'installation Silk Central>/wwwroot/silkroot/xsl/testplan.xsd`.

Exemple de service Web createTestPlan

Le code suivant utilise Apache HttpClient pour créer des tests.

```

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

long sessionID = mWebServiceHelper.getSessionId();

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

```

```

%d",
    "createTestPlan", sessionID,
    PARENT_NODE_ID));

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

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

```

Il n'est possible de télécharger qu'une seule pièce jointe par demande. Pour télécharger Apache HttpComponents, visitez le site <http://hc.apache.org/downloads.cgi>. Reportez-vous à la documentation du composant pour connaître les bibliothèques requises.

Exemple de test

Le code suivant affiche un exemple de test pouvant être téléchargé dans Silk Central à l'aide des services `createTestPlan` et `updateTestPlan`.

```

<?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>

```

Interface exportTestPlan

L'interface `exportTestPlan` permet d'exporter les plans de test sous forme de fichiers XML. Le tableau ci-dessous présente les paramètres de l'interface `exportTestPlan`.

URL de l'interface	Paramètre	Descriptions
http://<front-end URL>/servicesExchange?hid=exportTestPlan	sid	ID de session – Authentification de l'utilisateur
	nodeID	Le nœud associé à cet ID et, de façon récursive, tous les nœuds enfants de ce nœud sont exportés.
	userName	<i>Facultatif</i> : nom d'utilisateur – Utilisé à la place de sid
	passWord	<i>Facultatif</i> : mot de passe utilisateur – Utilisé à la place de sid

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

Exemple de service Web exportTestPlan

Le code ci-dessous utilise Apache HttpClient pour exporter les tests.

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

long sessionId = mWebServiceHelper.getSessionId();

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

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

Pour télécharger Apache HttpComponents, visitez le site <http://hc.apache.org/downloads.cgi>. Reportez-vous à la documentation du composant pour connaître les bibliothèques requises.

Interface updateTestPlan

L'interface `updateTestPlan` permet de mettre à jour les tests avec les nœuds racine existants des fichiers XML. La réponse HTTP de l'appel comprend la structure XML des tests modifiés. Les identifiants des nouveaux nœuds sont disponibles dans la structure du test XML mis à jour.

Le tableau ci-dessous présente les paramètres de l'interface `updateTestPlan`.

URL de l'interface	Paramètre	Descriptions
http://<front-end URL>/servicesExchange?hid=updateTestPlan	sid	ID de session – Authentification de l'utilisateur
	userName	<i>Facultatif</i> : nom d'utilisateur – Utilisé à la place de sid
	passWord	<i>Facultatif</i> : mot de passe utilisateur – Utilisé à la place de sid

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

Le fichier de définition du schéma XML qui sert à valider les plans de test peut être téléchargé au moyen de l'URL du serveur de présentation, http://<URL serveur de présentation>/silkroot/xsl/testplan.xsd, ou copié à partir du dossier d'installation du serveur de présentation, <dossier d'installation Silk Central>/wwwroot/silkroot/xsl/testplan.xsd.

Exemple de service Web updateTestPlan

Le code ci-dessous utilise Apache HttpClient pour mettre à jour les tests.

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

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

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

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

Il n'est possible de télécharger qu'une seule pièce jointe par demande. Pour télécharger Apache HttpComponents, visitez le site <http://hc.apache.org/downloads.cgi>. Reportez-vous à la documentation du composant pour connaître les bibliothèques requises.

Exemple de test

Le code suivant affiche un exemple de test pouvant être téléchargé dans Silk Central à l'aide des services createTestPlan et updateTestPlan.

```
<?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>

```

Interface createRequirements

L'interface `createRequirements` permet de créer des exigences. La réponse HTTP de l'appel comprend la structure XML des exigences modifiées. Les identifiants des nouveaux nœuds sont disponibles dans la structure de l'exigence XML mise à jour.

Le tableau ci-dessous présente les paramètres de l'interface `createRequirements`.

URL de l'interface	Paramètre	Descriptions
http://<front-end URL>/servicesExchange?hid=createRequirements	sid	ID de session – Authentification de l'utilisateur
	parentNodeID	ID du conteneur auquel la nouvelle exigence est ajoutée dans l'arborescence des exigences
	userName	<i>Facultatif</i> : nom d'utilisateur – Utilisé à la place de sid
	passWord	<i>Facultatif</i> : mot de passe utilisateur – Utilisé à la place de sid

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

Le fichier de définition du schéma XML qui sert à valider les exigences peut être téléchargé au moyen de l'URL du serveur de présentation, http://<URL serveur de présentation>/silkroot/xsl/requirements.xsd, ou copié à partir du dossier d'installation du serveur de présentation, <dossier d'installation Silk Central>/wwwroot/silkroot/xsl/requirements.xsd.

Exemple de service Web createRequirements

Le code suivant utilise Apache HttpClient pour créer des exigences.

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

long sessionID = mWebServiceHelper.getSessionId();

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

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

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

Il n'est possible de télécharger qu'une seule pièce jointe par demande. Pour télécharger Apache HttpComponents, visitez le site <http://hc.apache.org/downloads.cgi>. Reportez-vous à la documentation du composant pour connaître les bibliothèques requises.

Exemple d'exigence

Le code suivant affiche un exemple d'exigence pouvant être téléchargé dans Silk Central à l'aide des services createRequirements, updateRequirements et updateRequirementsByExtID.

```
<?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>/silkrout/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>

```

Interface exportRequirements

L'interface `exportRequirements` permet d'exporter les exigences sous forme de fichiers XML. Le tableau ci-dessous présente les paramètres de l'interface `exportRequirements`.

URL de l'interface	Paramètre	Descriptions
http://<front-end URL>/servicesExchange?hid=exportRequirements	sid	ID de session – Authentification de l'utilisateur
	nodeID	Le nœud associé à cet ID et, de façon récursive, tous les nœuds enfants de ce nœud sont exportés.
	userName	<i>Facultatif</i> : nom d'utilisateur – Utilisé à la place de sid
	passWord	<i>Facultatif</i> : mot de passe utilisateur – Utilisé à la place de sid
	includeObsolete	<i>Facultatif</i> : Spécifiez true ou false. Si vous omettez ce champ, la valeur par défaut est true. Spécifiez false pour exclure les exigences obsolètes.

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

Exemple de service Web exportRequirements

Le code ci-dessous utilise Apache HttpClient pour exporter les exigences.

```

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

long sessionId = mWebServiceHelper.getSessionId();

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

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

```

Pour télécharger Apache HttpComponents, visitez le site <http://hc.apache.org/downloads.cgi>. Reportez-vous à la documentation du composant pour connaître les bibliothèques requises.

Interface updateRequirements

L'interface `updateRequirements` permet de mettre à jour les exigences avec les nœuds racine existants des fichiers XML. Les exigences sont identifiées par leur ID de nœud Silk Central interne dans l'arborescence Exigences. Le nœud de l'arborescence Exigences et tous les nœuds enfants sont mis à jour. De nouveaux nœuds sont ajoutés, les nœuds manquants sont définis comme étant obsolètes et les nœuds déplacés sont également placés dans Silk Central. La réponse HTTP de l'appel comprend la structure XML des exigences modifiées. Les identifiants des nouveaux nœuds sont disponibles dans la structure de l'exigence XML mise à jour.

Le tableau ci-dessous présente les paramètres de l'interface `updateRequirements`.

URL de l'interface	Paramètre	Descriptions
http://<front-end URL>/servicesExchange?hid=updateRequirements	sid	ID de session – Authentification de l'utilisateur
	userName	<i>Facultatif</i> : nom d'utilisateur – Utilisé à la place de sid
	passWord	<i>Facultatif</i> : mot de passe utilisateur – Utilisé à la place de sid

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

Le fichier de définition du schéma XML qui sert à valider les exigences peut être téléchargé au moyen de l'URL du serveur de présentation, `http://<URL serveur de présentation>/silkroot/xsl/requirements.xsd`, ou copié à partir du dossier d'installation du serveur de présentation, `<dossier d'installation Silk Central>/wwwroot/silkroot/xsl/requirements.xsd`.

Exemple de service Web updateRequirements

Le code ci-dessous utilise Apache HttpClient pour mettre à jour les exigences.

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

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

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

filePost.setRequestEntity(new MultipartRequestEntity(parts,
    filePost.getParams()));
client.getHttpClientConnectionManager().getParams().setConnectionTimeout(60000);
int status = client.executeMethod(filePost);
```

```
System.out.println(filePost.getStatusLine());  
String responseXml = filePost.getResponseBodyAsString();
```

Il n'est possible de télécharger qu'une seule pièce jointe par demande. Pour télécharger Apache HttpComponents, visitez le site <http://hc.apache.org/downloads.cgi>. Reportez-vous à la documentation du composant pour connaître les bibliothèques requises.

Exemple d'exigence

Le code suivant affiche un exemple d'exigence pouvant être téléchargé dans Silk Central à l'aide des services `createRequirements`, `updateRequirements` et `updateRequirementsByExtID`.

```
<?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>
```

Interface `updateRequirementsByExtID`

L'interface `updateRequirementsByExtID` permet de mettre à jour les exigences avec les nœuds racine existants des fichiers XML. Les exigences sont identifiées par des ID externes. Le nœud de l'arborescence Exigences et tous les nœuds enfants sont mis à jour. De nouveaux nœuds sont ajoutés, les nœuds manquants sont définis comme étant obsolètes et les nœuds déplacés sont également placés dans Silk Central. La réponse HTTP de l'appel comprend la structure XML des exigences modifiées. Les identifiants des nouveaux nœuds sont disponibles dans la structure de l'exigence XML mise à jour.

Le tableau ci-dessous présente les paramètres de l'interface `updateRequirementsByExtID`.

URL de l'interface	Paramètre	Descriptions
http://<front-end URL>/servicesExchange?hid=updateRequirementsByExtID	sid	ID de session – Authentification de l'utilisateur
	nodeID	ID du nœud de l'arborescence des exigences à mettre à jour.
	userName	<i>Facultatif</i> : nom d'utilisateur – Utilisé à la place de sid
	passWord	<i>Facultatif</i> : mot de passe utilisateur – Utilisé à la place de sid

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

Le fichier de définition du schéma XML qui sert à valider les exigences peut être téléchargé au moyen de l'URL du serveur de présentation, http://<URL serveur de présentation>/silkroot/xsl/requirements.xsd, ou copié à partir du dossier d'installation du serveur de présentation, <dossier d'installation Silk Central>/wwwroot/silkroot/xsl/requirements.xsd.

Exemple de service Web updateRequirementsByExtID

Le code ci-dessous utilise Apache HttpClient pour mettre à jour les exigences.

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

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

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

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

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

Il n'est possible de télécharger qu'une seule pièce jointe par demande. Pour télécharger Apache HttpComponents, visitez le site <http://hc.apache.org/downloads.cgi>. Reportez-vous à la documentation du composant pour connaître les bibliothèques requises.

Exemple d'exigence

Le code suivant affiche un exemple d'exigence pouvant être téléchargé dans Silk Central à l'aide des services `createRequirements`, `updateRequirements` et `updateRequirementsByExtID`.

```
<?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>/silkkroot/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>
  </Requirement>
  <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>
```

Interface `createExecutionDefinitions`

L'interface `createExecutionDefinitions` permet de créer des plans d'exécution. La réponse HTTP de l'appel comprend la structure XML des plans d'exécution modifiés. Les identifiants des nouveaux nœuds sont disponibles dans la structure du plan d'exécution XML mis à jour.

Le tableau ci-dessous présente les paramètres de l'interface `createExecutionDefinitions`.

URL de l'interface	Paramètre	Descriptions
http://<front-end URL>/ servicesExchange? hid=createExecutionDefinitions	sid	ID de session – Authentification de l'utilisateur
	parentNodeID	ID du nœud auquel le nouveau plan d'exécution est ajouté dans l'arborescence des exécutions
	userName	<i>Facultatif</i> : nom d'utilisateur – Utilisé à la place de sid
	passWord	<i>Facultatif</i> : mot de passe utilisateur – Utilisé à la place de sid

Exemple : `http://<front-end URL>/servicesExchange?`

`hid=createExecutionDefinitions&parentNodeID=<id>&userName=<user>&passWord=<password>`

Le fichier de définition du schéma XML qui sert à valider les exécutions peut être téléchargé au moyen de l'URL du serveur de présentation, `http://<URL serveur de présentation>/silkroot/xsl/executionplan.xsd`, ou copié à partir du dossier d'installation du serveur de présentation, `<dossier d'installation Silk Central>/wwwroot/silkroot/xsl/executionplan.xsd`.

Exemple de service Web createExecutionDefinitions

Le code suivant utilise Apache HttpClient pour créer des plans d'exécution.

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

long sessionID = mWebServiceHelper.getSessionID();

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

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

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

Il n'est possible de télécharger qu'une seule pièce jointe par demande. Pour télécharger Apache HttpComponents, visitez le site <http://hc.apache.org/downloads.cgi>. Reportez-vous à la documentation du composant pour connaître les bibliothèques requises.

Exemple de plan d'exécution

Le code suivant affiche un exemple de plan d'exécution pouvant être téléchargé dans Silk Central à l'aide des services `createExecutionDefinitions` et `updateExecutionDefinitions`. L'exemple permet de créer une planification personnalisée pour l'une des définitions d'exécution et d'assigner des tests à un plan d'exécution, via une assignation manuelle et un filtre. L'exemple permet également de créer une suite de configurations avec des configurations.

```
<?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>

```

Interface exportExecutionDefinitions

L'interface `exportExecutionDefinitions` permet d'exporter les plans d'exécution sous forme de fichiers XML. Le tableau ci-dessous présente les paramètres de l'interface `exportExecutionDefinitions`.

URL de l'interface	Paramètre	Descriptions
http://<front-end URL>/ servicesExchange? hid=exportExecutionDefinitions	sid	ID de session – Authentification de l'utilisateur
	nodeID	Le nœud associé à cet ID et, de façon récursive, tous les nœuds enfants de ce nœud sont exportés.

URL de l'interface	Paramètre	Descriptions
	userName	<i>Facultatif</i> : nom d'utilisateur – Utilisé à la place de sid
	passWord	<i>Facultatif</i> : mot de passe utilisateur – Utilisé à la place de sid

Exemple : `http://<front-end URL>/servicesExchange?`

`hid=exportExecutionDefinitions&nodeID=<id>&userName=<user>&passWord=<password>`

Exemple de service Web exportExecutionDefinitions

Le code ci-dessous utilise Apache HttpClient pour exporter les plans d'exécution.

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

long sessionId = mWebServiceHelper.getSessionId();

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

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

Pour télécharger Apache HttpComponents, visitez le site <http://hc.apache.org/downloads.cgi>. Reportez-vous à la documentation du composant pour connaître les bibliothèques requises.

Interface updateExecutionDefinitions

L'interface `updateExecutionDefinitions` permet de mettre à jour les plans d'exécution des fichiers XML. La réponse HTTP de l'appel comprend la structure XML des plans d'exécution modifiés. Les identifiants des nouveaux nœuds sont disponibles dans la structure du plan d'exécution XML mis à jour.

Le tableau ci-dessous présente les paramètres de l'interface `updateExecutionDefinitions`.

URL de l'interface	Paramètre	Descriptions
<code>http://<front-end URL>/servicesExchange?hid=updateExecutionDefinitions</code>	sid	ID de session – Authentification de l'utilisateur
	userName	<i>Facultatif</i> : nom d'utilisateur – Utilisé à la place de sid
	passWord	<i>Facultatif</i> : mot de passe utilisateur – Utilisé à la place de sid

Exemple : `http://<front-end URL>/servicesExchange?`

`hid=updateExecutionDefinitions&userName=<user>&passWord=<password>`

Le fichier de définition du schéma XML qui sert à valider les exécutions peut être téléchargé au moyen de l'URL du serveur de présentation, `http://<URL serveur de présentation>/silkroot/xsl/executionplan.xsd`, ou copié à partir du dossier d'installation du serveur de présentation, <dossier d'installation Silk Central>/wwwroot/silkroot/xsl/executionplan.xsd.

Exemple de service Web updateExecutionDefinitions

Le code ci-dessous utilise Apache HttpClient pour mettre à jour les plans d'exécution.

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

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

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

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

Il n'est possible de télécharger qu'une seule pièce jointe par demande. Pour télécharger Apache HttpComponents, visitez le site <http://hc.apache.org/downloads.cgi>. Reportez-vous à la documentation du composant pour connaître les bibliothèques requises.

Exemple de plan d'exécution

Le code suivant affiche un exemple de plan d'exécution pouvant être téléchargé dans Silk Central à l'aide des services `createExecutionDefinitions` et `updateExecutionDefinitions`. L'exemple permet de créer une planification personnalisée pour l'une des définitions d'exécution et d'assigner des tests à un plan d'exécution, via une assignation manuelle et un filtre. L'exemple permet également de créer une suite de configurations avec des configurations.

```
<?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>
    </ExecDef>
  </Folder>
</ExecutionPlan>
```

```

    <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>

```

Interface createLibraries

L'interface `createLibraries` permet de créer des bibliothèques. La réponse HTTP de l'appel comprend la structure XML des bibliothèques modifiées. Les identifiants des nouveaux nœuds sont disponibles dans la structure de la bibliothèque XML mise à jour.

Le tableau ci-dessous présente les paramètres de l'interface `createLibraries`.

URL de l'interface	Paramètre	Descriptions
http://<front-end URL>/servicesExchange?hid=createLibraries	sid	ID de session – Authentification de l'utilisateur
	userName	<i>Facultatif</i> : nom d'utilisateur – Utilisé à la place de sid
	passWord	<i>Facultatif</i> : mot de passe utilisateur – Utilisé à la place de sid

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

Le fichier de définition du schéma XML qui sert à valider les bibliothèques peut être téléchargé au moyen de l'URL du serveur de présentation, `http://<URL serveur de présentation>/silkroot/xsl/libraries.xsd`, ou copié à partir du dossier d'installation du serveur de présentation, `<dossier d'installation Silk Central>/wwwroot/silkroot/xsl/libraries.xsd`.

Exemple de service Web createLibraries

Le code suivant utilise Apache HttpClient pour créer des bibliothèques.

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

long sessionID = mWebServiceHelper.getSessionId();

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

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

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

Pour télécharger Apache HttpComponents, visitez le site <http://hc.apache.org/downloads.cgi>. Reportez-vous à la documentation du composant pour connaître les bibliothèques requises.

Exemple de bibliothèques

Le code suivant affiche un exemple de bibliothèque pouvant être téléchargé dans Silk Central à l'aide du service createLibraries. L'utilisation d'une nouvelle bibliothèque n'est pas limitée à certains projets, à moins qu'un ou plusieurs projets soient définis dans la section GrantedProjects.

```
<?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>Successful 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>

```

Interface exportLibraryStructure

L'interface `exportLibraryStructure` permet d'exporter des objets bibliothèques, dossiers et pas de tests partagés sous forme de fichiers XML. Le tableau ci-dessous présente les paramètres de l'interface `exportLibraryStructure`.

URL de l'interface	Paramètre	Descriptions
<a href="http://<front-end URL>/servicesExchange?hid=exportLibraryStructure">http://<front-end URL>/servicesExchange?hid=exportLibraryStructure	sid	ID de session – Authentification de l'utilisateur
	userName	<i>Facultatif</i> : nom d'utilisateur – Utilisé à la place de sid
	passWord	<i>Facultatif</i> : mot de passe utilisateur – Utilisé à la place de sid
	nodeID	Dossier ou nœud de l'arborescence des bibliothèques qui doit être exporté. Les ID des nœuds de pas de tests partagés ne sont pas autorisés.

Exemple : [http://<front-end URL>/servicesExchange?](http://<front-end URL>/servicesExchange?hid=exportLibraryStructure&userName=<user>&passWord=<password>&nodeID=<id>)

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

Exemple de service Web exportLibraryStructure

Le code ci-dessous utilise Apache HttpClient pour exporter les bibliothèques.

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

```

long sessionID = mWebServiceHelper.getSessionId();

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

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

```

Pour télécharger Apache HttpComponents, visitez le site <http://hc.apache.org/downloads.cgi>. Reportez-vous à la documentation du composant pour connaître les bibliothèques requises.

Interface exportLibraryStructureWithoutSteps

L'interface `exportLibraryStructureWithoutSteps` permet d'exporter des objets bibliothèques, dossiers et pas de tests partagés sous forme de fichiers XML. Les pas de tests inclus dans les objets pas de tests partagés ne sont pas exportés. Le tableau ci-dessous présente les paramètres de l'interface `exportLibraryStructureWithoutSteps`.

URL de l'interface	Paramètre	Descriptions
http://<front-end URL>/servicesExchange?hid=exportLibraryStructureWithoutSteps	sid	ID de session – Authentification de l'utilisateur
	userName	<i>Facultatif</i> : nom d'utilisateur – Utilisé à la place de sid
	passWord	<i>Facultatif</i> : mot de passe utilisateur – Utilisé à la place de sid
	nodeID	Dossier ou nœud de l'arborescence des bibliothèques qui doit être exporté. Les ID des nœuds de pas de tests partagés ne sont pas autorisés.

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

Exemple de service Web exportLibraryStructureWithoutSteps

Le code ci-dessous utilise Apache HttpClient pour exporter les bibliothèques.

```

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

long sessionID = mWebServiceHelper.getSessionId();

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

```

```

"exportLibraryStructureWithoutSteps", sessionID, NODE_ID));

HttpClient client = new HttpClient();
client.getHttpClientManager().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);

```

Pour télécharger Apache HttpComponents, visitez le site <http://hc.apache.org/downloads.cgi>. Reportez-vous à la documentation du composant pour connaître les bibliothèques requises.

Interface getLibraryInfoByName

L'interface `getLibraryInfoByName` renvoie l'ID, le nom et la description de toutes les bibliothèques portant un nom défini. L'interface renvoie uniquement les propriétés des bibliothèques, pas leur structure. Le tableau ci-dessous présente les paramètres de l'interface `getLibraryInfoByName`.

URL de l'interface	Paramètre	Descriptions
http://<front-end URL>/servicesExchange? hid=getLibraryInfoByName	sid	ID de session – Authentification de l'utilisateur
	userName	<i>Facultatif</i> : nom d'utilisateur – Utilisé à la place de sid
	passWord	<i>Facultatif</i> : mot de passe utilisateur – Utilisé à la place de sid
	libraryName	Nom de la bibliothèque

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

Exemple de service Web getLibraryInfoByName

Le code suivant utilise Apache HttpClient pour obtenir les informations de la bibliothèque.

```

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

long sessionID = mWebServiceHelper.getSessionId();

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

HttpClient client = new HttpClient();
client.getHttpClientManager().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);
```

Pour télécharger Apache HttpComponents, visitez le site <http://hc.apache.org/downloads.cgi>. Reportez-vous à la documentation du composant pour connaître les bibliothèques requises.

Web Service Demo Client

L'outil Web Service Demo Client illustre le mode d'utilisation de Silk Central Web Services. Téléchargez le client depuis **Aide > Outils**.

L'outil Web Service Demo Client affiche tous les attributs disponibles sous **Tests > Gérer les Attributs de Test** pour chaque test et toutes les propriétés pour chaque type de test disponible.



Attention: L'outil Web Service Demo Client est destiné à illustrer l'utilisation des services Web. N'utilisez pas le client de démonstration dans un environnement de production.

Index

A

- Apache Axis 28
- API
 - intégration de couverture de code 7
- API, structure
 - plug-in de type de test tiers 19
- authentification
 - Service Web 33

C

- capture vidéo
 - indication de la fin 25
 - indication du début 25
- classe 16
- client de démonstration
 - interface de service Web 61
- cloud, plug-in 27
- couverture de code
 - API 7
- createExecutionDefinitions
 - exemple 49
 - interface 49
- createLibraries
 - exemple 56
 - interface 56
- createRequirements
 - exemple 43
 - interface 43
- createTestPlan
 - exemple 37
 - interface 37

D

- déploiement
 - plug-in 6
 - plug-in de type de test tiers 25
- distribution
 - plug-in 6
- données d'authentification 33

E

- espèce
 - plug-in 6
- exécuteur de processus
 - exemple de code 20
- exemple de code
 - plug-in de type de test tiers 20
- exportExecutionDefinitions
 - exemple 52
 - interface 52
- exportLibraryStructure
 - exemple 58
 - interface 58
- exportLibraryStructureWithoutSteps
 - exemple 59
 - interface 59

- exportRequirements
 - exemple 45
 - interface 45
- exportTestPlan
 - exemple 40
 - interface 40

G

- getLibraryInfoByName
 - exemple 60
 - interface 60

I

- icône
 - personnaliser 24
- icône personnalisée
 - plug-in de type de test tiers 24
- implémentation
 - plug-in de type de test tiers 18
- intégration
 - plug-in de type de test tiers 18
- intégration cloud 27
- intégration de la gestion des exigences 17
- intégration du suivi des incidents
 - vue d'ensemble 16
- interface
 - createExecutionDefinitions 49
 - createLibraries 56
 - createRequirements 43
 - createTestPlan 37
 - exportExecutionDefinitions 52
 - exportLibraryStructure 58
 - exportLibraryStructureWithoutSteps 59
 - exportRequirements 45
 - exportTestPlan 40
 - getLibraryInfoByName 60
 - Interface Java 16
 - référentiel tiers 14
 - reportData 34
 - TMAAttach 35
 - updateExecutionDefinitions 53
 - updateRequirements 46
 - updateRequirementsByExtID 47
 - updateTestPlan 40
- Interface de service Web
 - démarrage rapide 29
 - didacticiel 30
- Interface Java 16

M

- méta-informations
 - plug-in de type de test tiers 23
- méta-informations des propriétés des chaînes
 - plug-in de type de test tiers 24

- méta-informations des propriétés des fichiers
 - plug-in de type de test tiers 24
- méta-informations des propriétés générales
 - plug-in de type de test tiers 23
- mise en package
 - plug-in de type de test tiers 18

P

- paramètre prédéfini
 - transmission à des plug-ins de type de test tiers 19
- plug-in
 - cloud 27
 - compilation 6
 - déploiement 6
 - distribution 6
 - espèce 6
 - exigences 17
 - exigences, gestion 17
 - référentiel tiers 14
 - suivi des incidents 16
 - vue d'ensemble 6
- plug-in d'exigences 17
- plug-in de type de test tiers
 - API, structure 19
 - exemple de code 20
 - fichier de configuration XML 23, 25
 - icône personnalisée 24
 - implémentation 18
 - intégration 18
 - méta-informations 23
 - méta-informations des propriétés des chaînes 24
 - méta-informations des propriétés des fichiers 24
 - méta-informations des propriétés générales 23
 - mise en package 18
 - transmission de paramètres prédéfinis 19
- plug-in, compilation 6

R

- référentiel tiers
 - conventions de l'interface 15
 - intégration 14
 - interface 14
 - plug-in 14
- reportData
 - exemple 34

- interface 34

S

- service Web
 - cas d'utilisation, exemple 31
 - conditions préalables 29
- Service Web
 - cloud 27
 - disponible 33
 - exigences, gestion 17
 - vue d'ensemble 28
- Services Exchange 34
- session, gestion 33
- SOAP
 - enveloppe 30
 - pile 28
- suivi des incidents
 - plug-in 16
- synchronisation
 - exigence 17

T

- TMAAttach
 - exemple 35
 - interface 35

U

- updateExecutionDefinitions
 - exemple 53
 - interface 53
- updateRequirements
 - exemple 46
 - interface 46
- updateRequirementsByExtID
 - exemple 47
 - interface 47
- updateTestPlan
 - exemple 40
 - interface 40

W

- Web Service Demo Client 61